საქართველოს საპატრიარქოს წმ. ანდრია პირველწოდებულის სახელობის
ქართული უნივერსიტეტი

თ. კივილაძე

დაპროგრამების განაწილებული გარემო

(ლექციების კურსი)

თბილისი 2015

St. Andrew the First-Called

GEORGIAN UNIVERSITY

Of the Patriarchate of Georgia

Teimuraz Kiviladze

Programming in Distributed Environment

(Course of lectures)

Recommended by the Research Council of the Georgian University Department of Informatics, Mathematics and Natural Sciences 23 October, protocol № 4

Tbilisi 2015

UDC (უაკ) 004.42

კ-49

ლექციათა კურსში განხილულია თანამედროვე კომპიუტერებსა და კომპიუტერულ სისტემებშიგამოთვლების გადანაწილების შესაძლებლობები. განხილულია გამოთვლების გადანაწილების შესაძლებლობები როგორც ჩვეულებრივი პერსონალური კომპიუტერის პირობებში, ასევე მრავალპროცესორულ და მრავალკომპიუტერულ სისტემებში. შემოთავაზებულია აგრეთვე დაპროგრამების განაწილებული გარემოს პირობებში მატრიცული გამოთვლების ზოგიერთი პარალელური ალგორითმი.

ლექციათა კურსი პირველ რიგში განკუთვნილია ინფორმატიკის სპეციალობის მაგისტრანტებისათვის, აგრეთვე დოქტორანტებისათვის, დარგის სპეციალისტებისათვის და სტუდენტებისათვის, რომლებიც დაინტერესებული არიან გამოთვლების გადანაწილებით დაპროგრამების პარალელურ გარემოში.

რეკომენდებულია ქართული უნივერსიტეტის
ინფორმატიკის, მათემატიკისა და საბუნებისმეტყველო
მეცნიერებათა ფაკულტეტის სამეცნიერო საბჭოს მიერ

23 ოქტომბერი, ოქმი №4

რეცენზენტი: პროფესორი პაატა წერეთელი

რედაქტორი: პროფესორი გურამ ცერცვაძე

გამომცემლობა „ქართული უნივერსიტეტი“

ISBN 978-9941-9450-4-5 (PDF)

http://www.sangu.edu.ge

UDC 004.42

k-49

The possibilities of calculation distribution in modern computers and computer systems are presented in the course of lectures. The possibilities of calculation distribution are discussed in the conditions of usual personal computers as well as multiprocessor and multicomputer systems. Some parallel algorithms of matrix computation conditions of programming in distributed environment are also presented.

The course of lectures is first of all made for the magistrates as well as the doctorates of informatics specialty, IT specialists and students, interested in the distribution of calculations in the parallel environment of programming.

Recommended by the Scientific Council of the Georgian University
Department of Informatics, Mathematics and Natural Sciences

23 October, 2015, Protocol No.4

Reviewer: Professor Paata Tsereteli

Editor: Professor Guram Tsertsvadze

Published by Georgian University

ISBN 978-9941-9450-4-5 (PDF)

http://www.sangu.edu.ge

სარჩევი

შესავალი

კომპიუტერების მუშაობის სწრაფქმედება მუდმივად იზრდება, მაგრამ ასევე იზრდება მოთხოვნები მათ მიმართ. ასტრონომები ცდილობენ აღადგინონ სამყაროს ისტორია დიდი აფეთქების მომენტიდან დღემდე. საფრენი აპარატების მწარმოებლებს სჭირდებათ რთული ექსპერიმენტების ჩატარება ძვირადღირებულ აეროდინამიურ მილებში. მათთვის გაცილებით მოსახერხებელი და იაფი იქნებოდა ამ პროცესების კომპიუტერული მოდელირება. ასეთი მდგომარეობაა მეცნიერებისა და ტექნიკის ყველა დარგში. რაც არ უნდა სწრაფქმედი იყოს დღევანდელი კომპიუტერი, პრაქტიკული ამოცანებიდან გამომდინარე საჭიროა კიდევ უფრო სწრაფქმედი კომპიუტერების გამოყენება.

კომპიუტერებში მართალია ტაქტური სიხშირე მუდმივად იზრდება, მაგრამ შეუძლებელია კომუტაციის სიჩქარის უსასრულოდ გაზრდა. პრობლემა იმაში მდგომარეობს, რომ შეუძლებელია ელექტრონები ვაიძულოთ იმოძრაონ სინათლის სიჩქარეზე მეტი სიჩქარით. სწრაფი კომპიუტერების ასაგებად საჭირო ხდება სულ უფრო მეტი და მეტი ტრანზისტორის გამოყენება. მართალია, ტრანზისტორების ზომები მუდმივად მცირდება, მაგრამ ამასაც აქვს თავისი ზღვარი. და კიდევ, კომპიუტერებში ტრანზისტორების რაოდენობის სწრაფი ზრდა იწვევს მოხმარებული ელექტროენერგიის და გამოყოფილი სითბოს რაოდენობის ზრდას.

შექმნილი მდგომარეობიდან გამომდინარე, ზესწრაფი კომპიუტერების შესაქმნელად საჭირო გახდა კომპიუტერებში პარალელური მოქმედების პრინციპზე გადასვლა – შემდგომში პარალელური კომპიუტერები. შეუძლებელია ავაგოთ კომპიუტერი ერთი პროცესორით, რომელსაც ექნება ციკლის დრო 0,001 ნწმ, მაგრამ შესაძლებელია ავაგოთ კომპიუტერი 1000 პროცესორით, რომელთაგან ყოველი პროცესორის ციკლის დრო იქნება 1 ნწმ. მართალია, თითოეული პროცესორის სწრაფქმედება არ იქნება ძალიან მაღალი, მაგრამ ჯამური სწრაფქმედება იქნება მაღალი.

კომპიუტერში პარალელური მოქმედების პრინციპი შეიძლება განხორციელებული იქნას სხვადასხვა დონეზე. ყველაზე დაბალი დონე ეს არის პროცესორის დონე. აქ შესაძლებელია კონვეიერების გამოყენება და სუპერსკალარული არქიტექტურის გამოყენება რამდენიმე ფუნქციონალური ბლოკით. პარალელიზმის მაჩვენებლის გაუმჯობესება შესაძლებელია აგრეთვე მანქანურ ბრძანებებში სიტყვის სიგრძის გაზრდითაც. დამატებითი ფუნქციების საშუალებით შესაძლებელია პროცესორს „შევასწავლოთ“ რამდენიმე პროგრამული ნაკადის ერთდროულად დამუშავება. და ბოლოს, ერთ მიკროსქემაში ჩავსვათ რამდენიმე პროცესორი. მაგრამ, ყველა ამ მეთოდის ერთდროული გამოყენება შესაძლებელს ხდის კომპიუტერის სწრაფქმედება გავზარდოთ მხოლოდ 10–ჯერ, კლასიკური, მიმდევრობითი პრინციპით მომუშავე პროცესორთან შედარებით.

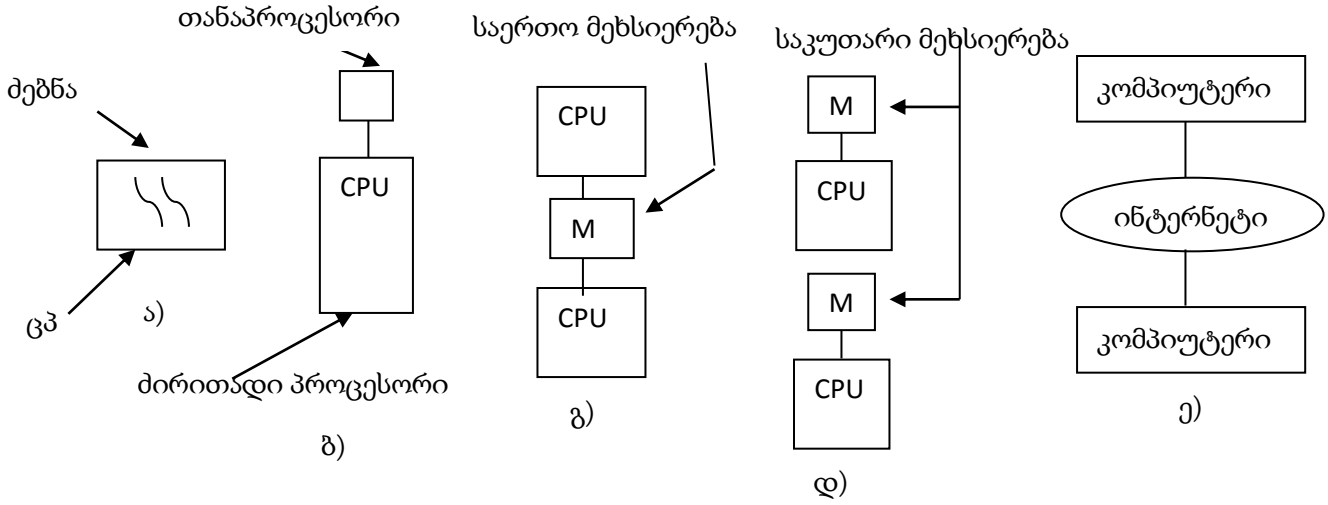
შემდეგი დონე, შესაძლებელია სისტემას მივუერთოთ ცენტრალური პროცესორების გარე პლატები, გაუმჯობესებული გამოთვლითი შესაძლებლობებით. როგორც წესი, მიერთებულ პრცესორებში ხდება სპეციალური ფუნქციების რეალიზება, ისეთების როგორიცაა საქსელო პაკეტების დამუშავება, მულტიმედიური მონაცემების დამუშავება, კრიპტოგრაფია და ა.შ. ამ დამატებითი პროცესორების საშუალებით კომპიუტერის საერთო მწარმოებლურობა შესაძლებელია გაზრდილი იქნას კიდევ 5–10–ჯერ.

კომპიუტერის მწარმოებლურობის 100–ჯერ, 1000–ჯერ ან 1000000–ჯერ გასაზრდელად საჭიროა ერთ სისტემაში გაერთიანებული იქნას პროცესორების დიდი რაოდენობა და უზრუნველ-

ყოფილი იქნას მათი ეფექტური მუშაობა და თანამოქმედება. ეს პრინციპი რეალიზებულია დიდ მულტიპროცესორულ სისტემებში და მულტიკომპიუტერებში (კლასტერული კომპიუტერები). ბუნებრივია, რომ პროცესორების ასეთი დიდი რაოდენობის ერთ სისტემაში გაერთიანება წარმოქმნის კიდევ ახალ პრობლემებს, რომლებიც გადაწყვეტილ უნდა იქნას.

და ბოლოს, დღეს უკვე შესაძლებელი გახდა ინტერნეტის საშუალებით გაერთიანებული იქნას მთელი ორგანიზაციები. ასეთ შემთხვევაში, წარმოიქმნება სუსტად შეკავშირებული განაწილებული გამოთვლითი ბადეები. მათი განვითარება ახლა მიმდინარეობს, მაგრამ მათ დიდი პოტენციალი გააჩნიათ.

როდესაც ორი პროცესორი ან ორი სისტემა ერთმანეთის გვერდიგვერდ დგას დაერთმანეთთან ახორციელებენ მონაცემების დიდი ნაკადების სწრაფ გაცვლას, ასეთ სისტემებს უწოდებენ **მჭიდროდ შეკავშირებულ** სისტემებს (tightly coupled). შესაბამისად, როდესაც ორი პროცესორი ან ორი სისტემა ერთმანეთისაგან დიდ მანძილითაა დაშორებული და მათ შორის მონაცემების გაცვლა ხდება მცირე მოცულობით და დროის დიდი შეყოვნებით, ასეთ სისტემებს უწოდებენ **სუსტად შეკავშირებულ** სისტემებს (loosely coupled). სხვადასხვა სახეობის პარალელიზმის მქონე სისტემები ნაჩვენებია ნახ. 1–ზე.



ნახ. 1. სხვადასხვა პარალელიზმის მქონე სისტემები

1. პარალელიზმი ერთ კომპიუტერში

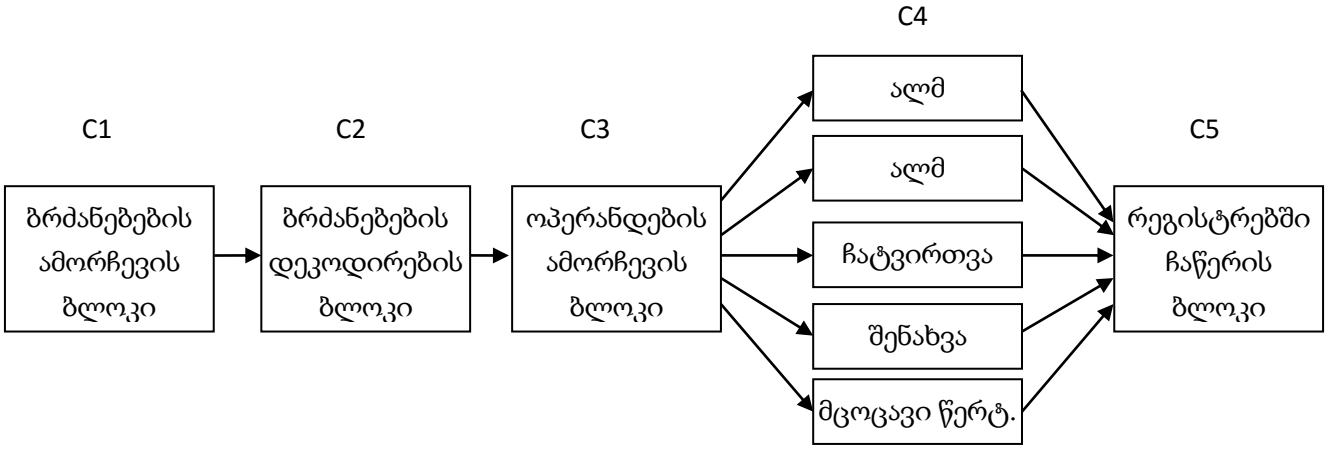
1.1. პროცესორსშიდა პარალელიზმი

მიკროსქემის მწარმოებლურობის გაზრდის ერთერთი აშკარა გზაა გავზარდოთ მასში დროის ერთეულში შესრულებული ოპერაციების რაოდენობა. ამ თავში განხილული იქნება სწრაფქმედების გაზრდისათვის დაპარალელება მიკროსქემების დონეზე, კერძოდ, დაპარალელება ბრძანებების დონეზე, მრავალნაკადურობა და მიკროსქემის შიგნით რამდენიმე პროცესორის განთავსება. ეს მეთოდები მართალია ერთმანეთისაგან განსხვავებული მეთოდებია, მაგრამ მათ აერთიანებთ საბაზო პრინციპი – ოპერაციების შესრულების დროში „შემჭიდროვება“.

1.1.1. პარალელიზმი ბრძანებების დონეზე

დაბალ დონეზე პარალელიზმის შემთხვევაში ერთი ტაქტური სიგნალის პერიოდში ხდება ერთდროულად რამდენიმე ბრძანების გამოძახება. ამ პრინციპით რეალიზებული პროცესორები იყოფა ორ კატეგორიად: სუპერსკალარულ და VLIW.

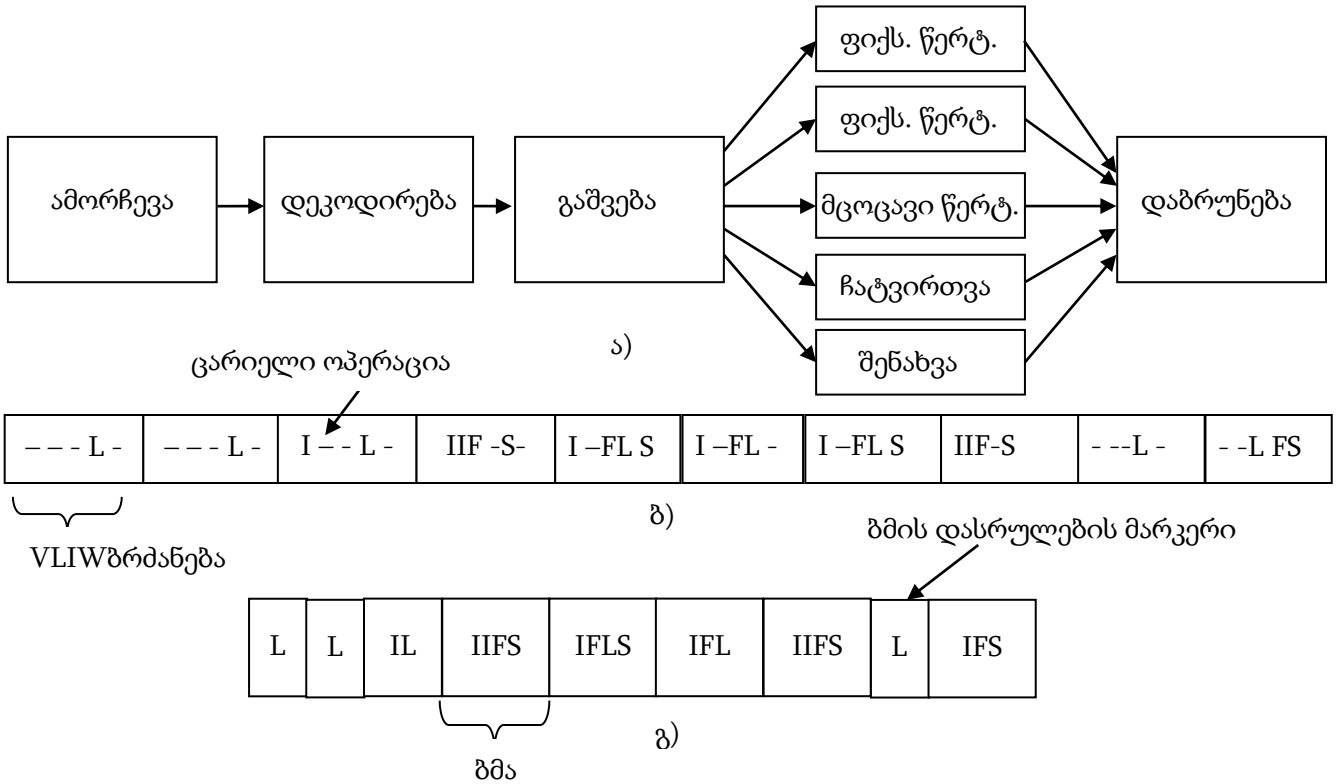
სუპერსკალარული პროცესორის სქემა მოყვანილია ნახ, 1.1.1–ზე. ყველაზე გავრცელებულ კონფიგურაციებში ბრძანება შესასრულებლად მზად უნდა იყოს კონვეიერის გარკვეულ საფეხურზე. სუპერსკალარულ პროცესორებს შეუძლიათ ერთი ტაქტური ციკლის განმავლობაში შეასრულონ რამდენიმე ბრძანება. ფაქტიურად შესრულებული ბრძანებების რაოდენობა დამოკიდებულია როგორც პროცესორის კონსტრუქციაზე, ასევე მიმდინარე სიტუაციაზე. აპარატული შეზღუდვები შესრულებული ბრძანებების რაოდენობას ზღუდავს რაოდენობით 2–დან 6–მდე. ამავდროულად, ბრძანების შესასრულებლად თუ საჭიროა ფუნქციონალური ბლოკი, რომელთანაც მოცემულ მომენტში წვდომა არაა შესაძლებელი, ან ჯერ არ არის შესრულებული ბრძანება, რომლის შედეგიც საჭიროა მოცემული ბრძანების შესასრულებლად, მაშინ ასეთი ბრძანება რა თქმა უნდა გამოძახებული არ იქნება, თუნდაც ამის ფიზიკური შესაძლებლობა არსებობდეს.



ნახ. 1.1.1. სუპერსკალარული პროცესორი 5 ფუნქციონალური ბლოკით

ბრძანებების დონეზე პარალელიზმის მეორე სახეობის რეალიზება ხდება პროცესორებში **ზეგრძელი საბრძანებო სიტყვებით** (Very Long Instruction Word, VLIW). VLIW–სისტემები თავიდან მართლაც გამოირჩეოდნენ გრძელი საბრძანებო სიტყვებით, რომლებითაც მიმართვა ხდებოდა ერთდროულად რამდენიმე ფუნქციონალურ ბლოკთან. მაგალითისათვის განვიხილოთ ნახ. 1.1.2,ა–ზე გამოსახული კონვეიერი. იგი მოიცავს 5 ფუნქციონალურ ბლოკს და შეუძლია ერთდროულად შეასრულოს 2 ფიქსირებულწერტილიანი ოპერაცია, 1 ოპერაცია მცოცავი წერტილით, 1 შენახვის და 1 ჩატვირთვის ოპერაცია. ამ VLIW–სისტემის ერთი ბრძანება მოიცავს 5 ოპერაციის კოდს და 5 ოპერანდების წყვილს თითოეული ფუნქციონალური ბლოკისათვის. იმის გათალისწინებით, რომ ოპერაციის კოდი იკავებს 6 ბიტს, რეგისტრი – 5 ბიტს, ხოლო ოპერატიული მეხსიერების უჯრედი 32 ბიტს, ბრძანების საერთო სიგრძემ შეიძლება მიაღწიოს 134 ბიტს, რაც სრულებით არ არის ცოტა.

ასეთი გადაწყვეტა აღიარებული იქნა არცთუ მაინცდამაინც ხელსაყრელ გადაწყვეტილებად. საქმე იმაშია რომ, ყველა ბრძანებს არ შეეძლო მიემართა შესაბამისი ფუნქციონელური ბლოკებისათვის, შედეგად ჭარბად წარმოიქმნებოდა უაზრო ცარიელი ოპერაციები (ნახ. 1.1.2,ბ). თანამედროვე VLIW–სისტემებში გათვალისწინებული უნდა იქნას ბრძანებების მარკირების ბმების რაიმე მექანიზმი, მაგალითად გამოყენებული იქნას „ბმების დასრულების“ ბიტი (ნახ. 1.1.2,გ). პროცესორს შეუძლია შეარჩიოს და გაუშვას ბმა მთლიანად. ამ ბრძანებების ბმების მომზადების შერჩევის ამოცანის გადაწყვეტა ხდება კომპილატორის საშუალებით.



ნახ. 1.1.2. პროცესორის კონვეიერი (ა); VLI–ბრძანებების მიმდევრობა (ბ); ბრძანებების ნაკადი მონიშნული ბმებით (გ)

ფაქტობრივად VLIW–სისტემებში ბრძანებების თავსებადობის პრობლემის გადაწყვეტა ბრძანებების შესრულების პერიოდიდან გადატანილია კომპილაციის სტადიაზე. ამის შედეგად პრობლემის აპარატული გადაწყვეტა მარტივდება და იაფდება. გარდა ამისა, რადგანაც კომპილა-

ტორის მუშაობის პერიოდში არ არსებობს ხისტი დროითი შეზღუდვები, ბრძანებების ბმულების ფორმირება ხდება უფრო გააზრებულად, ვიდრე ეს მოხდებოდა მათი შესრულების დროს.

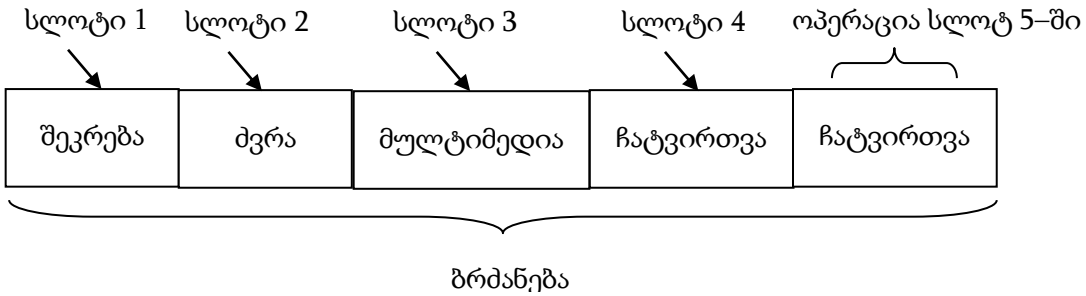
არ იქნება ზედმეტი აღვნიშნოთ, რომ პარალელიზმი ბრძანებების დონეზე არ წარმოადგენს დაბალი დონის პარალელიზმის ერთადერთ შესაძლო ფორმას. არსებობს აგრეთვე პარალელიზმი მეხსიერების დონეზე, რომელიც აგრეთვე მეხსიერებაში ითვალისწინებს მრავალი ოპერაციის ერთდროულად შესრულების შესაძლებლობას.

1.1.2. პროცესორი TriMedia

გავეცნოთ კომპანია Philips-ის მიერ წარმოებულ VLIW ტიპის TriMedia პროცესორს. TriMedia ეს არის გამოსახულებების წარმოსახვის მოწყობილობებისათვის განკუთვნილი ჩამონტაჟებული პროცესორი. გარდა ამისა ეს მოწყობილობა გამოიყენება აუდიო და ვიდეო მოწყობილობებისათვის, CD და DVD ჩამწერი მოწყობილობებისათვის, ინტერაქტიური ტელევიზორებისათვის, ციფრული ფოტოკამერებისათვის, ვიდეო კამერებისათვის და ა.შ.

TriMedia პროცესორის ერთ მანქანურ ბრძანებაში შეიძლება შედიოდეს 5 ოპერციამდე. ოპტიმალურ პირობებში ერთ ტაქტურ ციკლში გაიშვება ერთი ბრძანება, რომლის მიხედვითაც ამოირჩევა 5 ოპერაცია. პროცესორის ნორმალურ ტაქტურ სიხშირეს წარმოადგენს 266 ან 366 მეგა ჰერცი, მაგრამ რადგანაც ერთ ციკლში შეიძლება შესრულებული იქნას 5–მდე ოპერაცია, მისი ფაქტობრივი სწრაფქმედება 5–ჯერ მეტია. შემდეგი გარჩევების დროს გათვალისწინებული იქნება TM3260 პროცესორის ტექნიკური მონაცემები. პროცესორის სხვა რეალიზაციებს გააჩნიათ მცირედ განსხვავებული მონაცემები.

TriMedia პოროცესორის სტანდარტული ბრძანება გამოსახულია ნახ. 1.1.3–ზე. თავისი შინაარსის მიხედვით ბრძანებები ტიპი შეიძლება იცვლებოდეს 8–, 16– და 32–თანრიგიან ფიქსირებულწერტილიანი ბრძანებებიდან IEEE 754 სტანდარტის მცოცავწერტილიან ბრძანებებამდე და მულტიმედიური პარალელური დამუშავების ბრძანებებამდე. ერთ ციკლში 5 ოპერაციის შესრულება და მულტიმედიური მონაცემების პარალელური შესრულების ბრძანებების არსებობა TriMedia პროცესორს აძლევს ციფრული ვიდეოკამერიდან მიღებული ციფრული ვიდეოს ნაკადური დეკოდირების შესაძლებლობას.



ნახ. 1.1.3. TriMedia–ბრძანების სტანდარტი ხუთი ოპერაციით

TriMedia პოროცესორში გამოიყენება მეხსიერება ბაიტური ორგანიზებით, ხოლო შეტანა–გამოტანის რეგისტრები აისახება მეხსისიერების სივრცეში. ნახევარსიტყვების (16 ბიტი) და სრული სიტყვების (32 ბიტი) საზღვრების გასწორება ხდება არსებული საზღვრების მიხედვით. ბაიტე-

ბის მიმდევრობის რიგი შეიძლება იყოს როგორც პირდაპირი, ასევე შებრუნებული, რომელიც დამოკიდებულია პროგრამის მდგომარეობის სიტყვის ბიტზე, რომლის დაყენებაც ხდება ოპერაციული სისტემის მიერ. ეს ბიტიგანსაზღვრავს მონაცემების ჩატვირთვის ოპერაციების გადაცემის მექანიზმსოპერატიულ მეხსიერებასა და რეგისტრებს შორის. პროცესში გათვალისწინებულია გაყოფილი 8–თანრიგიანი ასოციაციური კეშ–მეხსიერება სტრიქონების ერთნაირი სიგრძით (64 ბიტი) კეშ–ბრძანებებსა და მეხსიერებაში. კეშ–ბრძანებების მეხსიერების მოცულობა წარმოადგენს 64 Kბაიტს, ხოლო მონაცემების კეშ–მეხსიერების მოცულობა წარმოადგენს 16 Kბაიტს.

არსებობს 128 უნივერსალური 32–თანრიგიანი რეგისტრი. R0 და R1 რეგისტრების მნიშვნელობები შესაბამისად უტოლდება აპარატულ 0–ს და 1–ს. დანარჩენი 126 რეგისტრი ფუნქციონალურად ექვივალენტურია და შეიძლება გამოყენებული იქნას ნებისმიერი დანიშნულებით. გარდა ამისა, გათვალისწინებულია ოთხი სპეციალიზებული 32–თანრიგიანი რეგისტრი: ბრძანებების მთვლელი, პროგრამის მდგომარეობის სიტყვის რეგისტრი და 2 წყვეტებთან დაკავშირებული რეგისტრი. და ბოლოს, ერთი 64–თანრიგიანი რეგისტრი ითვლის პროცესორის ციკლების რიცხვს უკანასკნელი ჩამოყრის შემდეგ. 300 MHz სიხშირის შემთხვევაში მთვლელის სრული ციკლი წარმოადგენს 2000 წელს.

TM3260 პროცესორში არსებობს 11 ფუნქციონალური ბლოკი, რომლებიც განკუთვნილია არითმეტიკული, ლოგიკური და მმართველი ოპერაციებისათვის (ასევე არსებობს კეშ–მეხსიერების მართვის ბლოკი). ყველა ესენი ჩამოთვლილია ცხრილ 1.1–ში. პირველ ორ სვეტში მოყვანილია ბლოკის სახელი და მათ მიერ შესრულებული ფუნქციები. მესამე სვეტში მითითებულია ბლოკის აპარატული ასლების რაოდენობა. მეოთხე სვეტში მითითებულია მოლოდინის მნიშვნელობა (უფრო სწორედ ციკლების რაოდენობა) ოპერაციის დასრულებამდე. აქვე, უნდა აღინიშნოს, რომ ყველა ფუნქციონალური ბლოკი, გარდა კვადრატული ფესვის ამოღების ბლოკისა და მცოცავწერტილიანი გაყოფის ბლოკისა, არის კონვეირიზებული. მოლოდინი თუმცა კი მიუთითებს ოპერაციის დასრულებამდე რა დროა დარჩენილი, მაგრამ, მიუხედავად ამისა, ყოველ ახალ ციკლში შეიძლება დაწყებული იქნას ახალი ოპერაცია. ყოველი სამი მიმდევნო ბრძანება შეიძლება მოიცავდეს ჩატვირთვის ორ ოპერაციას, მაშასადამე შესრულების სხვადასხვა ეტაპზე ერთდროულად შეიძლება იმყოფებოდეს ჩატვირთვის 6 ოპერაცია.

ცხრილი 1.1. TM3260 პროცესორის ფუნქციონალური ბლოკები მათი რაოდენობის, დაყოვნების დროის და საბრძანებო სლოტებთან შესაბამისობის მითითებით.

ბლოკი	აღწერა	#	მოლოდინი	1	2	3	4	5
ოპერაციები მუდმივებთან	ოპერაციები უშუალო დამისამართებით	5	1	კი	კი	კი	კი	კი
ალმ მთელი რიცხვებისათვის	32–თანრიგიანი არითმეტიკული და ლოგიკური ოპერაციები	5	1	კი	კი	კი	კი	კი
ძვრები	მრავალთანრიგიანი ძვრები	2	1	კი	კი	კი	კი	კი
ძვრები შენახვით	მიმართვები მეხსიერებასთან	2	3				კი	კი
ფიქსირებულ და მცოცავწერტილიანი რიცხვების გამრავლება	ფიქსირებულ და მცოცავწერტილიანი რიცხვების გამრავლების 32–თანრიგიანი ოპერაციები	2	3		კი	კი		

მცოცავწერტილიანი ოპერაციების ალმ	არითმეტიკული ოპერაციები მცოცავ-წერტილიანი რიცხვებისათვის	2	3	კი			კი	
მცოცავწერტილიანი რიცხვების შედარება	მცოცავწერტილიანი რიცხვების შედარების ოპერაციები	1	1			კი		
გაყოფა და კვადრატული ფესვის ამოღება მცოცავწერტილიანი რიცხვებისათვის	გაყოფისა და კვადრატული ფესვის ამოღების ოპერაციები მცოცავწერტი-ლიანი რიცხვებისათვის	1	17		კი			
განშტოებები	ნაკადების მართვა	3	3		კი	კი	კი	
სიგნალების ციფრული დამუშავების ალმ	არითმეტიკული ოპერაციები მულტი-მედიური მონაცემებისათვის (ორი 16–თანრიგიანი ან ოთხი 8–თანრიგიანი სიტყვა)	2	3	კი		კი		კი
გამმრავლებელი სიგნა-ლების ციფრული დამუშავებისათვის	გამრავლება მულტიმედიური მონაცე-მებისათვის (ორი 16–თანრიგიანი ან ოთხი 8–თანრიგიანი სიტყვა)	2	3		კი	კი		

უკანასკნელ 6 სვეტში ხდება ფუნქციონალურ ბლოკებთან ბრძნებების შესაბამისობის განსაზღვრა. მაგალითად, მცოცავწერტილიანი რიცხვების შედარების ოპერაციის შესრულება შესაძლებელია მხოლოდ მესამე საბრძანებო სლოტში.

კონსტანტებზე ოპერაციების ფუნქციონალური ბლოკი გამოიყენება ოპერაციებისათვის უშუალო დამისამართებით, მაგალითად, ოპერაციის ველიდან რიცხვის რეგისტრში ჩასატვირთად. ალმ ფიქსირებულწერტილიან მონაცემებზე ოპერაციების შესასრულებლად ახორციელებს შეკრე-ბას, გამოკლებას, სტანდარტულ ლოგიკურ ოპერაციებს და აგრეთვე შეფუთვისა და გახსნის ოპერაციებს. ძვრის ოპერაციების ბლოკს შეუძლია შეასრულოს რეგისტრების ძვრა ორივე მიმართულებით მითითებული ბიტების მიხედვით.

ჩატვირთვისა და შენახვის ბლოკი ახორციელებს მეხსიერებიდან სიტყვების წაკითხვას, რეგისტრებში ჩატვირთვას და პირიქით. TriMedia პროცესორი წარმოადგენს RISC–პროცესორს გაფართოებული ფუნქციონალური შესაძლებლობებით, ამიტომ ჩვეულებრივი ოპერაციები სრულ-დება რეგისტრების საშუალებით, ხოლო მეხსიერებასთან მიმართვები კი ჩატვირთვისა და შენახ-ვის ფუნქციონალური ბლოკის საშუალებით. შეიძლება გადაცემული იქნას 8, 16 ან 32 ბიტი. არით-მეტიკული და ლოგიკური ოპერაციების შესრულების დროს მეხსიერებასთან მიმართვები არ ხდება.

გამრავლების ბლოკი ასრულებს ოპერაციებს როგორც მთელ რიცხვებზე, ასევე მცოცავწერ-ტილიან მონაცემებზე. შემდეგი სამი ბლოკი ახორციელებს მცოცავწერტილიან მონაცემებზე შემდეგ ოპერაციებს: შეკრებისა და გამოკლების, შედარების, გაყოფისა და კვადრატული ფესვის ამოღების.

გადასვლის ოპერაციების შესრულება ხდება განშტოების ფუნქციონალური ბლოკის საშუალებით. გადასვლის შემდეგ ხდება დროითი დაყოვნება ფიქსირებული ხანგრძლივობით – 3 ციკლი, რომლის დროსაც ყოველთვის ხდება 3 ბრძანების შესრულება. ასე ხდება უპირობო გადასვლის შემთხვევაშიც.

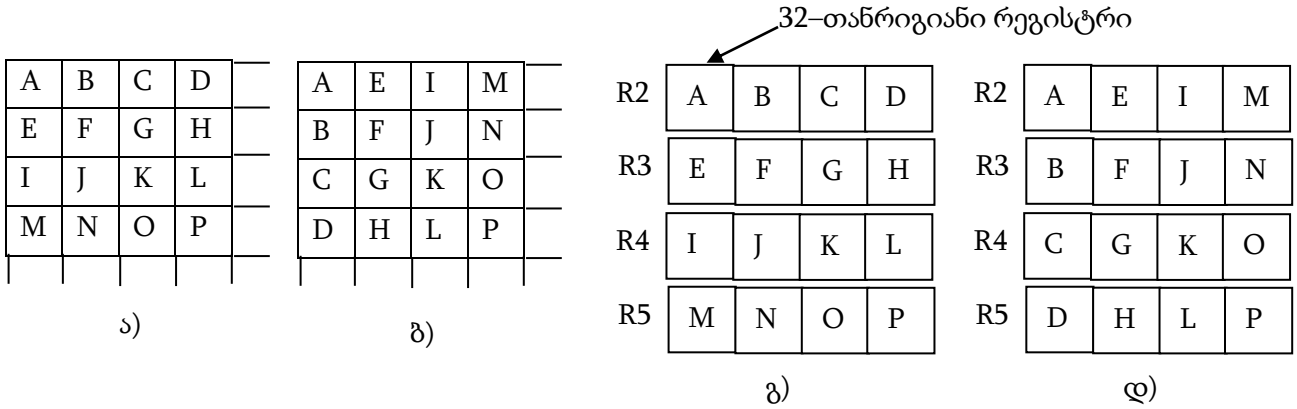
და ბოლოს, უკანასკნელი ორი ფუნქციონალური ბლოკი განკუთვნილია სპეციალური მულტიმედიური ოპერაციების შესრულებისათვის. მულტიმედიური ოპერაციების შესრულებას ფაქტობრივად ახორციელებს ციფრული სიგნალების დამმუშავებელი (Digital Signal Processor, DSP). აქ უნდა აღინიშნოს, რომ ფიქსირებულწერტილიან მონაცემებზე ოპერაციების დროს თუ გამოიყენება შევსება ორამდე, მულტიმედიური ოპერაციებისათვის გამოიყენება ინტენსიური არითმეტიკა (saturated arithmetic). გადავსების შედეგად ოპერაციის შედეგი თუ არ მიიღება, მაშინ შეირჩევა უახლოესი კორექტული რიცხვი. მაგალითად, 8–თანრიგიან რიცხვებთან მიმართებაში 130 და 130–ის შეკრების შედეგად აირჩევა 255.

ზოგიერთი ოპერაცია და საბრძანებო სლოტი შეიძლება იყოს შეუთავსებელი, მაშინ ბრძანებაში ჩაირთვება ხუთზე ნაკლები ოპერაცია. TriMedia–ბრძანებაში ოპერაციების შეთავსებადობაზე შემოწმება არ ხდება. ამიტომ, ოპერაციების შესრულება მოხდება მათი შეუთავსებლობის შემთხვევაშიც, რაც რა თქმა უნდა გამოიწვევს არასწორი შედეგის მიღებას. შეთავსებადობაზე შემოწმებაზე უარის თქმის გადაწყვეტილება დამპროექტებელმა მიიღო დროისა და ტრანზისტორების დაზოგვის მიზნით. Pentium პროცესორებში სუპერსკალარულ ოპერაციებში შეთავსებადობაზე შემოწმება ხდება, მაგრამ რთულდება გადაწყვეტილების მიღება, იზრდება დროითი დანახარჯები და იზრდება გამოყენებული ტრანზისტორების რაოდენობა. TriMedia პოროცესორში დაგეგმვის ამოცანა დაკისრებული აქვს კომპილატორს, რომელსაც ზედმეტი დროითი დანაკარგების გარეშე შეუძლია საბრძანებო სიტყვებში ოპერაციების განთავსების ოპტიმიზაცია. მაგრამ, ოპერაციის შესასრულებლად თუ საჭირო აღმოჩნდება მიუწვდომადი ფუნქციონალური ბლოკი, მაშინ მთელ ბრძანებას უწევს ლოდინი ვიდრე საჭირო ბლოკი არ განთავისუფლდება.

მულტიმედიური ოპერაციების შესრულება ძალიან იშვიათად ხდება 32–თანრიგიან მთელ რიცხვებზე. ეს დაკავშირებულია იმასთან, რომ გამოსახულების აგება ხდება RGB ფერად მოდელში, წითელი, მწვანე და ლურჯი პიქსელების 8–თანრიგიანი მონაცემებით. გამოსახულების დამუშავების დროს (მაგალითად შეკუმშვის დროს) მისი გამოსახვა ხდება სამი კომპონენტით – ერთი კომპონენტი თითოეულ ფერზე. გამოთვლების ძირითადი მოცულობა მოდის მატრიცებზე 8–თანრიგიანი უნიშნო მთელი რიცხვებით.

ასეთ მატრიცებზე მოქმედებების ეფექტურად შესრულების მიზნით TriMedia პრცესორში გათვალისწინებულია სპეციალიზებული ოპერაციები დიდი რაოდენობით. მაგალითისათვის, განვიხილოთ 8–თანრიგიანი მნიშვნელობების მქონე მატრიცის ზედა მარცხენა კუთხე, რომელიც შენახულია მეხსიერებაში ბაიტების პირდაპირი მიმდევრობით (ნახ. 1.1.4,ა). ამ კუთხეში ბლოკი ზომით 4×4 მოიცავს 8–თანრიგიან 16 მნიშვნელობას A-დან P–მდე. დავუშვათ, რომ გამოსახულების ტრანსპონირების შედეგად მიიღება ნახ. 1.1.4,ბ–ზე გამოსახული მატრიცი. როგორ მიიღება ეს შედეგი?

ტრანსპონირება შეიძლება შესრულებული იქნას 12 ოპერაციის საშუალებით, რომელთაგანაც თითოეული ახდენს ბაიტების ახალ რეგისტრებში ჩატვირთვას, რის შემდეგაც შესრულებული უნდა იქნას კიდევ 12 ოპერაცია, რომელთა საშუალებითაც ეს ბაიტები მოხვდება დანიშნულების ადგილზე (ტრანსპონირების დროს დიაგონალური ელემენტების მდებარეობა არ იცვლება). პრობლემა იმაში მდგომარეობს, რომ ამ სქემის მიხედვით საჭირო ხდება ოპერატიულ მეხსიერებასთან მიმართვის 24 ხანგრძლივი ოპერაციის შესრულება.



ნახ. 1.1.4. 8–თანრიგიანი ელემენტების მატრიცი (ა); ტრანსპონირებული მატრიცი (ბ); 4 რეგისტრში გადატანილი საწყისი მატრიცი (გ); ტრანსპონირებული მატრიცი 4 რეგისტრში

არსებობს სხვა ხერხიც. თავიდან სრულდება 4 ოპერაცია, რომელთაგან თითოეული თითო სიტყვას ათავსებს თითო რეგისტრში R2-დან R5–მდე, როგორც ნაჩვენებია ნახ. 1.1.4,გ–ზე. შემდეგ ნიღბისა და ძვრის ოპერაციების საშუალებით ხდება მიღებული ოთხი სიტყვის გაერთიანება და მიიღება სასურველი შედეგი (ნახ. 1.4,დ). დასასრულს, სიტყვების შენახვა ხდება ოპერატიულ მეხსიერებაში. მიუხედავად იმისა, რომ საგრძნობლად იქნა შემცირებული ოპერატიულ მეხსიერებასთან მიმართვების რაოდენობა (24–დან 8–მდე), ამ მეთოდის ეფექტურობა მაინცდამაინც მაღალი არ არის, რადგანაც ნიღბისა და ძვრის ოპერაციების საშუალებით ყველა ბაიტის საჭირო ადგილებზე დალაგებისათვის საჭირო ხდება მართალია მოკლე, მაგრამ მაინც ოპერაციების დიდი რაოდენობის შესრულება.

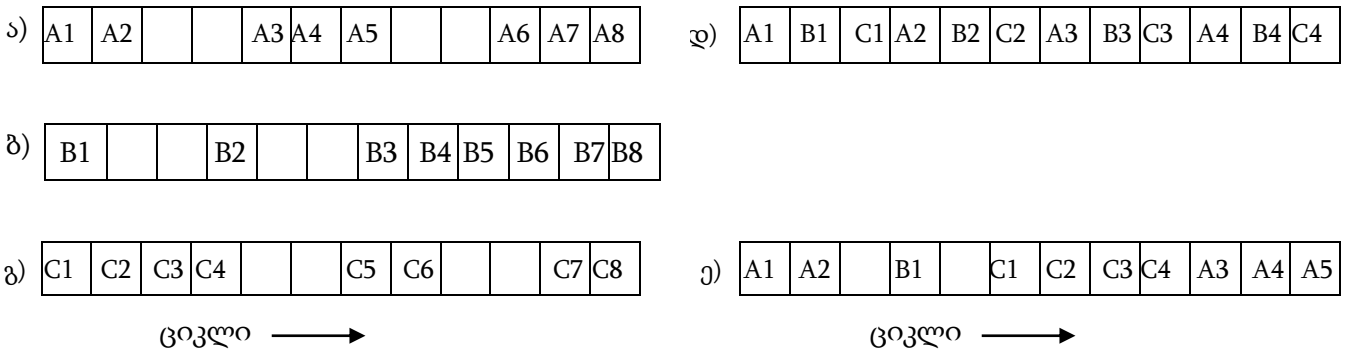
TriMedia პრცესორში რეალიზებულია უფრო უკეთესი მეთოდი. პირველ რიგში ხდება 4 სიტყვის შესაბამის რეგისტრებში განთავსება. ამასთან, შედეგის ფორმირება ხდება არა შენიღბვისა და ძვრის ოპერაციების საშუალებით, არამედ სპეციალური მულტიმედიური ოპერაციების საშუალებით. გამოსახულების ტრანსპონირებისათვის საჭირო ხდება 8 სპეციალური მუტიმედიური ბრძანების შესრულება და იმდენჯერვე ოპერატიული მეხსიერებისათვის მიმართვა. რვავე სპეციალური მულტიმედიური ოპერაციის და შენახვის ორი ბრძანების ჩატვირთვის შემდეგ მიიღება შედეგი. საბოლოო ჯამში, საჭირო ხდება სულ 6 ბრძანების შესრულება, თანაც 30 სლოტიდან 14 თავისუფალი რჩება და შესაძლებელია მათი გამოყენება სხვა ოპერაციების მიერ. სხვა მულტიმედიური ოპერაციებიც ასევე ეფექტური არის. ამ ოპერაციებისა და აგრეთვე, ბრძანებების 5 სლოტად დაყოფის შედეგად, TriMedia პრცესორი წაროადგენს მულტიმედიური მონაცემების დამუშავების მაღალეფექტურ საშუალებას.

1.1.3. პროცესორსშიდა მრავალნაკადურობა

ყველა თანამედროვე კონვეიზირებული პროცესორისათვის დამახასიათებელია ერთიდაიგივე პრობლემა – მეხსიერებასთან მიმართვის დროს მონაცემი თუ არ აღმოჩნდა პირველი ან მეორე დონის კეშ–მეხსიერებაში, მაშინ ამ მონაცემის კეშ–მეხსიერებაში ჩატვირთვისათვის საჭიროა დიდი დრო, რომლის განმავლობაშიც ხდება კონვეიერის მოცდენა. ამ პრობლემის გადაწყვეტის ერთერთ მეთოდს უწოდებენ **პროცესორსშიდა მრავალნაკადურობას** (on-chip multithreading).

ეს მეთოდი პროცესორს საშუალებას აძლევს ერთდროულად მართოს რამდენიმე პროგრამული ნაკადი, რითაც აკეთებს მოცდენების შენიღბვას. პროგრამული მრავალნაკადურობა მოკლედ შეიძლება ასე აიხსნას: პროგრამული ნაკადი 1 თუ იბლოკება, პროცესორს შეუძლია უზრუნველჰყოს სრული აპარატული მხარდაჭერა და გაუშვას პროგრამული ნკადი 2.

მეთოდის იდეა მარტივია, მისი რეალიზება კი შესაძლებელია რამდენიმე გზით. მათგან პირველს უწოდებენ **მცირემოდულურ მრავალნაკადურობას** (fine-grained multithreading). ამ შემთხვევაში პროცესორს ერთ ტაქტში შეუძლია გამოიძახოს ერთი ბრძანება (ნახ. 1.1.5). ნახ. 1.1.5.ა,ბ,გ–ზე გამოსახულია სამი პროგრამული ნაკადი (A,B,C), რომლებიც შეესაბამება 12 მანქანურ ციკლს. პირველი ციკლის განმავლობაში A ნაკადისათვის სრულდება A1 ბრძანება. რადგანაც ამ ბრძანების შესრულება მთავრდება ერთ ციკლში, მეორე ციკლის დაწყებისას გაიშვება A2 ბრძანებამისთვის საჭირო მონაცემი პირველი დონის კეშ–მეხსიერებაში არ აღმოჩნდა და ამიტჰმ, საჭირო ხდება მეორე დონის კეშ–მეხსიერებასთან მიმართვა, რასაც სჭირდება 2 მანქანური ციკლი. ნაკადის შესრულების გაგრძელება მოხდება მეხუთე ციკლში. როგორც ნახატზე ჩანს B და C ნაკადებშიც ადგილი აქვს ხშირ მოცდენებს. ამ მიდგომის შემთხვევაში, მომდევნო ბრძანების გამოძახება არ ხორციელდება, ვიდრე არ დასრულდაბა წინა ბრძანების შესრულება.



ნახ. 1.1.5. სამი პროგრამული ნაკადი. ცარიელი კვადრატები აღნიშნავს მეხსიერებიდან მონაცემების მოლოდინით გამოწვეულ მოცდენებს (ა, ბ, გ); მცირემოდულური მრავალნაკადურობა (დ); დიდმოდულური მრავალნაკადურობა (ე).

მცირემოდულური მრავალნაკადურობის შემთხვევაში მოცდენების თავიდან აცილება ხდება შემდეგნაირად: პროგრამული ნაკადების შესრულება ხდება მიმდევრობით, ანუ წრიულად, ანუ მიმდებარე ციკლებში ხდება სხვადასხვა ნაკადების ბრძანებების შესრულება (ნახ. 1.1.5.დ). მეოთხე ციკლის დაწყების მომენტში A1 ბრძანებით გამოწვეული მოცდენა დამთავრებულია და შესაძლებელია A2 ბრძანების შესრულების დაწყება. ამ შემთხვევაში მოცდენის ხანგრძლივობა 2 ციკლს არ აღემატება, ან სამი პროგრამული ნაკადის შემთხვევაში, ოპერაცია, რომელსაც მოცდა მოუწევდა, მაინც აკრძალულია. მოლოდინის დრო თუ იქნებოდა 4 ციკლის ხანგრძლივობის, მაშინ უწყვეტი მუშაობისათვის საჭირო გახდებოდა უკვე 4 პროგრამული ნაკადი და ა.შ.

რადგანაც სხვადასხვა პროგრამული ნაკადები ერთმანეთთან არანაირად დაკავშირებული არ არიან, ამიტომ თითოეულ მათგანს სჭირდება რეგისტრების სხვადასხვა ნაკრებები. ის მითითებული უნდა იყოს ყოველი გამოძახებული ბრძანებისათვის, მაშინ აპარატული უზრუნველყოფისათვის ცნობილი იქნება თუ რომელი ბრძანებისათვის რეგისტრების რომელ ნაკრებს უნდა მიმარ-

თოს. ცხადია, რომ ერთდროულად შესასრულებელი პროგრამული ნაკადების რაოდენობის განსაზღვრა უნდა მოხდეს თვითონ პროცესორის მიკროსქემის დამუშავების დროს.

პროცესორის მოცდენის ერთადერთ მიზეზს არ წარმოადგენს ოპერატიულ მეხსიერებასთან მიმართვები. ხანდახან მომდევნო ბრძანების შესასრულებლად საჭიროა წინა ბრძანების შედეგის არსებობა, მაგრამ შედეგის გამოთვლა კი შეიძლება დასრულებული არ იყოს. ასევე შეუძლებელია ბრძანების შესრულება, თუ ის პირობითი გადასვლის ოპერატორის შემდეგ უნდა შესრულდეს და გადასვლის მიმართულება კი ჯერ ცნობილი არ არის. ზოგადი წესის ფორმულირება შეიძლება შემდეგნაირად: კონვეიერს თუ გააჩნია k რაოდენობის საფეხური და შესასრულებელი პროგრამული ნაკადების რაოდენობა არ არის k–ზე ნაკლები, მაშინ ყოველ ნაკადში, ყოველ მოცემულ მომენტში შეიძლება შესრულებული იქნას არა უმეტეს ერთი ბრძანებისა. ასეთ შემთხვევაში პროცესორი მუშაობს მოცდენების გარეშე და მაქსიმალური სიჩქარით.

ბუნებრივია, რომ პროგრამული ნაკადების რაოდენობა ყოველთვის არ არის პროცესორის კონვეიერის საფეხურების რაოდენობის ტოლი, ამიტომ ზოგიერთი დამპროექტებელი უპირატესობას ანიჭებს **მსხვილმოდულური მრავალნაკადურობის** (coarse-grainid multithreading) მეთოდიკას, რომელიც ილუსტრირებულია ნახ. 1.1.5.ე–ზე. მოცემულ შემთხვევაში A ნაკადის ბრძანებების მიმდევრობის შესრულება მიმდევრობით გრძელდება მოლოდინის მომენტის დადგომამდე. ამ დროს ხდება ერთი ციკლის დაკარგვა. ამის შემდეგ ხდება გადართვა B ნაკადის პირველ ბრძანებაზე (B1). იმის გამო, რომ ამ ნაკადში პირველივე ბრძანების შემდეგ ადგილი აქვს მოლოდინს, 6–ე ციკლში იწყება უკვე C1 ბრძანების შესრულება. იმის გამო, რომ მოლოდინის მომენტის დადგომის შემთხვევაში ერთი ციკლი იკარგება, შეიძლება დავასკვნათ, რომ ეფექტურობის მხრივ მცირემოდულურ მრავალნაკადურობას უპირატესობა გააჩნია მსხვილმოდულურ მრავალნაკადურობასთან შედარებით, მაგრამ, მიუხედავად ამისა, მას გააჩნია ერთი არსებითი უპირატესობა, პროგრამული ნაკადების ნაკლები რაოდენობის შემთხვევაში მნიშვნელოვნად ზრდის პროცესორის ეფექტურობას. პროგრამული ნაკადების არასაკმარისი რაოდენობის შემთხვევაში მსხვილმოდულური მრავალნაკადურობის მეთოდიკა ოპტიმალურია.

მოყვანილი აღწერის მიხედვით, მსხვილმოდულური მრავალნაკადურობის შემთხვევაში ხდება გადართვები ნაკადებს შორის, მაგრამ მოცემულ მეთოდიკაში ეს არ წარმოადგენს მოქმედებების ერთადერთ ვარიანტს. არსებობს ვარიანტი, რომლის დროსაც შესაძლებელია განხორციელდეს დაუყოვნებელი გადართვები ბრძანებებიდან, რომლებსაც პოტენციალურად შეუძლიათ გამოიწვიონ მოლოდინი (მაგალითად, ჩატვირთვა, შენახვა და გადასვლა) და გამორიცხავენ ფუჭ ციკლებს. სხვა სიტყვებით რომ ვთქვათ, შესრულება გრძელდება იმ მომენტამდე, ვიდრე აღმოჩენილი არ იქნება პრობლემის წარმოქმნის მომენტი, რომლის შემდეგაც განხორციელებული უნდა იქნას გადართვა. ასეთი ხშირი გადართვები ერთმანეთს აახლოებს მცირემოდულური და მსხვილმოდულური მრავალნაკადურობის პრინციპებს.

მრავალნაკადურობის გამოყენებული ვარიანტისაგან მიუხედავად, საჭიროა თვალყურის მიდევნება იმაზე, თუ რომელი ბრძანება რომელ პროგრამულ ნაკადს მიეკუთვნება. მცირემოდულური მრავალნაკადურობის შემთხვევში ყოველ ბრძანებას მიენიჭება ნაკადის იდენტიფიკატორი, ამიტომ კონვეიერში მათი გადაადგილება ეჭვს არ იწვევს. მსხვილმოდულური მრავალნაკადურობა ითვალისწინებს კონვეიერის საფეხურების გასუფთავებას ყოველი მომდევნო ნაკადის გაშვების

წინ. ასეთ შემთხვევაში ცალსახად განისაზღვრება მოცემულ მომენტში შესრულებადი ნაკადის იდენტურობა. ცხადია, რომ მოცემული მეთოდიკა ეფექტურია მხოლოდ იმ შემთხვევაში, თუ გადართვებს შორის პაუზების ხანგრძლივობა მნიშვნელოვნად აღემატება კონვეიერის საფეხურების გასუფთავებისათვის საჭირო დროს.

ყოველივე ზემოთ აღნიშნული ეკუთვნის მხოლოდ იმ პროცესორებს, რომელთაც ერთი ტაქტური ციკლის განმავლობაში შეუძლიათ გამოიძახონ არა უმეტეს ერთი ბრძანებისა. მაგრამ, ვიცით, რომ თანამედროვე პროცესორებისათვის ეს შეზღუდვა არაა აქტუალური. ნახ. 1.1.6–ზე გამოსახულია შემთხვევა, როდესაც პროცესორს შეუძლია ერთი ციკლის განმავლობაში გამოიძახოს ორი ბრძანება, თუმცა შეზღუდვა შემდეგი ბრძანებების გამოძახების შესახებ წინა ბრძანებების შესრულების შეყოვნების შემთხვევაში, ძალაში რჩება. ნახ. 1.1.6,ა–ზე გამოსახულია მცირემოდულური მრავალნაკადურობის მექანიზმი გაორმაგებულ სუპერსკალარულ პროცესორში. როგორც ნახაზზე ჩანს, პირველ ციკლში გაიშვება A ნაკადის ორი ბრძანება, მაგრამ მეორე ციკლში გაიშვება B ნაკადის მხოლოდ ერთი ბრძანება.

A1	B1	C1	A3	B2	C3	A5	B3	C5	A6	B5	C7
A2		C2	A4		C4		B4	C6	A7	B6	C8

ციკლი ⟶

ა)

A1	B1	C1	C3	A3	A5	B2	C5	A6	A8	B3	B5
A2		C2	C4	A4			C6	A7		B4	B6

ციკლი ⟶

ბ)

A1	B1	C2	C4	A4	B2	C6	A7	B3	B5	B7	C7
A2	C1	C3	A3	A5	C5	A6	A8	B4	B6	B8	C8

გ)

ნახ.1.1.6. მრავალნაკადურობა გაორმაგებულ პროცესორში: მცირემოდულური მრავალნაკადურობა (ა); მსხვილმოდულური მრავალნაკადურობა (ბ); სინქრონული მრავალნაკადურობა (გ).

ნახ. 1.1.6,ბ–ზე გამოსახულია მსხვილმოდულური მრავალნაკადურობის რეალიზება გაორმაგებული პროცესორის შემთხვევაში სტატიკური დამგეგმავით, რომელიც ბრძანებების მოლოდინის პერიოდში გამორიცხავს უსასრულო ციკლებს. ამ შემთხვევაში პროგრამული ნაკადების შესრულება ხდება რიგრიგობით, ყოველ ციკლში პროცესორი გამოიძახებს ორ ბრძანებას თითოეული ნაკადიდან ვიდრე არ დადგება მოლოდინის დრო; მოცდენის შემდეგ ტაქტში იწყება შემდეგი ნაკადის შესრულება.

სუპერსკალარულ პროცესორებში არსებობს მრავალნაკადურობის რეალიზების კიდევ ერთი საშუალება, ეს არის **სინქრონული მრავალნაკადურობა** (simultaneous multithreading), რომელიც წარმოდგენილია ნახ. 1.1.6,გ–ზე. ეს მეთოდიკა წარმოადგენს მსხვილმოდულური მრავალნაკადურობის გაუმჯობესებულ ვარიანტს, რომლის დროსაც ყოველ პროგრამულ ნაკადს ერთ ციკლში შეუძლია გაუშვას ორი ბრძანება, მაგრამ მოლოდინის შემთხვევაში, პროცესორის მოცდენის თავიდან ასაცილებლად შეიძლება გაშვებული იქნას ბრძანება შემდეგი ნაკადიდან. სინქრონული მრავალნაკადურობის შემთხვევაში ხდება ყველა ფუნქციონალური ბლოკის სრულად დატვირთვა. თუ შესაძლებელი არ არის რომელიმე ბრძანების გაშვებაიმის გამო, რომ შესაბამისი ფუნქციონალური ბლოკი დაკავებულია, მაშინ გაიშვება ბრძანება შემდეგი ნაკადიდან.

1.1.4. მრავალნაკადურობა Pentium 4 პროცესორში

მრავალნაკადურობის რეალიზების პრაქტიკულ მაგალითად განვიხილოთ პროცესორი Pentium 4. მიკროსქემის დაპროექტების დროს Intel–ის ინჟინრები ბჭობდნენ, თუ როგორ მიეღწიათ პროცესორის მაქსიმალური სწრაფქმედებისათვის. განიხილებოდა ხუთი მიმართულება:

1. პროცესორის ტაქტური სიხშირის გაზრდა;
2. ერთ მიკროსქემაში ორი პროცესორის ჩადგმა;
3. ახალი ფუნქციონალური ბლოკების დამატება;
4. კონვეიერის დაგრძელება;
5. მრავალნაკადურობის გამოყენება.

პროცესორის სწრაფქმედების გასაზრდელად ყველაზე ტრივიალური საშუალებაა მისი ტაქტური სიხშირის გაზრდა, სხვა პარამეტრების შეუცვლელად. როგორც წესი, ყოველ შემდეგ მიკროპროცესორსმის წინამორბედთან შედარებით აქვს უფრო მაღალი ტაქტური სიხშირე. ტაქტური სიხშირის გაზრდის დროს დამპროექტებლები აწყდებიან ორ პრობლემას: იზრდება ენერგომოხმარება (რაც ძალიან აქტუალურია აკუმულტორზე მომუშავე კომპიუტერებისათვის) და გადახურება (რაც მოითხოვს უფრო ეფექტური გაცივების სისტემის შექმნას).

მეორე ხერხი – მიკროსქემაში ორი პროცესორის მოთავსება – შედარებით ადვილია, მაგრამ ორჯერ იზრდება მიკროსქემის ფართი. გარდა ამისა, თუ გავითვალისწინებთ, რომ თითოეულ პროცესორს უნდა ჰქონდეს თავისი კეშ–მეხსიერება, ვღებულობთ იმას, რომ წარმოების ხარჯები ორმაგდება. ორივე პროცესორისათვის თუ გამოყენებული იქნება საერთო კეშ–მეხსიერება, მაშინ კრისტალის ფართობის მნიშვნელოვანი გაზრდა შეიძლება თავიდან იყოს აცილებული, მაგრამ ამ შემთხვევაში თავს იჩენს სხვა პრობლემა – თითოეულ პროცესორზე გათვლით მცირდება კეშ–მეხსიერების მოცულობა, რაც აისახება პროცესორის მწარმოებლურობაზე.

ახალი ფუნქციონალური ბლოკების დამატებაც პრინციპიალურ პრობლემებს არ ქმნის, მაგრამ საჭიროა ბალანსის დაცვა. რა აზრი აქვს ათეულობით ალმ ბლოკების არსებობას, თუკი მიკრეოსქემას არ შეუძლია კონვეიერზე ბრძანებების იმ სიჩქარით გაცემა, რომელიც სრულად დატვირთავდა მის ბლოკებს?

კონვეიერში საფეხურების რაოდენობის გაზრდით შეიძლება შემცირებული იქნას ბრძანებების შესრულების დრო, რითაც იზრდება პროცესორის მწარმოებლურობა, მაგრამ, მეორეს მხრივ, იზრდება გადასვლების პროგნოზირებასთან დაკავშირებული ნეგატიური შედეგების ალბათობა. კეშ–აცილებების, წყვეტების და კიდევ ზოგიერთ შემთხვევაში საჭირო ხდება კონვეიერის თავიდან შევსება. მაქსიმალური სწრაფქმედება კი მიიღწევა მხოლოდ სრულად შევსებული კონვეიერის შემთხვევაში. გარდა ამისა, კონვეიერში საფეხურების რაოდენობის გაზრდა პირდაპირაა დაკავშირებული ტაქტური სიხშირის გაზრდასთან, რაც თავის მხრივ წამოჭრის უკვე განხილულ პრობლემებს.

და ბოლოს, შესაძლებელია მრავალნაკადურობის რელიზება. ამ ტექნოლოგიის უპირატესობა მდგომარეობს დამატებითი პროგრამული ნაკადის შემოღებაში, რომელიც დამატებით აამოქმედებს პროცესორის იმ რესურსებს, რომლებიც მანამდე გამოუყენებელი იყო. ექსპერიმენტალური კვლევების შედეგების მიხედვით Intel–ის დამპროექტებლებმა დაადგინეს, რომ მრავალნაკადურობის შემოღების მიზნით მიკროსქემის ფართობის 5%–ით გაზრდის შედეგად პროცესორის მწარმო-

ებლურობა იზრდებოდა 25%–ით. Intel–ის პირველი პროცესორი მრავალნაკადურობის მხარდაჭერით იყო 2002 წელს გამოშვებული პროცესორი Xeon. შემდეგ მრავალნაკადურობა შემოტანილი იქნა Pentium 4 პროცესორებში. მათში რეალიზებულ მრავალნაკადურობას უწოდებენ **ჰიპერნაკადურობას** (hiperthreding).

ჰიპერნაკადურობის ძითადი პრინციპია ორი პროგრამული ნაკადის გამოყენება (პროცესორი ერთმანეთისაგან არ განასხვავებს პროცესებსა და პროგრამულ ნაკადებს). ოპერაციული სისტემა ჰიპენაკადურ Pentium 4 პროცესორს განიხილავს როგორც ორპროცესორიან სისტემას საერთო კეშ–მეხსიერებით და ოპერატიული მეხსიერებით. ოპერაციული სისტემა თითოეული პროგრამული ნაკადისათვის ცალცალკე ახდენს დაგეგმვას. ამგვარად, ერთდროულად შეიძლება სრულდებოდეს ორი პროგრამა. მაგალითად, საფოსტო პროგრამა ფონურ რეჟიმში შეიძლება ღებულობდეს ან აგზავნიდეს შეტყობინებას, ვიდრე მომხმარებელი ურთიერთობს ინტერაქტიულ პროგრამასთან.

გამოყენებით პროგრამებს, რომელთა შესრულებაც შესაძლებელია რამდენიმე პროგრამული ნაკადის სახით, შეუძლიათ ერთდროულად აამოქმედონ ორივე „ვირტუალური პროცესორი“. მაგალითად, ვიდეო მონაცემების რედაქტირების პროგრამა მომხმარებლებს საშუალებას აძლევს ყველა კადრისათვის გამოიყენონ ფილტრები. ასეთი ფილტრების საშუალებით ხდება სიმკვეთრის, სიკაშკაშის, ფერთა ბალანსის და კადრების სხვა თვისებების კორექტირება. ასეთ შემთხვევაში პროგრამამ შეიძლება ერთი ვირტუალური პროცესორი გამოიყენოს კენტი კადრების დასამუშავებლად, ხოლო მეორე კი ლუწი კადრებისათვის. ამ დროს ორი პროცესორი მუშაობს ერთმანეთისაგან სრულიად დამოუკიდებლად.

რადგანაც პროგრამული ნაკადები მიმართავენ ერთიდაიგივე აპარატულ საშუალებებს, საჭირო ხდება ამ ნაკადების კოორდინაცია. ჰიპერნაკადურობის პირობებში Intel–ის დამპროექტებლებმა გამოყვეს 4 სტრატეგია რესურსების ერთდროულად გამოყენების მართვისათვის: რესურსების დუბლირება, რესურსების ხისტი, ზღურბლური და სრული დაყოფა.

რესურსების დუბლირება (resource duplication). როგორც ცნობილია, პროგრამული ნაკადების ორგანიზების მიზნით ხდება ზოგირთი რესურსის დუბლირება. მადალითად, რადგანაც ყოველ პროგრამულ ნაკადს სჭირდება ინდივიდუალური მართვა, საჭირო ხდება მეორე ბრძანებების მთვლელი. გარდა ამისა, საჭირო ხდება რეგისტრების (EAX,EBX, და ა.შ.) ასახვის მეორე ცხრილი ფიზიკურ რეგისტრებზე, ანალოგიურად საჭირო ხდება წყვეტების კონტროლერის დუბლირებაც, რადგანაც თითოეული ნაკადისათვის საჭირო ხდება წყვეტების ინდივიდუალური მართვა.

პროგრამულ ნაკადებს შორის **რესურსების ხისტი დაყოფა** (partitioned resource). მაგალითად, პროცესორში თუ გათვალისწინებულია რიგი კონვეიერის ორ ფუნქციონალურ საფეხურს შორის, მაშინ სლოტების ნახევარი შეიძლება გადაცემული იქმას პირველი ნაკადისათვის, ხოლო მეორე ნახევარი კი მეორე ნაკადისათვის. რესურსების გაყოფა ადვილად ხდება, არ იწვევს დისბალანსს და უზრუნველყოფს პროგრამული ნაკადების ერთმანეთისაგან სრულ დამოუკიდებლობას. რესურსების სრული გაყოფის შედეგად ერთი პროცესორი გადაიქცევა ორ პროცესორად. მეორეს მხრივ, შეიძლება შეიქმნას ისეთი სიტუაცია, რომლის დროსაც ერთი პროგრამული ნაკადი არ გამოიყენებს გამოყოფილი რესურსების ნაწილს იმ დროს, როდესაც ეს რესურსები ესაჭიროება

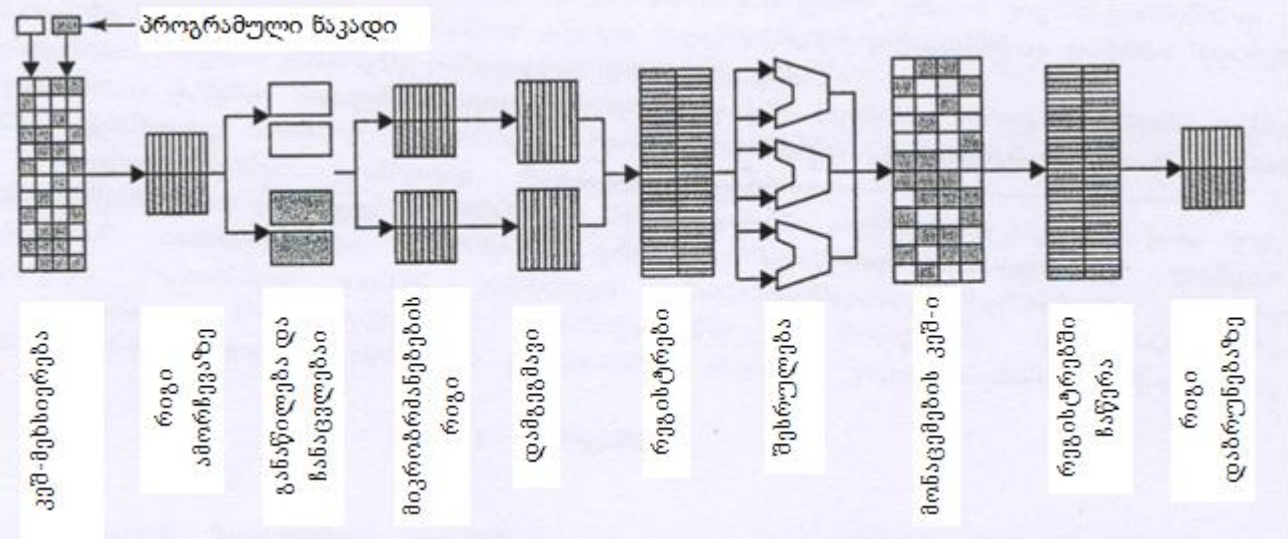
მეორე ნაკადს, მაგრამ მათთან წვდომის უფლებამოსილება არ გააჩნია. შედეგად ხდება ამ რესურსების მოცდენა.

პროგრამულ ნაკადებს შორის **რესურსების სრული დაყოფა** (full resource sharing). ამ სქემის მიხედვით საჭირო რესურსებთან წვდომის უფლება შეიძლება მიიღოს ნებისმიემა პროგრამულმა ნაკადმა, ხოლო მათი მომასხურება ხდება წვდომაზე მათი შემოსვლის დროის მიხედვით. განვიხილოთ შემთხვევა, როდესაც სწრაფი ნაკადი, რომელიც უპირატესად შედგება შეკრებისა და გამოკლების ოპერაციებისაგან, თანაარსებობს ნელ ნაკადთან, რომლის ფარგლებშიც ხდება გამრავლებისა და გაყოფის ოპერაციების შესრულება. თუ ბრძანებების გამოძახება ოპერატიული მეხსიერებიდან ხდება უფრო სწრაფად, ვიდრე ხდება გამრავლებისა და გაყოფის ოპერაციების შესრულება, მაშინ ნელი ნაკადის შემთხვევაში გამოძახებული ბრძანებების რაოდენობა თანდათან იზრდება. ბოლოსდაბოლოს ამ ბრძანებების ნაკადი შეავსებს რიგს და შედეგად რიგში ადგილების უკმარისობის გამო მოხდება სწრაფი ბრძანებების ნაკადის შეჩერება. რესიურსების სრული გაყოფა ხსნის რესურსების არაოპტიმალური გამოყენების პრობლემას, მაგრამ ქმნის დისბალანსს მათ გამოყენებაში – ერთ ნაკადს შეუძლია შეანელოს ან სულაც შეაჩეროს მეორე ნაკადი.

შუალედური სქემის რეალიზება ხდება პროგრამულ ნაკადებს შორის **რესურსების ზღურბლური დაყოფის** ფარგლებში (threshold resource sharing). ამ სქემის მიხედვით, ნებისმიერ პროგრამულ ნაკადს შეუძლია დინამიურად მიიღოს რესურსების განსაზღვრული (შეზღუდული) რაოდენობა. მაგალითად, თუ თითოეულ ნაკადს ავუკრძალავთ დაიკავოს ბრძანებების რიგის ¾–ზე მეტი, მაშინ ნელი პროცესების მიერ რესურსების გადამეტებული მოთხოვნა დაიწყებს სწრაფი ბრძანებებისათვის ხელის შეშლას.

Pentium 4 პროცესორის ჰიპერნაკადურობის მოდელი აერთიანებს რესურსების განაწილების სხვადასხვა სტრატეგიებს. ამგვარად, ადგილი აქვს თითოეული სტრატეგიისათვის არსებული ყველა პრობლემის გადაჭრის მცდელობას. დუბლირების რეალიზება ხდება იმ რესურსებთან მიმართებაში, რომლებთანაც წვდომას მოითხოვს ორივე პროგრამული ნაკადი (მაგალითად, ბრძანებების მთვლელი, რეგისტრების ასახვის ცხრილი, წყვეტების კონტროლერი). ამ რესურსების დუბლირებისათვის საჭირო ხდება მიკროსქემის ფართობის გაზრდა მხოლოდ 5%–ით, რაც სრულიად რეალური საფასურია მრავალნაკადურობის უზრუნველსაყოფად. რესურსების, რომლებიც მოცემულია იმ ოდენობით, რომ გამორიცხულია მათი სრული დაკავება რომელიმე პროგრამული ნაკადის მიერ, განაწილება ხდება დინამიურად. კონვეიერის მუშაობის მაკონტროლებელ რესურსებთან წვდომა განაწილებულია – ყოველი პროგრამული ნაკადისათვის გამოყოფილია სლოტის ნახევარი. Pentium 4 პროცესორში რეალიზებული მთავარი კონვეიერი ნაჩვენებია ნახ. 1.1.7–ზე. მასზე თეთრი და რუხი არეები მიუთითებენ რესურსების განაწილებას თეთრ და რუხ პროგრამულ ნაკადებს შორის.

როგორც ნახატიდან ჩანს, ყველა რიგი გაყოფილია – თითოეულ პროგრამულ ნაკადს მინიჭებული აქვს სლოტების ნახევარი. არცერთ პროგრამულ ნაკადს არ შეუძლია მეორე ნაკადის შესრულების შეზღუდავა. განაწილებისა და ჩანაცვლების ბლოკიც გაყოფილია. დამგეგმავის რესურსები განაწილებულია დინამიურად, მაგრამ გარკვეული ზღურბლის პრინციპის საფუძველზე, ისე რომ, არცერთ პროგრამულ ნაკადს არ შეუძლია დაიკავოს რიგის ყველა სლოტი. კონვეიერის ყველა დანარჩენი სლოტისათვის გამოიყენება სრული გაყოფის პრინციპი.



ნახ. 1.1.7. პროგრამულ ნაკადებს შორის რესურსების განაწილება Pentium 4 პროცესორში რეალიზებულ NetBurst მიკროარქიტექტურაში

მრავალნაკადურობასთან დაკავშირების საქმე არც ისე მარტივია. ასეთ პროგრესულ მეთოდიკასაც გააჩნია თავისი ნაკლები. რესურსების ხისტი განაწილების მეთოდი არ არის დაკავშირებული დიდ დანაკარგებთან. რესურსების დინამიური განაწილების მეთოდი კი, განსაკუთრებით თუ განაწილება ხდება ზღურბლის მნიშვნელობის გათვალისწინებით, ბრძანებების შესრულების პროცესში მოითხოვს რესურსების გამოყენებაზე კონტროლის განხორციელებას. გარდა ამისა, ზოგიერთ შემთხვევაში პროგრამები უკეთესად მუშაობენ მრავალნაკადური პრინციპის გარეშე, ვიდრე მის პირობებში. დავუშვათ, რომ ორი პროგრამული ნაკადის შესრულებისას ნორმალური ფუნქციონირებისათვის თითოეულ მათგანს ესაჭიროება კეშ–მეხსიერების ¾. ამ ნაკადების შესრულება რომ მომხდარიყო მიმდევრობით, მაშინ თითოეული მათგანი აჩვენებდა საკმაოდ მაღალ ეფექტურობას კეშ–აცილებების მცირე რაოდენობის გამო. ამ ნაკადების პარალელური შესრულების შემთხვევაში კი კეშ–აცილებების რაოდენობა გაიზრდება და შედეგიც შეიძლება მიღებული იქნას უარესი ვიდრე მრავალნაკადურობის გარეშე.

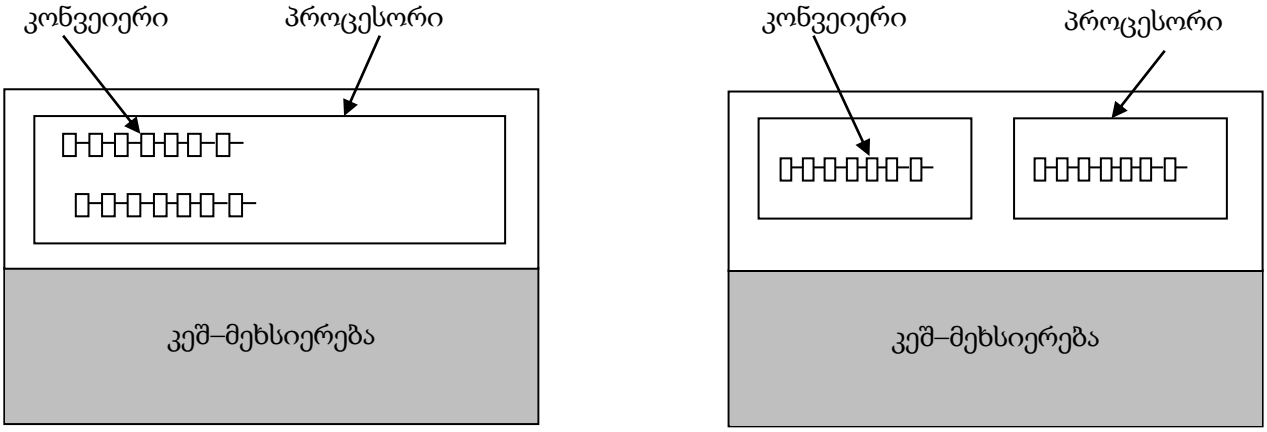
1.2. ერთკრისტალიანი მულტიპროცესორები

მრავალნაკადურობით შესაძლებელია გონივრული დანახარჯებით მიღწეული იქნას მწარმოებლურობის მნიშვნელოვანი გაზრდა, მგრამ ზოგიერთი ამოცანის გადასწყვეტად საჭიროა ბევრად უფრო მაღალი მწარმოებლურობა და ასეთ შემთხვევაში მრავალნაკადურობა უკვე საკმარისი აღარ იქნება. ასეთი ამოცანებისათვის არსებობს მულტიპროცესორული სქემები. ესაა სქემები, რომლებშიც ჩაშენებულია ორი ან მეტი პროცესორი.

1.2.1. ჰომოგენური ერთკრისტალიანი მულტიპროცესორები

ზედიდი ინტეგრალური სქემების ტექნოლოგიის განვითარების შედეგად მიკროსქემის ერთ კრისტალში შეიძლება მოთავსებული იქნა ორი და მეტი საკმაოდ მძლავრი პროცესორი. რადგანც ეს პროცესორები მიმართავენ მეხსიერების ერთიდაიგივე მოდულს (პირველი დონის კეშ, მეორე დონის კეშ და ოპერატიულ მეხსიერებებს), ისინი განიხილებიან, როგორც ერთიანი მულტიპროცესორი. როგორც წესი, მათი დაყენება ხდება ვებ–სერვერების დიდ ფერმებში. ორი პროცესორის ერთობლივად განლაგების, მეხსიერების რესურსების, დისკური და საქსელო ინტერფეისების განაწილების შემთხვევაში შეიძლება მიღწეული იქნას სერვერის მწარმოებლურობის მნიშვნელოვანი გაზრდა, თანაც ისე, რომ ამაზე გაწეული ხარჯები გაიზრდება, მაგრამ გაცილებით უფრო ნაკლებად.

მცირე ზომის ერთკრისტალიანი მულტიპროცესორებისათვის არსებობს გადაწყვეტის ორი საშუალება. პირველ შემთხვევაში (ნახ. 1.2.1.ა) არსებობს ერთი მიკროსქემა ორი კონვეიერით. ასეთ შემთხვევაში ბრძანებების შესრულების სიჩქარე თეორიულად ორმაგდება. მეორე შემთხვევაში



ა) ბ)

ნახ. 1.2.1. ერთკრისტალიანი მულტიპროცესორები: მიკროსქემა ორი კონვეიერით (ა); მიკროსქემა ორი ბირთვით (ბ)

(ნახ. 1.2.1.ბ) მიკროსქემაში გამოყენებულია ორი დამოუკიდებელი ბირთვი, რომელთაგან თითოეული წარმოადგენს სრულყოფილ პროცესორს. **ბირთვს** უწოდებენ დიდ მიკროსქემას (მაგალითად, პროცესორი, შეტანა–გამოტანის კონტროლერი, კეშ–მეხსიერება), რომელიც მიკროსქემაში განთავსებულია, როგორც მოდული, და თანაც, როგორც წესი, რამდენიმე სხვა ასეთივე ბირთვთან ერთად.

პირველი გადაწყვეტა უშვებს პროცესორის რესურსების ერთობლივ გამოყენებას (ფუნქციონალური ბლოკების) ანუ თითოეულ პროცესორს შეუძლია მიმართოს იმ რესურსებს, რომლებზც მოცემულ მომენტში არა არის მოთხოვნა მეორე პროცესორიდან. მეორე გადაწყვეტა მოითხოვს მიკროსქემის კონსტრუქციის შეცვლას და არ ითვალისწინებს ორზე მეტ პროცესორს. აქ უნდა აღინიშნოს, რომ ერთ მიკროსქემაში პროცესორის ორი ან მეტი ბირთვის განთავსება დიდ სირთულეს არ წარმოადგენს.

1.2.2. ჰეტეროგენული ერთკრისტალიანი მიკროპროცესორები

სერვერების გარდა ერთკრისტალიანი მულტიპროცესორების გამოყენება ხდება აგრეთვე ჩაშენებულ სისტემებში, მაგალითად, აუდიო–ვიზუალურ საყოფაცხოვრებო ელექტროტექნიკაში: ტელევიზორებში, DVD პლეერებში, ვიდეოკამერებში, მობილურ ტელეფონებში და ა.შ. ასეთ მოწყობილობებში დიდი მოთხოვნებია წაყენებული სწრაქმედების მიმართ და არსებობს აგრეთვე სხვადასხვა შეზღუდვები. მიუხედავად გარეგნული განსხვავებებისა, ეს მოწყობილობები ფაქტობრივად წარმოადგენს მინიატურულ კომპიუტერებს ერთი ან რამდენიმე პროცესორით, მეხსიერების მოდულებით, კონტროლერებით და შეტანა–გამოტანის სპეციალიზირებული მოწყობილობებით. მაგალითად, მობილური ტელეფონი წარმოადგენს კომპიუტერს, რომლის შემადგენლობაშიც შედის: პროცესორი, მეხსიერება, კომპაქტური კლავიატურა, მიკროფონი, ხმოვანი სიგნალის გახმოვანების სისტემა, უსადენო საქსელო ადაპტერი.

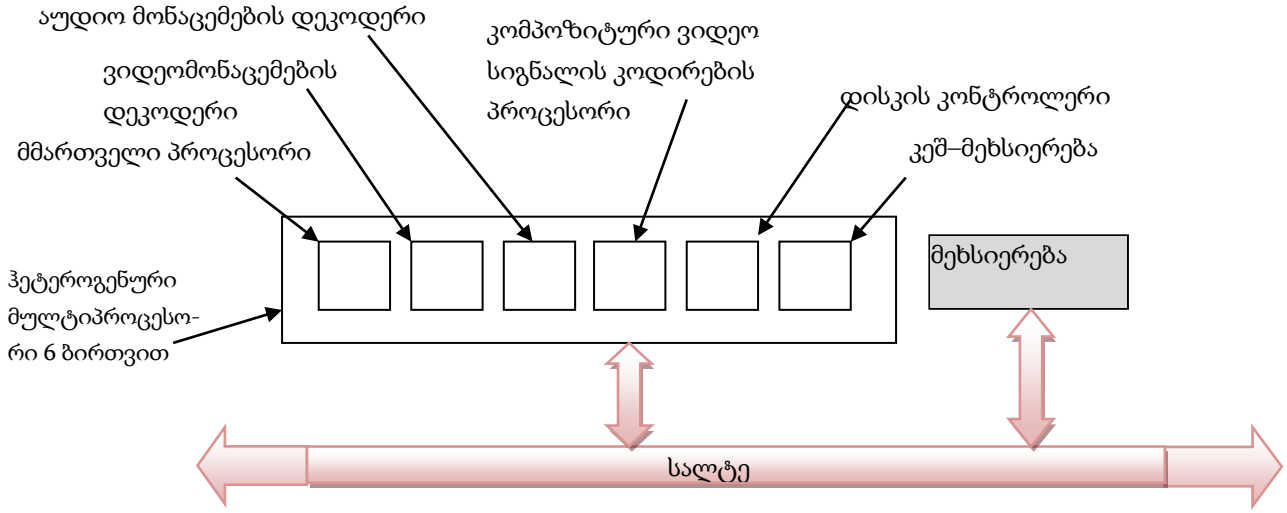
მაგალითისათვის განვიხილოთ DVD პლეერი. მასში გამოყენებული პროცესორის მიერ ხდება რამდენიმე ფუნქციის შესრულება:

1. სერვოძრავის მართვა, თავაკის მდებარეობის რეგულირებით,
2. შეცდომების კორექცია,
3. ანალოგური სიგნალის ციფრულ ფორმაში გადაყვანა,
4. დეშიფრაცია და უფლებების მართვა,
5. MPEG–2 ვიდეოფორმატის მონაცემების დეკომპრესია,
6. ხმოვანი სიგნალების დეკომპრესია, გამოსასვლელი მონაცემების კოდირება NTSC, PAL ან SECAM სისტემების ტელევიზორებისათვის.

ყველა ეს ფუნქცია შესრულებული უნდა იქნას დროის რეალურ მასშტაბში, დაცული უნდა იქნას მკაცრი მოთხოვნებიმომსახურების ხარისხზე, ენერგომოხმარებაზე, სითბოს გამოყოფაზე, ზომაზე, წონაზე და ფასზე.

DVD დისკებზე მონაცემები ჩაწერილია გრძელ სპირალზე. დისკის ბრუნვის დროს თავაკი უნდა მიჰყვებოდეს ამ სპირალს. ასეთი მოწყობილობების ფასი მაღალი არ არის იმის წყალობით, რომ გამოყენებულია მარტივი მექნიკური კონსტრუქცია თავაკის მდებარეობის კონტროლი ხდება პროგრამულად. თავაკიდან მოიხსნება ანალოგური სიგნალი, რომელიც დამუშავებამდე ჯერ გარდაქმნილი უნდა იქნას ციფრულ ფორმატში. გაციფრულების შემდეგ მიმდინარეობს შეცდომების ინტენსიური კორექცია, რომლებიც წარმოიქმნება დისკების დაშტამპვის დროს. დისკზე ვიდეო მონაცემები შენახულია MPEG–2 ფორმატში, რომელთა დეკომპრესიისათვისაც საჭიროა ფურიეს სწრაფი გარდაქმნის ტიპის რთული გამოთვლების ჩატარება. აუდიო მონაცემები შეკუმშულია ფსიქოაკუსტიკული მოდელის მიხედვით, რომლის დეკომპრესიაც ასევე რთულ საქმეს წარმოადგენს. და ბოლოს, აუდიო და ვიდეო მონაცემები მიყვანილი უნდა იქნას იმ ფორმამდე, რომლის წარმოდგენაც შესაძლებელი იქნება NTSC, PAL ან SECAM სისტემების ტელევიზორებში. ჩამოთვლილი სამუშაოები მოითხოვს საკმაოდ რთულ გამოთვლებს და დროის რეალურ მასშტაბში მათი პროგრამული გადაწყვეტა იაფფასიანი უნივერსალური პროცესორის საშუალებით შეუძლებელი იქნებოდა. ამგვარად, საჭიროა ჰეტეროგენული მულტიპროცესორი რამდენიმე სპეციალიზებული ბირთვით. DVD პლეერის ლოგიკური სქემა ნაჩვენებია ნახ. 1.2.2–ზე.

ნახ. 1.2.2–ზე გამოსახული ბირთვები ერთმანეთისაგან განსხვავდება ფუნქციონალური სპეციალიზაციის მიხედვით. ყოველი მათგანი დაპროექტებულია იმ გათვლით, რომ მინიმალური დანახარჯებით მიღწეული იქნას მაქსიმალური შედეგი. მაგალითად, შეკუმშული ვიდეოსიგნალები DVD დისკებზე ინახება MPEG–2 ფორმატში. შეკუმშვის დროს ყოველი კადრი იყოფა რამდენიმე ბლოკად და ყოველი მათგანისათვის ხდება რთული გარდაქმნების ჩატარება. კადრი შეიძლება შედგებოდეს მთლიანად შეცვლილი ბლოკებისაგან ან წინა კადრში არსებული ბლოკებისაგან, მაგრამ პიქსელებისათვის (Δx, Δy) წანაცვლების მითითებით. პროგრამულად ასეთი გამოთვლები სრულდება ძალიან ნელა, მაგრამ, MPEG–2 სიგნალების დეკოდირების პროცესორის არსებობის შემთხვევაში ეს პროცესი საკმაოდ სწრაფდება. ანალოგიურად, კომპოზიტური აუდიო–ვიდეო სიგნალის კოდირება და შემდეგ კვლავ დეკოდირება სატელევიზიო სიგნალების კოდირებისათვის მიღებულ რომელიმე სტანდარტში, ხელსაყრელია შესრულებული იქნას სპეციალიზებული აპარატული პროცესორის საშუალებით. აქედან გამომდინარე, სრულიად გასაგებია, რომ DVD პლეერებში და მსგავს მოწყობილობებში გამოყენებულია რამდენიმე ბირთვის მქონე ჰეტეროგენური მულტიპროცესორები. ამავდროულად, რადგანაც მმართველი პროცესორი წარმოადგენს უნივერსალურ დაპროგრამებად მოწყობილობას, ამიტომ მულტიპროცესორული მიკროსქემა შეიძლება გამოყენებული იქნას ფუნქციონალურად მსგავს მოწყობილობებში.



ნახ. 1.2.2. DVD პლეერის ლოგიკური სქემა 6 ბირთვიანი ჰეტეროგენული პროცესორით

ჰეტეროგენური მულტიპროცესორები ასევე გამოიყენება თანამედროვე მობილურ ტელეფონებში, რომელთაც გააჩნიათ ფოტო– და ვიდეოკამერები, თამაშები, ბრაუზერები, ელექტრონული ფოსტა, ციფრული თანამგზავრული სიგნალების მიმღებები, უსადენო ინტერნეტი (IEEE 802.11 ან WiFi). საყოფაცხოვრებო ტექნიკის ფუნქციონალური გართულების შედეგად ჰეტეროგენურ პროცესორებზე მოთხოვნილება მუდმივად გაიზრდება.

საკმაოდ მალე 500 მილიონი ტრანზისტორის შემცველი მიკროსქემები გახდება ჩვეულებრივი ამბავი. ასეთი დიდი მიკროსქემების მონოლითურ ვარიანტში დაპროექტება ძალიან რთულია. ეს იმდენად რთული, რომ არსებობს ალბათობა, რომ დაპროექტების დამთავრების მომენტისათვის ისინი უკვე მორალურად მოძველდებიან. გაცილებით გამართლებულია ერთ მიკროსქემაში მოთავსებული იქნას რამდენიმე ბირთვი და მოხდეს მათი ერთ სისტემაში გაერთიანება.

მმართველ პროცესორში პროგრამულ უზრუნველყოფაზე დატვირთვის ზრდა ანელებს მიკროსქემის მუშაობას, მაგრამ, ამავდროულად ამცირებს მის ზომებს და აიაფებს მას. მიკროსქემაში შეიძლება გამოყენებული იქმას რამდენიმე ერთნაირი სახეობის პროცესორი, მაგალითად აუდიო სიგნალების დამუშავების სპეციალიზებული პროცესორი. ასეთი მიდგომა იწვევს მიკროსქემის ფართობის გაზრდას, და რა თქმა უნდა მის გაძვირებას, მაგრამ მეორეს მხრივ, უზრუნველყოფს მიკროსქემის სტაბილურ და მაღალმწარმოებლურ მუშაობას უფრო დაბალ ტაქტურ სიხშირეზე, რის შედეგადაც მცირდება ენერგომოხმარება და შესაბამისად სითბოს გამოყოფაც. რა თქმა უნდა ეს ძალიან მნიშვნელოვანია.

აუდიო და ვიდეო მონაცემების დამუშავების პროგრამების საშუალებით ხდება ინფორმაციის ძალიან დიდი მოცულობის გადამუშავება. რადგანაც ამ მონაცემების გადამუშავება საჭიროა დროის რეალურ მასშტაბში, ამიტომ მათში საჭირო ხდება მიკროსქემის ფართობის 50%–დან 75%–მდე გამოყოფილი იქნას მეხსიერების ამა თუ ინ მოწყობილობისათვის. ამასთან დაკავშირებით წარმოიქმნება მრავალი შეკითხვა. ამა თუ იმ შემთხვევაში კეშ–მეხსიერების რამდენი დონეა საჭირო? როგორი უნდა იყოს კეშ–მეხსიერების მოდულები, დაყოფილი თუ გაერთიანებული? რამდენი უნდა იყოს თითოეული მოდულის მოცულობა? როგორ სწრაფად უნდა მუშაობდნენ ეს მოდულები? საჭიროა თუ არა მიკროსქემაში მეხსიერების სხვა ტიპის მოდულების განთავსება, მაგალითად SRAM თუ SDRAM? ამ კითხვებზე სწორი პასუხების გაცემით შეიძლება გაზრდილი იქნას მიკროსქემის მწარმოებლურობა, შემცირებული იქმას ენერგომოხმარება და სითბოს გამოყოფა.

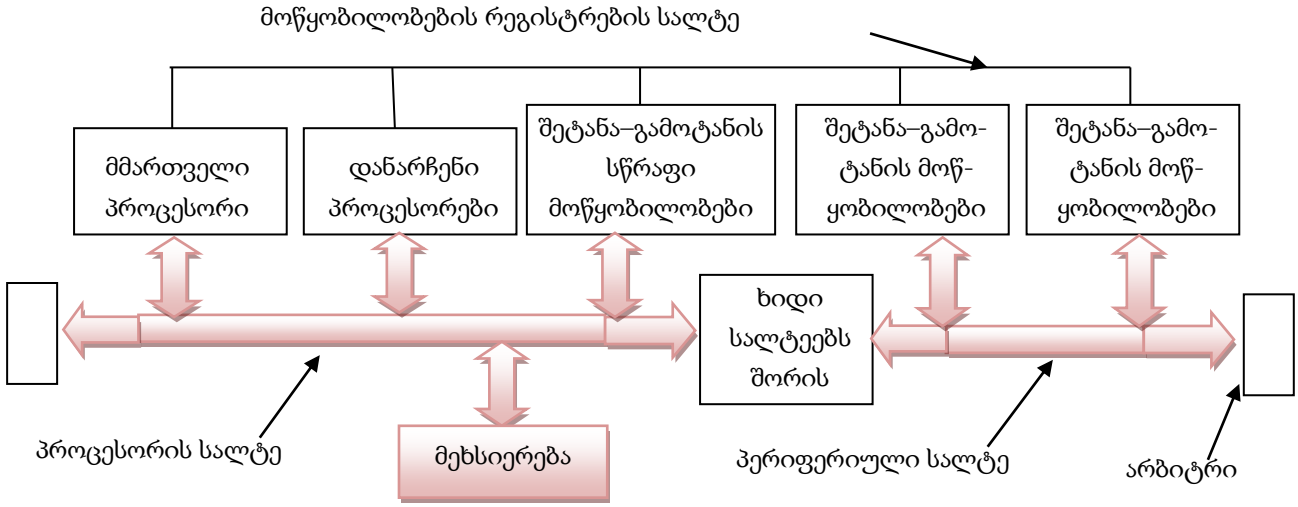
პროცესორებისა და მეხსიერების მოწყობილობების დაპროექტებასთან ერთად საჭიროა აგრეთვე ბირთვებს შორის ურთიერთქმედების სქემის დამუშავებაც. პატარა სისტემებში ამ მიზნისათვის სავსებით საკმარისია საერთო სალტე, მაგრამ უფრო დიდ სისტემებში ეს საერთო სალტე შეიძლება გახდეს ვიწრო ყელი მთელ სისტემაში. ხშირ შემთხვევებში გამოსავალს წარმოადგენს ერთდროულად რამდენიმე სალტის გამოყენება და მათი ერთმანეთთან დაკავშირება, ან რგოლური ტოპოლოგიის ორგანიზება. უკანასკნელ შემთხვევაში არბიტრაჟის განხორციელება ხდება რგოლზე სპეციალური მცირე პაკეტის – **მარკერის** (token) გაგზავნით. ბირთვს მონაცემების გადაცემა შეუძლია თუ არის მარკერის მფობელი და ინარჩუნებს ამ მარკერს მონაცემების გადაცემის მთელ პროცესში. მონაცემების გაცემის დამთავრების შემდეგ ბირთვი მარკერს გადასცემს სხვა ბირთვს. მარკერის გადაცემა ხდება წრიულად. ამგვარად, გამორიცხულია კონფლიქტები მონაცემების გადაცემის დროს.

ბირთვების თანამოქმედების მაგალითის სახით განვიხილოთ IBM კომპანიის მიერ დამუშავებული არქიტექტურა **CoreConnect** (ნახ. 1.2.3). ის განკუთვნილია ერთკრისტალიან ჰეტეროგენულ მულტიპროცესორებში ბირთვების გასაერთიანებლად. ერთკრისტალიან მულტიპროცესორებში CoreConnect ასრულებს იგივე ფუნქციას, რასაც PCI სალტე Pentium-ებში, მაგრამ იმ განსხვავებით, რომ მათში არაა გათვალისწინებული თავსებადობა წინამორბედ მოდელებთან.

CoreConnect არქიტექტურა შედგება სამი სალტისაგან. **პროცესორის სალტე** წარმოადგენს მაღალსიჩქარიან კონვეირიზირებულ სინქრონულ სალტეს 66, 133 ან 183MHz ტაქტურ სიხშირეზე მომუშავე 32, 64 ან 128 საინფორმაციო ხაზით. მისი მაქსიმალური გამტარუნარიანობაა 23,4 Gბიტი წამში (შედარებისათვის, PCI სალტისათვის იგივე მაჩვენებელი წარმოადგენს 4,2 Gბიტი წამში). კონვეიერიზაცია ბირთვებს საშუალებას აძლევთ მოითხოვონ სალტე მონაცემების გადაცემის პროცესში. გარდა ამისა, ისევე როგორც PCI სალტეში, აქაც ბირთვებს შეუძლიათ მონაცემები

გადასცენ სხვადასხვა ხაზებით. პროცესორის სალტე ოპტიმიზირებულია მონაცემების მცირე ზომის ბლოკების გადასაცემად და უზრუნველყოფს სწრაფ ბირთვებს შორის თანამოქმედებას.

მთელი მიკროსქემისათვის მონაცემების გაცვლის პროცესორის სალტე საკმარისი არაა,



ნახ. 1.2.3. IBM კომპანიის არქიტექტურა CoreConnect

ამიტომ, ნელ მოწყობილობებს შორის (UART, ტაიმერები, USB–კონტროლერები, შეტანა–გამოტანის მიმდევრობითი მოწყობილობები და ა.შ.) მონაცემების გაცვლისათვის დამატებით გაითვალისწინეს მეორე **პერიფერიული სალტე**. ის ამარტივებს მონაცემების გაცვლას 8–, 16– და 32–თანრიგიან პერიფერიულ მოწყობილობებს შორის და თანაც გამოიყენებს მხოლოდ რამდენიმე ასეულ ვენტილს. პერიფერიული სალტეც ასევე არის სინქრონული და მისი მაქსიმალური გამტარებლობა არის 300 Mბიტი წამში. ეს ორი სალტე ერთმანეთთან დაკავშირებულია ხიდის საშუალებით, რომელიც მოგვაგონებს PCI და ISA სალტეების დამაკავშირებელ ხიდს, ვიდრე ISA სალტეს საერთოდ არ ამოიღებდნენ კომპიუტერის შემადგენლობიდან.

CoreConnect არქიტექტურაში ასევე გამოყენებულია **მოწყობილობების რეგისტრების სალტე**. ეს უკიდურესად ნელი ასინქრონული სალტეა, რომელიც პროცესორებს აკავშირებს პერიფერიული მოწყობილობების რეგისტრებთან მათი მართვის მიზნით. ამ სალტეზე მონაცემების გაცვლა არარეგულარულია და მონაცემები გადაიცემა რამდენიმე ბაიტიანი პორციებით.

CoreConnect არქიტექტურის გარდა არსებობს მიკროსქემებში სალტეების ჩაშენების კიდევ მრავალი სხვა არქიტექტერა. მას პოპულარობით არაფრით არ ჩამოუვარდება სალტეების არქიტექტურა **AMBA** (Advanced Microcontroller Bus Architecture - გაფართოებული სალტური არქიტექტურა მიკროკონტროლერებისათვის). ცოტა ნაკლებად პოპულარულია **VCI** (Virtual Component Interconnect - ვირტუალური კომპონენტების თანამოქმედება) და **OCP-IP** (Open Core Protocol-International Partnership –ბირთვის ღია პროტოკოლის საერთაშორისო კონსორციუმი).

მიკროსქემების მწარმოებლები ვერ ახერხებენ ტაქტური სიხშირის მუდმივად ზრდას, რასაც აბრკოლებს ენერგომოხმარებისა და გამოყოფილი სითბოს რაოდენობის ზრდა. ამდენად, ერთკრისტალიანი მულტიპროცესორების წარმოება მეტად აქტუალურია. ამ გზაზე მიკროსქემაში

ჩადგმული სალტეების დაპროექტება ჯერ დასაწყისია. ახლა უკვე მიმდინარეობს მიკროსქემებში მთელი ქსელების პროექტირება.

1.3. თანაპროცსორები

პროცესორსშიდა პარალელიზმის გარჩევის შემდეგ, განვიხილოთ კომპიუტერის სწრაფქმედების გაზრდის ვარიანტები, რომლებშიც სწრაფქმედების გაზრდა მიიღწევა მის შემადგენლობაში მეორე, სპეციალიზებული პროცესორის შემოტანით. ასეთი **თანაპროცესორები** ძალიან მრავალფეროვანია. IBM მაინფრეიმებში და ამ სერიის მომდევნო მოდელებში მონაცემების შეტანა–გამოტანისათვის გათვალისწინებული იყო დამოუკიდებელი მულტიპლექსორული და სელექტორული არხები. CDC 6600–ის შემადგენლობაში იყო 10 დამოუკიდებელი პროცესორი, რომელთა საშუალებითაც ხდებოდა მონაცემების შეტანა–გამოტანის პროცესის მართვა. თანაპროცესორების გამოყენების სხვა სფეროა გრაფიკის დამუშავება და მცოცავწერტილიან მონაცემებზე არითმეტიკული ოპერაციების შესრულება. DMA კონტროლერიც კი შეიძლება განხილული იქნას, როგორც თანაპროცესორი. ზოგჯერ პროცესორი თანაპროცესორს შესასრულებლად გადასცემს ბრძანებას, ან ბრძანებების ნაკრებს, სხვა შემთხვევებში კი თანაპროცესორი მოქმედებს დამოუკიდებლად და ასრულებს თავის ბრძანებებს.

კონსტრუქციულად თანაპროცესორები შეიძლება შესრულებული იყოს სხვადასხვანაირად. თანაპროცესორი შეიძლება მოთავსებული იყოს დამოუკიდებელ კორპუსში (მაგალითად, შეტანა–გამოტანის არხი IBM 360-ში), ან როგორც მისაერთებელი პლატა (საქსელო პროცესორები), ან კიდევ ჩადგმული იყოს ძირითადი პროცესორის მიკროსქემაში (თანაპროცესორი მცოცავწერტილიანი მონაცემების დამუშავებისათვის). კონსტრუქციული გადაწყვეტა როგორიც არ უნდა იყოს, მათთვის საერთოა ის, რომ მათ ყველა შემთხვევაში გააჩნიათ ცენტრალურ პროცესორზე დამოკიდებული როლი. შემდეგში განხილული იქნება რამდენიმე სფერო, რომლებშიც თანაპროცესორების გამოყენება მნიშვნელოვნად ზრდის კომპიუტერის საერთო მწარმოებლურობას.

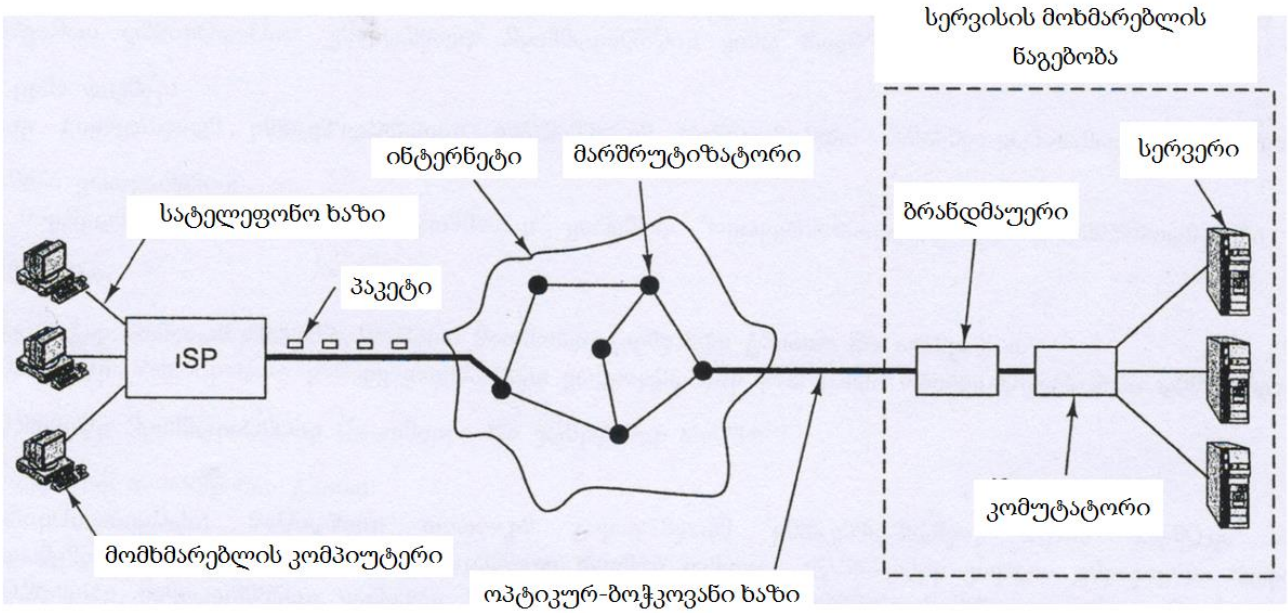
1.3.1. საქსელო პროცესორები

თანამედროვე კომპიუტერების უმეტესობა მიერთებლია ან ლოკალურ ქსელთან ან ინტერნეტთან. საქსელო მოწყობილობებში ტექნოლოგიური პროგრესის შედეგად თანამედროვე ქსელებში მონაცემების გადაცემა ხდება ისე სწრაფად, რომ შემოსული და გასული მონაცეების პროგრამული დამუშავება სულ უფრო და უფრო რთულდება. ამის გამო ხდება ტრაფიკის დამუშავებისთვის განკუთვნილი სპეციალური საქსელო პროცესორების დამუშავება. დღეს ამ პროცესორებით აღჭურვილია უკვე ბევრი პროფესიონალური გამოთვლითი სისტემა.

1.3.1.1. ქსელების შესახებ

გამოთვლითი ქსელები იყოფა ორ ძირითად ჯგუფად: **ლოკალური ქსელები** (Local Area Networks, LAN) აერთიანებენ ერთ შენობაში განლაგებულ კომპიუტერებს და **გლობალური ქსელები** (Wide Area Networks, WAN)ერთმანეთთან აკავშირებს დიდი მანძილით დაცილებულ

კომპიუტერებს. ლოკალური ქსელების ყველაზე პოპულარულ ტიპს წარმოადგენს Ethernet. თანამედროვე ქსელებში Ethernet კომპიუტერები უკავშირდება ცენტრალურ კომუტატორს (ნაჩვენებია ნახ. 1.3.1–ზე). Ethernet ქსელის პირველ ვერსიებში მონაცემების გადაცემის მაქსიმალური სიჩქარე იყო 3 Mბიტი წამში, პირველ კომერციულ ვერსიებში ეს სიჩქარე გაიზარდა 10 M ბიტიამდე წამში. შემდეგ გამოჩნდა Fast Ethernet (გადაცემის სიჩქარით 100 Mბიტი წამში) და Gigabit Ethernet (სიჩქარით 1 G ბიტი წამში). უკვე გამოშვებულია კომერციული ვერსიები სიჩქარით 10 G ბიტი წამში და 40 G ბიტი წამში.



ნახ. 1.3.1. ინტერნეტის სერვერებთან მომხმარებლების მიერთების სქემა

გლობალური ქსელების სტრუქტურა ასეთი ერთგვაროვანი არ არის. ასეთ ქსელებში გამოყენებულია სპეციალური კომპიუტერები – **მარშრუტიზატორები**, რომლებიც ერთმანეთთან დაკავშირებულია მავთულის ან უფრო ხშირად ოპტიკურ–ბოჭკოვანი კავშირის არხების საშუალებით. საწყისი კომპიუტერიდან ადრესატისათვის მონაცემები გადაიცემა პატარა ბლოკების სახით, მოცულობით 64–1500 ბაიტი, რომლებსაც **პაკეტებს** უწოდებენ. ყოველ სარეტრანსლაციო უბანზე, პაკეტი ჯერ შეინახება მარშრუტიზატორის მეხსიერებაში და შემდეგ, როგორც კი განთავისუფლდება საჭირო კავშირის არხი, გაეგზავნება შემდეგ მარშრუტიზატორს. ასეთ მიდგომას უწოდებენ **კომუტაციას პაკეტების შენახვითა და გაგზავნით** (store-and_forward packet switching).

ერთი შეხედვით ინტერნეტი წარმოადგენს გლობალურ ქსელს, მგარამ ტექნიკურად იგი წარმოადგენს დიდი რაოდენობით სხვადასხვა სახეობის ქსელების გაერთიანებას. ნახ. 1.3.1–ზე ინტერნეტის სტრუქტურა ნაჩვენებია სახლიდან მიერთებული მომხმარებლის პოზიციიდან. ასეთი სახის მომხმარებლები ვებ–სერვერებს უკავშირდებიან სატელეფონო კავშირის არხებით, რომლებთანაც მიერთებული არიან მოდემების საშუალებით. ქსელთან მიერთებისათვის შეიძლება გამოყენებული იყოს საკაბელო ტელევიზიის კაბელიც. მომხმარებლის კომპიუტერში ხდება შეტყობინების პაკეტებად დანაწევრება, რათა ამ სახით მიაწოდოს **ინტერნეტის მომსახურების მიმწოდებელს** (Internet Service Provider, **ISP**). თვითონ ინტერნეტ სერვისების მიმწოდებელი გამოიყენებს მაღალ-

სიჩქარიან (როგორც წესი, ოპტიკურ ბოჭკოვან კავშირის არხებს) კავშირის არხებს. პაკეტები შუალედური კვანძების გავლით გადაეცემა სერვერებს.

ვებ–მომსახურების მიმწოდებელი კომპანიების უმრავლესობაში გამოყენებულია ე.წ. **ბრანდმაუერები**, ანუ სპეციალიზებული კომპიუტერები, რომელთა საშუალებითაც ხდება შემომავალი ნაკადის გაფილტვრა არასასურველი შეტყობინებებისაგან ან ჰაკერების შემოტევებისაგან. უმეტეს შემთხვევებში, კომუტატორების საშუალებით ხდება ბრანდმაუერების მიერთება ლოკალურ ქსელებთან, რომელთა საშუალებითაც ხდება პაკეტების საჭირო სერვერისათვის მიწოდების უზრუნველყოფა. რა თქმა უნდა რეალობა გაცილებით რთულია, მაგრამ ნახ. 1.3.1–ზე წარმოდგენილი იდეები სწორია.

საქსელო პროგრამული უზრუნველყოფა შედგება რამდენიმე განაწესისაგან, რომელთაგანაც თითოეული წარმოადგენს წესებისა და გაცვლის მიმდევრობების ფორმატების ნაკრებს, რომელთა საშუალებითაც ხდება პაკეტების დანიშნულების განსაზღვრა. მაგალითად, როდესაც მომხმარებელს სჭირდება მიიღოს სერვერის ვებ–გვერდი, მისი ბრაუზერი **HTTP** (Hyper Text Transfer Protokol - ჰიპერტექსტების გადაცემის განაწესი) განაწესის მიხედვით სერვერს უგზავნის პაკეტებს GET PAGE შეკვეთებით, რომელმაც „იცის“ როგორ დაამუშაოს მიღებული პაკეტი. გადაცემის პროცესში, ხშირად ერთდროულად, გამოიყენება პაკეტების გადაცემის სხვადასხვა სახეობის განაწესები. ეს პაკეტები გაერთიანებულია იერარქიულ მრავალდონიან სტრუქტურაში, რომელშიც ზედა დონის განაწესები გადსცემენ პაკეტებს ქვედა დონის განაწესებს, ხოლო მონაცემების რეალურ გადაცემაზე პასუხისმგებელია ყველაზე ქვედა დონის განაწესი. მიმღებ მხრეზე პაკეტების მიღება ხდება საპირისპირო მიმართულებით, ქვედა დონიდან ზევით.

რადგანაც საქსელო პროცესორების დანიშნულებას წარმოადგენს განაწესების შესრულება, ამიტომ საქსელო პროცესორების მუშაობის გარჩევამდე კარგი იქნება განაწესების უფრო დაწვრილებით გარჩევა. დავუბრუნდეთ GET PAGE შეკვეთებს. როგორ ხვდება ის ვებ–სერვერზე? უპირველეს ყოვლისა, ბრაუზერი ვებ–სერვერთან ამყარებს კავშირს TCP (Transmission Control Protocol - გადაცემის მართვის განაწესი) განაწესის საშუალებით. ამ განაწესის რეალიზების პროგრამული უზრუნველყოფა თვალყურს ადევნებს, რომ ყველა პაკეტი სწორი თანმიმდევრობით მიტანილი იქნას ადრესატამდე. პაკეტის დაკარგვის შემთხვევაში, TCP განაწსი მაქსიმალურად სწრაფად იმეორებს დაკარგული პაკეტის გადაცემას, ვიდრე არ მოხდება მისი ადრესატისათვის გადაცემა.

რეალურად ხდება შემდეგი. ბრაუზერი ახდენს HTTP–შეტყობინების კორექტულად ფორმირებას GET PAGE შეკვეთით და შემდეგ გადასცემს TCP პროგრამულ უზრუნველყოფას, რომელიც უზრუნველყოფს მის შუალედურ კვანძებში გადაცემას. TCP პროგრამული უზრუნველყოფა პაკეტის დასაწყისში ამატებს სათაურს, რომელიც შეიცავს მის ნომერს და სხვა საჭირო ინფორმაციას. ამ სათაურს უწოდებენ **TCP–სათაურს**.

თავისი სამუშაოს შესრულების შემდეგ, TCP პროგრამულ უზრუნველყოფაTCP–სათაურს, მის დანარჩენ ნაწილთან ერთად გადასცემს კიდევ ერთ პროგრამას, რომლის საშუალებითაც ხდება **IP** (Internet Protocol –**საქსელო განაწესი**) განაწესის რეალიზება. ეს პროგრამა პაკეტის დასაწყისში IP-სათაურს ამატებს ინფორმაციას, რომელიც მოიცავს გამგზავნის და მიმღების მისამართებს, შუალედური კვანძების მაქსიმალურ რაოდენობას (რათა ამ რაოდენობა კვანძების გავლის შემთხვევაში

მოხდეს პაკეტის განადგურება, რათა პაკეტმა არ იარსებოს უსასრულოდ დიდხანს), საკონტროლო ჯამს (გადაცემის დროს დაშვებული შეცდომის აღმოსაჩენად) და კიდევ სხვა ველებს.

შემდეგ, IP- და TCP–სათაურების შემცველი პაკეტი გადაეცემა „ქვევით“, მონაცემების გადაცემის არხის დონეს, რომელიც ასევე დაუმატებს თავისი დონის შესაბამის სათაურს და გადასცემს კავშირის არხში. ამ დონეზე ასევე ხდება პაკეტის ბოლოს საკონტროლო ჯამის მიწერა, კერძოდ, CRC (Cyclic Redundancy Check– სიჭარბის ციკლური კონტროლი, რომლის საშუალებითაც შესაძლებელია გადაცემის პროცესში წარმოქმნილი შეცდომების აღმოჩენა). შეიძლება მოგვეჩვენოს, რომ ორი საკონტროლო ჯამი, IP დონეზე და მონაცემების გადაცემის არხის დონეზე, ცოტა ზედმეტია, მაგრამ ასეთი მიდგომა ზრდის საიმედოობას. პაკეტის CRC–კოდის შემოწმება ხდება ყველა შუალედურ კვანძში, რის შემდეგაც ხდება პაკეტის სათაურის ხელახალი ფორმირება წყარო აბონენტის მონაცემების გადაცემის დონის მოთხოვნების შესაბამისად. ნახ. 1.3.2–ზე ნაჩვენებია პაკეტის სახე Ethernet ქსელისათვის. პაკეტების სათაურების დამუშავება ქსელის ერთერთი მნიშვნელოვანი ამოცანაა, რომლის გადაწყვეტაც ხდება საქსელო პროცესორების საშუალებით.

Ethernet სათაური	IP სათაური	TCP სათაური	პაკეტის მონაცემები	CRC

ნახ. 1.3.2. Ethernet ქსელის პაკეტი

1.3.1.2. საქსელო პროცესორების ძირითადი მონაცემები

კომპიუტერულ ქსელებთან ხდება სხვადასხვა სახეობის მოწყობილობების მიერთება. მომხმარებლებისათვის, უპირველეს ყოვლისა, ესაა პერსონალური კომპიუტერები და სერვერები. ქსელებში გამოჩნდა კიდევ ახალი მოწყობილობები, როგორიცაა: სათამაშო კონსოლები, პერსონალური ელექტრონული მდივნები (ჯიბის კომპიუტერები), მობილური ტელეფონები. ამათ გარდა, ქსელებში უამრავი რაოდენობით ფუნქციონირებს კიდევ სხვასხვაგვარი შუალედური მოწყობილობა, მათ რიცხვში შედის: მარშრუტიზატორები, კომუტატორები, ბრანდმაუერები, პროქსი–სერვერები, დატვირთვის ბალანსირების სისტემები. აღსანიშნავია, რომ ამ შუალედურ მოწყობილობებს წაყენებული აქვს საკმაოდ მკაცრი მოთხოვნები, სწორედ მათ უნდა უზრუნველყონ ქსელების მაღალი გამტარუნარიანობა. სერიოზული მოთხოვნები წაეყენება აგრეთვე სერვერებსაც. ყველაზე ნაკლები მოთხოვნები წაყენებული აქვს მომხმარებლების კომპიუტერებს.

პაკეტებს კავშირის არხებში გაცემის წინ, ან გამოყენებითი პროგრამებისათვის დაბრუნების წინ შეიძლება დასჭირდეთ გარკვეული დამუშავება, რომელიც დამოკიდებულია, როგორც ქსელის, ასევე პაკეტის სახეობაზე. პაკეტის დამუშავება მოიცავს შემდეგ სახეობებს: გადაწყვეტილების მიღებას იმის შესახებ, თუ სად გადაიგზავნოს მოცემული პაკეტი, პაკეტის დაშლას ნაწილებად, ან პირიქით, მის აწყობას ნაწილებისაგან, მომსახურების ხარისხის მართვას (განსაკუთრებით აუდიო და ვიდეო პაკეტებისათვის), მონაცემების დაცვას (კოდირება და დეკოდირება), მონაცემების კომპრესიასა და დეკომპრესიას და ა.შ.

როდესაც ლოკალურ ქსელში მონაცემების გადაცემის სიჩქარე აღწევს 40 G ბაიტს წამში და პაკეტების ზომაა 1 K ბაიტი, მაშინ საქსელო კომპიუტერს უწევს წამში 5 მილიონი პაკეტის გადამუშავება. პაკეტების ზომა თუ იქნება 64 ბაიტი, მაშინ პაკეტების დამუშავების სიჩქარე იზრდება

წამში 80 მილიონ პაკეტამდე. ყოველი პაკეტისათვის ჩამოთვლილი სამუშაოების შესასრულებლად რჩება დრო 12–200 ნანო წამი. ცხადია, რომ პროგრამულად ამ სამუშაოების შესრულება ვერანაირად ვერ მოხერხდება და აპარატული მხარდაჭერა პრინციპული ხდება.

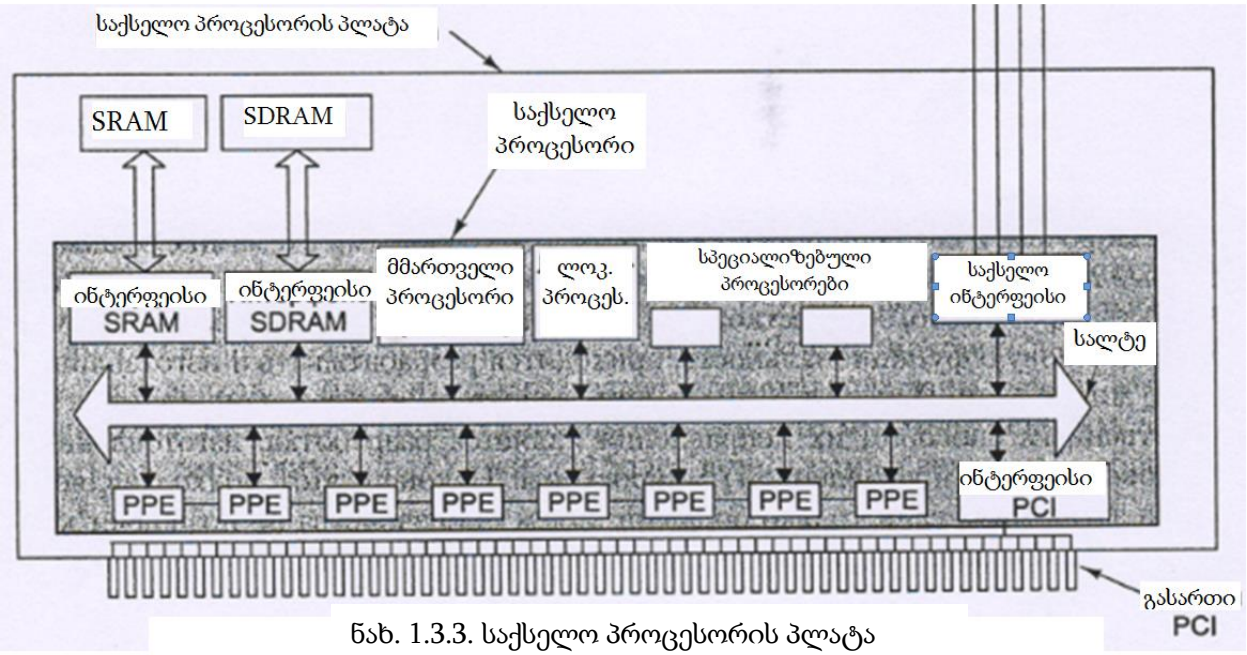
პაკეტების სწრაფი დამუშავების პრობლემის აპარატული გადაწყვეტის ერთერთ გზას წარმოადგენს **სპეციალიზებული ინტეგრალური სქემების** (Application-Specific Integrated Circuit, **ASIC**) გამოყენება. ასეთი მიკროსქემა წააგავს პროგრამის აპარატულ რეალიზებას, რომელსაც შეუძლია წინასწარგათვალისწინებული ნებისმიერი მოქმედების შესრულება. თანამედროვე მარშრუტიზატორების საფუძველს სწორედ ეს მიკროსქემები წარმოადგენს. სპეციალიზებული მიკროსქემების გამოყენებასთანაც არის დაკავშირებული გარკვეული პრობლემები. უპირველეს ყოვლისა მათი დაპროექტება და წარმოება დიდ დროს მოითხოვს. და კიდევ, ეს მიკროსქემები წარმოადგენენ ხისტაი ლოგიკით დამზადებულ მიკროპროცესორებს და ექსპლუატაციის შედეგად, მათ ფუნქციებში თუ ცვლილების შეტანა გახდა საჭირო, მაშინ მთელი მიკროსქემა თავიდან ხდება დასაპროექტებელი და დასამზადებელი. კიდევ უფრო უარეს პრობლემას წარმოადგენს შეცდომები. დაპროექტების დროს თუ დაშვებული იქნა შეცდომა, მისი აღმოჩენაც რთულია და შეცდომის აღმოჩენის შემდეგ, გასწორებული მიკროსქემა თავიდან ხდება დასაპროექტებელი და დასამზადებელი. ცხადია, რომ ყველაფერი ეს დაკავშირებულია დიდ დანახარჯებთან, რომლის გამართლებაც შესაძლებელი იქნება დაპროექტებული მიკროსქემების დიდი რაოდენობით დამზადებით და დანერგვით.

მეორე მიდგომა დამყარებულია **დაპროგრამებადი ველტილური მატრიცების** (Field Programmable Gate Array, **FPGA**) გამოყენებაზე. ასეთი მატრიცი წარმოადგენს ველტილების ნაკრებს, რომელთა გადაკომუტირების საშუალებითაც ხდება საჭირო სქემების აგება. სპეციალურ ინტეგრალურ სქემებთან შედარებით, დაპროგრამებადი ველტილური მატრიცების დანერგვის დრო გაცილებით ნაკლებია. გარდა ამისა, შესაძლებელია ასეთი მატრიცების „საველე პირობებში“ გადაპროგრამება სპეციალური პროგრამატორების საშუალებით. და ბოლოს, ამ მატრიცების შესახებ შეიძლება ითქვას, რომ ისინი არიან საკმაოდ რთული, ძვირი და ნელი ASIC სქემებთან შედარებით, ამიტომაც მათ ვერ მოიპოვეს საყოველთაო გავრცელება და გამოიყენება სპეციალიზებულ შემთხვევებში.

და ბოლოს, საქსელო პროცესორები – წარმოადგენენ მოწყობილობებს, რომელთაც შეუძლიათ შემავალი და გამომავალი პაკეტების დამუშავება კავშირის არხებში მათი გადაცემის სიჩქარით, ანუ მუშაობენ დროის რეალურ მასშტაბში. როგორც წესი, მათი დამზადება ხდება დამოუკიდებელი პლატების სახით, რომელზეც მოთავსებულია საქსელო პროცესორის კრისტალი, მეხსიერება და დამხმარე ლოგიკა. პლატასთან შესაძლებელია ერთი ან რამდენიმე საქსელო კავშირის არხის მიერთება. პროცესორი კავშირის არხებიდან ღებულობს პაკეტებს, ახდენს მათ გადამუშავებას და აგზავნის სისტემურ სალტეზე (PCI სალტეზე). კომუტატორებისა და მარშრუტიზატორების შემთხვევაში კი პაკეტები მიიღება კავშირის ერთი არხიდან და გადამუშავების შემდეგ ხდება მათი გაგზავნე სხვა არხში. ტიპიური საქსელო პროცესორი და მისი პლატა ნაჩვენებია ნახ. 1.3.3–ზე.

საქსელო პლატაზე ჩვეულებრივ გამოყენებულია როგორც სტატიკური (SRAM), ასევე სინქრონული დინამიური ოპერატიული მეხსიერება (SDRAM). მეხსიერების ეს სახეობები გამოიყენება სხვადასხვა დანიშნულებით. SRAM მეხსიერება უფრო სწრაფია ვიდრე SDRAM მეხსიერება,

მაგრამ სიძვირის გამო, მისი მოცულობა ნაკლებია. ის გამოიყენება მარშრუტიზაციის ცხრილების და მონაცემების სხვა ძირითადი სტრუქტურების შესანახად. SDRAM მეხსიერებაში კი ხდება გადასამუშავებელი პაკეტების შენახვა. იმის გამო, რომ მეხსიერების ეს სახეობები განთავსებულია პროცესორის კრისტალისაგან დამოუკიდებლად, შესაძლებელია მეხსიერების მოცულობის მიმართ უფრო მოქნილი მიდგომის გამოყენება. მარტივ შემთხვევებში, კავშირის ერთი არხის არსებობის შემთხვევაში (პერსონალური კომპიუტერები ან სერვერები) მეხსიერების მოცულობა შეიძლება არ იყოს დიდი, ხოლო მარშრუტიზატორების შემთხვევაში მეხსიერება, მოითხოვება გაცილებით დიდი მოცულობით.



ნახ. 1.3.3. საქსელო პროცესორის პლატა

საქსელო პროცესორები ოპტიმიზებულია შემავალი და გამომავალი პაკეტების გაცილებით უფრო დიდი რაოდენობის დასამუშავებლად. ეს ნიშნავს, რომ კავშირის ყოველ არხში წამში ხდება მილიონობით პაკეტების გატარება, ხოლო მარშრუტიზატორმა უნდა მოახდინოს ათეულობით ასეთი კავშირის არხის მხარდაჭერა. ასეთი სერიოზული მაჩვენებლების მიღწევა შესაძლებელია მხოლოდ შიდა პარალელიზმის მაღალი დონის მქონე საქსელო პროცესორებში. გარდა ამისა, საქსელო პროცესორის შემადგენლობაში აუცილებლად შედის რამდენიმე **PPE–კონტროლერი** (Protocol/Programmable/Packet Processing Engine – პაკეტებისა და განაწესების დამუშავების დაპროგრამებადი სისტემა), რომელთაგან თითოეული მოიცავს RISC ბირთვს და მცირე მოცულობის შიდა მეხსიერებას პროგრამისა და რამდენიმე ცვლადის მნიშვნელობების დასამახსოვრებლად.

არსებობს PPE–კონტროლერების ორგანიზების ორი მიდგომა. მარტივ შემთხვევაში PPE–კონტროლერების აგება ხდება იდენტურად. საქსელო პროცესორში პაკეტის შემოსვლის შემდეგ ის შესასრულებლად გადაეცემა PPE–კონტროლერს, რომელიც მოცემულ მომენტში უქმადაა. თუ თავისუფალი PPE–კონტროლერი არ არსებობს, მაშინ პაკეტი იკავებს რიგს პლატის SDRAM მეხსიერებაში და უცდის ერთერთი PPE–კონტროლერის გათავისუფლებას. ასეთი ორგანიზაციის შემთხვევაში, ნახ. 1.3.3–ზე ნაჩვენები ჰორიზონტალური კავშირები არ არსებობს, რადგანაც სხვადასხვა დონის PPE–კონტროლერებს არ სჭირდებათ ერთმანეთთან ურთიერთობა.

PPE–კონტროლერების ორგანიზების სხვა შემთხვევაში, კონვეიერში თითოეული PPE–კონტროლერი ასრულებს დამუშავების ერთ ეტაპს, რის შემდეგაც მიღებულ პაკეტზე მიმთითებელს გადასცემს კონვეიერის შემდეგ PPE–კონტროლერს. ასეთი კონვეიერი მუშაობს ცენტრალური პროცესორის კონვეიერის ანალოგიურად. PPE–კონტროლერების ორგანიზების ორივე შემთხვევაში მათი ორგანიზების პრინციპი არის დაპროგრამებადი.

უფრო სრულყოფილ საქსელო პროცესორებში PPE–კონტროლერებში ხდება მრავალნაკადურობის პრინციპის მხარდაჭერა, ანუ ყოველ პროგრამულ ნაკდს გააჩნია რეგისტრების რამდენიმე ნაკრები და აპარატული რეგისტრი, რომელიც მიუთითებს, თუ რომელი მათგანი გამოიყენება. ეს საშუალებას იძლევა ერთდროულად შესრულებული იქნას რამდენიმე პროგრამა (ან პროგრამული ნაკადი) ან მოხდეს მათ შორის გადართვები „რეგისტრების მომდინარე ნაკრების“ გადართვების შეცვლის საშუალებით. როდესაც ერთერთი პროგამული ნაკადი იძულებულია მოიცადოს (მაგალითად SDRAM მეხსიერებასთან მიმართვის დროს, რასაც რამდენიმე ციკლი სჭირდება), მაშინ PPE–კონტროლერი შეიძლება მყისიერად გადართული იქნას იმ ნაკადზე, რომლის მუშაობის გაგრძელებაც შესაძლებელია. ეს შესაძლებლობას იძლევა მიღწეული იქნას PPE–კონტროლერების დატვირთვის მაღალი მაჩვენებელი, მიუხედავად იმისა, რომ SDRAM მეხსიერებასთან ან სხვა გარე მოწყობილობებთან მონაცემების გაცვლის პროცესში იქმნება მოლოდინის საჭიროება.

ყველა საქსელო პროცესორს გააჩნია მმართველი პროცესორი იმ ოპერაციების შესასრულებლად, რომლებიც არ არიან უშუალოდ დაკავშირებული პაკეტების დამუშავებასთან (მაგალითად, მარშრუტიზაციის ცხრილების განახლება). ჩვეულებრივ ის წარმოადგენს საერთო დანიშნულების RISC–პროცესორს, რომლის ბრძანებების და მონაცემების მეხსიერებაც მოთავსებულია ერთ კრისტალში პროცესორთან ერთად. უფრო მეტიც, საქსელო პროცესორში შეიძლება გამოყენებული იყოს ძალიან მნიშვნელოვანი ოპერაციების შესრულებისათვის განკუთვნილი რამდენიმე სპეციალიზებული პროცესორი. ისინი წარმოადგენენ მცირე ზომის სპეციალიზებულ ინტეგრალურ სქემებს (ASIC), რომელთაც შეუძლიათ შეასრულონ მხოლოდ ერთი მარტივი მოქმედება, ისეთი, როგორიცაა მარშრუტიზაციის ცხრილში მიზნობრივი მისამართის მოძებნა. საქსელო პროცესორის ყველა კომპონენტი ერთმანეთთან კავშირს ამყარებს მულტიგეგაბიტური სიჩქარით კრისტალში განთავსებული ერთი ან რამდენიმე სალტის საშუალებით.

1.3.1.3. პაკეტების დამუშავება

მიუხედავად იმისა, პროცესორს გააჩნია კონვეიერული თუ პარალელური ორგანიზება, პროცესორში შემოსული ყოველი პაკეტისათვის სრულდება დამუშავების რამდენიმე ეტაპი. ზოგიერთი პროცესორისათვის ეს ეტაპები იყოფა **შემომავალ** (ingress processing) და **გამომავალ** (egress processing) **დამუშავებად**. პირველ ჯგუფს განეკუთვნება ოპერაციები გარედან (საქსელო კავშირის არხიდან ან სისტემური სალტიდან) მოსულ პაკეტებზე, ხოლო მეორე ჯგუფს კი განეკუთვნება გაგზავნის წინ პაკეტებზე შესრულებული ოპერაციები. ამგვარად, ყოველი პაკეტისათვის ჯერ სრულდება შემავალი დამუშავება, და შემდეგ, გამომავალი დამუშავება. ეს დაყოფა პირობითია, რადგანაც ზოგიერთი ოპერაციის შესრულება შესაძლებელია ნებისმიერ ეტაპზე (მაგალითად, ტრაფიკზე მონაცემების შეგროვება).

განვიხილოთ ეს ეტაპები იმ თანმიმდევრობით, როგორც მათი შესრულებაა შესაძლებელი:

1. ***საკონტროლო ჯამის შემოწმება.*** შემომავალი პაკეტი თუ Ethernet ქსელიდან შემოდის, მაშინ ხდება მისი საკონტროლო ჯამის (CRC–კოდის) დათვლა და პაკეტში არსებულ კოდთან შედარება, რის შედეგადაც ვრწმუნდებით მიღებული პაკეტის სისწორეში. საკონტროლო ჯამის ორივე მნიშვნელობა თუ ერთმანეთს ემთხვევა ან Ethernet–პაკეტი CRC ველს არ შეიცავს, მაშინ გამოითვლება IP–პაკეტის საკონტროლო ჯამი და მოხდება მოსული პაკეტის საკონტროლო ჯამთან შედარება. ამის საშუალებით ვრწმუნდებით, რომ გამგზავნის მიერ IP–პაკეტის საკონტროლო ჯამის შექმნის შემდეგ შეცდომას არ ჰქონია ადგილი. ყველა შემოწმების გავლის შემდეგ პაკეტი გადაეცემა შემდგომი დამუშავებისათვის. წინააღმდეგ შემთხვევაში ხდება მისი უბრალოდ გადაგდება.
2. ***ველების მნიშვნელობების ამოღება.*** ანალიზის საფუძველზე ხდება საჭირო სათაურის მდებარეობის განსაზღვრა და პაკეტიდან ხდება სათაურის შესაბამისი გასაღები ველების ამოღება. Ethernet კომუტატორში ხდება მხოლოდ Ethernet სათაურის გამოკვლევა, IP –მარშრუტიზატორში კი გამოიკვლევა მხოლოდ IP–სათაური. გასაღები ველების მნიშვნელობების შენახვა ხდება ან რეგისტრებში (PPE–კონტროლერების პარალელური ორგანიზების შემთხვევაში), ან SRAM მეხსიერებაში (კონვეიერული რეალიზების შემთხვევაში).
3. ***პაკეტების კლასიფიკაცია.*** პაკეტების კლასიფიცირება ხდება მთელი რიგი პროგრამული წესების გათვალისწინებით. უმარტივეს შემთხვევაში ხდება მონაცემების პაკეტებისა და მმართველი პაკეტების განცალკევება.
4. ***გზის არჩევა.*** საქსელო პროცესორების უმეტესობაში არსებობს განსაკუთრებით სწრაფი გზა, რომელიც ოპტიმიზებულია მონაცემების პაკეტების მთელი მრავალფეროვნების გადასაცემად, იმ დროს, როდესაც დანარჩენი პაკეტების დამუშავება თავისებურად ხდება, ჩვეულებრივ, მმართველი პროცესორის მიერ. შესაბამისად, არჩეული უნდა იქნას ან ყველაზე სწრაფი გზა, ან ერთერთი ნელი გზა.
5. ***მიზნობრივი ქსელის არჩევა.*** IP პაკეტები მოიცავს მიმღების 32–თანრიგიან მისამართს. მაგრამ, შეუძლებელია (და არასასურველიც) ყოველი პაკეტის მიმღების განსაზღვრისათვის მთელი ცხრილის გამოკვლევა, რომელიც შეიცავს 2^{32} ჩანაწერს. ამიტომ, როგორც წესი, მისამართის მარცხენა ნაწილი წარმოადგენს ქსელის მისამრთს, ხოლო მარჯვენა ნაწილი კი მიუთითებს აბონენტს ამ ქსელში. ქსელის მისამართის სიგრძე არაა ფიქსირებული, ამიტომ, მისი განსაზღვრა არაა ტრივიალური ამოცანა, თანაც ამას კიდევ ართულებს ისიც, რომ შესაძლებელია იყოს რამდენიმე ვარიანტიც, რომელთაგანაც სწორად ითვლება ყველაზე გრძელი. ამ ნაბიჯზე ხშირად გამოიყენება სპეციალიზებული ინტეგრალური სქემა.
6. ***მარშრუტის შერჩევა.*** SRAM მეხსიერებაში შენახული ცხრილის მიხედვით მიზნობრივი ქსელის მისამართის განსაზღვრის შემდეგ განისაზღვრება თუ კავშირის რომელი არხით მოხდეს პაკეტის გაგზავნა. ამ ნაბიჯზეც შეიძლება გამოყენებული იქნას სპეციალიზებული ინტეგრალური სქემა.
7. ***დაშლა და აწყობა.*** გამოყენებითი პროგრამები ხშირად მაქსმალურად ზრდიან TCP–პაკეტების სასარგებლო დატვირთვას სისტემური გამოძახებების შემცირების მიზნით, მაგრამ TCP, IP და Ethernet ქსელებშიც არსებობს შეზღუდვა პაკეტის მაქსიმალურ ზომაზე. ამ შეზღუდვების

როგორც შედეგი, შეიძლება საჭირო გახდეს გაგზავნის წინ პაკეტების ნაწილებად გაყოფა. ეს ფუნქციები შეიძლება შესრულებული იქნას საქსელო პროცესორის მიერ.

8. ***გამოთვლები.*** ხშირად მონაცემებზე საჭირო ხდება ამა თუ იმ რთული გამოთვლების ჩატარება, მაგალითად კომპრესიის ან დეკომპრესიის ჩატარება, კოდირება და დეკოდირება. ეს მოქმედებები შეიძლება დავალებული ჰქონდეს საქსელო პროცესორს.
9. ***სათაურებით მართვა.*** ზოგჯერ საჭირო ხდება სათაურის დამატება ან მოცილება, ასევე ზოგიერთი ველისათვის მნიშვნელობის შეცვლა. მაგალითად, IP–სათაურში არის შუალედური კვანძების მთვლელი, რომელთა გავლის შემდეგაც პაკეტმა უნდა მოახდინოს თვითგანადგურება. ყოველი შუალედური კვანძის გავლის შემდეგ ხდება ამ მთვლელის დეკრემენტირება. ამ ფუნქციის შესრულებაც თავისუფლად შეუძლია საქსელო პროცესორს.
10. ***რიგების მართვა.*** შემომავალ და გამავალ პაკეტებს ხშირად უწევთ დამუშავების მოლოდინში რიგებში დგომა. მულტიმედიური პაკეტებისათვის საჭიროა, რომ მოლოდინის დრო არ აღემატებოდეს გარკვეულ მნიშვნელობას. გარდა ამისა, ბრანდმაუერს ან მარშრუტიზატორს შეიძლება დასჭირდეს შემავალი ნაკადის რამდენიმე გამავალ ხაზზე გარკვეული წესებით გადანაწილება. ამ ამოცანების შესრულება შესაძლებელია საქსელო პროცესორის საშუალებით.
11. ***საკონტროლო ჯამების შექმნა.*** გამავალ პაკეტებს უნდა გააჩნდეთ საკონტროლო ჯამები. IP–პაკეტების საკონტროლო ჯამები შეიძლება გამოთვლილი იქნას საქსელო პროცესორის მიერ, იმ დროს როდესაც Ethernet–პაკეტების საკონტროლო ჯამების გამოთვლა ხდება აპარატულად.
12. ***აღრიცხვა.*** ზოგიერთ შემთხვევაში პაკეტების გავლის დროს საჭირო ხედება ტრაფიკის დათვლა. ეს განსაკუთრებით საჭიროა მაშინ, თუ ერთერთი კავშირის არხი კომერციული თვალსაზრისით გამოიყენება ტრაფიკის ტრანზიტისათვის. აღრიცხვა შეიძლება აწარმოოს საქსელო პროცესორმა.
13. ***სტატისტიკის შეგროვება.*** მრავალ კომპანიას სჭირდება ტრაფიკის სტატისტიკა. საქსელო პროცესორებში შეიძლება მოგროვებული იყოს ასეთი მონაცემები.

1.3.1.4. მწარმოებლურობის გაზრდა

საქსელო პროცესორების ყველაზე მთავარ მახასიათებელს წარმოადგენს მწარმოებლურობა. რა შეიძლება გაკეთდეს მის ასამაღლებლად? ამ შეკითხვაზე პასუხის გაცემამდე, ჯერ უნდა განისაზღვროს რით განისაზღვრება საქსელო პროცესორის მწარმოებლურობა. მის ერთერთ ზომას წარმოადგენს წამში გადაცემული პაკეტების რაოდენობა. მეორე ზომა შეიძლება იყოს წამში გადაცემული ბაიტების რაოდენობა. შეფასების ეს ორი სისტემა დამყარებულია სხვადასხვა მიდგომაზე. პატარა პაკეტებთან კარგად მომუშავე სქემა შეიძლება მოუქნელი აღმოჩნდეს დიდი პაკეტებისათვის. მაგალითად, პატარა პაკეტების გადაცემის დროს მწარმოებლურობის მნიშვნელოვანი გაზრდა შესაძლებელია დანიშნულების მისამართის განსაზღვრის პროცესის დაჩქარებით. იგივე მექანიზმი დიდი პაკეტების გადაცემის შემთხვევისათვის მნიშვნელოვან დაჩქარებას ვერ უზრუნველყოფს.

მწარმოებლურობის გაზრდის ყველაზე ნაღდ გზას წარმოადგენს საქსელო პროცესორის ტაქტური სიხშირის გაზრდა. მაგრამ, მწარმოებლურობა საქსელო პროცესორის ტაქტური სიხშირის პროპორციული არ არის, რადგანაც გათვალისწინებული უნდა იქნას მეხსიერებასთან მიმართვები

და კიდევ სხვა ფაქტორები. გარდა ამისა, სიხშირის გაზრდასთან დაკავშირებულია გამოყოფილი სითბოს გაზრდაც, რაც პრობლემატურია.

ყველაზე გამართლებულ გამოსავალს წარმოადგენს PPE–კონტროლერების რაოდენობის გაზრდა. ეს მიდგომა განსაკუთრებით ეფექტურია პარალელური არქიტექტურებისათვის. გამოსავალს შეიძლება წარმოადგენდეს აგრეთვე კონვეიერის სიგრძის გაზრდა, მაგრამ მხოლოდ იმ შემთხვევაში, თუ მოხერხდება პაკეტების დამუშავების პროცესის საკმაოდ მრტივ ეტაპებად დაშლა.

მწარმოებლურობის გაზრდის კიდევ ერთი გზა შეიძლება იყოს დამატებითი სპეციალიზებული ინტეგრალური სქემების ან სპეციალიზებული პროცესორების რაოდენობის გაზრდა. მწარმოებლურობის გაზრდა შესაძლებელია აგრეთვე სისტემაში პაკეტების გავლის დროის შემცირებით, რაც შესაძლებელია დამატებითი სალტეების გამოყენებით. მწარმოებლურობის გაზრდა შესაძლებელია კიდევ მეხსიერების მიკროსქემების შეცვლით, მაგალითად, SDRAM ნელი მიკროსქემების ნაცვლად თუ გამოყენებული იქნება უფრო სწრაფი SRAM მიკროსქემები, მაგრამ ეს დაკავშირებულია დიდ მატერიალურ დანახარჯებთან.

1.3.2. მულტიმედია პროცესორები

თანაპროცესორების გამოყენების კიდევ ერთი სფეროა – მაღალი რეზოლუციის ფოტოსურათების დამუშავება, აუდიო– და ვიდეო–ნაკადების დამუშავება. ცენტრალური პროცესორი მაინცდამაინც ეფექტური არ არის ამ ტიპის ინფორმაციის დასამუშავებლად. გარდა ამისა ხელსაყრელიც არაა ამ ტიპის სამუშაოებზე ცენტრალური პროცესორის მოცდენა. ამიტომ, ზოგიერთ თანამედროვე პერსონალურ კომპიუტერში გამოყენებულია სპეციალიზებული თანაპროცესორი მულტიმედიური მონაცემების დასამუშავებლად.

1.3.2.1. მულტიმედია პროცესორი Nexiperia

თანამედროვე კომპიუტერებში მულტიმედიური ინფორმაციის გადამუშავების მნიშვნელობა სულ უფრო და უფრო იზრდება. ამ დანიშნულების თანაპროცესორების კლასს განეკუთვნება კომპანია Philips-ის მიერ დამუშავებული თანაპროცესორი Nexiperia, რომელიც წარმოადგენს დამოუკიდებელ ერთკრისტალიან ჰეტეროგენულ მულტიპროცესორს (ნახ. 3.2). ის შეიცავს რამდენიმე ბირთვს, მათ შორის მმართველ VLIW-პროცესორს TriMedia-ს, და რამდენიმე ბირთვს ვიდეო გამოსახულების, აგრეთვე აუდიო, ვიდეო და საქსელო ოპერაციების დამუშავებისათვის. Nexiperia შეიძლება გამოყენებული იქნას როგორც დამოუკიდებელი ცენტრალური პროცესორი CD-, DVD-, MP3-პლეერებისათვის, ტელემიმღებისათვის, ფოტო და ვიდეო კამერებისათვის და ა.შ. გარდა ამისა, მას შეუძლია შეასრულოს თანაპროცესორის მოვალეობებიც, რომელიც განკუთვნილია გამოსახულებებისა და მულტიმედია ნაკადების დამუშავებისათვის თანამედროვე პერსონალურ კომპიუტერებში. Nexiperia პროცესორი ნებისმიერ კონფიგურაციაში მუშაობს დროის რეალურ მასშტაბში მომუშავე საკუთარი მინიატურული ოპერაციული სისტემის მართვით.

Nexiperia პროცესორის საშუალებით ხდება სამი ამოცანის შესრულება: შემომავალი მონაცემების დაჭერა და მეხიერების შესაბამის სტრუქტურებში გარდაქმნა, ამ სტრუქტურების გადამუშავება და, ბოლოს, მათი გამოტანა მიერთებული პერიფერიული მოწყობილობების

ფორმატებში. მაგალითად, როდესაც პერსონალური კომპიუტერი გამოიყენება, როგორც DVD–პლეერი, Nexiperia პროცესორი შეიძლება დაპროგრამებული იქნას DVD-დისკიდან შეკუმშული ვიდეონაკადის წასაკითხად, მისი დეკოდირებისათვის, დეკომპრესიისათვის და გამოტანისათვის განკუთვნილი ფანჯრის ზომებთან მისადაგებით. მას შემდეგ, რაც DVD–პლეერის პროგრამის ჩატვირთვა მოხდება Nexiperia პროცესორში, ყველაფერი ეს კეთდება ფონურ რეჟიმში, საქმეში ცენტრალური პროცესორის ჩართვის გარეშე.

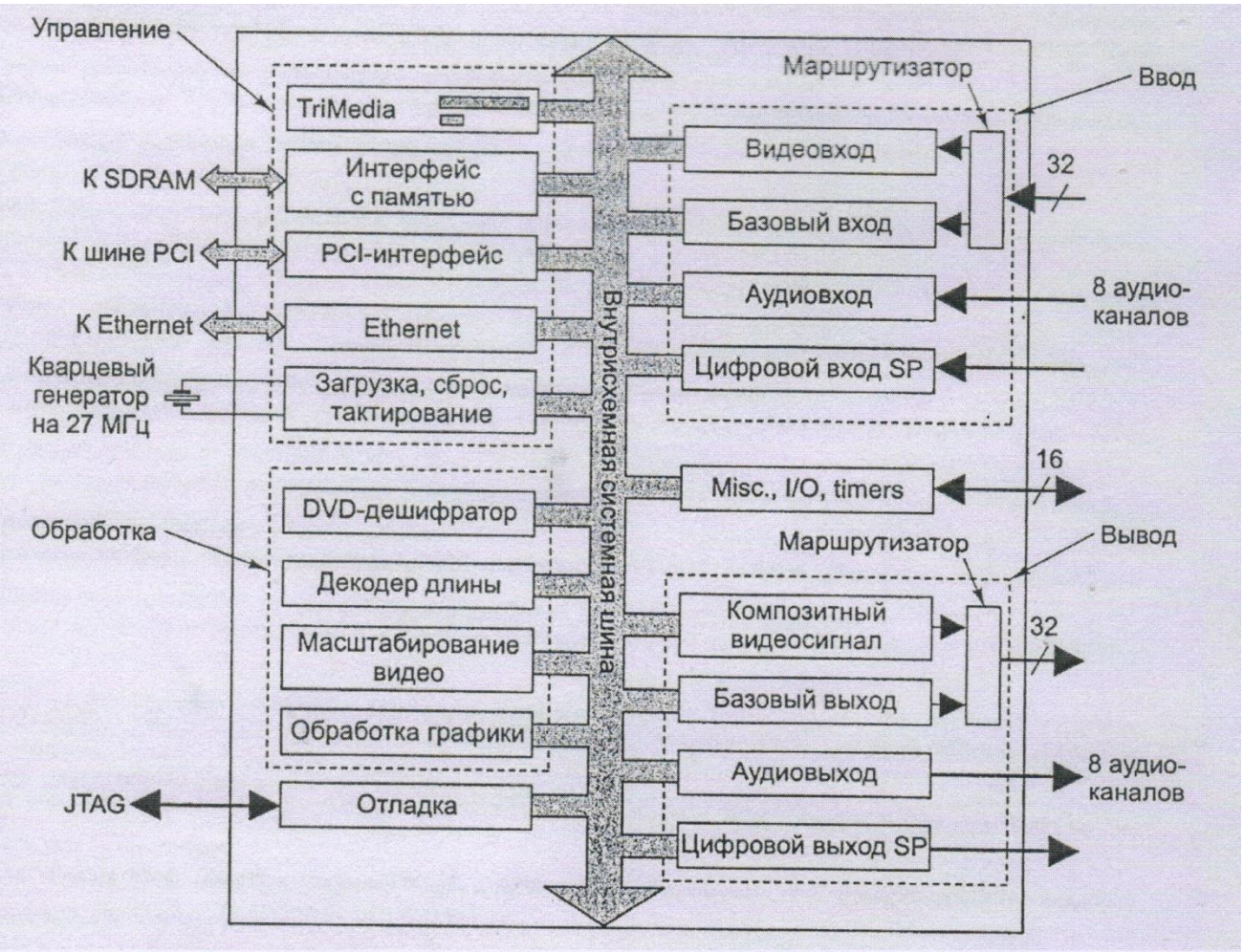
დამუშავებისათვის შემოსული მონაცემების შენახვა ხდება ჯერ მეხსიერებაში, ანუ ინფომაციის წყაროსა და მიმღებებს შორის არ არსებობს უშუალო კავშირები. შემომავალი მონაცემების დაჭერაში, ანუ ოპერატიულ მეხსიერებაში შენახვაში, იგულისხმება ვიდეოების სხვადასხვა ზომებისა და ფორმატების (მათ შორის MPEG-1, MPEG-2 და MPEG4) და აუდიო (მათ შორის AAC, Dolbi და MP3) დეკოდირება, აგრეთვე დეკოდირებული მონაცემების შენახვა შესაბამისი სტრუქტურების სახით შემდგომი დამუშავებისა და გამოტანისათვის. შემომავალი მონაცემების შემოსვლა შესაძლებელია PCI სალტიდან, Ethernet ქსელიდან ან სპეციალური შემოსასვლელიდან, როდესაც მიკროფონი ან სტერეოსისტემა მიერთებულია უშუალოდ მიკროსქემასთან. Nexiperia მიკროსქემას აქვს 456 გამომყვანი, რომელთაგანც ზოგიერთებს შეუძლია მულტიმედიური ნაკადების მიღება ან გადაცემა.

მონაცემების დამუშავების კონტროლი ხდება TriMedia ცენტრალური პროცესორის საშუალებით, რომელიც შეიძლება დაპროგრამებული იქნას ნებისმიერი დავალების გადასაწყვეტად. ტიპიურ ამოცანებს წარმოადგენს: სიმკვეთრის, სიკაშკაშის და ფერთა გადაცემის ხარისხის, გამოსახულების მასშტაბირების, ვიდეოფორმატების გარდაქმნის, ხმაურის ჩახშობის მაჩვენებლების გაუმჯობესება. სამუშაოების შესრულების დროს ცენტრალური პროცესორი ამ დავალებების ძირითად ნაწილს გადასცემს მიკროსქემის სპეციალიზებულ ბირთვებს.

გამოტანის ფუნქციებში შედის მონაცემების სტრუქტურების გარდაქმნა წარმოებული მოწყობილობებისათვის მისაღებ სტრუქტურებში, რომლებშიც ხდება მონაცემების რამდენიმე წყაროს გაერთიანება (ვიდეო, აუდიო, გამოსახულება, ორგანზომილებიანი გრაფიკა) და გამომავალი მოწყობილობების მართვა. მონაცემების გამოტანა, ისევე როგორც შეტანა, შეიძლება განხორციელებული იქნას PCI სალტის, Ethernet ქსელის ან ცალკეული გამომყვანი მოწყობილობების (მაგალითად, დინამიკებთან ან გამაძლიერებელთან მიერთებული) საშუალებით.

Nexiperia ოჯახის PNX 1500 პროცესორის ბლოკ–სქემა ნაჩვენებია ნახ. 1.3.4–ზე. ამ ოჯახის სხვა პროცესორები მცირედ განსხვავდებიან ნახ. 1.3.4–ზე წარმოდგენილი პროცესორისგან, ამიტომ შემდეგში Nexiperia პროცესორები გაიგივებული იქნება PNX 1500 პროცესორთან. PNX 1500 პროცესორს გააჩნია 4 ძირითადი სექცია, რომლებიც უზრუნველყოფენ მართვას, მონაცემების შეტანას, დამუშავებას და გამოტანას. ცენტრალური პროცესორის ფუნქციებს ასრულებს 32–თანრიგიანი VLIW პროცესორი Trimedia, რომელიც 300 MHz სიხშირეზე მუშაობს. პროცესორის ფუნქციონალურობას განაპირობებს დაპროგრამების C ან C++ ენაზე დაწერილი პროგრამები. Nexiperia პროცესორები გაიგივებული იქნება PNX 1500 პროცესორთან. PNX 1500 პროცესორს გააჩნია 4 ძირითადი სექცია, რომლებიც უზრუნველყოფენ მართვას, მონაცემების შეტანას, დამუშავებას და გამოტანას. ცენტრალური პროცესორის ფუნქციებს ასრულებს 32–თანრიგიანი

VLIW პროცესორი Trimedia, რომელიც 300 MHz სიხშირეზე მუშაობს. პროცესორის ფუნქციონალურობას განაპირობებს დაპროგრამების C ან C++ ენაზე დაწერილი პროგრამები.



ნახ. 1.3.4. ჰეტეროგენული ერთკრისტალიანი მულტიპროცესორი Nexiperia

Trimedia ცენტრალური პროცესორის 2 კეშ–მეხსიერებას თუ არ ჩავთვლით, Nexiperia მიკროსქემაში მეხსიერების მოწყობილობები არაა გამოყენებული. ამის მაგივრად გამოყენებულია 8–დან 256 მეგაბაიტამდე მოცულობის მულტიმედია გამოყენებებისათვის მიღებული DDR SDRAM მეხსიერების მიერთებისათვის განკუთვნილი ინტერფეისი. 200 MHz ტაქტური სიხშირისათვის მეხსიერების გამტარუნარიანობა წარმოადგენს 1,6 მეგაბაიტს წამში.

მიკროსქემა მეხსიერებასთან ინტერფეისის გარდა აღჭურვილია სრულადფუნქციონალური 8, 16 და 32 თანრიგიანი PCI–ინტერფეისით 33 MHz სიხშირეზე. ძირითადი პროცესორის ფუნქციების შესრულების დროს PCI ინტერფეისის ელექტრონიკას შეუძლია სალტის არბიტრის ფუნქციების შესარულება. ეს ინტერფეისი შეიძლება გამოყენებული იქნას მაგალითად, DVD დისკთან ურთიერთობისთვისაც.

Ethernet ქსელთან უშუალო მიერთების რეალიზება ხდება სპეციალურად გამოყოფილი ბირთვის საშუალებით, რომლის საშუალებითაც ხდება მიერთება 10 და 100 მეგაბიტი წამში ნსიჩქარით. შესაბამისად, Nexiperia პროცესორის ბაზაზე აგებულ ვიდეოკამერას შეუძლია

პირდაპირ მოახდინოს Ethernet ქსელის საშუალებით ციფრული ვიდეონაკადის პირდაპირი ტრანსლირება ვიდეო მონაცემების შეტანის ან გამოტანის დაშორებულ მოწყობილობებზე.

არსებული ბირთვი პასუხს აგებს ჩატვირთვის, ჩამოყრის, ტაქტირების და რამდენიმე სხვა ფუნქციაზე. რომელიმე განსაზღვრულ გამომყვანს თუ მიეწოდება შემომავალი სიგნალი, მაშინ ხდება პროცესორის ჩამოყრა. ბირთვი შეიძლება დაპროგრამებული იქნას, როგორც საონტროლო ტაიმერი, რომლის საშუალებითაც ხდება ცენტრალური პროცესორის გადატვირთვა, თუ განსაზღვრულ დროში არ ხდება მისი გამორთვა.

შემდეგი მოწყობილობა პასუხს აგებს ჩატვირთვის, ჩამოყრის, ტაქტირების და რამდენიმე სხვა ფუნქციაზე. გარკვეული გამომყვანიდან თუ შემოდის სიგნალი, ხდება პროცესორის ჩამოყრა. ბირთვი შეიძლება დაპროგრამებული იქნას, როგორც საკონტროლო ტაიმერი, რომელიც ახდენს ცენტრალური პროცესორის გადატვირთვას, თუკი ის წინასწარ დადგენილი დროის განმავლობაში არ გამოირთო. ავტონომიურ მოწყობილობებში გადატვირთვა შეიძლება მოხდეს ფლეშ–მეხსიერებიდანაც.

ბირთვის მართვა ხდება 27 MHz სიხშირის კვარცული გენერატორიდან, რომლის სიხშირის გამრავლება ხდება 64–ზე და საბოლოო ჯამში მიიღება 1,728 GHz სიხშირის სიგნალი, რომელიც პროცესორში ყველგან გამოიყენება. როგორც წესი, ცენტრალური პროცესორი მუშაობს სრული სიჩქარით, ხოლო დანარჩენი კომპონენტები მუშაობენ იმ სიჩქარით, რომელიც მისაღებია მათი დავალებების შესასრულებლად. ელექტროენერგიის დაზოგვის მიზნით გათვალისწინებულია აგრეთვე ტაქტური სიხშირის შემცირება. გარდა ამისა, გათვალისწინებულია აგრეთვე ძილის რეჟიმიც, რომლის დროსაც ხდება მოწყობილობების უმრავლესობის გამორთვა. ეს განსაკუთრებით მნიშვნელოვანია მობილური მოწყობილობებისთვის, რომელთათვისაც ხდება აკუმულატორების ეფექტური გამოყენება.

ამ ბირთვში გამოყენებულია აგრეთვე 16 სემაფორი, რომლებიც გამოიყენება მოწყობილობების სინქრონიზაციისათვის. სემაფორს თუ გააჩნია ნულოვანი მნიშვნელობა და ბირთვი მიანიჭებს არანულოვან მნიშვნელობას, მაშინ ჩანაწერი ხორციელდება, წინააღმდეგ შემთხვევაში ჩანაწერი არ ხორციელდება და სემაფორის მნიშვნელობა რჩება უცვლელი. ნულის ჩაწერა ყოველთვის წარმატებით სრულდება. ჩაწერის ოპერაცია არის ატომარული, რადგანაც დროის კონკრეტულ მომენტში მხოლოდ ერთ ბირთვს შეუძლია სალტის დაკავება. სემაფორების გამოყენება გამორიცხავს სალტის ერთდროულად რამდენიმე მოწყობილობის მიერ დაკავებას. იმისათვის, რომ რესურსთან მიღწეული იქნას წვდომა, ბირთვი ცდილობს რომელიმე სემაფორში არანულოვანი მნიშვნელობის ჩაწერას. ჩაწერა თუ განხორციელდა, მაშინ ბირთვი ახორციელებს სალტესთან წვდომას, ანუ შესაბამისად საჭირო მოწყობილობასთან წვდომას. სალტის განთავისუფლებისათვის სემაფორში იწერება 0. საჭირო მოწყობილობის დაჭერა თუ არ განხორციელდა, მაშინ ბირთვი იმეორებს მცდელობებს.

განვიხილოთ ბირთვის სექციები. ვიდეოშესასვლელის ბირთვი ღებულობს 10–თანრიგიან ციფრულ ვიდეონაკადს, გარდაქმნის მას 8–თანრიგიან ვიდეონაკადად და ჩაწერს მას გარე SDRAM მეხსიერებაში. ამ დროს სალტის სიხშირე წარმოადგენს 100 MHz–ს. გარდა ამისა, ბირთვს შეუძლია დაიჭიროს სტრუქტურირებული მონაცემები ჭდეებით, რომელთა საშუალებითაც მონიშნულია ჩაწერის საზღვრები. ორი ვიდეო შესასვლელიდან მოსული ვიდეოსიგნალების განცალკევება

ხდება მარშრუტიზატორის საშუალებით. განცალკევება საჭიროა იმისათვის, რომ ერთიდაიგივე გარე კონტაქტები გამოიყენება როგორც ვიდეო, ასევე ბაზური შესასვლელებისათვისაც.

აუდიოშესასვლელის ბირთვს შეუძლია ერთდროულად დაიჭიროს ხმის ან სტერეოფონუ-ლი მუსიკის 8–მდე 8, 16 ან 32 თანრიგიანი არხი 96 KHz–მდე სიხშირეზე და შეინახოს SDRAM მეხსიერებაში. გარდა ამისა, ბირთვს მონაცემების შენახვამდე შეუძლია განახორციელოს შეკუმშუ-ლი ფორმატების დეკომპრესია, არხების შერევა, დისკრეტიზაციის სიხშირის შეცვლა და ფილტრე-ბის გამოყენება.

ციფრული SP შესასვლის ბირთვს შეუძლია მიიღოს ციფრული აუდიოსიგნალები Sony-Philips (IEC 1937) სტანდარტით. მოწყობილობებს შორის ციფრული აუდიოსიგნალების გადაცემა ხდება ხარისხის შემცირების გარეშე.

აუდიო, ვიდეო ან სხვა სახის მონაცემების მიღების შემდეგ საჭიროა მათი გადამუშავება, რაც ხორციელდება პროცესორის სექციის საშუალებით. კოპირებისაგან დაცვის მიზნით ხდება DVD ფაილების შიფრაცია. DVD–დეშიფრატორი ასრულებს დეშიფრირებას, ვიდეო მონაცემების MPEG–2 ფორმატში მისაღებად. დეშიფრირება არის მეხსიერება–მეხსიერება ტიპის ოპერაცია.

სიგრძის დეკოდერი აგრძელებს წინა ბირთვის მიერ დაწყებული დეკოდირების ოპერა-ციას, სირთულე დაკავშირებულია სიტყვის ცვლად სიგრძესთან, რაც დამახასიათებელია MPEG-2 ფორმატისათვის (ასევე MPEG-1 ფორმატისათვისაც). ამ ოპერაციის შემდეგ შეკუმშული მონაცე-მები მიიღება ნახევრად გაშლილ მდგომარეობაში და გადაეცემა MPEG ფორმატის მონაცემების დამუშავების ბლოკს (TriMedia პროცესორში ეს რელიზებულია პროგრამულად), სადაც ხდება დეკოდირების ოპერაციის დასრულება. ასეთი გაყოფის მიზეზი მდგომარეობს იმაში, რომ ცვლადი სიგრძის მქონე სიტყვების დეკოდირების პროცესი (დაფუძნებულია ჰაფმანისა და ჯგუფურ დეკოდირებაზე) ეფექტურად ვერ იყენებს TriMedia პროცესორს, ამიტომ გამართლებულად ჩათვალეს მიკროპროცესორის კრისტალისათვის დაემატებინათ კიდევ რამდენიმე კვადრატული მილიმეტრი და მოეხდინათ ამ ალგორითმის აპარატული რელიზება. ამ ოპერაციების შესრულების შედეგად მეხსიერებაში მიიღება სრული პიქსელური რუქა.

პიქსელური რუქა შეიძლება წარმოდგენილი იქნას სამიდან ერთერთ ფორმატში. პირველი ფორმატის სახელია **ინდექსირებული ფერები**, რომელშიც გამოიყენება **ფერების კოდური ცხრილი** (Color Look Up Table, **CLUT**). ამ ცხრილში ინახება ფერების 24–თანრიგიანი კოდები, რომელთაც დამატებული აქვს 8–თანრიგიანი ალფა არხის ნიღაბი. RGB ფორმატში, რომელშიც მონიტორი მუშაობს, ცალცალკე ხდება წითელი, მწვანე და ლურჯი ფერების ინტენსივობების მითითება. YUV ფორმატში კი ხდება სატელევიზიო გამოსახულების კოდირება. ამ ფორმატში, იმის მაგივრად, რომ ცალცალკე ხდებოდეს წითელი, მწვანე და ლურჯი ფერების კოდირება, ვიდეო კამერაშივე ხდება გარდაქმნა, რომლის შედეგადაც მიიღება სიკაშკაშის ერთი და ფერების ორი არხი. სიკაშკაშის არხს სისტემა გამოუყოფს გატარების უფრო ფართო ზოლს, რაც უზრუნველყოფს გადაცემის პროცესში შეფერხებებისადმი უფრო მაღალ მდგრადობას. იმის წყალობით, რომ გამოსახულების წარმოდგენა ხდება მხოლოდ რამდენიმე ფორმატში, პროცესორის ყოველ ბირთვს „ესმის“ თუ რა ხდება სხვა ბირთვებში.

ვიდეო მონაცემების მასშტაბირების ბირთვი წამში 120 მილიონი პიქსელის სისწრაფით იღებს და ასრულებს მასშტაბირების დავალებებს, რომელთა შორის შეიძლება აღინიშნოს:

- „სავარცხლის“ ეფექტის მოცილება;
- ჰორიზონტალური და ვერტიკალური მასშტაბირება;
- გარდაქმნები სხვადასხვა ფერის ფორმატებს შორის;
- სიკაშკაშის ჰისტოგრამის აგება;
- ციმციმის მოცილება.

სატელევიზიო გამოსახულებაში ადგილი აქვს „სავარცხლის“ ეფექტს, რომელიც წარმოიქმნება ანალოგური სატევიზიო სიგნალის ციფრულ ფორმატში გადაყვანის შედეგად, როდესაც ყოველი კადრისათვის, რომელიც შეიცავს 525 სტრიქონს (PAL და SECAM ფორმატები კი 625 სტრიქონს), ჯერ გაიცემა კენტ ნომრიანი სტრიქონები და შემდეგ კი ლუწ ნომრიანი სტრიქონები. „სავარცხლის“ ეფექტის მოცილების შემდეგ მიიღება უფრო ხარისხიანი გამოსახულება**პროგრესიული გაშლით** (progressive scan), როდესაც სტრიქონების გადაცემა ან დამუშავება ხდება მათი ჩვეულებრივი თანმიმდევრობით. კადრების განახლების სიხშირე NTSC სისტემაში არის 29,97 კადრი წამში, ხოლო Pal და SECAM სისტემებში კი 25 კადრი წამში. ეს სიხშირე არის ორჯერ მეტი **სტრიქონის გამოტოვებით გაშლასთან** (interlaced scan) შედარებით. ჰორიზონტალური და ვერტიკალური მასშტაბირების შედეგად შეიძლება გამოსახულების ზომის გაზრდა ან შემცირება. სტანდარტულ ტელევიზიაში კადრის ზომის მახასიათებლად გამოიყენება შეფარდება სიგანე/სიმაღლე. ამ მაჩვენებლის ნორმალური მნიშვნელობა არის 4/3, ხოლო ფართეეკრანიანი გამოსახულებისათვის კი 16/9, რომელიც უკეთ მიესადაგება ტრადიციულ 35–მილიმეტრიან კინოფირს. მასშტაბირების ბირთვს შეუძლია კადრის ზომების შეცვლა წრფივი ან არაწრფივი ალგორითმით. გარდა ამისა, მას შეუძლია კიდევ ფერების წარმოდგენის ფორმატების (ინდექსირებული ფერები, RGB, YUV) ურთიერთგადაყვანა და სიკაშკაშის ჰისტოგრამის აგება, რომელიც საჭიროა გამოსასვლელზე გამოსახულების ხარისხის გასაუმჯობესებლად. და ბოლოს, შეიძლება შესრულებული იქნას გარკვეული გარდაქმნები ციმციმის მოსაცილებლად.

გრაფიკის დამუშავების ბირთვი ობიექტების აღწერილობების მიხედვით აგებს ორგანზომილებიან გამოსახულებებს. გარდა ამისა, მას შეუძლია შეკრული კონტურების ფერებით შევსება და **მონაცემების ბლოკური გადაცემის** (bitblt) გრაფიკული ოპერაციების შესრულება, რომელთა შესრულების შემდეგაც შესაძლებელი ხდება პიქსელური რუქების გაერთიანება AND, OR, XOR და ბულის სხვა ლოგიკური ფუნქციების საშუალებით.

აუდიო მონაცემების დამუშავებისათვის ცალკე ბირთვები არ გამოიყენება. მათი მოცულობა იმდენად მცირეა, რომ შესაძლებელია მათი პროგრამული დამუშავება, რაც ხდება კიდევაც TriMedia ცენტრალურ პროცესორში.

გამართვის ბლოკი ეხმარება დამპროექტებლებს პროცესორის აპარატული და პროგრამული საშუალებების გამართვაში. ეს ბირთვი წარმოადგენს ინტერფეისს **JTAG** (Joint Test Action Group – ტელევიზიის ავტომატიზების გაერთიანებული მუშა ჯგუფი) IEEE 1149.1 სტანდარტის შესაბამისად.

გამოტანის სექცია უზრუნველყოფს დამუშავებული მონაცემების მეხსიერებიდან წაკითხვას და მათ გამოტანას. კომპოზიტური ვიდეო სიგნალის ბირთვი მონაცემების გამოტანის წინ ახდენს ახდენს ერთი ან რამდენიმე პიქსელური სტრუქტურის მონაცემების ნორმალიზებას და შერევას. ხდება ინდექსირებული პიქსელების „დეინდექსირება“ ცალკეულ პიქსელებში, ხოლო

შეუთავსებელი ფორმატებისათვის კომპოზიტური ვიდეოსიგნალის ბირთვი ასრულებს წინასწარ გარდაქმნას. გარდა ამისა, ეს ბირთვი, საჭიროების შემთხვევაში, ასრულებს სიკაშკაშისა და კონტრასტულობის კორექციას. ბირთვის მუშაობის საბოლოო შედეგია NTSC, PAL ან SECAM ფორმატში მიღებული ვიდეო გამოსახულება.

მომავალში Nexiperia პროცესორის ბაზაზე შექმნილი სისტემები ავტომატურად გაარჩევენ სატელევიზიო ფორმატების სამივე სახეობას. ეს ცვლილება დიდ სირთულეს არ წარმოადგენს, მაგრამ გააჩნია დიდი კომერციული პერსპექტივა. ანალოგიურად, **მაღალი სიმკვეთრის ტელევიზიის** (High Definition Television, **HDTV**) მხარდაჭერა მოითხოვს მხოლოდ პროგრამული კოდის გართულებას, რომელმაც უნდა მოახდინოს ვიდეო მონაცემების მეხსიერების სტრუქტურაში გადაყვანა და პირიქით.

ბაზური გამოსასვლელის ბირთვი ასრულებს მხოლოდ 8, 16 და 32 თანრიგიანი მონაცემების გამოტანას ციკლში 100 MHz სიხშირით, რაც საბოლოო ჯამში იძლევა გამტარუნარიანობას 3,2 Gბიტი წამში. ერთი Nexiperia პროცესორის გამოსასვლელს თუ მივუერთებთ მეორე პროცესორის შესასვლალს, შესაძლებელი გახდება მათ შორის ფაილების გაცვლა უფრო მაღალი სიჩქარით, ვიდრე ეს შესაძლებელია Gigabit Ethernet ქსელში.

გამოსასვლელი მარშრუტიზატორი განსაზღვრავს თუ ორი ბირთვიდან რომლის სიგნალები უნდა მიეწოდოს მიკროსქემის გამოსასვლელ კონტაქტებზე. გრდა ამისა, მას შეუძლია შეასრულოს კიდევ რამდენიმე დამატებითი მოქმედება.

აუდიო გამოსასვლელის ბირთვს შეუძლია მოახდინოს 8–მდე სტერეოარხის გენერირება 32 ბიტი სიზუსტით და დისკრეტიზაციის სიხშირით 96 KHz. ციფრული SP გამოსასვლელი შეიძლება მიერთებული იქნას მოწყობილობებთან, რომელთაც გააჩნიათ Sony-Philips ციფრული ხმის სტანდარტის მხარდაჭერა.

პროცესორის უკანასკნელი ბირთვი უზრუნველყოფს საერთო დანიშნულების სიგნალების შეტანასა და გამოტანას. ამ ბირთვის 16 კონტაქტი შეიძლება ნებისმიერი დანიშნულებით იქნას გამოყენებული, მაგალითად ღილაკებთან, კლავიშებთან, გადამრთველებთან, შუქდიოდებთან მისაერთებლად მათი პროგრამულად სამართავად. ამ ბირთვში არის კიდევ სხვადასხვა ტაიმერიმთვლელი და სხვა.

დასკვნის სახით, Nexiperia პროცესორის შესახებ შეიძლება ითქვას, რომ მას გააჩნია საკმაოდ დიდი გამოთვლითი სიმძლავრეები აუდიო და ვიზუალური მონაცემებისათვის. ამ თანაპროცესორის გამოთვლითი სიმძლავრეები სინამდვილეში უფრო მეტია, ვიდრე ერთი შეხედვით შეიძლება მოგვეჩვენოს. მის ყველა ბირთვს შეუძლია ცენტრალურ პროცესორთან ერთდოულად მუშაობა .

1.3.3. კრიპტოპროცესორები

უსაფრთხოება და განსაკუთრებით საქსელო უსაფრთხოება არის კიდევ ერთი სფერო, სადაც ფართოდ გამოიყენება თანაპროცესორები. როდესაც კლიენტსა და სერვერს შორის მყარდება კავშირი, საჭირო ხდება მათი ურთიერთაუტენტიფიკაცია. ასეთი გზით დამყარებული კავშირის შემდეგ, მითუმეტეს თუ მონაცემების კოდირებაც გამოიყენება, შეიძლება მშვიდად იქნას განხორ-

ციელებული მონაცემების გადაცემა ისე, რომ არ ვიფიქროთ კავშირის არხების მოსმენაზე და ჰაკერებზე.

აქ პრობლემა იმაში მდგომარეობს, რომ უსაფრთხოების უზრუნველყოფა ხდება კრიპტოგრაფიული მეთოდებით, რომლებიც მოითხოვენ ძალიან დიდი რაოდენობით გამოთვლების ჩატარებას. კრიპტოგრაფიაში მონაცემების დაცვის მიართ დღეს გავრცელებულია ორი მიდგომა: **დაშიფვრა სიმეტრიული გასაღებით** და **დაშიფვრა ღია გასაღებით.** პირველი დაფუძნებულია ბიტების საგულდაგულოდ არევაზე (თითქოს მონაცემების მოთავსება ხდება სპეციალურ მიქსერში). მეორე მეთოდს საფუძვლად უდევს დიდი რიცხვების (1024–თანრიგიანი რიცხვების) გამრავლება და ხარისხში აყვანა, რაც დაკავშირებულია დიდ დროით დანახარჯებთან.

სხვადასხვა კომპანიებს გამოშვებული აქვთ კრიპტოგრაფიული თანაპროცესორები, რომელთა საშუალებითაც მონაცემთა უსაფრთხო გაცემის მიზნით ხდება გადაცემის წინ მონაცემების დაშიფვრა, ხოლო მიღებისას კი მონაცემების დეშიფრაცია. ძირითადად, ესაა PCI გასართში ჩასაყენებელი მცირე ზომის პლატები. სპეციალური აპარატული საშუალებების წყალობით, ამ პროცესორებს შეუძლიათ საჭირო კრიპტოგრაფიული გამოთვლები ჩაატარონ გაცილებით უფრო სწრაფად, ვიდრე ცენტრალურ პროცესორს და ეს გამოთვლები არ გამოიწვევს ცენტრალური პროცესორის ზედმეტად დაკავებას.

2. მულტიპროცესორები

აქამდე განიხილებოდა ერთპროცესორიან სისტემაში პარალელიზმის შეტანის საკითხები. შემდეგი ნაბიჯი პარალელიზმის სამყაროში იქნება რამდენიმე სრულფასოვანი პროცესორის ერთიან სისტემაში გაერთიანება. რამდენიმე ცენტრალური პროცესორის შემცველი სისტემები შეიძლება დაიყოს მულტიპროცესორებად და მულტიკომპიუტერებად.

2.1. მულტიპროცესორები და მულტიკომპიუტერები

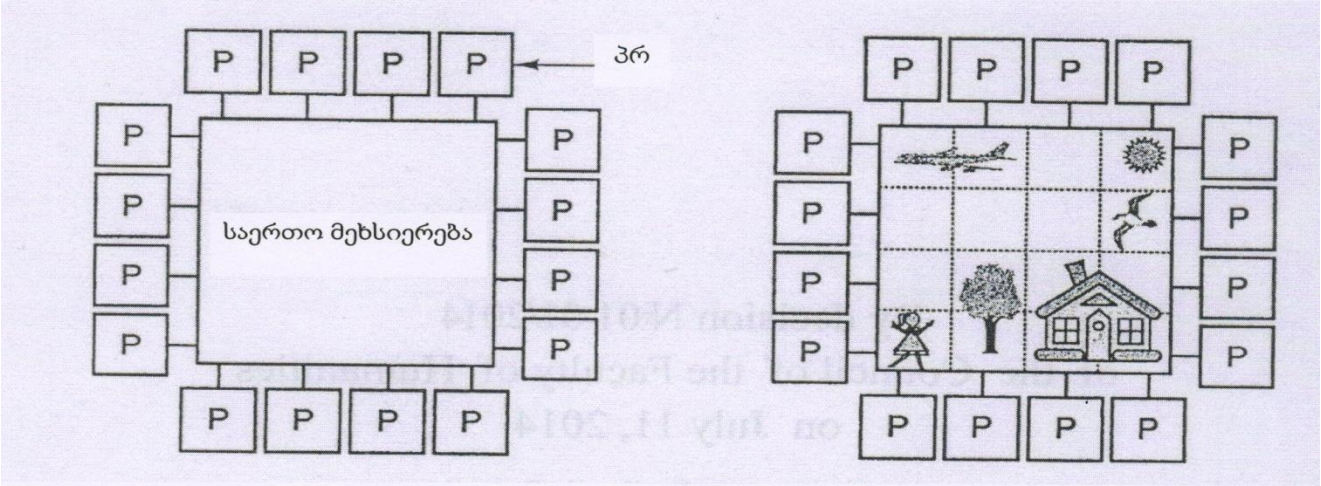
პარალელური კომპიუტერების ნებისმიერ სისტემაში ერთიანი სამუშაოს შესრულებით დაკავებულ პროცესორებს ერთმანეთთან უნდა ჰქონდეს კავშირი ინფორმაციის გასაცვლელად. როგორ უნდა ხორციელდებოდეს მონაცემების ეს გაცვლა? ამ მიზნით შემუშავებული იქნა ორი სტრატეგია: მულტიპროცესორები და მულრიკომპიუტერები. ამ ორ სტრატეგიას შორის ძირითადი განსხვავება გამოიხატება იმაში, არსებობს თუ არა საერთო მეხსიერება. ეს განსხვავება აისახება როგორც კომპიუტერების კონსტრუქციაზე, მოწყობილობებზე და დაპროგრამების სისტემაზე, ასევე მათ ზომებსა და ფასზე.

2.1.1. მულტიპროცესორები

პარალელურ კომპიუტერს, რომელშიც პროცესორები ერთობლივად გამოიყენებენ საერთო ოპერატიულ მეხსიერებას, უწოდებენ მულტიპროცესორს, ანუ სისტემას საერთო ოპერატიული

მეხსიერებით (ნახ. 2.1.1.ა). მულტიპროცესორულ სისტემაში მომუშავე ყველა პროცესორს გააჩნია ვირტუალური მეხსიერების ერთი საერთო არე. ნებისმიერ პროცესს LOAD და STORE ბრძანებების საშუალებით შეუძლიათ საერთო ოპერატიულ მეხსიერებაში სიტყვის ჩაწერა ან წაკითხვა. ამის

საშუალებით, მულტიპროცესორულ სისტემაში ორ პროცესს ადვილად შეუძლია მონაცემების გაცვლა. ამისათვის საჭიროა ერთმა პროცესმა ოპერატიულ მეხსიერებაში ჩაწეროს მონაცემები, ხოლო მეორე პროცესმა კი იქიდან წაიკითხოს ეს მონაცემები.



ნახ. 2.1.1. 16 პროცესორის შემცველი მულტიპროცესორული სისტემა საერთო ოპერატიული მეხსიერებით (ა); საერთო ოპერატიული მეხსიერება დაყოფილია 16 სექციად, რომელთაგან თითოეულს აკონტროლებს ცალკეული პროცესორი (ბ)

ორ ან მეტ პროცესს შორის მონაცემების გაცვლის სისტემის სიმარტივის გამო მულტიპროცესორები ძალიან პოპულარულია. მეხსიერების აგების ეს მოდელი პროგრამისტებისთვის მისაღებია და მისი საშუალებით შესაძლებელია გადაწყვეტილი იქნას ამოცანების ფართე სპექტრი. მაგალითისათვის განვიხილოთ პროგრამა, რომელიც ახდენს ბიტური ასახვის ანალიზს და ადგენს ყველა ობიექტის სიას. გამოსახულების ერთი ასლი ინახება მეხსიერებაში, როგორც ეს ნაჩვენებია ნახ. 2.1.1ბ–ზე. პროცესორებიდან თითოეული უშვებს ერთ პროცესს, რომელიც ახდენს სექციებიდან ერთერთის ანალიზს. პროცესი თუ აღმოაჩენს, რომ მისი ობიექტი გასცდა სექციის საზღვარს, მაშინ ეს პროცესიც გადავა შემდეგ სექციაში და გააგრძელებს მუშაობას. მოყვანილ მაგალითში ზოგიერთი ობიექტის დამუშავებას ახდენს რამდენიმე პროცესი, ამიტომ ბოლოს საჭირო გახდება გარკვეული კოორდინაცია, რათა გაირკვეს სახლების, ხეების და თვითმფრინავების სწორი რაოდენობა.

მულტიპროცესორის ყოველი პროცესორი, რადგანაც გამოიყენებს მეხსიერების ერთ სამისამართო სივრცეს, ამიტომ ფუნქციონირებს ოპერაციული სისტემის მხოლოდ ერთი ასლი. შესაბამისად, არსებობს მეხსიერების ფურცლების მხოლოდ ერთი რუქა და პროცესორების ერთი ცხრილი. როდესაც ხდება პროცესის ბლოკირება, მისი პროცესორი ინახავს თავის მდგომარეობას ოპერაციული სისტემის ცხრილებში, შემდეგ ათვალიერებს ამ ცხრილებს და იწყებს სხვა პროცესის ძებნას, რომლის გაშვებაც უნდა მოხდეს.

მულტიპროცესორი, ისევე როგორც ნებისმიერი კომპიუტერი, უნდა შეიცავდეს შეტანა–გამოტანის (დისკები, საქსელო ადაპტერები და სხვა) მოწყობილობებს. ზოგიერთ მულტიპროცესორულ სისტემაში მხოლოდ ცალკეულ პროცესორებს შეუძლიათ განახორციელონ წვდომა შეტანა–გამოტანის მოწყობილობებთან და შესაბამისად გააჩნიათ შეტანა–გამოტანის სპეციალური საშუალებები. სხვა მულტიპროცესორულ სისტემებში ყოველ პროცესორს შეუძლია განახორციელოს წვდომა შეტანა–გამოტანის ნებისმიერ მოწყობილობასთან. თუ ყველა პროცესორს გააჩნია

მეხსიერების მოდულებთან და შეტანა–გამოტანის მოწყობილობებთან წვდომის თანაბარი უფლებები, და პროცესორებს შეუძლიათ ურთიერთშენაცვლება, მაშინ ასეთ მულტიპროცესორს უწოდებენ **სიმეტრიულს** (Symmetric MultiProcessor, **SMP**).

2.1.2. მულტიკომპიუტერები

პარალელურ არქიტექტურის მეორე ვარიანტში ყოველ პროცესორს გააჩნია საკუთარი მეხსიერება, რომელთანაც წვდომა შეუძლია მხოლოდ მას. ასეთ სქემას უწოდებენ **მულტიკომპიუტერს,** ანუ **სისტემას განაწილებული მეხსიერებით** (ნახ. 2.1.2,ა). მულტიპროცესორსა და მულტიკომპიუტერს შორის არსებითი განსხვავება მდგომარეობსიმაში, რომ მულტიკომპიუტერში ყოველ პროცესორს გააჩნია საკუთარი ლოკალური მეხსიერება, რომელთან LOAD და STORE ბრძანებებით მიმართვა შეუძლია მხოლოდ ამ პროცესორს. ამგვარად, მულტიპროცესორს გააჩნია ერთი ფიზიკური სამისამართო სივრცე, რომლის განაწილებაც ხდება ყველა პროცესორს შორის, ხოლო მულტიკომპიუტერი შეიცავს ცალკეულ სამისამართო სივრცეებს თითოეულო პროცესორისათვის.

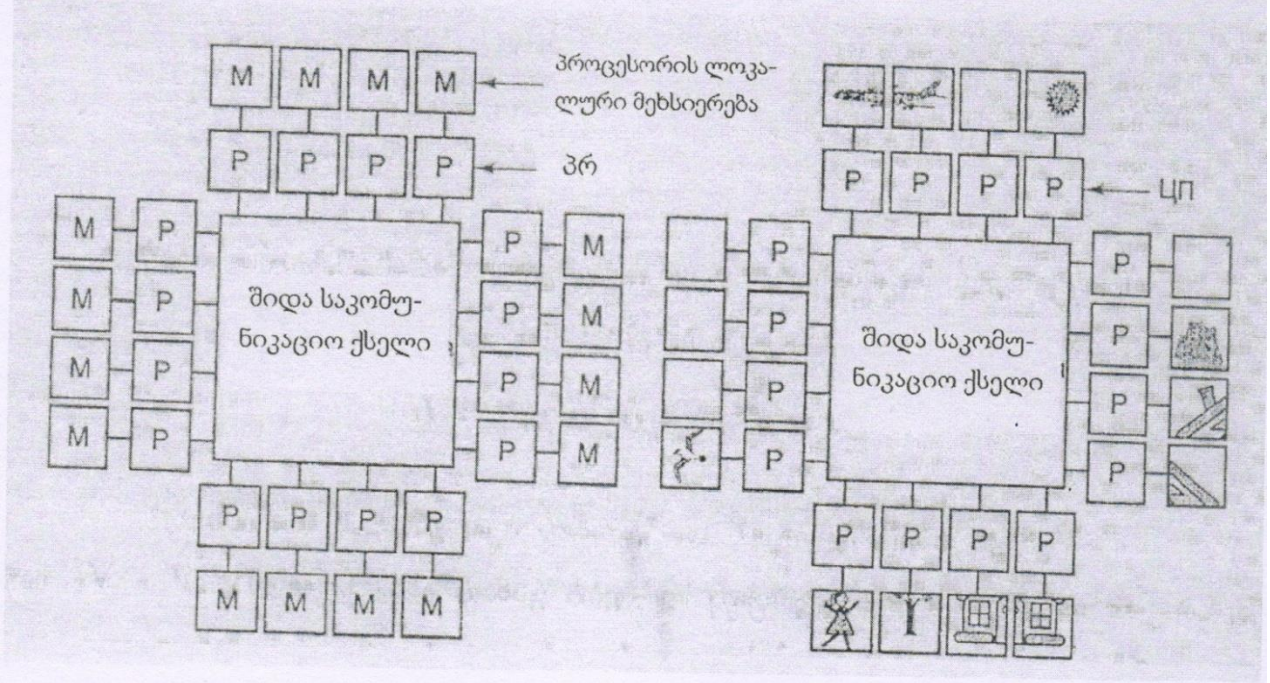
მულტიკომპიუტერებში პროცესორებს არ შეუძლიათ მონაცემების გაცვლა საერთო მეხსიერებასთან მიმართვებით. მათ შორის მონაცემების გაცვლა უნდა მოხდეს მათი დამაკავშირებელი საკომუნიკაციო ქსელის საშუალებით. მულტიკომპიუტერების მაგალითებად შეიძლება დასახელებული იქნას IBM BlueGene/L, Red Storm და Google–ს კლასტერი.

მულტიკომპიუტერებში არ არსებობს აპარატულად რეალიზებული საერთო მეხსიერება, მისი ჩანაცვლება ხდება გარკვეული პროგრამული სტრუქტურის საშუალებით. მულტიკომპიუტერებში შეუძლებელია პროცესორებისათვის ერთიანი ვირტუალური სამისამართო სივრცის არსებობა, რომელშიც LOAD და STORE ბრძანებებით შესაძლებელი იქნებოდა მონაცემების ჩაწერა და წაკითხვა. მაგალითად, ნახ 2.1.1,ბ–ზე გამოსახულ სტრუქტურაში ზედა მარცხენა პროცესორი (ამ პროცესორს მივანიჭოთ ნომერი 0) თუ აღმოაჩენს, რომ მისი ობიექტის ნაწილი მოხვედრილია სხვა სექციაში, რომელიც ეკუთვნის შემდეგ პროცესორს (ეს იყოს პროცესორი 1), მას შეუძლია თვითმფრინავის კუდის გამოსახულების წასაკითხად გააგრძელოს საერთო მეხსიერებიდან მონაცემების წაკითხვა. საქმე თუ გვექნება მულტიკომპიუტერთან (ნახ. 2.1.2,ბ), მაშინ პროცესორი 0 ვერ შეძლებს ობიექტის შესახებ მონაცემების წაკითხვას პაროცესორი 1–ის მეხსიერებიდან. ამ შემთხვევაში მონაცემების მიღების ალგორითმი სხვა უნდა იყოს.

ჯერ 0 პროცესორმა როგორღაც უნდა დაადგინოს რომელ პროცესორში იმყოფება მისთვის საჭირო მონაცენები და შემდეგ გაუგზავნოს ამ პრცესორს შეტყობინება, რომლითაც მოითხოვს მისგან საჭირო მონაცემების ასლს. პროცესორი 0 იბლოკება პასუხის მიღებამდე. შეტყობინების მიღების შემდეგ პროცესორი 1 პროგრამულად აანალიზებს მას და მოთხოვნილ მონაცემებს უგზავნის პროცესორ 0–ს. პროცესორ 0–ში შეტყობინების მოსვლის შემდეგ, მისი ბლოკირება პროგრამულად იხსნება და პროცესორი 0 აგრძელებს მუშაობას.

მულტიკომპიუტერებში კომპიუტერების თანამოქმედებისათვის ხშირად გამოიყენება პრიმიტივები send და receive. ამიტომ, მულტიკომპიუტერის პროგრამულ უზრუნველყოფას აქვს უფრო რთული სტრუქტურა მულტიპროცესორებთან შედარებით. მულტიკომპიუტერებში მნიშვნელოვან პრობლემას წარმოადგენს მონაცემების სწორი განაწილება და კომპიუტერებში განთავ-

სება. მულტიპროცესორებში მონაცემების განაწილების და განთავსების ამოცანა საერთოდ არისმება, ყველა პროცესორს მონაცემებთან წვდომის თანაბარი უფლებები აქვს. როგორც ვხედავთ, მულტიკომპიუტერების დაპროგრამება უფრო რთული საქმეა, ვიდრე მულტიპროცესორების დაპროგრამება.



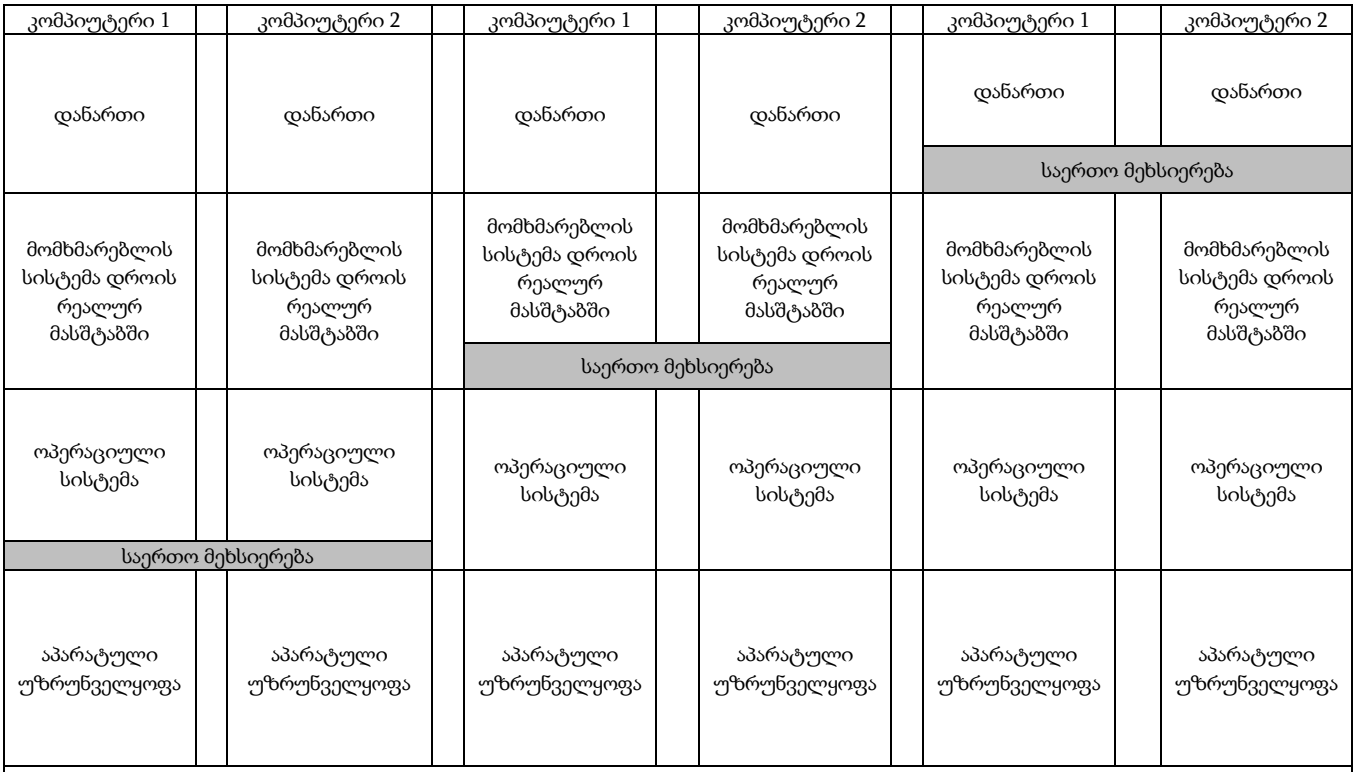
ნახ. 2.1.2. მულტიკომპიუტერი 16 პროცესორით, რომელთაგანც თითოეულს გააჩნია საკუთარი მეხსიერება (ა); გამოსახულებების ბიტური რუქა, განაწილებული მეხსიერების 16 მოდულს შორის (ბ)

ისმება შეკითხვა, მაშინ რაღა საჭიროა მულტიკომპიუტერების შექმნა, თუ მულტიპროცესორებოს დაპროგრამება უფრო ადვილია? პასუხი მარტივია: დიდი მულტიკომპიუტერის შექმნა გაცილებით უფრო ადვილია და იაფი იგივე შესაძლებლობების მულტიპროცესორთან შედარებთ. საერთო მეხსიერების შექმნა, რომლითაც ერთდროულად რამდენიმე ასეულო პროცესორი ისარგებლებს, საკმაოდ რთული საქმეა, ხოლო მულტიკომპიუტერის აგება, რომელშიც გაერთიანებული იქნება ათასობით და მილიონობით პრცესორი, უფრო ადვილია.

ამგვარად, იქმნება ასეთი დილემა: მულტიპროცესორების დაპროექტება რთულია, მაგრამ მათი დაპროგრამებაა ადვილი, ხოლო მულტიკომპიუტერების დაპროექტება ადვილია, მაგრამ მათი დაპროგრამებაა რთული. ამიტომ, ხშირია მათი გაერთიანების, ანუ **ჰიბრიდული** სისტემების შექმნის მცდელობა. ამ მცდელობების შედეგად დადგინდა, რომ საერთო მეხსიერების შექმნა შესაძლებელია რამდენიმე გზით, თანაც ყოველ ვარიანტს ექნება როგორც დადებითი ასევე უარყოფითი მხრეები. ჰიბრიდული პარალელური გამოთვლითი სისტემების შექმნა ძალიან პერსპექტიული მიმართულებაა, მასში გაერთიანებული იქნება ორივე მხარისათვის დამახასიათებელი დადებითი თვისებები. ასეთ სისტემებში მიღწეული უნდა იქნას მასშტაბირებადობა, ანუ დამუშავებული უნდა იქნას ისეთი სისტემა, რომელშიც ახალ ახალი პროცესორების დამატება პრობლემას არ შექმნის.

ერთერთი მიდგომა დაფუძნებულია იმ გარემოებაზე, რომ თანამედროვე კომპიუტერები არაა მონოლითური და გააჩნიათ მრავალდონიანი სტრუქტურა. ეს იძლევა საერთო მეხსიერების

სხვადასხვა დონეზე რეალიზების საშუალებას, როგორც ეს ნაჩვენებია ნახ. 2.1.3–ზე. ნახ 2.1.3,ა–ზე საერთო მეხსიერება რეალიზებულია აპარატულად, ისე, როგორც ეს კეთდება მულტიპროცესორებში. ასეთი ორგანიზების დროს მუშაობს ოპერაციული სისტემის ერთი ასლი ცხრილების ერთი ნაკრებით, კერძოდ გამოიყენება მეხსიერების განაწილების ცხრილი. პროცესს თუ სჭირდება უფრო მეტი მეხსიერება, მაშინ წყვეტს ოპერაციული სისტემის მუშაობას. ოპერაციული სისტემა ცხრილში იწყებს თავისუფალი ფურცლის ძებნას და ასახავს მას გამოძახებული პროცესის სამისამართო სივრცეში. რაც შეეხება ოპერაციულ სისტემას, ის მართავს ერთიან მეხსიერებას და ინახავს მონაცემებს იმის შესახებ, თუ მეხსიერების რომელი ფურცელი რომელ პროცესს ეკუთვნის. საერთო მეხსიერების აპარატული ორგანიზების ბევრი ხერხი არსებობს.



ნახ. 2.1.3. დონეები, რომლებზეც შესაძლებელია საერთო მეხსიერების ორგანიზება: აპარატული რეალიზება (ა); ოპერაციული სისტემა (ბ); პროგრამული რეალიზება (გ)

მეორე მიდგომა გულისხმობს მულტიკომპიუტერის აპარატული უზრუნველყოფისა და ოპერაციული სისტემის გამოყენებას, რომლის საშუალებითაც ხდება ფურცლებად დაყოფილი ერთიანი საერთო მეხსიერების მოდელირება ერთიანი ვირტუალური სამისამართო სივრცით. ასეთი მიდგომის შედეგად იქმნება **განაწილებული საერთო მეხსიერება** (Distributed Shared Memory, **DSM**), რომელშიც ყოველი ფურცელი მოთავსებულია მეხსიერების ერთერთ მოდულში (ნახ. 2.1.2,ა), ანუ ყოველ პროცესორს გააჩნია საკუთარი ვირტუალური მეხსიერება და ფურცლების ცხრილი. პროცესორი თუ ასრულებს LOAD ან STORE ბრძანებას და მიმართავს ფურცელს, რომელიც მის მეხსიერებაში არ არის, მაშინ ხდება სისტემური უარყოფა. ამის შემდეგ ოპერაციული სისტემა აღმოაჩენს საჭირო ფურცელს და მიმართავს შესაბამის პროცესორს, რათა მან ამოტვირთოს ეს ფურცელი თავისი მეხსიერებიდან და შიდა საკომუნიკაციო ქსელის საშუალებით გადაგზავნოს საჭირო პროცესორის მეხსიერებაში. როდესაც ეს ფურცელი აღმოჩნდება საჭირო პროცესორის

ოპერატიულ მეხსიერებაში, მაშინ შეწყვეტილი ბრძანების შესრულება გრძელდება. პრაქტიკულად ხდება შემდეგი:ოპერაციული სისტემა ღებულობს საჭირო ფურცლებს არა დისკიდან, არამედ ოპერატიული მეხსიერებიდან. ამ დროს, მომხმარებელს ექმნება შთაბეჭდილება, რომ პროცესორს აქვს ერთიანი ოპერატიული მეხსიერება.

მესამე მიდგომა გულისხმობს პროგრამულ მომხმარებლური საერთო სისტემის შექმნას დროის რეალური მასშტაბით. ასეთი მიდგომის დროს საერთო მეხსიერების აბსტრაქციას ქმნის დაპროგრამების ენა, და ამ აბსტარაქციის შექმნა ხდება კომპილატორის საშუალებით (ანუ, საერთო მეხსიერების მოდელი შეიძლება დამოკიდებული იყოს დაპროგრამების გამოყენებულ ენაზე). მაგალითად, მოდელი Linda დაფუძნებულია კორტეჟების წარმოდგენის საერთო სივრცეზე (მონაცემების ჩანაწერი მოიცავს ველების ნაკრებს). ნებისმიერი პროცესორის პროცესებს შეუძლიათ აიღონ კორტეჟები საერთო სივრციდან ან გააგზავნონ საერთო სივრცეში. რადგანაც ამ სივრცესთან წვდომა პროგრამულად სრულად კონტროლირებადია (რეალური დროის სისტემა Linda), ამიტომ არანაირი აპარატული მხარდაჭერა ან ოპერაციული სისტემა საჭირო არაა.

საერთო მეხსიერებაზე დაფუძნებულ დროის რეალურ მასშტაბზე მომუშავე გამოყენებითი რეალიზაციის მეორე მაგალითს წარმოადენს მონაცემების საერთო ობიექტებზე დაფუძნებული მოდელი Orca სისტემაში. Orca მოდელში Linda მოდელის მსგავსად ერთობლივად გამოიყენებენ არა კორტეჟებს, არამედ ბაზურ ობიექტებს, რომელთათვისაც იძახებენ ამა თუ იმ მეთოდს. მეთოდი თუ ცვლის ობიექტის შიდა მდგომარეობას, მაშინ რეალური დროის სისტემამ თვალყური უნდა მიადევნოს,რომ ერთდროულად შეცვლილი იქნას ამ სისტემის ყველა ასლი ყველა პროცესორზე. და კიდევ, ობიექტების პროგრამული რეალიზაცია შეიძლება რეალიზებული იქნას, ოპერაციულის სისტემის ჩარევის გარეშე ან აპარატული რეალიზების საშუალებით.

2.1.3. პარალელური კომპიუტერული სისტემების კლასიფიკაცია

პარალელური კომპიუტერული სისტემების რეალიზების შესახებ ბევრი შეიძლება ითქვას, მაგრამ ახლა დავუბრუნდეთ ძირითად თემას – ასეთი სისტემების არქიტექტურას. ხანგრძლივი დროის განმავლობაში აგებული იქნა პარალელური კომპიუტერული სისტემების მრავალი სახეობა. ამიტომ, ჩნდება მათი კლასიფიცირების საჭიროება. ამის გაკეთების მრავალი მცდელობა იყო, მაგრამ სამწუხაროდ კლასიფიცირების კარგი სისტემა დღესაც არ გვაქვს. ყველაზე ხშირად გამოიყენება კლასიფიცირების ფლინის სისტემა, მაგრამ ისიც კი საკმაოდ უხეშია.

პარალელური კომპიუტერების კლასიფიცირების სისტემას საფუძვლად უდევს ბრძანებების და ნაკადების ცნებებები. პროცესორების n რაოდენობის შემცველ სისტემას გააჩნია ბრძანებების n მთვლელი და ამგვარად, ბრძანებების n ნაკადი. მონაცემების ნაკადი შედგება ოპერანდების ნაკადისაგან.

ბრძანებებისა და მონაცემების ნაკადები გარკვეულწილად დამოუკიდებელია, ამიტომ არსებობს ასეთი ნაკადების 4 სახეობა (იხ. ცხრილი 2.1.1). **SISD** (Single Instruction stream Single Data stream – ბრძანებების ერთი ნაკადი მონაცემების ერთი ნაკადით) – იგი წარმოადგენს კომპიუტერების აგების კლასიკურ ფონ ნეიმანის სქემას. კომპიუტერების აგების ფონ ნეიმანის სქემაში გათვალისწინებულია ბრძანებების ერთი ნაკადი და მონაცემებისაც ერთი ნაკადი და ყოველ კონკრე-

ტულ მომენტში შეიძლება შესრულებული იქნას მხოლოდ ერთი მოქმედება. კომპიუტერებში, რომლებიც მიეკუთვნებიან **SIMD** (Single Instruction-stream Multiple Data-stream–ბრძანებების ერთი ნაკადი მონაცემების რამდენიმე ნაკადით) კატეგორიას, გამოყენებულია მართვის ერთი ბლოკი, რომელიც შესასრულებლად გასცემს თითოთითო ბრძანებას, მაგრამ ამ დროს არსებობს ბევრი ალმ, რომელთაც შეუძლიათ ერთდროულად შეასრულოს ბრძანებები მონაცემების ნაკადებზე. SIMD არქიტექტურის პროტოტიპია კომპიუტერი ILLIAC IV. SIMD კომპიუტერები არაა საყოველთაოდ გავრცელებული კომპიუტერები, მაგრამ თანამედროვე კომპიუტერებში მულტიმედიური მონაცემების დამუშავებისათვის გამოიყენება SIMD კატეგორიის ბრძანებები. თანამედროვე Pentium კომპიუტერებში SSE ტიპის ბრძანებები განეკუთვნება SIMD არქიტექტურის კატეგორიას. ამ ბრძანებების ტიპი პირველ პლანზე გამოდის ნაკადური პროცესორების შემთხვევაში. ნაკადური პროცესორები დამუშავებულია სპეციალურად მულტიმედიური მონაცემების დასამუშავებლად.

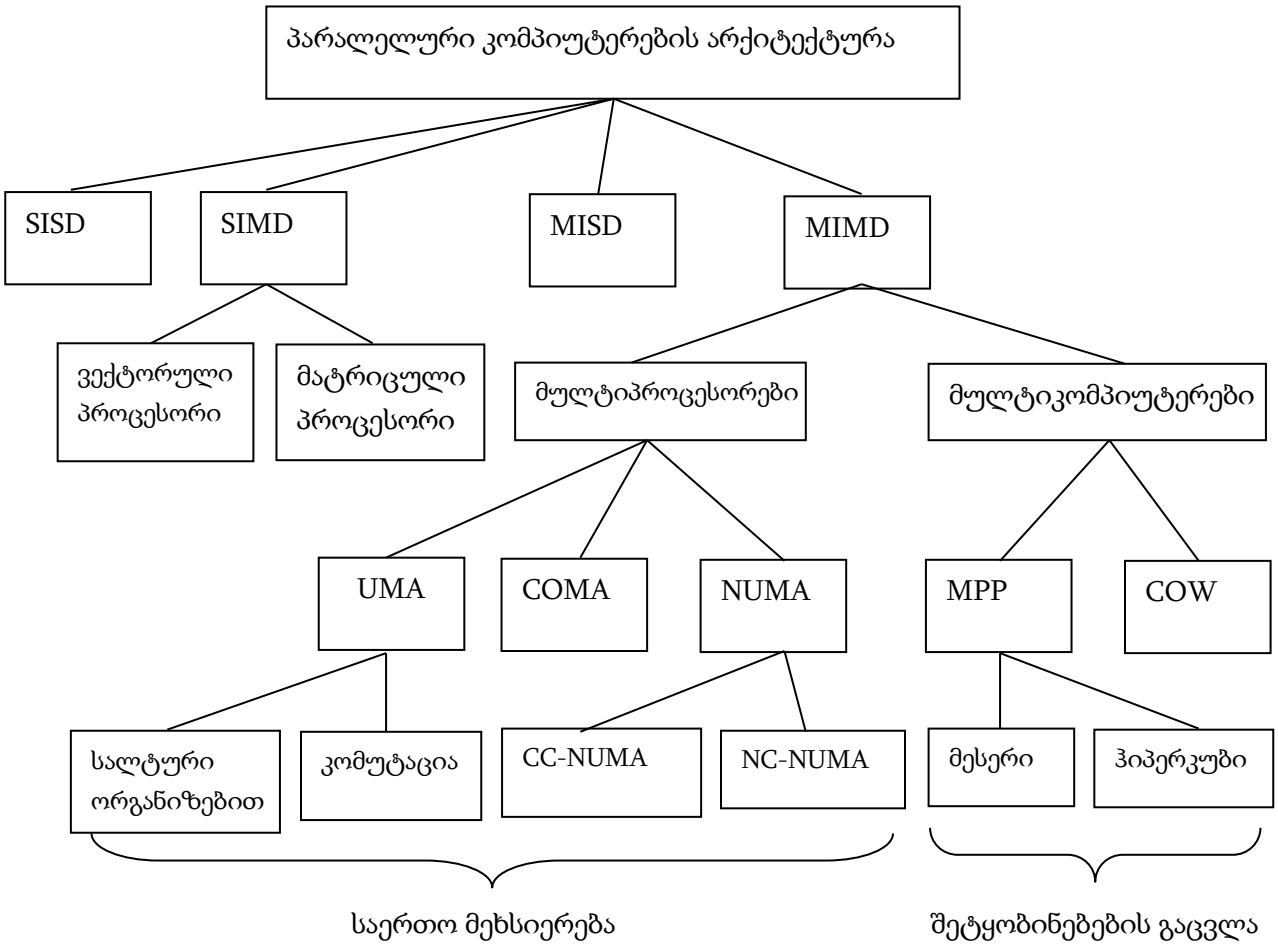
ცხრილი 2.1.1. კომპიუტერების კლასფიკაციის ფლინის მეთოდი

ბრძანებები ნაკადი	მონაცემების ნაკადი	კატეგორია	მაგალითები
1	1	SISD	ფონ ნეიმანის კლასიკური კომპიუტერი
1	ბევრი	SIMD	ვექტორული და მატრიცული კომპიუტერები
ბევრი	1	MISD	კომპიუტერები კონვეიერული ტიპის პროცესორებით
ბევრი	ბევრი	MIMD	მულტიპროცესორები, მულტიკომპიუტერები

არქიტექტურა **MISD** (Multiple Instruction-stream Single Data-stream - ბრძანებების რამდენიმე ნაკადი მონაცემების ერთი ნაკადით) წარმოადგენს ცოტა უცნაურ კატეგორიას. აქ გამოიყენება ბრძანებების რამდენიმე ნაკადი, რომელთა საშუალებითაც ხდება მონაცემების ერთი ნაკადის დამუშავება. კომპიუტერების ამ კატეგორიას მიეკუთვნება ის კომპიუტერები, რომელთა პროცესორებიც შეიცავს კონვეიერებს.

უკანასკნელი კატეგირიაა **MIMD** (Multiple Instruction-stream Multiple Data-stream - ბრძანებების რამდენომე ნაკადი მონაცემების რამდენიმე ნაკადით). ამ შემთხვევაში რამდენიმე დამოუკიდებელი პროცესორი მუშაობს როგორც ერთი სისტემის ნაწილი. ამ კატეგორიაში გაერთიანებულია პარალელური კომპიუტერების უმრავლესობა. მულტიპროცესორებიც და მულტიკომპიუტერებიც წარმოადგენენ MIMD კომპიუტერებს.

ფლინის კლასიფიკაციის გაფართოებული სურათი ნაჩვენებია ნახ. 2.1.4–ზე. აქ SIMD კომპიუტერები გაყოფილია ორ ქვეჯგუფად. პირველ ქვეჯგუფში გაერთიანებულია სუპერკომპიუტერები, რომლებშიც მოქმედებები ხდება ვექტორებზე, ანუ ერთი და იგივე ოპერაციები სრულდება ვექტორების ელემენტებზე. მეორე ქვეჯგუფში კი გაერთიანებულია ILLIAC IV ტიპის კომპიუტერები, სადაც მართვის ბლოკიდან ბრძანებები შესასრულებლად იგზავნება დამოუკიდებელ ალმ–ებზე.



ნახ. 2.1.4. პარალელური მოქმედების კომპიუტერების კლასიფიკაცია

MIMD კატეგორიის კომპიუტერები დაიყო ორ ქვეჯგუფად: მულტიპროცესორებად (კომპიუტერები საერთო მეხსიერებით) და მულტოკომპიუტერებად (კომპიუტერები შეტყობინებების გაცვლით). არსებობს მულტიპროცესორების სამი ტიპი. ისინი ერთმანეთისაგან განსხვავდება საერთო მეხსიერებასთან წვდომის მექანიზმით. ესენია: **UMA** (Uniform Memory Access - **მეხსიერებასთან ერთგვაროვანი წვდომა**), **NUMA** (Non Uniform Memory Access –**მეხსიერებასთა არაერთგვაროვანი წვდომა**) და **COMA** (Cache Only Memory Access - **მხოლოდ კეშ–მეხსიერებასთან წვდომა**). ასეთ ქვეკატეგორიებად დაყოფას აზრი აქვს, რადგანაც დიდ მულტიპროცესორებში მეხსიერება დაყოფილია რამდენიმე მოდულად. UMA კომპიუტერებში ყოველ მოდულს გააჩნია ერთნაირი წვდომა მეხსიერების ნებისმიერ მოდულთან. ეს ტექნიკურად თუ შესაძლებელი არაა, მაშინ ხდება სწრაფი მიმართვების შეყოვნება და მათი სიჩქარე გაუტოლდება ნელი მიმართვების სიჩქარეს და პროგრამისტი მათ შორის განსხვავებას ვეღარ ამჩნევს. სწორედ ეს ნიშნავს ერთგვაროვან წვდომას. ეს ერთგვაროვნება მწარმოებლურობას ხდის პროგნოზირებადს, რაც ძალიან მნიშვნელოვანია ეფექტური პროგრამების შექმნისათვის.

NUMA კომპიუტერებს კი პირიქით არ გააჩნიათ ერთგვაროვნების თვისება. ყოველ პროცესორს გააჩნია მეხსიერების მოდული, რომელიც მასთან ყველაზე ახლოსაა მოთავსებული და ამიტომ მასთან მიმართვა ხდება ყველაზე სწრაფად. ასეთ შემთხვევაში მაღალი მწარმოებლურობის უზრუნველსაყოფად ძალიან მნიშვნელოვანია სად აღმოჩნდება პროგრამა და მონაცემები. COMA

კომპიუტერებშიც წვდომა არაერთგვაროვანია, მაგრამ აქ მიზეზი სხვაა. შემდეგ, ყოველი შემთხვევა დაწვრილებით იქნება განხილული.

MIMD კატეგორიის კომპიუტერების მეორე ქვეჯგუფში მოხვდა მულტიკომპიუტერები. მათ მულტიპროცესორებისგან განსხვავებით არ გააჩნიათ საერთო მეხსიერება. ანუ, მულტიკომპიუტერის შემადგენლობაში შემავალი პროცესორის ოპერაციულ სისტემას არქიტექტურულ დონეზე არ შეუძლია წვდომა განახორციელოს სხვა პროცესორის მეხსიერებასთან LOAD და STORE ბრძანებების საშუალებით. ამ შემთხვევაში პროცესორმა უნდა გააგზავნოს შეტყობინება და უნდა დაელოდოს პასუხს. ოპერაციული სისტემების საშუალებით მომხმარებლის პროგრამებისათვის იქმნება ილუზია, რომ გააჩნიათ კავშირები ნებისმიერ მოდულთან. მულტიკომპიუტერებში პროცესორებს არ გააჩნიათ სხვა პროცესორის მეხსიერებასთან ფიზიკური წვდომის საშუალება. მათ ზოგჯერ მიაკუთვნებენ **NORMA** (NO Remote Memory Access –**მეხსიერებასთან„ დაშორებული წვდომა არ არის**) კატეგორიას.

მულტიკომპიუტერების მეორე ჯგუფი მოიცავს ჩვეულებრივ პერსონალურ კომპიუტერებს და სამუშაო სადგურებს, რომლებიც გაერთიანებულია ამათუიმ კომერციული საკომუმუნიკაციო ტექნოლოგიით. ასეთ შემთხვევაში მართალია გამოთვლითი სიმძლავრეების ათვისება ისეთი ეფექტურობით არხდება, როგორც მულტიპროცესორებში, მაგრამ მათი აგება ჯდება ბევრად უფრო იაფი MPP კომპიუტერებთან შედარებით. ამ ჯგუფის სისტემებს ხშირად უწოდებენ: **სამუშაო სადგურების ქსელს** (Network Of Workstations, **NOW**), **სამუშაო სადგურების კლასტერს** (Cluster Of Workstations, **COW**), ან უბრალოდ **კლასტერს** (Cluster).

2.1.4. მეხსიერების სემანტიკა

ყველა მულტიპროცესორში პროცესორებს მიენიჭებათ მეხსიერების მთელ სამისამართო სივრცესთან წვდომის უფლება. ხშირად, საერთო მეხსიერებასთან ერთად კიდევ არსებობს მოდულების სიმრავლე, რომელთაგან თითოეულში ინახება ფიზიკური მეხსიერების ესა თუ ის ფრაგმენტი. პროცესორები და მეხსიერების მოდულები ერთმანეთთან დაკავშირებულია რთული საკომუნიკაციო ქსელით. წარმოვიდგინოთ სიტუაცია, რამდენიმე პროცესორს ერთდროულად დასჭირდა მეხსიერებიდან სიტყვის წაკითხვა; ამავდროულად სხვა პროცესორები ცდილობენ მათში ჩაწერას; შეტყობინებები ადრესატებამდე შეიძლება მივიდეს განსხვავებული თანმიმდევრობით, ვიდრე გაგზავნილი იყო; ამას კიდევ თუ დაემატება, რომ მეხსიერებაში მონაცემები შეიძლება კოპირებული იყოს, მაშინ შეიძლება მივიღოთ სრული ქაოსი მეხსიერებაში. ამის საწინააღმდეგოდ საჭიროა კონტროლონისძიებების გატარება.

მეხსიერების სემანტიკა შეიძლება განხილული იქნას როგორც მეხსიერების აპარატულ და პროგრამულ უზრუნველყოფებს შორის კავშირი. აპარატული საშუალებები თუ იცავს გარკვეულ წესებს, მაშინ მეხსიერება გასცემს ან მიიღებს გარკვეულ მონაცემებს. ძირითად პრობლემას აქ სწორედ ეს წესები წარმოადგენს, რომლებსაც უწოდებენ **უნარიანობის მოდელებს**. შემოთავაზებული და დამუშავებული იქნა ამ წესების მთელი რიგი.

პრობლემის არსის წარმოსადგენად დავუშვათ, რომ პროცესორმა 0–მა მეხსიერების რომელიმე სიტყვაში ჩაწერა მნიშვნელობა 1, მოგვიანებით მეხსიერების იგივე სიტყვაში პროცესორი 1

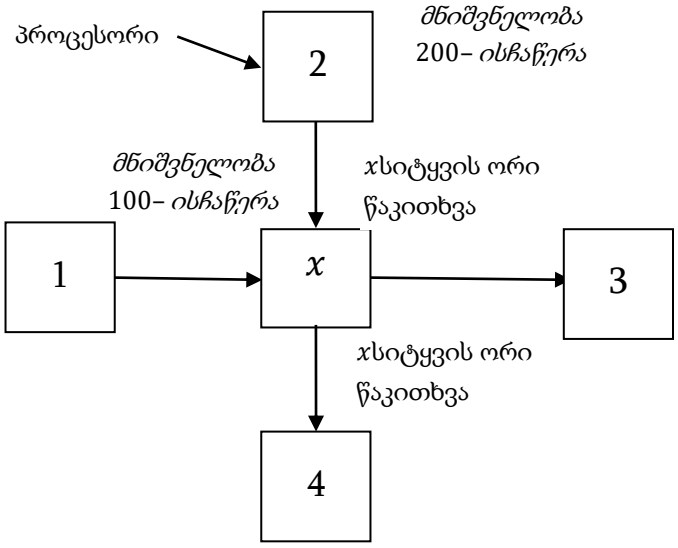
წერს მნიშვნელობა2–ს. პროცესორი 2 კითხულობს ამ სიტყვას და ღებულობს მონაცემ 1–ს. რა ხდება? ადგილი აქვს შეცდომას? ეს დამოკიდებულია იმაზე, თუ რა წესებით მუშაობს მეხსიერება.

მკაცრი უნარიანობა

მეხსიერების მუშაობის ყველაზე მარტივი მოდელია **მკაცრი უნარიანობა**. ამ მოდელის მიხდვით x მისამართიდან ყოველი ამოკითხვის შედეგად დაბრუნებული იქნება მასში უკანასკნელად ჩაწერილი მონაცემი. პროგრამისტებს ეს მოდელი ძალიან მოსწონთ, მაგრამ მისი რეალიზება შესაძლებელია მხოლოდ შემდეგნაირად: უნდა იყოს მეხსიერების მხოლოდ ერთი მოდული, რომელიც ემსახურება ყველა შეკვეთას მათი შემოსვლის რიგის მიხედვით, მონაცემების კეშირება და დუბლირება დაშვებული არ არის. მეხსიერეის მუშაობის ასეთი მიდგომა მნიშვნელოვნად შეანელებდა მონაცემებთან წვდომას. ამიტომ, ეს მოდელი აღარ იქნება განხილვის საგანი.

სეკვენციალური უნარიანობა

განვიხილოთ **სეკვენციალური უნარიანობის** მოდელი. ამ მოდელის მიხედვით მონაცემების ჩაწერასა და წაკითხვაზე რამდენიმე მოთხოვნის არსებობის შემთხვევაში მათ შორის რიგითობის განსაზღვრა ხდება აპარატულად, და ეს რიგითობა საერთოა ყველა პროცესორისათვის.



ნახ. 2.1.5. საერთო მეხსიერებაში ორი პროცესორი წერს და ორი კიდევ კითხულობს ერთიდაიგივე სიტყვას

განვიხილოთ შემთხვევა. ვთქვათ პროცესორი 1 წერს მნიშვნელობას 100 მეხსიერების x სიტყვაში, ხოლო 1 ნწმ–ის შემდეგ პროცესორი 2 იგივე მისამართზე წერს მნიშვნელობას 200. ეხლა წარმოვიდგინოთ, რომ ჩაწერის მეორე ოპერაციის დაწყებიდან 1 ნწმ–ის შემდეგ (ჩაწერის ოპერაცია ჯერ არ არის დასრულებული) სხვა ორი პროცესორი 3 და 4 კითხულობს x სიტყვას ორ–ორჯერ (ნახ. 2.1.5).

აღწერილ სიტუაციაში შესაძლებელია 6 სხვადასხვა შემთხვევა. ეს შემთხვევები მოყვანილია ცხრილში 2.1.2.

პირველ ვარიანტში ორივე პროცესორი წაკითხვის შედეგად მიიღებს მნიშვნელობას 200. მეორე ვარიანტში წაკითხვის ორი ოპერაციის შედეგად პროცესორი 3 მიიღებს მნიშვნელობებს 100 და 200, ხოლო პროცესორი 4 კი ორივეჯერ მნიშვნელობას 200. მესამე ვარიანტში პროცესორი 3 ორჯერ მიიღებს მნიშვნელობას, ხოლო პროცესორი 4 კი მნიშვნელობებს 200 და 100. ყველა ეს ვარიანტი დასაშვებია.

ცხრილი 2.1.2. მოვლენების რიგითობის შესაძლო ვარიანტები ნახ. 2.1.5–ის მიხედვით

ვარიანტი 1	ვარიანტი 2	ვარიანტი 3
მნიშვნელობა 100–ის ჩაწერა	მნიშვნელობა 100–ის ჩაწერა	მნიშვნელობა 200–ის ჩაწერა
მნიშვნელობა 200–ის ჩაწერა	მნიშვნელობა 100–ის წაკითხვა პროცესორი 3–ის მიერ	მნიშვნელობა 200–ის წაკითხვა პროცესორი 4–ის მიერ
მნიშვნელობა 200–ის წაკითხვა პროცესორი 3–ის მიერ	მნიშვნელობა 200–ის ჩაწერა	მნიშვნელობა 100–ის ჩაწერა
მნიშვნელობა 200–ის წაკითხვა პროცესორი 3–ის მიერ	მნიშვნელობა 200–ის წაკითხვა პროცესორი 4–ის მიერ	მნიშვნელობა 100–ის წაკითხვა პროცესორი 3–ის მიერ
მნიშვნელობა 200–ის წაკითხვა პროცესორი 4–ის მიერ	მნიშვნელობა 200–ის წაკითხვა პროცესორი 3–ის მიერ	მნიშვნელობა 100–ის წაკითხვა პროცესორი 4–ის მიერ
მნიშვნელობა 200–ის წაკითხვა პროცესორი 4–ის მიერ	მნიშვნელობა 200–ის წაკითხვა პროცესორი 4–ის მიერ	მნიშვნელობა 100–ის წაკითხვა პროცესორი 3–ის მიერ

სეკვენციალური მოდელის საფუძველზე აგებული მეხსისრება არ დართავს ნებას პროცესორ 3–ს მიიღოს მონაცემები 100 და 200, თუ პროცესორი ღებულობს მონაცემებს 200 და 100. ეს თუ ასე მოხდა, მაშინ პროცესორ 3–ის პოზიციიდან გამოდის, რომ პროცესორ 1–მა მნიშვნელობა 100–ის ჩაწერა დაამთავრა უფრო ადრე, ვიდრე პროცესორი 2 ჩაწერდა მნიშვნელობას 200. ეს სრულიად შესაძლებელია. მაგრამ, პროცესორ 4–ის პოზიციიდან გამომდინარე ეს ნიშნავს, რომ პროცესორ 2–მა მნიშვნელობა 200–ის ჩაწერა დაამთავრა უფრო ადრე, ვიდრე პროცესორ 1–მა ჩაწერა მნიშვნელობა 100. ეს შედეგიც სავსებით შესაძლებელია, მაგრამ ეწინააღმდეგება პირველ შედეგს. სეკვენციალური უნარიანობა უზრუნველყოფს ჩაწერის ოპერაციების ერთიან გლობალურ თანმიმდევრობას (შეიძლება იყოს ნებისმიერ). პროცესორი 3–ის პოზიციებიდან თუ პირველად ჩაიწერა მნიშვნელობა 100, მაშინ პროცესორ 4–მაც იგივე უნდა დაინახოს.

მიუხედავად იმისა, რომ სეკვენციალური უნარიანობის მოდელი არ გამოიყურება ისე „მკაცრად“, როგორც მკაცრი უნარიანობის მოდელი, ის მაინც ძალიან სასარგებლო მოდელია. იმ შემთხვევაშიც კი, თუ ორი მოვლენა ერთდროულად ხდება, ჩაითვლება ეს მოვლენები თანმიმდევრულად მოხდა (თანმიმდევრობა შეიძლება ნებისმიერი იყოს) და ყველა პროცესორი იმოქმედებს ამ თანმიმდევრობით.

პროცესორული უნარიანობა

პროცესორული უნარიანობა ძალიამ მკაცრი მოდელია, მაგრამ დიდ მულტიპროცესორებში ადვილია მისი რეალიზება. მას გააჩნია ორი თვისება:

1. ყველა პროცესორი ხედავს სხვა პროცესორების მიერ მეხსიერებაში ჩაწრის ოპერაციებს იმ თანმიმდევრობით, რა თანმიმდევრობითაც მოხდა მათი განხორციელება.
2. ყველა პროცესორი მეხსიერების სიტყვაში ჩაწერის ოპერაციებს ხედავს ერთიდაიგივე თანმიმდევრობით.

ორივე ეს პუნქტი ძალიან მნიშვნელოვანია. პირველ პუნქტში ლაპარაკია იმაზე, რომ თუ პროცესორ 1–მა მეხსიერების რომელიმე ადგილზე დაიწყო 1A, 1B და 1C მონაცემების ჩაწერა ამ თანმიმდევრობით, მაშინ ყველა დანარჩენი პროცესორი ამ მონაცემებს დაინახავს იგივე თანმიმდევრობით. მეორე პუნქტი საჭიროა იმისათვის, რომ ყოველ სიტყვას მეხსიერებაში ჰქონდეს გარკვეული და არაორაზროვანი მნიშვნელობა მას შემდეგ, რაც პროცესორმა ამ სიტყვაში განახორციელა რამდენიმე ჩაწერა და გაჩერდა. ყველა პროცესორმა უნდა დაინახოს ერთიდაიგივე უკანასკნელი მნიშვნელობა.

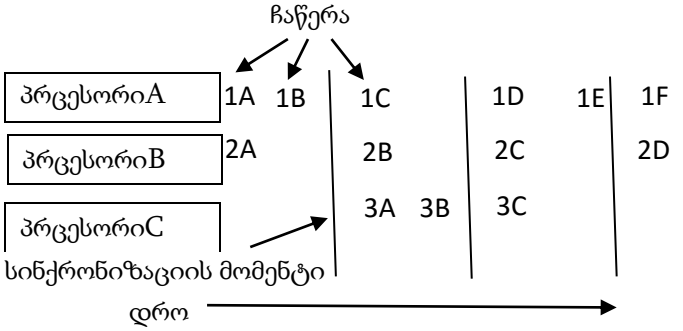
ასეთი შეზღუდვების პირობებშიც კი დამპროექტებელს გააჩია მრავალი შესაძლებლობა. ვნახოთ რა მოხდება თუ პროცესორ 2–მა პროცესორ 1–თან ერთდროულად დაიწყო 2A, 2B და2C მნიშვნელობების ჩაწერა. სხვა პროცესორები, რომლებიც ახდენენ მეხსიერებიდან წაკითხვას, ჩაწერის 6 ოპერაციიდან დაინახავენ რომელიმე თანმიმდევრობას, მაგალითად 1A, 1B, 2A, 2B, 1C, 2C ან 2A, 1A, 2B, 1B, 2C, 1C ან რომელიმე სხვა მიმდევრობას. პროცესორული უნარიანობის შედეგად არ ხდება იმის გარანტირება, რომ ყველა პროცესორი ხედავს ერთიდაიგივე თანმიმდევრობას (სეკვენციალური უნარიანობისაგან განსხვავებით). შესაძლებელია ისეც მოხდეს, რომ ზოგმა პროცესორმა დაინახოს თანმიმდევრობა 1A, 1B, 2A, 2B, 1C, 2C, ხოლო ზოგმა კი 2A, 1A, 2B, 2C, 1B, 1C თანმიმდევრობა. ერთადერთი რისი გარანტირებაც შეიძლება, არის ის, რომ ერთი პროცესორის მიერ განხორციელებული მიმართვების თანმიმდევრობა არ დაირღვევა და დარჩება სწორი ყველა პროცესორისათვის.

სუსტი უნარიანობა

შემთხვევაში არ ხდება იმის გარანტირება, რომ ერთერთი პროცესორის მიერ განხორციელებული ჩაწერის ოპერაციები სხვა პროცესორების მიერ აღქმული იქნება იგივე თანმიმდევრობით. ერთმა პროცესორმა შეიძლება ჯერ დაინახოს ოპერაცია 1A და შემდეგ 1B, მეორე პროცესორმა კი შეიძლება ჯერ დაინახოს ოპერაცია 1B და შემდეგ კი ოპერაცია 1A. ამ ქაოსში სიცხადის შემოტანისათვის საჭიროა მეხსიერების სინქრონიზაციის ცვლადების შემოტანა სინქრონიზაციის ოპერაციების მხარდაჭერა. სინქრონიზაციის შემთხვევაში ხდება ჩაწერის ყველა დაწყებული ოპერაციის დასრულება. არ მოხდება ახალი ჩაწერის ოპერაციის დაწყება, ვიდრე ჩაწერის ყველა დაწყებული ოპერაცია დასრულებული არ იქნება და არ დასრულდება თვით სინქრონიზაციაც. სინქრონიზაციას მეხსიერება მოჰყავს მდგრად მდგომარეობაში, მის შემდეგ სისტემაში არ რჩება ჩაწერის არცერთი დაუსრულებელი ოპერაცია. თვით სინქრონიზაციის ოპერაციები წარმოადგენენ სეკვენციალური უნარიანობის ოპერაციებს, ანუ თუ მათი ინიცირება ხდება რამდენიმე პროცესორის მიერ, მაშინ მოხდებაოპერაციების შესრულების გარკვეული თანმიმდევრობის ამორჩევა და ყველა პროცესორი იცავს ამ თანმიმდევრობას.

სუსტი უნარიანობის შემთხვევაში დრო იყოფა მკაცრად თანმიმდევრულ პერიოდებად, რომლებიც ერთმანეტისაგან გამოყოფილია სინქრონიზაციის ოპერაციებით (ნახ. 2.1.6). 1A და 1B ჩაწერის ოპერაციებისათვის რაიმე განსაკუთრებული თანმიმდევრობის გარანტირება არ ხდება. სხვადასხვა პროცესორებმა ეს პროცესები შეიძლება სხვადასხვა თანმიმდევრობით დაინახონ, ანუ ერთი პროცესორის თვალთახედვით შეიძლება გამოჩნდეს ჯერ 1A და შემდეგ 1B, ხოლო მეორე პროცესორისათვის კი პირიქით ჯერ 1B და შემდეგ 1A. ასეთი სიტუაცია დასაშვებია. ყველა

პროცესორისათვის ოპერაცია 1B შესრულდება უფრო ადრე ვიდრე ოპერაცია 1C, რადგანაც 1C, 2B, 3A, 3B ჩაწერის ოპერაციები დაიწყება მხოლოდ მასშემდეგ, რაც სინქრონიზაციის ოპერაციების განმავლობაში მოხდება 1A, 1B, და 2A ჩაწერის ოპერაციების დასრულება. ამგვარად, სინქრონიზაციის ოპერაციების საშუალებით პროგრამულად შეიძლება შემოტანილი იქნას მოვლენების გარკვეული მიმდევრობითობა, თუმცა კი ამას სჭირდება გარკვეული დრო, რადგანაც ეს დაკავშირებულია მეხსიერების კონვეიერის დაცლასთან და შემდეგ ხელახალ შევსებასთან.



ნახ. 2.1.6. სინქრონიზაცია სუსტი უნარიანობის მეხსიერებაში

თავისუფალი უნარიანობა

სუსტი უნარიანობა არ წარმოადგენს მაღალი ეფექტურობის მქონე მოდელს, რადგანაც მოითხოვს მეხსიერებასთან დაწყებული ოპერაციების დასრულებას და აფერხებს ახალი ოპერაციების დაწყებას დაწყებული ოპერაციების დასრულებამდე. **თავისუფალი უნარიანობის მოდელში** საქმე უკეთესადაა, რადგანაც აქ გამოიყენება პროგრამების კრიტიკული სექციის მსგავსი მექანიზმი. იდეა შემდეგში მდგომარეობს. პროგრამა თუ გამოდის კრიტიკული არიდან, ეს არ ნიშნავს იმას, რომ ჩაწერის ყველა პროცესი დასრულებული უნდა იქნას. მოითხოვება მხოლოდ ის, რომ დაწყებული პროცესების დასრულება მოხდეს მანამდე, ვიდრე კრიტიკულ არეში არ შევა კიდევ რომელიმე სხვა როცესი.

ამ მოდელში სინქრონიზაციის ოპერაცია გაყოფილია ორ სხვადასხვა ოპერაციად. პროცესორს (უფრო სწორად მის პროგრამულ უზრუნველყოფას) თუ სჭირდება რომელიმე ცვლადის მნიშვნელობის მეხსიერებაში ჩაწერა ან წაკითხვა, სინქრონიზაციის ცვლადისათვის ჯერ უნდა შეასრულოს acquire ოპერაცია, რომელიც მიანიჭებს მას საერთო მანაცემებთან მონოპოლიური წვდომის უფლებას. ამის შემდეგ პროცესორს შეუძლია შეასრულოს ნებისმიერი მოქმედება მონაცემებთან. მოქმედებების დასრულების შემდეგ სინქრონიზაციის ცვლადისათვის უნდა შეასრულოს release ოპერაცია, რითაც ის ადასტურებს, რომ მუშაობა დაასრულა. ოპერაცია release არ მოითხოვს ჩაწერის დაუსრულებელი ოპერაციების დასრულებას, მაგრამ თვითონ ამ ოპერაციის დასრულება არ მოხდება, ვიდრე არ დასრულდება დაწყებული ჩაწერის ოპერაციები.

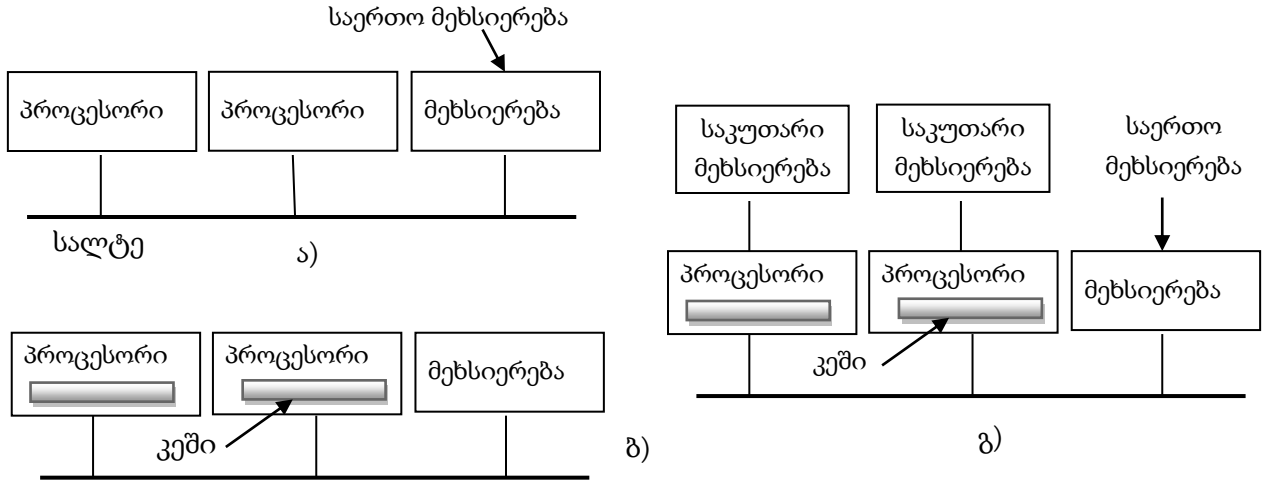
შემდეგი acquire ოპერაციის დაწყებამდე ჯერ მოწმდება დასრულებულია თუ არა ყველა დაწყებული release ოპერაცია. თუ რომელიმე დაწყებული release ოპერაცია დასრულებული არაა, მაშინ ხდება acquire ოპერაციის დაწყების შეჩერება. release ოპერაციის დასრულებამდე ყველა დაწყებული ჩაწერის ოპერაცია დასრულებული იქნება. ამგვარად, შემდეგი acquire ოპერაციის დაწყება უკანასკნელი release ოპერაციის დასრულების შემდეგ თუ დიდი დროითი ინტერვალის შემდეგ ხდება, მაშინ მას არ სჭირდება მოცდა და პირდაპირ შეუძლია კრიტიკულ არეში შესვლა. თუ acquire ოპერაცია უკანასკნელი release ოპერაციის დასრულების შემდეგ მალევე იწყება, მაშინ ის (და შესაბამისად მისი მომდევნო ყველა ბრძანება) იცდის ყველა release ოპერაციის

დასრულებამდე. ეს იძლევა იმის გარანტიას, რომ კრიტიკულ არეში მოხდება ყველა მონაცემის განახლება. ეს მოდელი სუსტი უნარიანობის მოდელზე ცოტა უფრო რთულია, მაგრამ მას გააჩნია არსებითი უპირატესობა: აქ არ ხდება ბრძანებების შესრულების შეჩერება ისე ხშირად, როგორც ეს ხდებოდა სუსტი უნარიანობის მოდელში.

ამით არ სრულდება მეხსიერების უნარიანობის საკითხის შესწავლა, მეცნიერები დღესაც გვთავაზობენ ახალ მოდელებს.

2.2.1. UMA–მულტიპროცესორები სიმეტრიულ მულტიპროცესორულ არქიტექტურაში

ყველაზე მარტივ მულტიპროცესორებს გააჩნიათ ერთი სალტე (ნახ. 2.2.1,ა). ორი ან მეტი პროცესორი და მეხსიერების ერთი ან რამდენიმე მოდული იყენებს ამ სალტეს მონაცემების გაცვლისათვის. პროცესორს მეხსიერებიდან თუ სჭირდება სიტყვის წაკითხვა, ის ჯერ ამოწმებს თავისუფალია თუ არა სალტე. სალტე თუ თავისუფალია, პროცესორი სალტეზე დადებს საჭირო სიტყვის მისამართს, დააყენებს რამდენიმე მმართველ სიგნალს და დაელოდება, ვიდრე მეხსიერება სალტეზე დადებს მოთხოვნილი სიტყვის მნიშვნელობას.



ნახ. 2.2.1. საერთო სალტეზე მულტიპროცესორის სამი ვარიანტი: კეშ–მეხსიერების გარეშე (ა); კეშ–მეხსიერებით (ბ); კეშ–მეხსიერებით და ლოკალური მეხსიერებით (გ)

სალტე თუ დაკავებულია, მაშინ პროცესორი უცდის მის გათავისუფლებას. ამ სქემასთან დაკავშირებულია ერთი პრობლემა. ორი და სამი პროცესორის შემთხვევაში სალტესთან წვდომის მართვა რთული არაა. პრობლემა წარმოიქმნება როდესაც სალტესთან მიერთებულია ბევრი პროცესორი, მაგალითად 32. ამ შემთხვევაში მთელი სისტემის მწარმოებლურობას განსაზღვრავს სალტის გამტარუნარიანობა. პროცესორებს ხშირად უწევთ ლოდინი სალტის გათავისუფლებისათვის.

პრობლემის გადაწყვეტა შესაძლებელია პროცესორებისათვის კეშ–მეხსიერების დამატებით (ნახ. 2.2.1,ბ). კეშ–მეხსიერება შეიძლება მოთავსებული იქნას პროცესორის მიკროსქემაში ან პროცესორის მიკროსქემის გვერდით ან პროცესორის პლატაზე. ამ სამი ვარიანტიდან შესაძლებელია ნებისმიერი კომბინაცია. ამ შემთხვევაში პროცესორი ბევრ მონაცემებს აიღებს კეშ–

მეხსიერებიდან, რაც სალტეზე ტრაფიკს შეამცირებს და სისტემა შეძლებს უფრო მეტი პროცესორის მომსახურებას. ამგვარად, კეშ მეხსიერების გამოყენება ზრდის სისტემის ეფექტურობას.

შემდეგ სქემაზე პროცესორს გააჩნია არა მრტო კეშ–მეხსიერება, არამედ საკუთარი ლოკალური მეხსიერება, რომელთანაც წვდომას ახორციელებს სპეციალურად გამოყოფილი ლოკალური სალტით (ნახ. 2.2.1,გ). ამ კონფიგურაციის ოპტიმალურად ასამოქმედებლად კომპილატორმა ლოკალური მეხსიერების მოდულებში უნდა ჩადოს მთელი პროგრამული კოდი, სტრიქონები, კონსტანტები და სხვა მონაცემები, რომლებიც განკუთვნილია მხოლოდ ჩაწერისათვის, აგრეთვე სტეკები და ლოკალური ცვლადები. საერთო მეხსიერების გამოყენება მოხდება მხოლოდ ერთობლივად გამოსაყენებელი მონაცემების შესანახად. ხშირ შემთხვევებში მონაცემების ასეთი გონივრული განაწილებით მიიღწევა საერთო სალტეზე ტრაფიკის მნიშვნელოვანი შემცირება და აღარ ხდება საჭირო პროცესებში კომპილატორის აქტიური ჩარევა.

2.2.2. კეშ–მეხსიერებების შეთანხმებულობა

დავუშვათ მეხსიერება არის სეკვენციალურად უნარიანი. მაშინ რა მოხდება როდესაც პროცესორი 1 თავის კეშ–მეხსიერებაში ინახავს რაიმე სტრიქონს, ხოლო პროცესორი 2 ცდილობს წაიკითხოს სიტყვა, რომელიც შეესაბამება კეშ–ის იმავე სტრიქონს? სპეციალური წესების არ არსებობის შემთხვევაში პროცესორი 2 თავის კეშ–მეხსიერებაში მიიღებს ამ სტრიქონის ასლს. ე.ი. მოხდება ერთიდაიგივე მონაცემის ორმაგი კეშირება. ეს სავსებით დასაშვებია. ეხლა დავუშვათ პროცესორ 1–მა შეცვალა თავისი სტრიქონი და მალევე ამის შემდეგ პროცესორი 2 კითხულობს ამ სტრიქონის ასლს თავისი კეშ–მეხსიერებიდან. შედეგად მივიღეთ, რომ პროცესორ 2–მა მიიღო **მოძველებული მონაცემი**. ამას პროცესორ 2–ში მიმდინარე პროგრამისათვის მოჰყვება ცუდი შედეგი.

ეს არის მნიშვნელოვანი პრობლემა და მას უწოდებენ **კეშ–მეხსიერებების შეთანხმებულობას**. ეს პრობლემა თუ გადაჭრილი არ იქნება, მაშინ მულტიპროცესორებში შეუძლებელი გახდება კეშ–მეხსიერების გამოყენება და მათში საჭირო გახდება პროცესორების რაოდენობის შეზღუდვა 2–3–მდე. სპეციალისტების მიერ შემოთავაზებული იქნა ამ პრობლემის ბევრნაირი გადაწყვეტა. შემოთავაზებული ალგორითმები ატარებს **კეშ–მეხსიერებების შეთანხმების განაწესის** სახელს. ეს ალგორითმები დეტალებში ერთმანეთისაგან განსხვავდება, მაგრამ ისინი უშვებენ ერთიდაიგივე სტრიქონის სხვადასხვა ვერსიის მოხვედრის შესაძლებლობას სხვადასხვა კეშ–მეხსიერებაში.

ყველა შემთხვევაში კეშ–მეხსიერების კონტროლერი დამუშავებულია ისეთნაირად, რომ შესაძლებელი იყოს პროცესორებიდან ან სხვა კეშ–მეხსიერებებიდან მომავალი შეკვეთების მონიტორინგი. ყოველ კონკრეტულ შემთხვევაში მიმართავენ ამა თუ იმ მოქმედებებს. ასეთმა მოწყობილობამ მიიღო **„მოთვალთვალე“ კეშ–მეხსიერების** (snooping cache) სახელწოდება, რადგანაც კეშ–მეხსიერება თითქოს „უთვალთვალებს“ სალტეს. წესების ერთობლიობას, რომელთაც იცავს კეშ–მეხსიერება, პროცესორები და ძირითადი მეხსიერება, რათა თავიდან იქნას აცილებული სხვადასხვა კეშ–მეხსიერებაში მონაცემების სხვადასხვა ვერსიების მოხვედრის შესაძლებლობა, უწოდებენ კეშ–მეხსიერებების შეთანხმების განაწესს. კეშ–მეხსიერებაში გადაცემისა და შენახვის ერთეულს წარმოადგენს **კეშ–ის სტრიქონი**, რომლის სიგრძეც შეიძლება იყოს 32 ან 64 ბაიტი.

კეშ–მეხსიერების შეთანხმებულობის ყველაზე მარტივ განაწესს უწოდებენ **გამჭოლ ჩაწერას** (write through). ამ განაწესის მოქმედების მექანიზმის უკეთ გასაგებად განვიხილოთ ცხრილ 2.2.1–ში ჩამოთვლილი 4 შემთხვევა. პროცესორი თუ ცდილობს კეშ–მეხსიერებიდან წაიკითხოს მასში არ არსებული სიტყვა, მაშინ კეშ–მეხსიერების კონტროლერი მასში ჩატვირთავს ამ სიტყვის შეცველ სტრიქონს. სტრიქონი აიღება ძირითადი მეხსიერებიდან, რომელიც ამ განაწესის მიხედვით ყოველთვის უნდა ინახავდეს განახლებულ მონაცემებს. ამის შემდეგ ინფორმაცია წაიკითხება კეშ–მეხსიერებიდან.

ცხრილი 2.2.1. გამჭოლი ჩაწერა (ცარიელი გრაფები აღნიშნავს, რომ მოქმებება არ სრულდება)

მოქმედება	ლოკალური შეკვეთა	დაშორებული შეკვეთა
წაკითხვის კეშ–აცილება	მეხსიერებიდან მონაცემების გამოძახება	
წაკითხვის კეშ–მოხვედრება	მონაცემების გამოყენება ლოკალური კეშ–მეხსიერებიდან	
ჩაწერის კეშ–აცილება	მეხსიერებაში მონაცემების განახლება	
ჩაწერის კეშ–მოხვედრება	ძირითადი და კეშ–მეხსიერებების განახლება	კეშ–მეხსიერებაში განახლებული ელემენტი არასწორია

კეშ–აცილების შემთხვევაში მოდიფიცირებული სიტყვების ჩაწერა ხდება ძირითად მეხსიერებაში. საჭირო სიტყვის შემცველი სტრიქონი არ ჩაიტვირთება კეშ–მეხსიერებაში. კეშ–მოხვედრების შემთხვევაში ხდება კეშ–ჩანაწერების განახლება, ხოლო სიტყვა ამავდროულად ჩაიწერება ძირითად მეხსიერებაში. განაწესის აზრი მდგომარეობს იმაში, რომ ჩაწერის ყველა ოპერაციის შედეგად ჩასაწერი სიტყვა აუცილებლად გაივლის ძირითად მეხსიერებაში, რათა ძირითად მეხსიერებაში ყოველთვის ინახებოდეს „განახლებული“ მონაცემები.

ყველა ეს მოქმედება განვიხილოთ თავიდან, მაგრამ „მოთვალთვალე“ კეშ–მეხსიერების პოზიციიდან (2.2.1 ცხრილის მარჯვენა სვეტი). მოქმედებების შემსრულებელ კეშ–ს ვუწოდოთ კეში 1, ხოლო მომდევნო კეშ–ს კი კეში 2. კეშ 1–ში თუ ხდება წაკითხვის კეშ–აცილება, მაშინ კეშ 1 მიმართავს სალტეს ძირითადი მეხსიერებიდან საჭირო სტრიქონის მისაღებად. კეშ 2 ხედავს ამას, მაგრამ არაფერს არ აკეთებს. კეშ 1 მეხსიერებაში წაკითხვის დროს თუ ხდება კეშ–მოხვედრება (ანუ საჭირო სტრიქონი არსებობს კეშ 1 მეხსიერებაში), მაშინ სალტესთან მიმართვა აღარ ხდება, ამიტომ კეშ 2 მეხსიერებისათვის არაფერი არაა ცნობილი კეშ 1 მეხსიერებაში კეშ–მოხვედრებების შესახებ.

ჩაწერის პროცესი კიდევ უფრო საინტერესოა. პროცესორი 1 თუ ჩაწერს სიტყვას, მაშინ კეშ 1 მიმართავს სალტეს ისე, როგორც აცილების შემთხვევაში. ასევე ხდება კეშ–მოხვედრების შემთხვევაშიც. ნებისმიერი ჩაწერის შემთხვევაში კეშ 2 ამოწმებს ჩაწერილ სიტყვას თავისთან. ჩაწერილი სიტყვა თუ მასთან არ არის, მაშინ ის ამ სიტუაციას აფასებს როგორც დაშორებული მეხსიერების კეშ–აცილებას და არაფერს არ აკეთებს. (2.2.1 ცხრილის მიხედვით დაშორებული მეხსიერების კეშ–აცილება ნიშნავს, რომ „მოთვალთვალე“ კეშ–მეხსიერებაში სიტყვა არ არის, ხოლო ინიციატორ კეშ–მეხსისრებაში ეს სიტყვა არის თუ არ არის, მნიშვნელობა არ აქვს. ამგვარად, ერთი და იგივე მიმართვის შედეგად შეიძლება მივიღოთ მხოლოდ კეშ–მოხვედრება ან კეშ–აცილება).

ეხლა დავუშვათ კეშ 1–ში იწერება სიტყვა, რომელიც არსებობს კეშ 2–ში. თუ კეშ 2–მა არაფერი არ გააკეთა, მაშინ მას ექნება მოძველებული მონაცემები. ამიტომ, კეშ 2 შეცვლილი სიტყვის შემცველ ელემენტს აღნიშნავს, როგორც არაქმედითუნარიანს, რის შედეგადაც ხდება მეხსიე-

რებიდან მისი ამოგდება. რადგანაც ყოველი კეშ–მეხსიერება თვალყურს ადევნებს სალტესთან მიმართვებს, ამიტომ, სიტყვის ყოველი ჩაწერის ოპერაციის შემდეგ ხდება მისი განახლება კეშ–ინიციატორში და ძირითად მეხსიერებაში და ამავდროულად ყველა დანარჩენ კეშ–მეხსიერებაში მისი განადგურება. ამგვარად ხდება შეთანხმებულობის მიღწევა.

ცხადია, კეშ 2–ის პროცესორს შემდეგ ციკლში შეუძლია წაიკითხოს იგივე სიტყვა. ამ შემთხვევაში კეშ 2–ში მიიღება უკვე განახლებული სიტყვა. ამ მომენტში კეშ 1, კეშ 2 და ძირითადი მეხსიერება შეიცავს ამ სიტყვის იდენტურ ასლებს. თუ რომელიმე სხვა პროცესორი განახორციელებს ჩაწერას, მაშინ მოხდება დანარჩენი პროცესორების კეშ–მეხსიერებების გაწმენდა და ძირითადი მეხსიერების კვლავ განახლება.

შესაძლებელია ამ ძირითადი განაწესის სხვადასხვა ვარიანტი. მაგალითად, ჩაწერის კეშ–მოხვედრების შემთხვევაში შემდეგი კეშ–მეხსიერება არასწორად აცხადებს ჩასაწერი სიტყვის შემცველ ელემენტს. ელემენტის არასწორად გამოცხადების მაგივრად შესაძლებელია აგრეთვე ახალი მნიშვნელობის მიღება და კეშ–მეხსიერების განახლება. არსებითად, კეშ–მეხსიერების განახლება იგივეა, რაც ელემენტის არასწორად გამოცხადება და შემდეგ საჭირო სიტყვის წაკითხვა მთავარი მეხსიერებიდან. კეშირების ყველა განაწესში საჭიროა გაკეთებული იქნას არჩევანი **განახლების სტრატეგიასა** და **მონაცემების არასწორად გამოცხადებას შორის**. ეს განაწესები სხვადასხვა დატვირთვის პირობებში სხვადასხვანაირად მუშაობს. განახლების შეტყობინებებს გააჩნიათ სასარგებლო დატვირთვა და შესაბამისად მოცულობითაც მეტია მონაცემების არასწორად გამოცხადების შეტყობინებებთან შედარებით, მაგრამ მათი საშუალებით შესაძლებელია შემდგომი კეშ–აცილების თავიდან აცილება.

კიდევ ერთი ვარიანტი – შემდეგი კეშ–მეხსიერების ჩატვირთვა ჩაწერის კეშ–აცილების შემთხვევაში. ასეთი ჩატვირთვა არანაირ გავლენას არ ახდენს ალგორითმის სისწორეზე, ის გავლენას ახდენს მხოლოდ მწარმოებლურობაზე. ისმება შეკითხვა: როგორია ახლად ჩაწერილი სიტყვის მალევე ხელახალი ჩაწერის ალბათობა? ეს ალბათობა თუ მაღალია, მაშინ ჩაწერის კეშ–აცილებების შემთხვევაში უპირატესობა ენიჭება კეშ–მეხსიერების ჩატვირთვას (**ჩაწერით შევსების პოლიტიკა**). ამ ალბათობის მნიშვნელობა თუ მცირეა, მაშინ ჩაწერის კეშ–აცილების შემთხვევაში უმჯობესია არ მოხდეს კეშ–მეხსიერების განახლება. თუ მოცემული სიტყვა მალევე წაკითხული უნდა იქნას, მაშინ წაკითხვის კეშ–აცილების შემდეგ მაინც მოხდება მისი ჩატვირთვა, ამიტომ ჩაწერის კეშ–აცილების შემდეგ აზრი არ აქვს მის განახლებას.

ეს გადაწყვეტილება არ გამოირჩევა მაღალი ეფექტურობით. ეს იმის შედეგია, რომ ჩაწერის ყოველი ოპერაცია მოითხოვს ძირითადი მეხსიერებისათვის მონაცემების გადაცემას სალტის საშუალებით და პროცესორების დიდი რაოდენობის შემთხვევაში სალტე ხდება ვიწრო ადგილი. ამიტომ, დამუშავებული იქნა სხვა განაწესები. ყოველ მათანს აქვს ერთი საერთო თვისება: ჩაწერის ყოველი ოპერაცია არ იწვევს ჩაწერას ძირითად მეხსიერებაში. ამის მაგივრად, კეშ–მეხსიერებაში სტრიქონის შეცვლის შემთხვევაში თვითონ კეშ–მეხსიერებაში ხდება სპეციალური ბიტის გამოყოფა, რომლის საშუალებითაც დგინდება, რომ კეშ–მეხსიერებაში სტრიქონი სწორია, ხოლო ძირითად მეხსიერებაში კი არასწორი. საბოლოო ჯამში, ამ სტრიქონის ძირითად მეხსიერებაში ჩაწერა მაინც მოხდება, მაგრამ ეს მოხდება ჩაწერის რამდენიმე ოპერაციის შემდეგ. განაწესების ასეთ ტიპს უწოდებენ **გადადებული ჩაწერის განაწესს**.

2.2.3. განაწესი MESI

გადადებული ჩაწერის განაწესებიდან ერთერთი პოპულარულია განაწსი **MESI** (Invalid, Shared, Exclusive, Modified - **მოქმედი, განაწილებული, ექსკლუზიური, მოდიფიცირებული**). დასახელების 4 ასო აღებულია კეშ–მეხსიერების 4 მდგომარეობის შესაბამისი სიტყვებიდან. MESI განაწესი გამოყენებულია Pentium 4 და სხვა პროცესორებში სალტის მართვისათვის. ამ განაწესის მიხედვით კეშ–მეხსიერების ყოველი ელემენტი შეიძლება იმყოფეოდეს ოთხი მდგოარეობიდან ერთერთში:

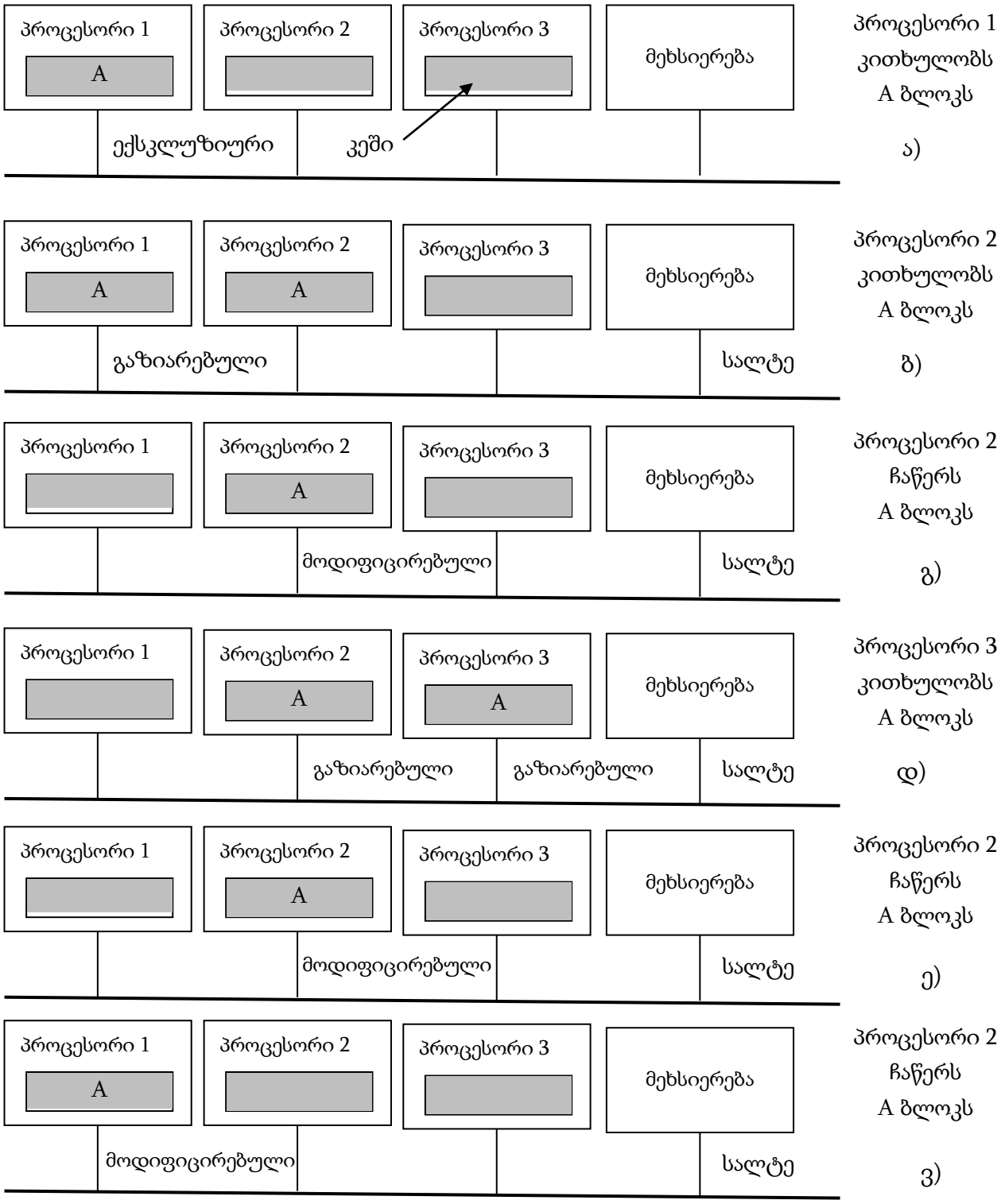
- არასწორი – კეშ–მეხსიერების ელემენტი შეიცავს არასწორ მონაცემებს;
- გაყოფადი – ელემენტი შეიძლება შენახული იყოს რამდენიმე კეშ–მეხსიერებაში, მეხსიერება განახლებულია;
- ექსკუზიური – ელემენტი მოთავსებულია მხოლოდ მოცემულ კეშ–მეხსიერებააში (სხვა კეშ–მეხსიერებებში ის არ არის), მეხსიერება განახლებულია;
- მოდიფიცირებული – ელემენტი სწორია, ძირითადი მეხსიერება არასწორია, ელემენტის ასლები არ არსებობს.

პროცესორის ჩატვირთვის დროს კეშ–მეხსიერების ყველა ელემენტი მოინიშნება, როგორც არასწორი. ძირითადი მეხსიერებიდან პირველი წაკითხვის დროს ხდება საჭირო სტრიქონის გამოძახება მოცემული პროცესორის კეშ–მეხსიერებაში და მოინიშნება როგორც ექსკლუზიური, რადგანაც ის ერთადერთი კეშირებული ასლია (ნახ. 2.2.2,ა). შემდგომი წაკითხვების დროს პროცესორი გამოიყენებს ამ სტრიქონს სალტესთან მიმართვის გარეშე. სხვა პროცესორს შეუძლია ამ სტრიქონის გამოძახება და კეშ–მეხსიერებაში მოთავსება. ამ შემთხვევაში სტრიქონის პირველი მფლობელი (პროცესორი 1) თვალყურის მიდევნების შედეგად აღმოაჩენს, რომ ის უკვე აღარ არის სტრიქონის ერთადერთი მფლობელი და სალტეზე განაცხადებს, რომ მას გააჩნია ასლი. ორივე ასლი გამოცხადდება როგორც გაზიარებული (ნახ. 2.2.2,ბ). სხვა სიტყვებით რომ ვთქვათ, მდგომარეობა „გაზიარებული“ აღნიშნავს, რომ მეხსიერება განახლებულია, და სტრიქონი წაკითხვის შემდეგ იმყოფება ერთ ან რამდენიმე კეშ–მეხსიერებაში. გაზიარებული სტრიქნის შემდგომი წაკითხვების დროს პროცესორი აღარ გამოიყენებს სალტეს და აღარ ცვლის სტრიქონის მდგომარეობას.

ეხლა განვიხილოთ თუ რა ხდება როდესაც პროცესორი 2 ახორციელებს ჩაწერას კეშ–მეხსიერების გაზიარებულ სტრიქონში. ასეთ შემთხვევაში პროცესორი სალტეზე დადებს სპეციალურ სიგნალს, რომლის საშუალებითაც დანარჩენ პროცესორებს ატყობინებს, რომ მათი ასლები არასწორია, ხოლო თვითონ პროცესორი 2–ის კეშ–მეხსიერებაშიასლი გადადის მდგომარეობაში „მოდიფიცირებულია“ (ნახ. 2.2.2,გ). მეხსიერებაში ეს სტრიქონი არ ჩაიწერება. აღსანიშნავია, რომ აღნიშნული სტრიქონი თუ არის ექსკლუზიური, მაშინ სალტეზესტრიქონის არასწორობაზე არავითარი სიგნალის გადაცემა არ არის საჭირო, რადგანაც ამ სტრიქონის არანაირი ასლი არ არსებობს.

ახლა გავარჩიოთ თუ რა ხედება, როდესაც ამ სტრიქონს კითხულობს პროცესორი 3. პროცესორ 2–მა, რომელიც ამ მომენტში არის სტრიქონის მფლობელი, იცის, რომ მეხსიერებაში ასლი არასწორია, ამიტომ, ის სალტეზე გასცემს სიგნალს, რომლითაც ის პროცესორ 3–ს ატყობიებს, რომ მან მოიცადოს ვიდრე ის მეხსიერებაში ამ სტრიქონს ჩაწერს. ჩაწერის შემდეგ პროცესორი 3

მეხსიერებიდან გამოიძახებს სტრიქონის ახლად ჩაწერილ ასლს და ორივე კეშ–მეხსიერებაში სტრიქონი მოინიშნება როგორც გაზიარებული (ნახ. 2.2.2,დ). შემდეგ, როდესაც პროცესორი 2 ამ სტრიქონის მნიშვნელობას შეცვლის (ნახ. 2.2.2,გ), მისი ასლი პროცესორ 3–ის კეშ–მეხსიერებაში გახდება არასწორი.

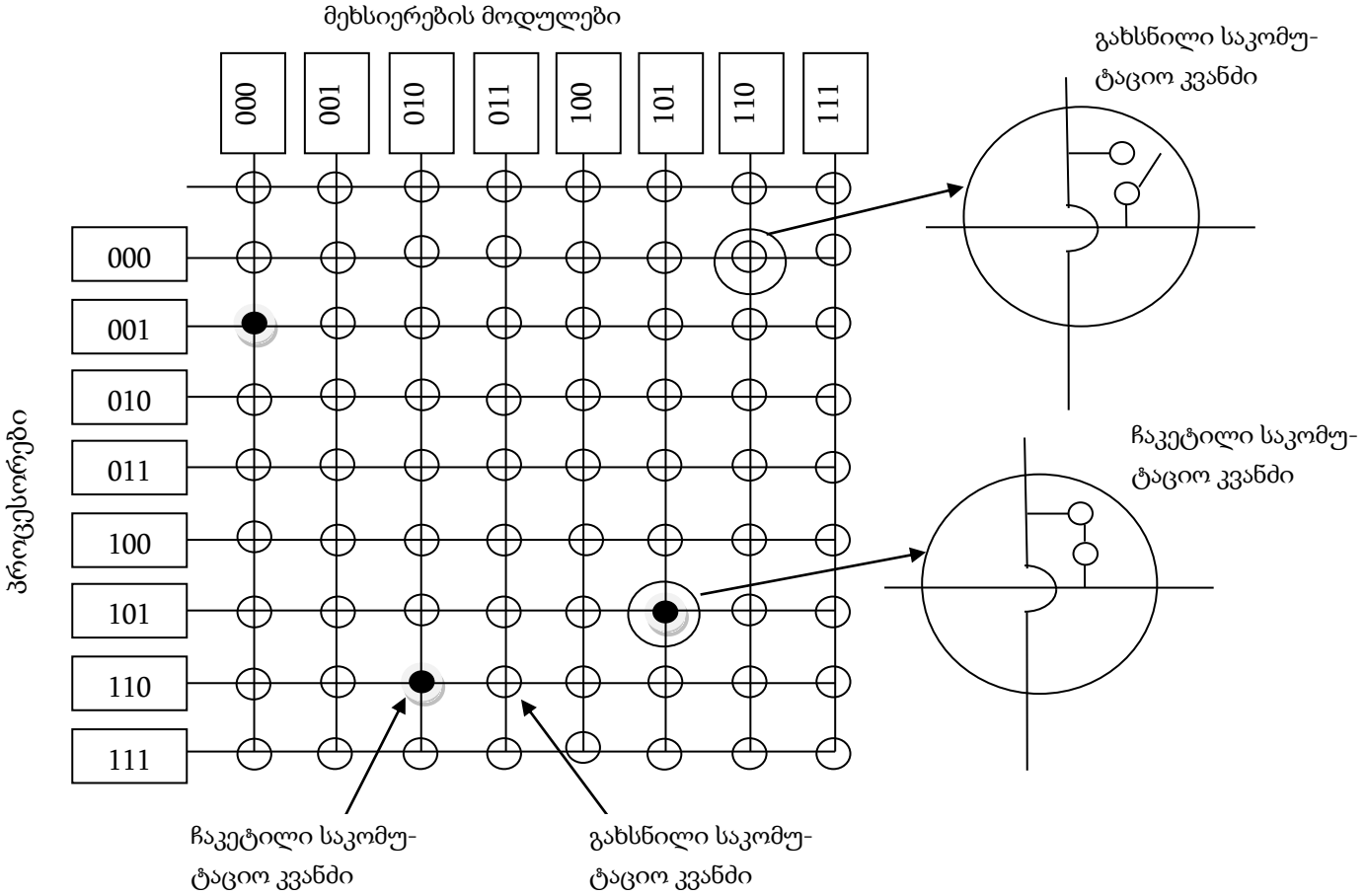


ნახ. 2.2.2. MESI განაწესის ილუსტრირება

და ბოლოს, ვთქვათ სტრიქონში სიტყვას წერს პროცესორი 1. პროცესორი 2 ხედავს, რომ ადგილი აქვს ჩაწერის მცდელობას და სალტეზე დააყენებს სიგნალს, რომლითაც პროცესორ 1–ს ატყობინებს, რომ მან მოიცადოს, ვიდრე პროცესორი 2 თავის სიტყვას ჩაწერს მეხსიერებაში. ჩაწერის დასრულების შემდეგ პროცესორი 2 მონიშნავს თავის ასლს როგორც არასწორს, რადგანაც იცის, რომ სხვა პროცესორი აპირებს მისი მნიშვნელობის შეცვლას. შეიძლება შეიქმნას სიტუაცია, რომლის დროსაც პროცესორი აპირებს ჩაწერას არაკეშირებად სტრიქონში. თუ გამოიყენება ჩაწერით შევსების პოლიტიკა, მაშინ სტრიქონი ჩაიწერება კეშ–მეხსიერებაში და მოინიშნება როგორც მოდიფიცირებული (ნახ. 2.2.2,ვ). თუ ჩაწერით შევსების პოლიტიკა არ გამოიყენება, მაშინ ჩაწერა ხდება პირდაპირ მეხსიერებაში და სტრიქონის კეშირება აღარ ხდება.

2.2.4. UMA–მულტიპროცესორები ჯვარედინი კომუტაციით

UMA–მულტიპროცესორში, იმის გამო, რომ მხოლოდ ერთი სალტეა, ოპტიმიზირების შემდეგაც კი შეუძლებელია 16 ან 32 პროცესორზე მეტის გაერთიანება. პროცესორების უფრო დიდი რაოდენობის გამოყენებისათვის საჭიროა სხვა ტიპის საკომუნიკაციო ქსელის გამოყენება. **ჯვარედინი კომურაცია** არის ყველაზე მარტივი სქემა, რომლითაც შესაძლებელია n რაოდენობის პროცესორების k რაოდენობის მეხსიერების მოდულებთან დაკავშირება (ნახ. 2.2.3). ჯვარედინი კომუტაცია უკვე დიდი ხნის წინ გამოიყენებოდა სატელეფონო კომუტატორებში, რომლის საშუალებითაც ხდებოდა შემავალი გამომავალი ხაზების ჯგუფების ერთმანეთთან მიერთება.



2.2.3. ჯვარედინი კომუტაციის სქემა 8×8 (ა); გახსნილი კვანძი (ბ); ჩაკეტილი კვანძი (გ)

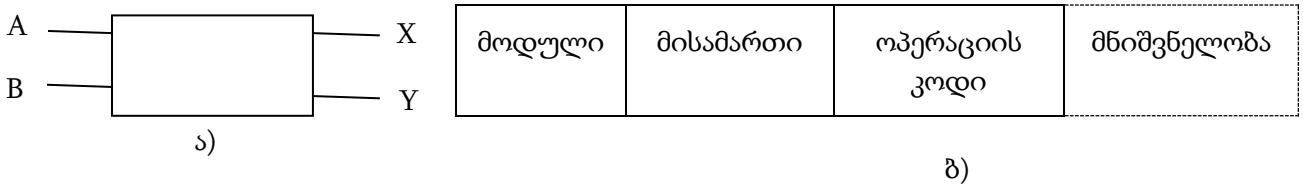
ჯვარედინი კომუტაციის სქემაში ჰორიზონტალური და ვერტიკალური ხაზების გადაკვეთაზე გამოყენებულია **საკომუტაციო კვანძი** (crosspoint), რომელიც შესაძლებელია იქნას გახსნილი ან დაკეტილი იმისდა მიხედვით გვჭირდება თუ არა ჰორიზონტალური და ვერტიკალური ხაზების ერთმანეთთან მიერთება. ნახ. 2.2.3, ა–ზე ვხედავთ, რომ 3 კვანძი ჩაკეტილია, რის შედეგადაც ერთდროულად მყარდება კავშირი პროცესორებისა და მეხსიერების მოდულების შემდეგ წყვლებს შორის (001, 000), (101, 101), (110, 010). შეაძლებელია ნებისმიერი სხვა კომბინაციაც. შესაძლო კომბინაციების რაოდენობა იგივეა რაც საჭადრაკო დაფაზე 8 ლაზიერის დაყენებისა ისე, რომ არცერთი ლაზიერი არ იყოს სხვა ლაზიერის დარტყმის ქვეშ.

ჯვარედინი კომუტაციის სქემის ერთერთი ყველაზე კარგი თვისებაა ის, რომ ის არის **არაბლოკირებადი**. ეს ნიშნავს, რომ პროცესორს ყოველთვის შეუძლია დაუკავშირდეს მეხსიერების საჭირო მოდულს მაშინაც კი, თუ ზოგიერთი კვანძი ან ხაზი დაკავებულია, მთავარია დაკავებული არ იყოს მეხსიერების საჭირო მოდული. უფრო მეტიც, კავშირის დასამყარებლად არავითარი წინასწარი დაგეგმვა საჭირო არ არის.

ჯვარედინი კომუტაციის სქემის ცუდ თვისებას წარმოადგენს ის, რომ მასში კვანძების რაოდენობა იზრდება როგორც n^2. საშუალო ზომის სისტემებისათვის ჯვარედინი კომუტაციის სქემის გამოყენება საკითხის კარგი გადაწყვეტაა, რის საილუსტრაციოდ შემდეგ განხილული იქნება NUMA–მულტიპროცესორი Sun Fire E25K. დიდ სისტემებში კი, სადაც პროცესორების რაოდენობა 1000–ს აჭარბებს, საკომუნიკაციო კვანძების რაოდენობა უკვე მილიონობითაა. ეს კი მიუღებელია. ასეთ შემთხვევებში მონახული უნდა იქნას სხვა გამოსავალი.

2.2.5. UMA–მულტიპროცესორები მრავალსაფეხურიანი კომუტაციით

მრავალსაფეხურიანი კომუტაციის სქემას საფუძვლად უდევს მცირე კომუტატორი 2×2 (ნახ. 2.2.4, ა) ორი შესასვლელით და ორი გამოსასვლელით. ნებისმიერ შესასვლელზე მოსული შეტყობინება შეიძლება გაცემული იქნას ნებისმიერ გამოსასვლელზე. ჩვენ მაგალითში შეტყობინება შეიძლება შედგებოდეს 4 ნაწილისაგან (ნახ. 2.2.4, ბ). მოდულის ველი მიუთითებს თუ მეხსიერების რომელი მოდულია მოთხოვნილი. მისამართის ველი განსაზღვრავს მისამართს მოთხოვნილ მოდულში. ოპერაციის კოდი ველში ჩაიწერება ერთერთი შესაძლო ოპერაცია, მაგალითად READ ან WRITE. და ბოლოს, მნიშვნელობის ველში შეიძლება მოცემული იყოს 32–თანრიგიანი სიტყვა, რომლის მეხსიერებაში ჩაწერაც უნდა მოხდეს WRITE ბრძანების შემთხვევაში. კომუტატორი ამოწმებს მოდულის ველს და მის მიხედვით განსაზღვრავს თუ რომელი გამოსასვლელი ხაზიდან უნდა იქნას გაცემული შეტყობინება: X ან Y.

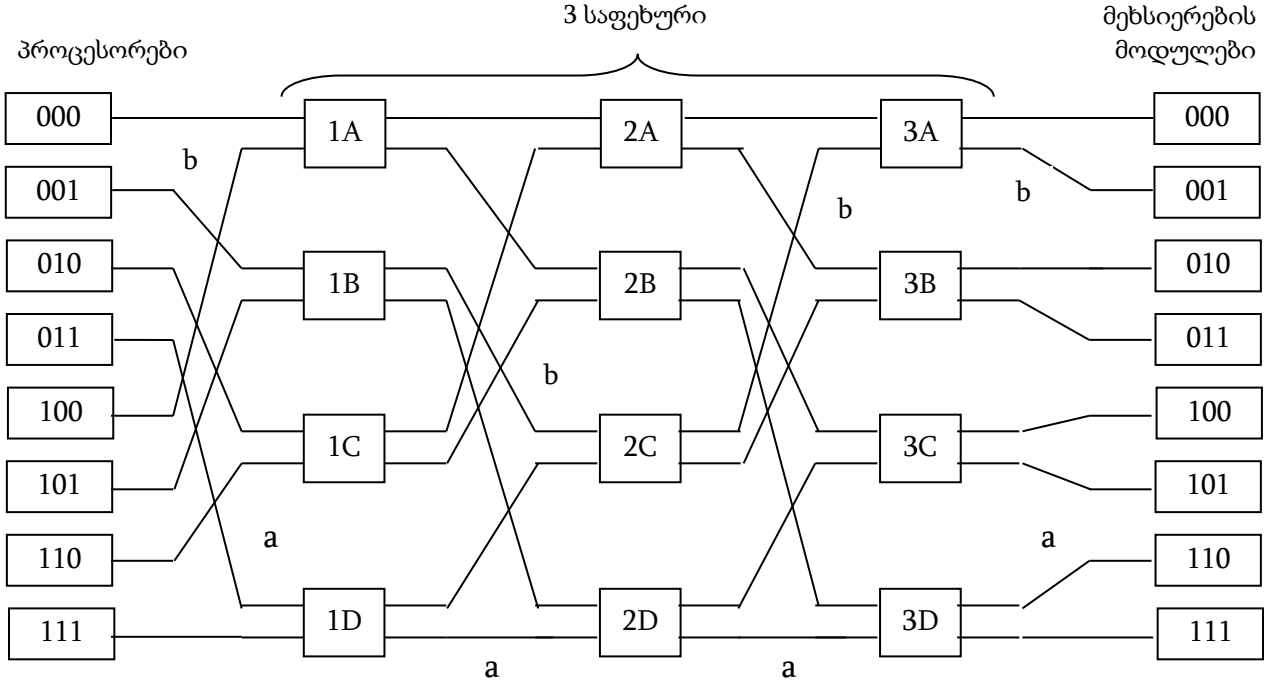


ნახ. 2.2.4. კომუტატორი 2×2 (ა); შეტყობინების ფორმატი (ბ)

ამ მარტივი კომუტატორებით შეიძლება აგებული იქნას უფრო რთული სქემები. სწორედ ამ გზით მიიღება **მრავალდონიანი კომუტაციის ქსელი**. ასეთი ქსელის ერთერთი მაგალითია

ქსელი **omega** (ნახ. 2.2.5). აქ 8 პროცესორი დაკავშირებულია მეხსიერების 8 მოდულთან და გამოყენებულია 12 კომუტატორი. n პროცესორისა და მეხსიერების n მოდულის ერთმანეთთან დასაკავშირებლად საჭირო გახდება $log_2 n$ საფეხური და $n/2$ კომუტატორი ყოველი საფეხურისათვის. ანუ საბოლოო ჯამში დაგვჭირდება $(n/2)\ log_2 n$ კომუტატორი, რაც ბევრად ნაკლებია ჯვარედინი კომუტაქიის სქემისათვის საჭირო კვანძების რაოდენობაზე, n^2–ზე.

omega ქსელში მავთულების გაყვანის სქემას უწოდებენ **სრულ აჭრას** (perfect shuffle), რადგანაც ყოველ საფეხურზე სიგნალების არევა ხდება ბანქოს ქაღალდების მსგავსად. omega ქსელის მუშაობის გასარკვევად დავუშვათ, რომ 011 პროცესორს სჭირდება სიტყვის წაკითხვა მეხსიერების 110 მოდულიდან. პროცესორი აგზავნის READ შეტყობინებას და გადართავს 1D კომუტატორს. ამ შეტყობინების მოდულის ველში მითითებულია 110. კომუტატორი მოდულის ნომრიდან განიხილავს უკიდურეს მარცხენა ბიტს და მის მიხედვით განსაზღვრავს გადართვის მიმართულებას (0 მიუთითებს ზედა გამოსასვლელს, ხოლო 1 კი ქვედა გამოსასვლელს). ჩვენ შემთხვევაში რადგანაც ამ ბიტის მნიშვნელობა არის 1, შეტყობინება ქვედა გამოსასვლელით გაეგზავნება 2D კომუტატორს.



ნახ.2.2.5. omega ქსელი მრავალსაფეხურიანი კომუტაციით

მეორე საფეხურის ყველა კომუტატორი გადართვის მიმართულების დასადგენად გამოიყენებს მეხსიერების მოდულის ნომრის მეორე თანრიგს. ჩვენ შემთხვევაში მისი მნიშვნელობაა 1, ამიტომ შეტყობინება ქვედა გამომყვანით გადაეგზავნება 3D კომუტატორს, რომელიც გააანალიზებს უკვე ნომრის მესამე ბიტს. რადგანაც მისი მნიშვნელობა 0–ის ტოლია, ამიტომ შეტყობინება კომუტატორის ზედა გამოსასვლელით გადაეცემა მეხსიერების 110 მოდულს. შეტყობინების მიერ გავლილი გზა აღნიშნულია a ასოთი.

ქსელში შეტყობინების გადაადგილების დროს ყოველი საფეხურის გავლის შემდეგ ფუნქციას კარგავს მეხსიერების მოდულის ნომერში ამ საფეხურის შესაბამისი თანრიგი. ეს თანრიგები შეიძლება გამოყენებული იქნას შემავალი ხაზების ნომრების ჩასაწერად. ამ მისამართით

მოხდება პასუხის დაბრუნება. a მარშრუტისათვის შემავალი ხაზები იქნება 0 (1D კომუტატორის ზედა გამოსასვლელი), 1 (2D კომუტატორის ქვედა გამოსასვლელი) და 1 (3D კომუტატორის ქვედა გამოსასვლელი). ამგვარად, პასუხის გაგზავნის დროს ასევე გამოიყენება მიმდევრობა 011, ოღონდ ამ შემთხვევაში ამ მიმდევრობის წაკითხვა უნდა მოხდეს საპირისპირო მიმართულებით– მარჯვნიდან მარცხნივ.

ვთქვათ იმ დროს, როდესაც ეს ხდება, პროცესორმა 001 აგრეთვე გადაწყვიტა სიტყვის ჩაწერა მეხსიერების მოდულში 001. აქ ადგილი აქვს ანალოგურ პროცესს. შეტყობინება გაიგზავნება კომუტატორის ზედა ან ქვედა გამოსასვლელიდან. ნახ. 2.2.5 ეს მარშრუტი მონიშნულია ასო b–თი. როდესაც შეტყობინება მიაღწევს დანიშნულების ადგილს, მისი ფორმატის მოდულის ველში მითითებული იქნება მიმდევრობა 001, რომელიც მიუთითებს ამ შეტყობინების მიერ გავლილ გზას. რადგანაც ეს ორი შეტყობინება გაივლის ორ სხვადასხვა კომუტატორს, მეხსიერების მოდულებსა და ხაზებს, ამიტომ მათი შესრულება შესაძლებელია პარალელურად.

ეხლა განვიხილოთ რა შეიძლება მომხდარიყო იმ შემთხვევში თუ პროცესორ 000–ს დასჭირდებოდა მეხსიერების მოდულ 000–თან წვდომა. მისი შეკვეთა მოვიდოდა კონფლიქტში 3A კომუტატორის 001 პროცესორის შეკვეთასთან და მათგან ერთერთს მოუწევდა ლოდინი. ანუ, ჯვარედინი კომუტაციის ქსელისაგან განსხვავებით, ქსელი omega არის **ბლოკირებადი ქსელი**. ერთდროულად არ შეიძლება გადაცემული იქნას შეკვეთების ნებისმიერი ნაკრები. კონფლიქტები შეიძლება წარმოიქმნას როგორც შეკეთებს შორის (ერთიდაიგივე ხაზების ან ერთიდაიგივე კომუტატორების გამოყენების შემთხვევაში), ასევე შეკვეთებს (მეხსიერებასთან) და პასუხებს (მეხსიერებიდან) შორის.

ცხადია, რომ მეხსიერებასთან მიმართვები განაწილებული უნდა იქნას თანაბრად მეხსიერების მოდულებს შორის. ერთერთი შესაძლებელი ხერხია, რომ მისამართების უმცროსი ბიტები გამოყენებული იქნას, როგორც მეხსიერების მოდულის ნომერი. განვიხილოთ კომპიუტერების სამისამრთო სივრცე ბიტური დამისამართებით, რომელსაც ძირითადად სჭირდება 32–თნრიგიან სიტყვებთან წვდომა. უმცროსი ორი ბიტი როგორც წესი 00–ის ტოლია, მაგრამ შემდეგი სამი ბიტი განაწილებულია თანაბრად. ამ სამ ბიტს თუ გამოვიყენებთ მეხსიერების მოდულის მისამართად, მაშინ მიმდევრობით დამისამართებული სიტყვები აღმოჩნდება მიმდევრობით განლაგებულ მოდულებში. მეხსიერების სისტემას, რომელშიც მიმდევრობითი სიტყვები განლაგებულია მეხსიერების სხვადასხვა მოდულებში, უწოდებენ **განშრევებულ მეხსიერებას**. განშრევებულ მეხსიერებას პარალელიზმი დაჰყავს აბსოლუტურობამდე, რადგანაც მეხსიერებასთან მიმართვების უმრავლესობა არის მიმდევრობით მისამართებთან მიმართვა. შესაძლებელია აგრეთვე არაბლოკირებული ქსელის დამუშავებაც, რომელშიც ტრაფიკის ოპტიმიზირებისათვის შემოთავაზებულია რამდენიმე გზა ყოველი პროცესორიდან მეხსიერების ყოველი მოდულისაკენ.

2.2.6. NUMA-მულტიპროცესორები

ერთი სალტის მქონე UMA–მულტიპროცესორებში პროცესორების რაოდენობა შეზღუდულია რამოდენიმე ათეულით, ხოლო ჯვარედინი ან მრავასაფეხურიანი კომუტაციის მქონე სისტემებში საჭიროა ძვირადღირებული სისტემების გამოყენება, და თანაც ასეთ სისტემებში პროცესორების რაოდენობა არ არის ძალიან დიდი. იმისათვის, რომ ერთ მულტიპროცესორში გაერთიანე-

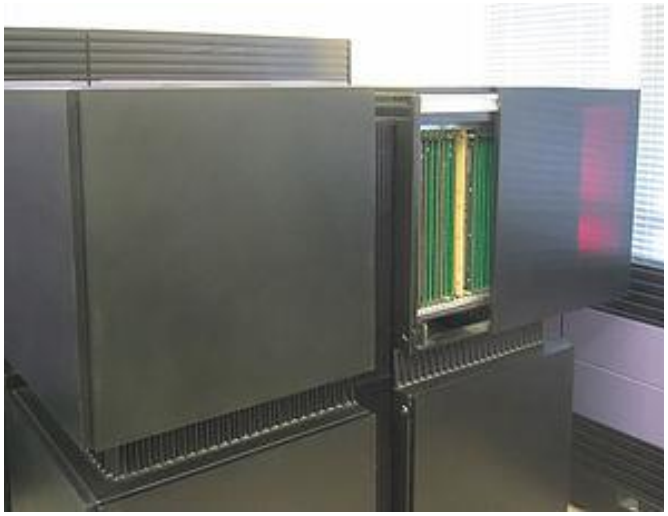
ბული იქნას 100 პროცესორზე მეტი, საჭიროა სხვა სქემის გამოყენება. ადრე მიღებული იყო, რომ მეხსიერების ყოველ მოდულთან წვდომის დრო ერთიდაიგივე იყო. ამ მიდგომას დოგმად თუ არ მივიღებთ, მაშინ შეიძლება შექმნილი იქნას **მულტიპროცესორები მეხსიერებასთან არაერთგვაროვანი წვდომით** (Non Uniform Memory Access, **NUMA**). Uma–მულტიპროცესორების მსგავსად NUMA–მულტიპროცესორებიც გამოიყენებენ მეხსიერების საერთო სივრცეს ყველა პროცესორისათვის, მაგრამ UMA–პროცესორებისაგან განსხვავებით მეხსიერების ლოკალურ მოდულებთან წვდომა ხორციელდება უფრო სწრაფად, ვიდრე მეხსიერების დაშორებულ მოდულებთან. ამგვარად, UMA–პროგრამებს ცვლილებების გარეშე შეუძლიათ NUMA–კომპიუტერებთან მუშაობა, მაგრამ იგივე ტაქტური სიხშირის მქონე NUMA–კომიუტერებთან შედარებით მწამოებლურობა იქნება ნაკლები.

NUMA–კომპიუტერებს გააჩნიათ სამი ძირითად თვისება, რომლებიც მათ არსებითად განასხვავებს სხვა მულტიპროცესორებისაგან:

- არსებობს ერთიანი სამისამართო სივრცე, რომელიც ხილულია ყველა პროცესორისათვის;
- მეხსიერების დაშორებულ მოდულებთან წვდომა ხორციელდება LOAD და STORE ბრძანებების საშუალებით;
- მეხსიერეის დაშორებულ მოდულებთან წვდომას სჭირდება მეხსიერების ლოკალურ მოდულებთან წვდომისათვის საჭირო დროზე მეტი დრო.

სისტემას, რომელშიც მეხსიერების დაშორებულ მოდულებთან წვდომის დრო არ არის მასკირებული კეშირებით (კეშ–მეხსიერება არ არსებობს), უწოდებენ NC-NUMA (No Caching NUMA – NUMA კეშირების გარეშე) სისტემას. პროგრამისტები ასეთ სისტემას ხშირად უწოდებენ **სისტემებს აპარატულად განაწილებული საერთო მეხსიერებით**, რადგანაც ეს სისტემა არსებითად პროგრამულად რეალიზებული განაწილებული საერთო მეხსიერების სისტემის (DSM) ანალოგიურია, მაგრამ ხდება მისი აპარატული მხარდაჭერა მცირე ზომის ფურცლებით.

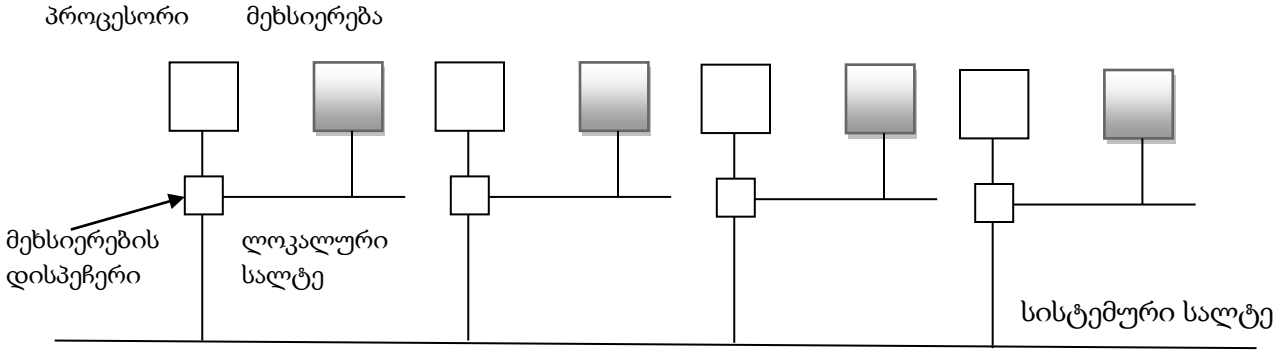
NC-NUMA-კომპიუტერების ერთერთი პირველი წარმომადგენელია Carnegi-Mellon-ის მიერ წარმოებული მულტიპროცესორი Cm*. მისი სურათი ნაჩვენებია ნახ 2.2.6-ზე, ხოლო გამარტივებული ფუნქციონალური სქემა ნაჩვენებია ნახ. 2.2.7–ზე. ეს მულტიპროცესორი შედგება LSI-11 პროცესორების ნაკრებისაგან. ამ პროცესორებიდან თითოეულს გააჩნია საკუთარი მეხსიერება, რომელთანაც მიმართვა ხდება ლოკალური სალტის საშუალებით. (LSI-11 ერთპროცესორიანი ვარიანტი ძალიან პოპულარული იყო გასული საუკუნის 70–იან წლებში. ის ცნობილი იყო DEC-ის მინიკომპიუტერ PDP-11–ის სახელით). LSI-11 პროცესორები ერთმანეთთან დაკავშირებული იყო სისტემური სალტის საშუალებით. მეხსიერებასთან მიმართვის



ნახ. 2.2.6. სუპერკომპიუტერი Cm1

დროს შეკვეთა ჯერ ხვდებოდა მეხსიერების დისპეჩერთან, რომელიც ამოწმებდა არსებობს თუ არა საჭირო სიტყვა ლოკალურ მეხსიერებაში. სიტყვა თუ ლოკალურ მეხსიერებაში არსებობს, მაშინ ლოკალური სალტით ხდება ლოკალურ მეხსიერებასთან მიმართვა, ხოლო თუ არ არსებობს, მაშინ სისტემური სალტის საშუალებით შეკვეთა გადაეცემა ამ სიტყვის შემცველ სისტემას. სხვა სისტემიდან სიტყვის წამოღებას რასაკვირველია სჭირდებოდა გაცილებით მეტი დრო. დაშორებულ მეხსიერებაში შენახული პროგრამის შესრულებას სჭირდებოდა 10–ჯერ მეტი დრო ლოკალურ მეხსიერებაში არსებულ პროგრამასთან შედარებით.

NC-NUMA-კომპიუტერში მეხსიერების შეთანხმებულობა გარანტირებულია, რადგანაც კეშ–მეხსიერება არა გამოყენებული. მეხსიერების ყოველი სიტყვა შეიძლება შენახული იყოს მხოლოდ ერთ ადგილზე, ამიტომ ასლების გაჩენის არანაირი საშიშროება არ არსებობს. ასეთ სისტემაში დიდი მნიშვნელობა ენიჭება იმის ცოდნას, თუ რომელი მონაცემი რომელ მეხსიერებაში



ნახ. 2.2.7. NUMA-კომპიუტერი სალტეების ორი დონით

არის განლაგებული. მწარმოებლურობის მაქსიმალურად გასაზრდელად NC-NUMA-კომპიუტერებში რეალიზებული იქნა ფურცლების გადაადგილებაზე თვალყურის მიდევნების შემდეგი სქემა.

რამდენინე წამში ერთხელ გაიშვებოდა სპეციალური „სადარაჯო“ პროცესი – **ფურცლების სკანერი**. მისი დანიშნულებაა თვალყური მიადევნოს ფურცლების გამოყენების სტატისტიკას და გადაადგილოს ისინი ისეთნაირად, რომ გაიზარდოს მწარმოებლურობა. ფურცელი თუ აღმოჩნდებოდა „არასწორ“ ადგილზე, მაშინ ფურცლების სკანერი ამოშლიდა ასეთ ფურცელს მეხსიერებიდან. შემდეგში, როდესაც მოხდებოდა ამ ფურცელთან მიმართვა, დაფიქსირდებოდა მეხსიერების შეცდომა – ფურცლის არარსებობა. ამ შემთხვევაში გამოჩნდებოდა თუ სად იყო საჭირო ამ ფურცლის ჩატვირთვა (სავარაუდოდ მეხსიერების სხვა მოდულში). ფუჭი მოქმედებების თავიდან აცილების მიზნით მიღებული იყო, რომ ფურცლის გადაადგილების შემდეგ, ის ახალ ადგილზე უნდა დარჩენილიყო ΔT დროის განმავლობაში. შემოთავაზებული იყო კიდევ სხვა ალგორითმებიც, მაგრამ საუკეთესო შედეგს ვერცერთი ვერ იძლეოდა.

2.2.7. CC-NUMA–მულტიპროცესორები

ნახ. 2.2.7–ზე გამოსახული ტიპის მულტიპროცესორებში, კეშ–მეხსიერეის არარსებობის გამო, რთულ პრობლემას წარმოადგენდა მასშტაბირება. ყოველთვის, როდესაც კი საჭირო სიტყვა არ აღმოჩნდებოდა ლოკალურ მეხსიერებაში საჭირო ხდებოდა დაშორებულ მეხსიერებასთან მიმართვა, ეს კი იწვევდა დიდ დროით დანაკარგებს და ამცირებდა მულტიპროცესორის მწარმოებ-

ლურობას. კეშ მეხსიერების დამატებით კი წარმოიქმნება შეთანხმებულობის პრობლემა. კეშ–მეხსიერებების შეთანხმებულობის მიღწევის ერთერთ ხერხს წარმოადგენს სისტემური სალტისათვის თვალყურის მიდევნება. ტექნიკურად ამის განხორციელება რთული არაა, მაგრამ პროცესორების რაღაც რაოდენობის შემდეგ ამის განხორციელება შეუძლებელი ხდება. დიდი რაოდენობით პროცესორების შემცველი მულტიპროცესორების დასამზადებლად საჭიროა სხვა მიდგომა.

დღეისათვის CC-NUMA ტიპის დიდი ზომის მულტიპროცესორების შესაქმნელად ყველაზე პოპულარულ მიდგომას წარმოადგენს მულტიპროცესორი კატალოგის საფუძველზე. ამ მეთოდის საფუძველს წარმოადგენს მონაცემთა ბაზა, რომელიც მოიცავს ინფორმაციას იმის შესხებ, თუ კეშ–მეხსიერების თითოეული ფურცელი სად არის განთავსებული და როგორია მისი მდგომარეობა. კეშ–მეხსიერების სტრიქონთან მიმართვის დროს ბაზაში იგზავნება შეკვეთა, რომლის მიხედვითაც დადგენილი უნდა იქნას სად ინახება ეს სტრიქონი და ის არის „სუფთა“ თუ „ჭუჭყიანი“ (მოდიფიცირებული). მონაცემთა ბაზასთან მიმართვა საჭირო ხდება მეხსიერებასთან მიმართვის ყოველი ბრძანების შესრულებისას, ამიტომ მონაცემთა ბაზას სჭირდება მაღალსიჩქარიანი სპეციალიზებული აპარატული მხარდაჭერა.

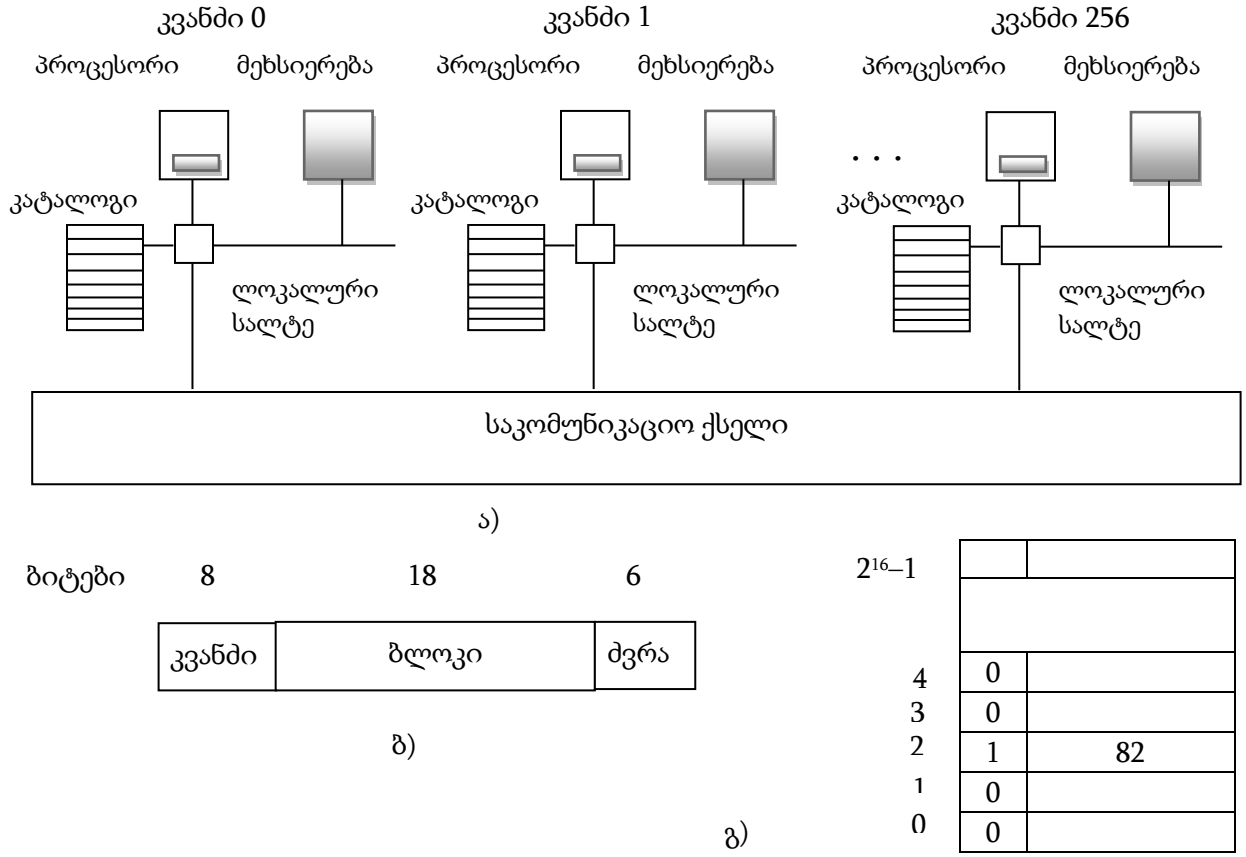
იმისათვის, რომ უკეთ გავარკვიოთ თუ როგორ მუშაობს მულტიპროცესორი კატალოგის საფუძველზე, გავარჩიოთ 256 კვანძის შემცველი სისტემის მუშაობა. თითოეული კვანძი შედგება ერთი პროცესორისაგან და 16 მეგაბაიტი მოცულობის მქონე ოპერატიული მეხსიერებისაგან. პროცესორი და ოპერატიული მეხსიერება ერთმანეთთან დაკაშირებულია ლოკალური სალტის საშუალებით. მეხსიერების საერთო მოცულობაა 2^{32} ბაიტი. ის დაყოფილია 64 ბაიტი სიგრძის მქონე 2^{26} კეშ–სტრიქონად. მეხსიერება სტატიკურად განაწილებულია კვანძებს შორის: მისამართები 0–16M განლაგებული კვანძში 0, მისამართები 16-32M – კვანძში 1 და ა.შ. კვანძები ერთმანეთთან დაკავშირებულია საკომუნიკაციო ქსელის საშუალებით (ნახ. 2.2.7, ა). საკომუნიკაციო ქსელი შეიძლება რეალიზებული იყოს მესერის, ჰიპერკუბის ან რომელიმე სხვა ტოპოლოგიით. ყოველ კვანძს გააჩნია კატალოგის ელემენტი 64–ბაიტიანი სტრიქონების 2^{18} ჩანაწერისათვის, რაც საბოლოო ჯამში იძლევა 2^{24} ბაიტ მეხსიერებას. სიმარტივისათვის დავუშვათ, რომ სტრიქონი შეიძლება არსებობდეს მხოლოდ ერთ კეშ–მეხსიერებაში.

კატალოგთან მუშაობის წესების გასარკვევად განვიხილოთ კონკრეტული ბრძანების შესრულება. 20–ე პროცესორმა გასცა ბრძანება LOAD. პირველ რიგში ბრძანების გამცემი პროცესორი გადასცემს მას მეხსიერების დისპეჩერს, რომელიც ჯერ ახდენს ბრძანების ტრანსლირებას, რათა მიიღოს ფიზიკური მისამართი, მაგალითად 0x24000108. მეხსიერების დისპეჩერი ამ ბრძანებას გაყოფს სამ ნაწილად, როგორც ეს ნაჩვენებია ნახ. 2.2.7, ბ–ზე. თვლის ათობით სისტემაში ეს ნაწილები წარმოადგენს კვანძი 36, სტრიქონი 4 და ძვრა 8. მეხსიერების დიპეჩერი ხედავს, რომ მეხსიერების სიტყვა, რომელთანაც კეთდება მიმართვა, იმყოფება არა 20–ე კვანძში, არამედ 36–ში, ამიტომ საკომუნიკაციო ქსელით აგზავნის შეკვეთას 36–ე კვანძში, სადაც იმყოფება საჭირო სტრიქონი, დაადგენს არის თუ არა ეს სტრიქონი კეშ–მეხსიერებაში და თუ არა, მაშინ სად არის.

როდესაც შეკვეთა მივა 36–ე კვანძში, ის გაიგზავნება კატალოგის მოწყობილობაში. ეს მოწყობილობა ამოწმებს კატალოგის 2^{18} ელემენტს (ერთი ელემენტი კეშ–ის ყოველ სტრიქონზე) და ამოიიღებს იქედან 4–ე ელემენტს. ნახ. 2.2.8, გ–ზე ჩანს, რომ კეშ–მეხსიერებაში ეს სტრიქონი არ არის, ამიტომ მოწყობილობა გამოიძახებს 4–ე სტრიქონს ლოკალური მეხსიერებიდან, გაუგზავნის

20–ე კვანძს და განაახლებს კატალოგის 4–ე ელემენტს და უჩვენებს, რომ ეს სტრიქონი იმყოფება 20–ე კვანძის კეშ–მეხსიერებაში.

ეხლა განვიხილოთ მეორე შეკვეთა. ახლა შეკვეთა კეთდება 36–ე კვანძის 2–ე სტრიქონზე. ნახ. 2.2.8, გ–ზე ჩანს, რომ კეშ–მეხსიერებაში ეს სტრიქონი არსებობს და შეიცავს მონაცემს 82. ამ მომენტში მოწყობილობამ შეიძლება განაახლოს კატალოგის 2–ე სტრიქონი და აჩვენოს, რომ სტრიქონი ამჟამად იმყოფება 20–ე კვანძში, ხოლო თავისი მონაცემი გამოაცხადოს არასწორად. უნდა აღინიშნოს, რომ საერთო მეხსიერების მქონე მულტიპლექსორებში ყველგან საჭიროა დიდი რაოდენობით შეტყობინებების გადაცემა.



ნახ. 2.2.8. 256 კვანძის შემცველი მულტიპროცესორი კატალოგის საფუძველზე (ა); 32–თანრიგიანი მისამართის ველებად დაშლა (ბ); კატალოგი 36–ე კვანძში (გ)

განვიხილოთ რა მოცულობას იკავებენ კატალოგები. ყოველ კვანძს გააჩნია ოპერატიული მეხსიერება მოცულობით 16 Mბაიტი და 2^{18} ცალი 9–თანრიგიანი ელემენტი ამ მეხსიერების ტრასირებისათვის. ამგვარად 16 Mბაიტი ოპერატიული მეხსიერებიდან კატალოგს მიაქვს 9×2^{18} ბიტი, რაც წარმოადგენს მთელი მოცულობის 1,76% და ეს სავსებით მისაღებია. კეშ–მეხსიერების სტრიქონის სიგრძე რომ ყოფილიყო 32 ბაიტი, მაშინ კატალოგზე მეხსიერების დანაკარგი იქნებოდა 4%–ზე ნაკლები, ხოლო 128 ბაიტიანი სტრიქონებისათვის დანაკარგი 1%–ზე ნაკლები იქნებოდა.

ამ სქემის აშკარა ნაკლს წარმოადგენს ის, რომ სტრიქონი შეიძლება კეშირებული იქნას მხოლოდ ერთი კვანძის მიერ. იმისათვის, რომ სტრიქონების კეშირება მოხდეს რამოდენიმე კვანძში, საჭიროა მათი მოძიების რაიმე სხვა ხერხი. არსებობს რამდენიმე ასეთი ვარიანტი.

ერთი ვარიანტი შეიძლება იყოს ასეთი – სხვა კვანძების იდენტიფიცირებისათვის კატალოგის ყოველ ელემენტს მივანიჭოთ k ველი, რაც მოგვცემს იმის საშუალებას, რომ მოვახდინოთ თითოეული სტრიქონის კეშირება რამდენიმე კვანძში (k –მდე). **მეორე ვარიანტია** – კვანძიში ნომრის ველი შეიცვალოს ბიტური ნიშნით, თითო ბიტი თითო კვანძისათვის. ამ შემთხვევაში ასლების რაოდენობაზე შეზღუდვა არ არის, მაგრამ მნიშვნელოვნად იზრდება დამხმარე დანახარჯები.კატალოგმა, რომელიც შეიცავს 256 ბიტს კეშის 64 ბაიტიან სტრიქონზე, შეიძლება დაიჭიროს მეხსიერების 50%–მდე. **მესამე ვარიანტი** – კატალოგის ყოველ ელემენტში ხდება 8–თანრიგიანი ველის შენახვა, რომელიც შემდეგ გამოყენებული იქნება ბმული სიის სათაურად. ამ ბმულ სიაში გაერთიანებული იქნება კეშ–სტრიქონის ყველა ასლი. ასეთი სტრატეგიის რეალიზებისათვის კოველი კვანძის მეხსიერებაში საჭიროა დამატებითი სივრცე ბმული სიის მიმთითებლებისათვის. გარდა ამისა საჭიროა კიდევ ამ სიის გადასინჯვა, რათა საჭიროების შემთხვევაში მონახული იქნას ყოველი ასლი. განხილული სტრატეგიებიდან თითოეულს აქვს დადებითი და უარყოფითი თვისებები. ამიტომაც, პრაქტიკაში სამივე გამოიყენება.

მოცემული სქემის კიდევ ერთი პრობლემა – როგორ შევამოწმოთ საწყისი მეხსიერება განახლებულია თუ არა? თუ საჭიროა კეშ მეხსიერების სტრიქონის წაკითხვა, რომელშიც ცვლილება არ მომხდარა, შეკვეთა შეიძლება დაკმაყოფილებული იქნას ძირითადი მეხსიერებიდან და ამ დროს არ იქნება საჭირო კეშ–მეხსიერებაში შეკვეთის გაგზავნა. თუკი საჭირო იქნება კეშ–მეხსიერებიდან შეცვლილი სტრიქონის წაკითხვა, მაშინშეკვეთა უნდა გაიგზავნოს იმ კვანძში, რომელშიც იმყოფება კეშის მოცემული სტრიქონი, რადგანაც მხოლოდ აქ იმყოფება მისი ნამდვილი ასლი. თუკი ნებადართულია კეშ–მეხსიერების სტრიქონის მხოლოდ ერთი ასლის შენახვა, როგორც ეს ნაჩვენებია ნახ. 2.2.8–ზე, მაშინ კეშ–მეხსიერების სტრიქონების ცვლილების კონტროლს აზრი არა აქვს, რადგანაც ყოველი შეკვეთა მოითხოვს შეტყობინების გადაცემას არსებული ასლის შესახებ, რომ გამოაცხადოს ის არასწორად.

ცხადია, კეშ–მეხსიერების ყოველი სტრიქონის შესახებ ინფორმაციის შესანახად საჭიროა საწყისი კვანძისათვის ინფორმაციის მიწოდება კეშ–მეხსიერების ყოველ სტრიქონში ცვლილებების შეტანის შესახებ, იმ შემთხვევაშიც კი თუ სისტემაში არსებობს ამ სტრიქონის მხოლოდ ერთი ასლი. თუკი ასეთი ასლი რამდენიმეა, მაშინ ერთერთის ცვლილება აუცილებელს ხდის ყველა დანარჩენი ასლის არასწორად გამოცხადებას, ამიტომ საჭიროა რაიმე განაწესი, რომელიც თავიდან აგვაცილებს ასეთ ქმედებებს. მაგალითად, კეშ–მეხსიერების ერთობლივად გამოსაყენებლად სტრიქონში ცვლილების შესატანად ამ სტრიქონის ერთერთ მფლობელს სტრიქონში ცვლილების შეტანმდე შეუძლია მოითხოვოს მასთან მონოპოლიური წვდომა. ასეთი მოთხოვნა აღნიშნავდა დანარჩენი ასლების არასწორად გამოცხდებას.

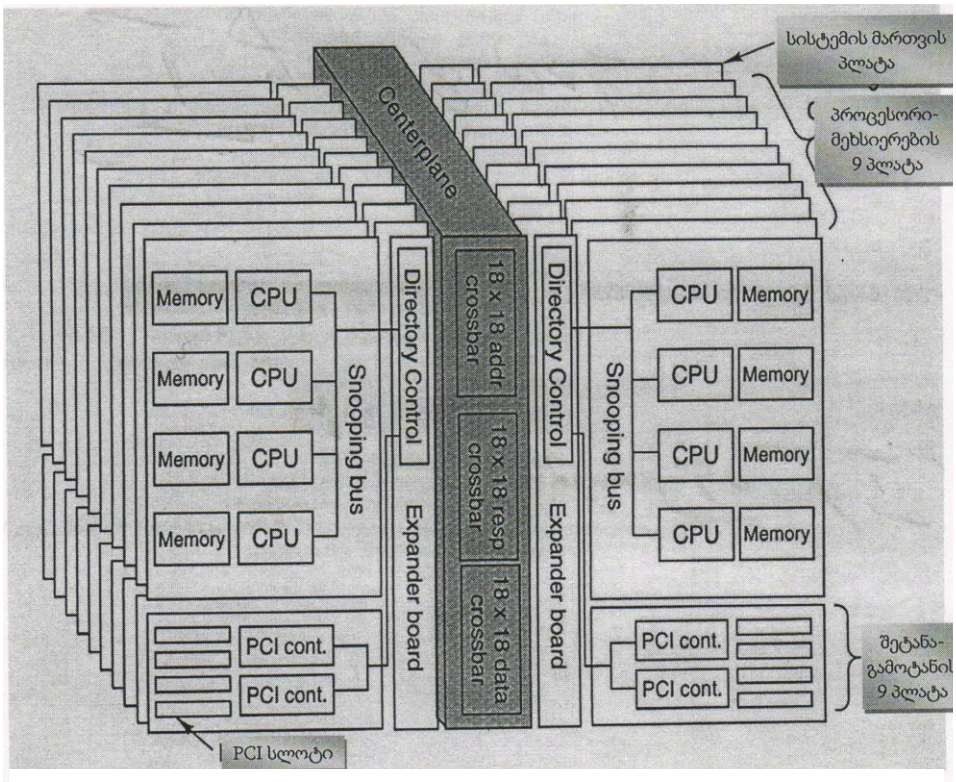
2.2.8. NUMA-მულტიპროცესორი Sun Fire E25K

საერთო მეხსიერეის მქონე NUMA–კომპიუტერების მაგალითად განვიხილოთ Sun Microsystems კომპანიის მულტიპროცესორების ოჯახი Sun Fire. მულტიპროცესორების ეს ოჯახი

აერთიანებს სხვადასხვა მოდელს, რომელთგანაც განხილული იქნება მულტიპროცესორი E25K, რომელიც მოიცავს 74 პროცესორს UltraSPARC IV. მისი სურათი ნაჩვენების ნახ. 2.2.9-ზე. ამ პროცესორებიდან თითოეული წარმოადგენს UltraSPARC III Cu პროცესორების წყვილს საერთო კეშ და ოპერატიული მეხსიერებით. სისტემა E15K განსხვავდება მხოლოდ იმით, რომ გაორმაგებულის ნაცვლად გამოიყენება თითოთითო პროცესორი. მულტიპროცესორების ოჯახში შედის უფრო მარტივი კომპიუტრებიც, მაგრამ ჩვენთვის ინტერეს იწვევს უფრო რთული მოდელები, რათა განვიხილოთ თუ როგორ იმართება პროცესორების მაქსიმალური რაოდენობის შემცველი მულტიპროცესორების მუშაობა

ნახ. 2.2.9 მულტიპროცესორი E25K

სისტემა E25K მოიცავს პლატების 18 ნაკრებს. თითოეული ნაკრები შედგება პროცესორი–მეხსიერება, PCI ოთხი სლოტის შემცველი შეტანა–გამოტანის და გაფართოების პლატებისაგან. გაფართოების პლატა წყვილ–წყვილად აერთიანებს პროცესორ–მეხსიერების და შეტანა–გამოტანის პლატებს და უკავშირებს მათ ცენტრალურ პანელს, რომელსაც ასევე უკავშირდება სისტემის ყველა პლატა და ახორციელებს მათ კომუტაციას. პროცესორ–მეხსიერების ყოველ პლატაზე მოთავსებულია 4 პროცესორი და ოპერატიული მეხსიერების 4 მოდული, თითოეული მოცულობით 4 გიგაბაიტი. საერთო ჯამში E25K სისტემაში არის 144 პროცესორი, ოპერატიული მეხსიერება საერთო მოცულობით 576 –გიგაბაიტი და 72 PCI–სლოტი. სისტემის სტრუქტურა ნაჩვენებია ნახ. 2.2.10–ზე. E25K სისტემაში გათვალისწინებულია პროცესორის პლატების 18 ნაკრები. ეს რიცხვი შერჩეული იქნა გაბარიტების მოთხოვნის გათვალისწინებით. ამ ზომის მოდული თავისუფლად ეტევა სტანდარტული ზომის კარებში.



ნახ. 2.2.10. NUMA-მულტიპროცესორი Sun Fire E25K

პლატების 18 ნაკრების მისაერთებლად ცენტრალურ პანელზე დაყენებულია ჯვარედინი კომუტაციის სამი სქემა, თითოეული ზომით 18×18. თითო თითო სქემა სამისამართო ხაზებისათვის, პასუხებისა და მონაცემებისათვის. ყველაფერი ამის გარდა ცენტრალურ პანელში ირთვება

კიდევ სისტემის მართვის პლატა, რომელიც პროცესორის გარდა შეიცავს კიდევ ინტერფეისებს დისკური დამგროვებლების, მაგნიტურ ლენტებზე დამგროვებლების და სხვა პერიფერიული მოწყობილობების მისაერთებლად, რომელთა საშუალებითაც უნდა მოხდეს სისტემის ჩატვირთვა და სხვადასვა მონაცემების დაგროვება.

მეხსიერების ქვესისტემა ნებისმიერი მულტიპროცესორის მნიშვნელოვანი ნაწილია. როგორ არის სისტემაში განაწილებული მეხსიერება მიერთებული 144 პროცესორთან? შეიძლება გამოყენებული იქნას ფართე საერთო სალტე ან ჯვარედინი კომუტაციის სქემა განზომილებით 144×72. არცერთი ეს ვარიანტი მისაღები არაა. საერთო სალტე მისაღები არაა იმიტომ, რომ საერთო სალტე სისტემისათვის გახდება ვიწრო ყელი, ხოლო ჯვარედინი კომუტაციის სქემა ასეთი ზომებისათვის იქნება ძალიან რთული და ძვირი. ამიტომ, E25K სისტემის მსგავს დიდ მულტიპროცესორებში საჭირო ხდება მეხსიერების არატრივიალური ქვესისტემების გამოყენება.

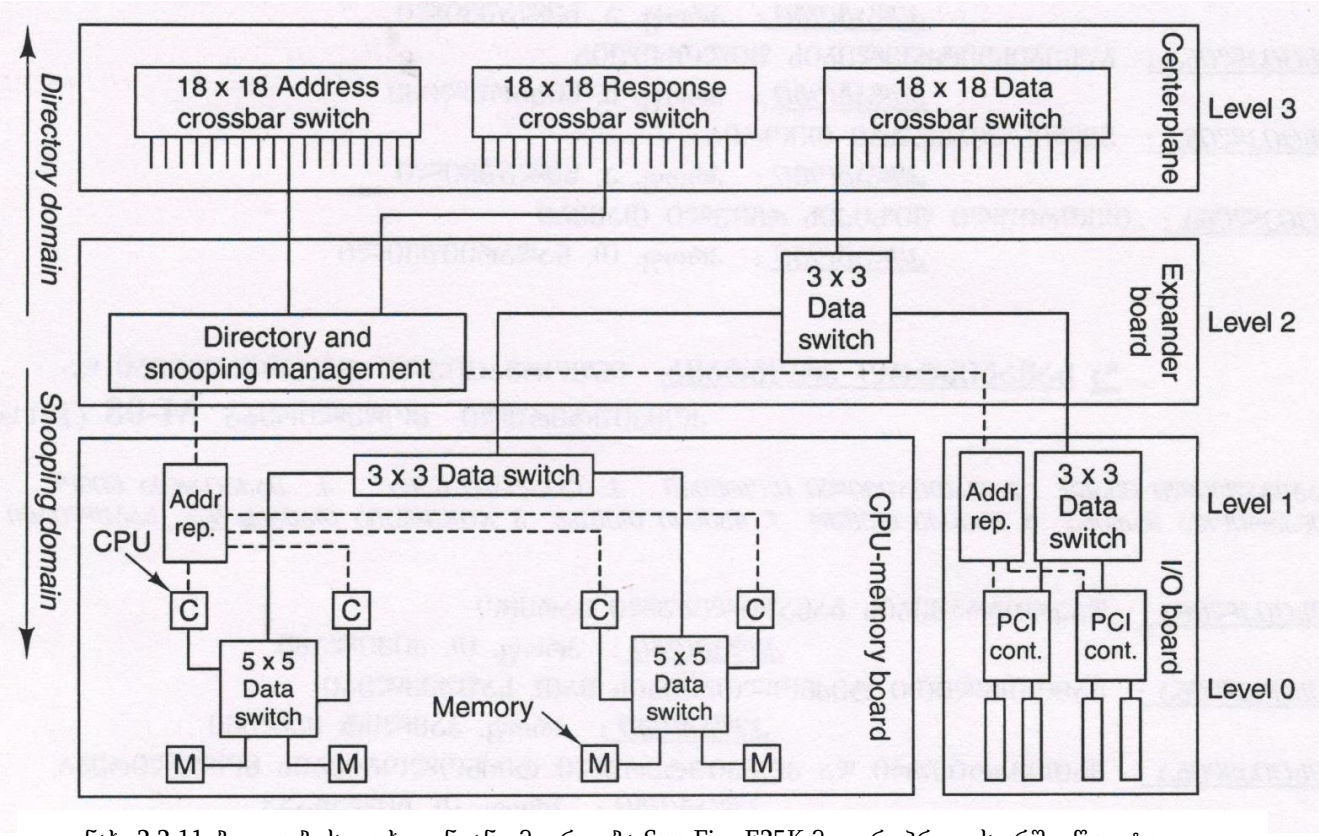
პლატების ნაკრებების დონეზე გამოყენებულია თვალყურის მიდევნების ლოგიკა, რომლის საშუალებითაც ხდება შემოსული შეკვეთების შედარება საკუთარ ლოკალურ კეშში ბლოკების სიასთან. როდესაც პროცესორი მიმართავს სიტყვას მეხსიერებაში, ის ჯერ ვირტუალურ მისამართს გარდაქმნის ფიზიკურ მისამართად და ამოწმებს არის თუ არა საჭირო ბლოკი კეშში. (ფიზიკური მისამართები 43–თანრიგიანია, მაგრამ გაბარიტული მოსაზრებების გამო სისტემაში გამოყენებულია მეხსიერება მოცულობით 576 Gბაიტი). საჭირო ბლოკი თუ აღმოჩენილი იქნება საკუთარ კეშში, მაშინ მოთხოვნილი სიტყვა პირდაპირ მიიღება. წინააღმდეგ შემთხვევაში თვალყურის მიდევნების ლოგიკა ამოწმებს არის თუ არა მოთხოვნილი სიტყვა საკუთარ ნაკრებში. თუ არის, მაშინ ხდება შეკვეთის შესრულება. წინააღმდეგ შემთხვევაში, შეკვეთა გადაიგზავნება ჯვარედინი კომუტაციის სამისამართო სალტეების საშუალებით. თვალყურის მიდევნების ლოგიკას შეუძლია მოემსახუროს ტაქტში ერთ მოთხოვნას. რადგანაც სისტემის ტაქტური სიხშირე 150 მეგაჰერცია, ამიტომ შესაძლებელია წამში 150 მილიონი შეკვეთის მომსახურება, ანუ მთელი სისტემის მასშტაბით ეს იქნება 2,7 მილიარდი შეკვეთა წამში.

ნახ. 2.2.10–ზე თვალყურის მიდევნების ლოგიკა ნაჩვენებია სალტის სახით, მაგრამ სინამდვილეში ის წარმოადგენს მოწყობილობების ხის მაგვარ სტრუქტურას, რომელზეც მონაცემები გადაეცემა როგორც ზევიდან ქვევით, ასევე ქვევიდან ზევითაც. როდესაც პროცესორიდან ან PCI სალტიდან შემოდის მისამართი, ის ჯერ გადაეცემა მისამართის გამმეორებელს, როგორც ეს ნაჩვენებია ნახ. 2.2.11–ზე. მისამართის გამმეორებელი ორია. ნებისმიერი მათგანიდან მისამართი მოხვდება გაფართოების პლატაზე, საიდანაც გადაეცემა ხეზე ქვევით, მოწყობილობებისაკენ. ამ სქემის საშუალებით შესაძლებელია სალტის გამოყენებისაგან თავის არიდება.

მონაცემების გაცვლისათვის გამოიყენება ოთხდონიანი მიერთება (ნახ. 2.2.11). ასეთი მიდგომა უზრუნველყოფს მაღალ მწარმოებლურობას. 0 დონეზე პროცესორების და მეხსიერების პლატების წყვილი ერთმანეთთან მიერთებულია ჯვარედინი კომუტაციის პატარა სქემებით, რომლებიც ასევე უკავშირდებიან დონეს 1. პროცესორ მეხსიერების პლატების მეორე წყვილი დონეზე 1 ერთმანეთს უკავშირდება ჯვარედინი კომუტაციის მეორე სქემით. ეს კომუტატორები შესრულებულია სპეციალიზებული ინტეგრალური სქემების სახით. მათ აქვთ შესასვლელები საკომუტაციო ბადის ყოველი სტრიქონისა და სვეტისათვის, თუმცა კი სტრიქონებისა და სვეტების

ზოგიერთი კომბინაცია არ გამოიყენება. პლატების კომუტაციის მთელი ლოგიკა აგებულია ჯვარედინი კომუტაციის სქემებზე ზომით 3×3.

პლატებეის ყოველი ნაკრები შედგება სამი პლატისაგან: პროცესორ–მეხსიერების პლატის, შეტანა–გამოტანის პლატის და გაფართოების პლატისაგან. 2–ე დონის კომუტატორი მოთავსებულია გაფართოების პლატაზე და მეხსიერებას აკავშირებს შეტანა–გამოტანის პორტებთან. პლატების ნაკრებში შემავალი ან გამომავალი ყველა მონაცემი გაივლის 2–ე დონის კომუტატორს. და ბოლოს, სხვადასხვა პლატებს შორის მონაცემების გაცვლა ხდება 3–ე დონეზე ჯვარედინი კომუტაციის სქემის საშუალებით, რომელსაც გააჩნია გაწზომილება 18×18, მონაცემების გადაცემა ხდება 32 ბაიტიაბიანი ბლოკების სახით, ანუ სტანდარტული 64 ბაიტიანი ბლოკის გადასაცემად საჭიროა 2 ტაქტი.



ნახ. 2.2.11. ბლოკების ოთხდონიანი მიერთება Sun Fire E25K მულრიპროცესორში. წყვეტილი ხაზებით აღნიშნულია მისამართების გადაცემა, ხოლო უწყვეტი ხაზებით კი მონაცემების გადაცემა

ეხლა განვიხილოთ, როგორაა განლაგებული მისი კომპონენტები, განვიხილოთ საერთო მეხსიერება. 576 Gბაიტის მოცულობის სამისამართო სივრცე ყველაზე დაბალ დონეზე დაიყოფა 64 ბაიტიან ბლოკებად. მათი რაოდენობა არის 2^{29}. ეს ბლოკები არის მეხსიერების გაუყოფელი ნაწილები. ყოველ ბლოკს აქვს თავისი „მშობლიური“ პლატა, სადაც ის „ცხოვრობს“, მანამდე, სანამ სხვა პლატიდან არ მოითხოვენ. ბლოკების უმეტესობა დროის უმეტეს ნაწილს ატარებენ თავის პლატაზე. როდესაც პროცესორს დასჭირდება რომელიმე ბლოკი, იქნება ეს თავისი პლატიდან თუ რომელიმე ნებისმიერი სხვა პლატიდან, ის ჯერ მოითხოვს ასლს თავის კეშ–მეხსიერებაში, შემდეგ კი მუშაობს კეშირებულ ასლთან. E25K სისტემის ყოველ მიკროსქემაში მოთავსებულია 2 პროცესორი. მათ გააჩნიათ საერთო სამისამართო სივრცე და შესაბამისად საერთო კეშ–ბლოკი.

მეხსიერების ყოველი ბლოკი (და ყველა მიკროსქემის კეშ–სტრიქონი) შეიძლება იმყოფებოდეს ჩამოთვლილი სამიდან მხოლოდ ერთ მდგომარეობაში:

- ექსკლუზიური წვდომა (ჩაწერისათვის);
- ერთობლივი წვდომა (წაკითხვისათვის);
- არასწორი (ანუ ცარიელი).

როდესაც პროცესორს სჭირდება მეხსიერებაში სიტყვის ჩაწერა ან მეხსიერებიდან სიტყვის წაკითხვა, ის უპირველეს ყოვლისა ამოწმებს თავის კეშ–მეხსიერებას. ბლოკი იქ თუ არ აღმოჩნდა, მაშინ იქმნება ბლოკის ფიზიკური მისამართის შეკვეთა, რომელიც საყოველთაოდ ვრცელდება პლატების თავისი ნაკრების ფარგლებში. საჭირო ბლოკი თუ აღმოჩენილი იქნა პლატების თავისი ნაკრების კეშ–მეხსიერებაში, მაშინ თვალყურის მიდევნების ლოგიკა დაადგენს კეშ–მოხვედრების ფაქტს და პასუხობს შეკვეთას. სტრიქონი თუ იმყოფება ექსკლუზიური წვდომის მდგომარეობაში, მაშინ ის გადაეცემა მის შემკვეთ პროცესორს და საწყისი ასლი მოინიშნება როგორც არასწორი. სტრიქონი თუ იმყოფება ერთობლივი წვდომის მდგომარეობაში, მაშინ ის არ გადაეცემა მის შემკვეთ პროცესორს, რადგანაც მეხსიერება თვითონ უგზავნის პასუხს, როდესაც მოხდება კეშ–სტრიქონის გასუფთავება.

თვალყურის მიდევნების ლოგიკა თუ ვერ აღმოაჩენს კეშ–სტრიქონს ან აღმოჩენილი სტრიქონი თუ იმყოფება ერთობლივი წვდომის მდგომარეობაში, მაშინ ცენტრალური პანელის გავლით „მშობლიურ“ პლატას ეგზავნება შეკვეთა, რომლითაც უნდა მოხდეს ბლოკის ადგილმდებარეობის დადგენა. ყოველი ბლოკის მდგომარეობა ინახება მის ECC–ბიტებში, ამიტომ, პლატას მაშინვე შეუძლია დაადგინოს მისი მდგომარეობა. ბლოკი თუ არ იმყოფება ერთობლივი წვდომის მდგომარეობაში ან იმყოფება ერთობლივი წვდომის მდგომარეობაში ერთი ან რამდენიმე დაშორებული პლატისათვის, მაშინ „მშობლიურ“ პლატაზე მოხდება მეხსიერების განახლება, ამიტომ „მშობლიური“ პლატა შეძლებს შეკვეთის შესრულებას. ასეთ შემთხვევაში კეშ–სტრიქონის ასლი მონაცემების ჯვარედინი კომუტაციის სქემის საშუალებით გადაეცემა მონაცემების შემკვეთ პროცესორს.

თუ შეიქმნა შეკვეთა წაკითხვაზე, მაშინ ბლოკის „მშობლიური“ პლატის კატალოგში შეიტანება ინფორმაცია, რომ მოცემულ კეშ–სტრიქონს გამოიყენებს კიდევ ერთი კლიენტი (ანუ ის იმყოფება ერთობლივი წვდომის მდგომარეობაში) და ამით ტრანზაქცია მთავრდება. თუკი შეიქმნა შეკვეთა ჩაწერაზე, მაშინ ყველა პლატას, რომელსაც გააჩნია მოცემული ბლოკის ასლი (თუკი ასეთები არსებობს), გადაეცემა შეტყობინება, რომ ბლოკი არასწორია. ამის შედეგად ჩაწერაზე შეკვეთის შემდეგ სისტემაში რჩება ბლოკის მხოლოდ ერთი ასლი.

ეხლა დავუშვათ, რომ ბლოკი იმყოფება დაშორებული პლატისათვის ექსკლუზიური წვდომის მდგომარეობაში. „მშობლიური“ პლატა, როდესაც მიიღებს შეკვეთას, კატალოგის მიხედვით დაადგენს საჭირო დაშორებული პლატის მისამართს და შეკვეთაზე უპასუხებს შეტყობინებით, რომლითაც შეატყობინებს თუ სად იმყოფება კეშ–სტრიქონი. ამის შემდეგ გამგზავნი აღმოჩენილ პლატას უგზავნის ახალ შეტყობინებას. როდესაც ის მიიღებს შეტყობინებას, პასუხად უგზავნის საჭირო კეშ–სტრიქონს. ამის შემდეგ, წაკითხვაზე მოთხოვნის შემთხვევაში სტრიქონი მოინიშნება როგორც ერთობლივი წვდომის მდგომარეობაში მყოფი და ასლი გადაეგზავნება „მშობლიურ“ პლატას. ჩაწერაზე მოთხოვნის შემთხვევაში კი მოპასუხე მხარე თავის ასლს გამოაცხადებს არასწორად, რის შედეგადაც გადამცემი მხარის ასლი გახდება ექსკლუზიური.

რადგანაც ყოველ პლატას გააჩნია მეხსიერების 2^{29} ბლოკი, ამიტომ უარეს შემთხვევაში კატალოგში უნდა არსებიბდეს 2^{29} ჩანაწერი. რეალურად კატალოგის მოცულობა ბევრად ნაკლებია, ამიტომ შეიძლება აღმოჩნდეს, რომ მასში არ არის საკმარისი ადგილი ზოგიერთი ჩანაწერისათვის (კატალოგში ჩანაწერების ძიება ხორციელდება ასოციაციურად). ასეთ შემთხვევაში პლატების „მშობლიურ“ ნაკრებს ბლოკის ადგილმდებარეობის განსაზღვირასათვის დასჭირდება საყოველთაო შეკვეთის გაგზავნა პლატების დანარჩენი 17 ნაკრებისათვის. კატალოგების შეთანხმებულობის უზრუნველყოფის ვალდებულებები და განახლების განაწესის შესრულება დაკისრებული აქვს პასუხების ჯვარედინი კომუტაციის სქემას, რომლის მიერაც ხდება გამგზავნისაკენ მიმართული ტრაფიკის დიდი ნაწილის გადამუშავება. სისტემის გამტარუნარიანობის მაღალ დონეზე შენარჩუნება ხერხდება იმის წყალობით, რომ განაწესის ტრაფიკი გადანაწილებულია ორ სალტეზე (მისამართების და პასუხების) და მონაცემები მესამე სალტეზე.

Sun Fire E25K სისტემას შეუძლია მუშაობა ძალიან მაღალი მწარმოებლურობით, რადგანაც მასში დატვირთვა გადანაწილებულია სხვადასხვა პლატებზე და სხვადასხვა მოწყობილობებზე. ადრე უკვე ნახსენები იყო წამში 2,7 მილიარდი შეკვეთის დამუშავება. ცენტრალურ პანელს შეუძლია უზრუნველყოს მონაცემების ერთდროული გაცვლა 9 პლატა–გადამცემსა და 9 პლატა–მიმღებს შორის. რადგანაც მონაცემების ჯვარედინი კომუტაციის სქემის სიფართე არის 32 ბაიტი, ამიტომ ყოველი ტაქტის განმავლობაში შეიძლება გადაცემული იქნას მონაცემების 288 ბაიტი. ტაქტური სიხშირის მნიშვნელობისათვის 150 Mჰერცი მაქსიმალური გამტარუნარიანობა იქნება 40 Gბაიტი წამში, როდესაც ყველა მიმართვები მიმართულია დაშორებული პლატებისაკენ. თუკი პროგრამულად შესაძლებელი ინება მეხსიერების ფურცლების ისე განლაგება, რომ მიმართვების უმეტესობა იყოს ლოკალური, მაშინ სისტემის მაქსიმალური გამტარუნარიანობა მნიშვნელოვნად გაიზრდება.

2.2.9. COMA–მულტიპროცესორები

NUMA და CC-NUMA–კომპიუტერებს ახასიათებთ ერთი სერიოზული ნაკლი: დაშორებულ მეხსიერებასთან მიმართვები ხორციელდება ბევრად უფრო ნელა ვიდრე ლოკალური მიმართვები. CC-NUMA–კომპიუტერებში ეს განსხვავება გარკვეულწილად ნიველირებულია კეშ–მეხსიერების ხარჯზე. მაგრამ, თუკი მოთხოვნილი დაშორებული მონაცემების მოცულობა აღემატება კეშ–მეხსიერების მოცულობას, ხშირად ექნება ადგილი კეშ–აცილებებს, რაც ცუდად აისახება სისტემის მწარმოებლურობაზე.

UMA–კომპიუტერები ხასიათდება მაღალი მწარმოებლურობით, მაგრამ მათში გამოყენებული პროცესორების რაოდენობა არ არის დიდი და თანაც მათი ფასი მაღალია. NC-NUMA-კომპიუტერებში კარგად ხდება მასშტაბირება, მაგრამ მათში საჭირო ხდება მეხსიერების ფურცლების ხელით ან ნახევრადავტომატური განლაგება, რაც ყოველთვის არ იძლევა კარგ შედეგს. საქმე იმაშია, რომ ძნელია იმის წინასწარმეტყველება, თუ როდის რომელი ფურცელი დაგვჭირდება. გარდა ამისა, ფურცლების გადაადგილება რთულია მათი დიდი ზომის გამო. CC-NUMA-კომპიუტერები, მაგალითად Sun Fire E25K, იწყებენ ნელა მუშაობას თუ პროცესორების დიდ რაოდენობას ერთდროულად დასჭირდება დიდი რაოდენობით დაშორებული მონაცემები. მოკლედ, განხილულ სქემებს გააჩნიათ არსებითი ნაკლი.

არსებობს მულტიპროცესორი, რომელშიც ეს პრობლემები გადაწყვეტილია ყოველი პროცესორის ძირითადი მეხსიერების კეშ–მეხსიერებად გამოყენების ხარჯზე. ასეთ სისტემას ეწოდება **COMA** (Cache Only Memory Access - წვდომა მხოლოდ კეშ–მეხსიერებასთან). მასში მეხსიერების ფურცლებს არ გააჩნიათ საკუთარი "შინაური" პროცესორები, ისე როგორც ეს ხდება NUMA და CC-NUMA სისტემებში. ამ სისტემაში ფურცლები არაა მიმაგრებული რომელიმე პროცესორთან.

ამის მაგივრად, ფიზიკური სამისამართო სივრცე დაყოფილია კეშ–მეხსიერების სტრიქონებად, რომლებიც მოთხოვნის შემთხვევაში თავისუფლად გადაადგილდებიან სისტემაში. მეხსიერების ბლოკებს არ გააჩნიათ საკუთარი პროცესორები. ისინი მომთაბარეობენ, ანუ მათი სახლი არის იქ სადაც აღმოჩნდებიან. მეხსიერებას, რომელიც საჭიროებისამებრ მიიზიდავს სტრიქონებს, უწოდებენ **მიმზიდველს**. ძირითადი მეხსიერების კეშ–მეხსიერებად გამოყენება ზრდის კეშ–მოხვერდებების პროცენტს, ანუ შესაბამისად სისტემის მწარმოებლურობას.

სამწუხაროდ იდეალური არაფერი არ არსებობს. COMA სისტემასთანაც დაკავშირებულია ორი პრობლემა:

- როგორ ხდება კეშ–სტრიქონების განლაგება?
- როგორ მოვიქცეთ მაშინ, თუ მეხსიერებიდან განდევნილი სტრიქონი წარმოადგენს უკანასკნელ ასლს?

პირველი პრობლემა დაკავშირებულია შემდეგ ფაქტთან. როგორც ცნობილია, მეხსიერების დიპეჩერი ახდენს ვირტუალური მისამართის ტრანსლირებას ფიზიკურ მისამართად. ტრანსლირების შემდეგ თუ აღმოჩნდა, რომ სტრიქონი არ არის აპარატულ კეშში, ძალიან რთული ხდება იმის თქმა, საძებნი სტრიქონი საერთოდ არის თუ არა ძირითად მეხსიერებაში. მეხსიერების სტრიქონებად დაყოფის აპარატული მხარდაჭერა აქ ვერ გვიშველის, რადგანაც ყოველი ფურცელი შედგება დიდი რაოდენობით ცალკეული კეშ–სტრიქონებისგან, რომლებიც სისტემაში განლაგებულია ერთმანეთისაგან დამოუკიდებლად. მაშინაც კი თუ ცნობილია, რომ სტრიქონი ძირითად მეხსიერებაში არ არის, როგორ დავადგინოთ მისი მდებარეობა? ამ შემთხვევაში ვერ მოვიძიებთ მის „შინაურ"პროცესორს და ვერ ვკითხავთ მას, რადგანაც ასეთი სისტემაში არ არსებობს.

შემოთავაზებული იყო ამ პრობლემის რამდენიმე გადაწყვეტა. იმისათვის, რომ ვიცოდეთ კეშ–სტრიქონი არის თუ არა ძირითად მეხსიერებაში, საჭიროა ყოველი კეშ–სტრიქონისათვის შექმნილი იქნას სპეციალური აპარატული ტეგი. მაშინ, მეხსიერების დისპეჩერს შეეძლება მოცემული სტრიქონის ტეგის შედარება ყველა დანარჩენი კეშ–სტრიქონების ტეგებთან, ვიდრე ადგილი არ ექნება დამთხვევას.

მეორე გადაწყვეტაა – ასახული იქნას ფურცლები მთლიანად, მაგრამ ამ დროს მოთხოვნილი უნდა იქნას ყველა კეშ–სტრიქონის არსებობა. მაშინ ყოველი ფურცლისათვის საჭირო გახდება აპარატულად აგებული იქნას ბიტური რუქა, რომელშიც ყოველ სტრიქონს შეესაბამება 1 ბიტი, რომელიც მიუთითებს ამ სტრიქონის არსებობას ან არ არსებობას. ამ სქემაში, რომელსაც **უბრალო COMA სქემას** უწოდებენ, სტრიქონი თუ ადგილზეა, მაშინ ის თავის ფურცელშია და სწორ პოზიციაზეა. სტრიქონი თუ ადგილზე არაა, მაშინ ის მოძიებული უნდა იქნას პროგრამულად.

ამგვარად, სისტემა მოძებნის მხოლოდ იმ ფურცლებს, რომლებიც იმყოფება დაშორებულ მეხსიერებაში. კიდევ ერთი გადაწყვეტა – ყოველ ფურცელს მიენიჭოს „საშინაო" პროცესორი (საშინაო იმ გაგებით, რომ კატალოგში მისთვის გაოიყოფა ჩანაწერი, და არა იმ გაგებით, რომ ის იქ

ინახება). მაშინ, იმისათვის რომ დადგინდეს, თუ სად უნდა მოხდეს ამ ფურცლის მოძებნა, უნდა გაეგზავნოს შეტყობინება მის „საშინაო“ პროცესორს. მეორე გადაწყვეტა – ორგანიზებული იქნას მეხსიერება ხის მაგვარი სტრუქტურით და მოხდეს მასზე გადაადგილება ქვევიდან ზევით, ვიდრე არ მიძებნება საჭირო სტრიქონი.

მეორე პრობლემა დაკავშირებულია უკანასკნელი ასლის წაშლასთან. ისევე, როგორც CC-NUMA კომპიუტერში, კეშ–სტრიქონი შეიძლება ერთდროულად იმყოფებოდეს რამდენიმე კვანძში. თუ ადგილი ექნა კეშ–აცილებას, მაშინ უნდა მოხდეს სტრიქონის წაკითხვა, ეს კი მის წაშლას ნიშნავს. რა მოხდება მაშინ, თუ არჩეული სტრიქონი აღმოჩნდება უკანასკნელი ასლი? მაშინ მისი წაშლა არ უნდა მოხდეს.

ერთერთი შესაძლო გადაწყვეტაა – კატალოგთან დაბრუნება და შემოწმება არსებობს თუ არა სხვა ასლები. თუ არსებობს, მაშინ თავისუფლად შეიძლება სტრიქონის წაშლა. თუ არ არსებობს, მაშინ სადმე უნდა მოხდეს მისი განთავსება. მეორე გადაწყვეტა – ყოველი კეშ–სტრიქონის ერთი ასლი განთავსდეს როგორც მთავარი და არასდროს არ მოხდეს მისი წაშლა. ასეთი მიდგომის დროს კატალოგის შემოწმება საჭირო აღარ გახდება. ყოველ შემთხვევაში პოტენციალურად COMA-კომპიუტერს CC-NUMA-კომპიუტერზე უფრო მაღალი მწარმოებლურობა უნდა ჰქონდეს, მაგრამ ჯერჯერობით შექმნილია მხოლოდ რამდენიმე COMA-კომპიუტერი. ამ ტიპის მულტიპროცესორების სრული პოტენციალის რეალიზებისათვის კი საჭიროა გამოცდილების დაგროვება. პირველი COMA-კომპიუტერები იყო KSR-1 და Data Diffusion Machine.

3. მულტიკომპიუტერები

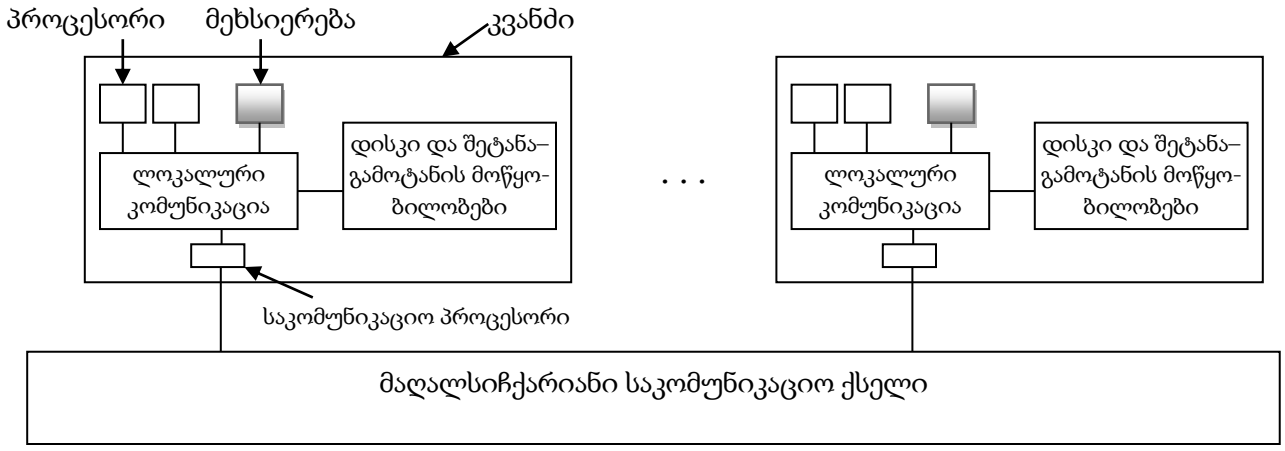
როგორც ნახ. 2.1.4 –ზე ნაჩვენები იყო, MIMD კატეგორიაში შედის პარალელური არქიტექტურის მქონე სისტემების ორი სახეობა: მულტიპროცესორები და მულტიკომპიუტერები. წინა თავში განხილული იყო მულტიპროცესორები. დადგენილი იქნა, რომ მულტიპროცესორებს შეიძლება ჰქონდეს საერთო მეხსიერება, რომელთანაც მიმართვა ხორციელდება ჩვეულებრივი ბრძანებებით LOAD და STORE. ასეთი მეხსიერების ორგანიზებისათვის შეიძლება გამოყენებული იქნას რამდენიმე სქემა: სალტეები თვალყურის მიდევნებით, ჯვარედინი და მრავალსაფეხურიანი კომუტაციის ქსელები, სხვადასხვა სქემები კატალოგის საფუძველზე. ყველა შემთხვევაში მულტიპროცესორებისათვის დაწერილმა პროგრამებმა შეიძლება მიიღონ წვდომა მეხსიერების ნებისმიერ ადგილთან ისე, რომ არ ჰქონდეთ წამოდგენა სისტემის ტოპოლოგიაზე ან მისი რეალიზაციის სქემაზე. სწორედ ამის გამო, მულტიპროცესორები სარგებლობენ დიდი პოპულარობით პროგრამისტებს შორის.

მიუხედავად ამისა, მულტიპროცესორებს გააჩნიათ გარკვეული უარყოფითი თვისებებიც და ეს ავტომატურად იწვევს მულტიკომპიუტერების როლის გაძლიერებას. პირველ რიგში, მულტიპროცესორებში გართულებულია მასშტაბირებადობა. უკვე განხილული გვაქვს რამდენი სახეობის მოწყობილობის გამოყენება დასჭირდათ Sun კომპანიის სპეციალისტებს E25K სისტემაში 72 პროცესორის ერთობლივად ფუნქციონირების უზრუნველსაყოფად. რაც შეეხება მულტიკომპიუტერებს, შემდეგში განხილული იქნება 65 536 კომპიუტერის შემცველი სისტემა. დღეს უკვე არსებობს მულტიკომპიუტერული სისტემები, სადაც კომპიუტერების რაოდენობამ მილიონს გადააჭარბა. ასეთი მასშტაბების მიღწევა მულტიპროცესორულ სისტემებში ძალიან რთულია.

შემდეგ, მულტიპროცესორების მწარმოებლურობაზე შეიძლება ძალიან ცუდად აისახოს კონკურენცია მეხსიერებაზე წვდომისათვის. თუ 100 პროცესორი მუდმივად ცდილობს მეხსიერებაში ჩაწეროს ან წაიკითხოს ერთიდაიგივე მონაცემი, ეს გამოიწვევს დიდ კონკურენციას სისტემის რესურსებზე (მეხსიერების მოდულებზე, სალტეებზე, კატალოგებზე) და შეამცირებს საერთო მწარმოებლურობას.

ამ ორი ფაქტორის შედეგად სპეციალისტები ავლენენ დიდ ინტერესს ისეთი პარალელური კომპიუტერული არქიტექტურების მიმართ, რომლებშიც ყოველ პროცესორს აქვს საკუთარი მეხსიერება, პირდაპირი გზით მიუწვდომელი სხვა პროცესორებისათვის. ასეთ არქიტექტურას წარმოადგენს მულტიკომპიუტერები. რადგანაც სხვადასხვა კომპიუტერების პროგრამებს არ გააჩნიათ ბრძანებებით LOAD და STORE პირდაპირი წვდომა სხვა კომპიუტერების მეხსიერებასთან, ისინი ერთმანეთთან თანამოქმედებას ახორციელებენ send და receive პრიმიტივების საშუალებით, რომლებიც გამოიყენება შეტყობინებების გადასაცემად. ეს განსხვავება არსებითად ცვლის დაპროგრამების მოდელს.

მულტიკომპიუტერის ყოველი კვანძი შედგება ერთი ან რამდენიმე პროცესორისაგან, ოპერატიული მეხსიერებისაგან (საერთო მხოლოდ მოცემული კვანძის პროცესორებისათვის), ხისტი დიკისაგან და (ან) შეტანა–გამოტანის სხვა მოწყობილობებისაგან. კვანძში გამოყენებულია აგრეთვე საკომუნიკაციო პროცესორი. საკომუნიკაციო პროცესორები ერთმანეთთან დაკავშირებულია მაღალსიჩქარიანი საკომუნიკაციო ქსელით. ასეთ სისტემებში გამოყენებულია სხვადასხვა ტოპოლოგია, კომუტაციის სქემა და მარშრუტის ამორჩევის ალგორითმი. მიუხედავად ამისა, ყველა მულტიპროცესორს აქვს ერთი თავისებურება: როდესაც პროგრამა ასრულებს send პრიმიტივს საკომუნიკაციო პროცესორი ღებულობს შეტყობინებას ამის შესახებ და მონაცემების ბლოკს გადასცემს ადრესატ კომპიუტერს (შესაძლოა წინასწარი შეკვეთისა და დასტურის მიღების შემდეგ). მულტიკომპიუტერის საერთო სქემა ნაჩვენებია ნახ. 3.1–ზე.



ნახ. 3.1. მულტიკომპიუტერის ზოგადი სქემა

3.1. საკომუნიკაციო ქსელები

მულტიკომპიუტერულ სისტემებში კვანძები ერთმანეთს უკავშირდება მაღალსიჩქარიანი საკომუნიკაციო ქსელის საშუალებით (ნახ. 3.1). განვიხილოთ ეს საკომუნიკაციო ქსელი უფრო

დაწვრილებით. აღსანიშნავია, რომ ამ მხრივ მულტიპროცესორები და მულტიკომპიუტერები ერთმანეთს ძალიან ჰგავს, რადგანაც მულტიპროცესორებშიც გამოიყენება ერთდროულად ბევრი პროცესორი და მეხსიერების ბევრი მოდული, რომელთა შორისაც დამყარებული უნდა იქნას მრავლობითი კავშირები. ამ ორი სახეობის სისტემის საკომუნიკაციო ქცელების მსგავსებას განაპირობებს ის, რომ ორივე შემთხვევაში ხდება შეტყობინებების გადაცემა. ერთპროცესორიან კომპიუტერშიც კი, როდესაც პროცესორს სჭირდება სიტყვის ჩაწერა ან წაკითხვა, საჭირო ხდება სალტეებზე გარკვეული ხაზების გააქტიურება და მათზე სიგნალების გადაცემა და პასუხის დაცდა. ეს დაახლოებით იგივეა, რაც შეტყობინების გაგზავნა, ინიციატორი აგზავნის შეტყობინებას და შემდეგ უცდის პასუხს. დიდ მულტიპროცესორებში, როდესაც მყარდება კავშირი პროცესორსა და მეხსიერების დაშორებულ მოდულს შორის, პროცესორი ყოველთვის აგზავნის შეტყობინებას მეხსიერების მოდულთან (ე.წ. **პაკეტს**), რომლითაც მოითხოვს გარკვეულ მონაცემებს, მეხსიერების მოდული კი უბრუნებს პროცესორს საპასუხო პაკეტს.

3.2. საკომუნიკაციო ქსელის ტოპოლოგია

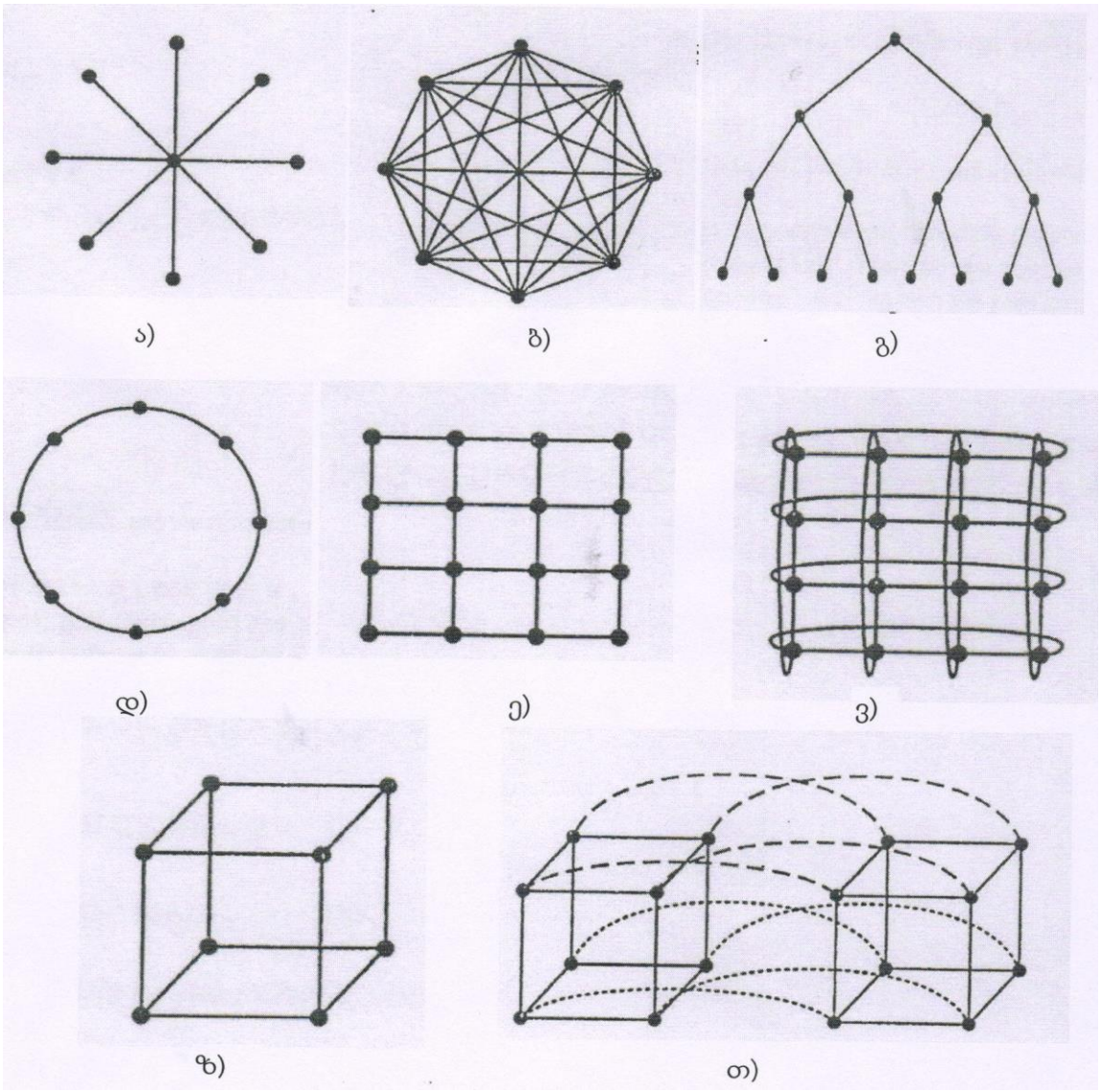
საკომუნიკაციო ქსელის ტოპოლოგია წარმოადგენს კომუტატორებისა და კავშირის ხაზების განლაგებისა და ურთიერთკავშირის აღწერას. ქსელების ტოპოლოგიას გრაფების სახით წარმოადგენენ, რომელთა წახნაგები კავშირის არხებს წარმოადგენს, ხოლო კვანძები კი კომუტატორებს (ნახ. 3.2). ქსელის ყოველ კვანძთან დაკავშირებულია კავშირის ხაზების გარკვეული რაოდენობა. ეს რაოდენობა განსაზღვრავს ქსელის განშტოებულობის კოეფიციენტს. განშტოებულობა რაც უფრო მაღალია, მით უფრო მაღალია განსხვავებული მარშრუტების მოძებნის ალბათობა და ასევე მაღალია სისტემის მტყუნებამედეგობა.

საკომუნიკაციო ქსელის შემდეგ მახასიათებელს წარმოადგენს მისი **დიამეტრი**. გრაფზე ორ კვანძს შორის მანძილი წარმოადგენს მათ შორის მოთავსებული კვანძების რაოდენობას. გრაფის დიამეტრს კი უწოდებენ მანძილს ქსელის ორ ყველაზე დაშორებულ კვანძს შორის. საკომუნიკაციო ქსელის დიამეტრი განსაზღვრავს პაკეტების გადაცემის ქსელში შესაძლო მაქსიმალურ დაყოვნებას. ქსელის დიამეტრი რაც უფრო მცირეა, მით უფრო მაღალია მწარმოებლურობა. გრაფზე შეიძლება განსაზღვრული იყოს კვანძებს შორის საშუალო მანძილი, რომლის საშუალებითაც განისაზღვრება პაკეტების გადაცემის საშუალო დრო.

საკომუნიკაციო ქსელის კიდევ ერთ მნიშვნელოვან მახასიათებელს წარმოადგენს მისი **გამტარუნარიანობა,** ანუ დროის ერთეულში გატარებული ინფორმაციის რაოდენობა. გამტარუნარიანობის საუკეთესო მახასიათებელს წარმოადგენს **კვეთის გამტარუნარიანობა.**ამ მახასიათებლის მნიშვნელობა შემდეგნაირად გამოითვლება – საკომუნიკაციო ქსელი წარმოსახვითიწახნაგებისა და კვანძების ამოგდებით უნდა გაიყოს კვანძების რაოდენობის მიხედვით ორ თანაბარ, ერთმანეთთან დაუკავშირებელ ნაწილად, და შემდეგ გამოითვალოს ამოგდებული კვანძებისა და წახნაგების საერთო გამტარუნარიანობა. მაგალითად, კვეთის გამტარუნარიანობა წარმოადგენს 800 ბიტი/წმ, ეს ნიშნავს, რომ თუ ქსელში ბევრია კვანძები და წახნაგები, მაშინ ქსელის საერთო გამტარუნარიანობის მინიმალური მნიშვნელობა იქნება 800 ბიტი/წმ. სპეციალისტების აზრით კვეთის გამტარუნარიანობა წარმოადგენს საკომუნიკაციო ქსელების ყველაზე მნიშვნელოვან მახასიათებელს. ხშირად,

საკომუნიკაციო ქსელების დაპროექტების დროს მიზნად ისახავენ კვეთის გამტარუნარიანობის მნიშვნელობის მაქსიმალურად გაზრდას.

საკომუნიკაციო ქსელის დასახასიათებლად შეიძლება გამოყენებული იქნას მისი **განზომილება**. საკომუნიკაციო ქსელის განზომილება განისაზღვრება გადამცემი კვანძიდან მიმღებ კვანძში გადასვლის მარშრუტების ვარიანტების მაქსიმალური რაოდენობით. არჩევანი თუ არ არსებობს, ანუ ქსელის ნებისმიერ ორ კვანძს შორის არსებობს მხოლოდ ერთადერთი მარშრუტი, მაშინ მისი განზომილება ნულია. ქსელის ნებისმიერ ორ კვანძს შორის თუ არსებობს ორი მარშრუტი, მაშინ მისი განზომილება ერთის ტოლია. თუ არსებობს ორი ღერძი და პაკეტი შეიძლება გაიგზავნოს აღმოსავლეთით ან დასავლეთით ან კიდევ ჩრდილოეთით ან სამხრეთით, მაშინ ქსელის განზომილება ორის ტოლია და ა.შ.



ნახ. 3.2. ტოპოლოგიის სახეობები. კვანძებს წარმოადგენენ კომუტატორები. პრცესორები და მეხსიერების მოდულები ნაჩვენები არაა: ვარსკვლავი (ა); სრულკავშირებიანი (ბ); იერარქიული (გ); რგოლი (დ); მესერი (ე); ორმაგი ტორი (ვ); კუბი (ზ); ოთხგანზომილებიანი ჰიპერკუბი (თ)

ნახ. 3.2–ზე ნაჩვენებია ტოპოლოგიის რამდენიმე სახეობა. ნახ. 3.2,ა–ზე ნაჩვენებია ტოპოლოგია **ვარსკვლავი**, სადაც პროცესორები და მეხსიერების მოდულები მიერთებულია გარე კვანძებთან, ხოლო მათ მიერთებას ახორციელებს ცენტრალური კვანძი, ანუ არსებობს მხოლოდ ერთადერთი მარშრუტი, შესაბამისად ამ ტოპოლოგიის განზომილება ნულის ტოლია. ასეთი სქემა ძალიან მარტივია, მაგრამ მისი საიმედობა არაა მაღალი და გარდა ამისა, ცენტრალური კვანძი წარმოადგენს ვიწრო ყელს და მისი მტყუნება ნიშნავს მთელი სისტემის მტყუნებას.

ნახ. 3.2,ბ–ზე ნაჩვენებია **სრულკავშირებიანი** ტოპოლოგია, რომლის განზომილებაც ასევე ნულის ტოლია. აქ ქსელის ნებისმიერი კვანძი დაკავშირებულია ყველა დანარჩენ კვანძებთან. ამ ტოპოლოგიის კვეთის გამტარუნარიანობას გააჩნია მაქსიმალური მნიშვნელობა, ხოლო დიამეტრი მინიმალურია და მტყუნებამედეგობა კი ძალიან მაღალი. მის ნაკლს წარმოადგენს ის, რომ kკვანძის შემთხვევაში საჭირო ხდება k(k-1)/2 რაოდენობის კავშირის ხაზი, ეს კი k–ს დიდი მნიშვნელობისათვის მიუღებელი ხდება.

ნახ. 3.2,გ–ზე ნაჩვენებია **იერარქიული** ტოპოლოგია. ამ ტოპოლოგიის პრობლემა იმაში მდგომარეობს, რომ კვეთის გამტარუნარიანობა კავშირის არხის გამტარუნარიანობის ტოლია. ყველაზე დიდი ტრაფიკი მოდის იერარქიის წვეროზე, ამიტომ სწორედ ქსელის ზედა კვანძები იქცევა სისტემის ვიწრო ყელად. ამ პრობლემის გადასაწყვეტად საჭიროა კვეთის გამტარუნარიანობის გაზრდა, რისთვისაც საჭიროა ზედა კვანძების კავშირის არხების გამტარუნარიანობის გაზრდა. მაგალითად, ყველაზე ქვედა კავშირის არხების გამტარუნარიანობა შეიძლება იყოს b, შემდეგი დონის კი 2b, იმის ზევით 4b და ა.შ. ასეთ სქემას უწოდებენ **სქელ ხეს** (fat tree). ასეთი სქემა გამოყენებული იყო კომერციულ მულტიკომპიუტრში Thinking Machines' CM5.

ტოპოლოგა **რგოლი** (ნახ. 3.2,დ). ამ ტოპოლოგიის განზომილება 1–ის ტოლია, რადგანაც ყოველი კვანძიდან პაკეტი შეიძლება გაიგზავნოს ან მარჯვნივ ან მარცხნივ.

ტოპოლოგია **მესერი** ან **ბადე** (ნახ. 3.2, ე). ამ ტოპოლოგიის განზომილება უკვე ორის ტოლია. ეს ტოპოლოგია ხშირად გამოიყენება კომერციულ სისტემებში. ის ხასიათდება რეგულარულობით და ადვილადაა შესაძლებელი მისი მასშტაბირება. მისი დიამეტრი არის კვანძების რაოდენობიდან ამოღებული კვადრატული ფესვის ტოლი. ანუ გამოდის, რომ სისტემის მასშტაბირების შედეგად მისი დიამეტრი დიდად არ იზრდება.

ტოპოლოგია **ორმაგი ტორი** (ნახ. 3.2, ვ) წარმოადგენს ტოპოლოგია მესერის ნაირსახეობას. ეს ტოპოლოგია ხასიათდება უფრო მაღალი მტყუნებამედეგობით და ნაკლები დიამეტრით სტანდარტულ მესერთან შედარებით. არსებობს კიდევ ერთი პოპულარული ტოპოლოგია – სამგანზომილებიანი ტორი. ამ ტოპოლოგიის აღწერა ხდება სამგანზომილებიანი სტრუქტურით, რომლის კვანძებიც იმყოფება წერტილებში (I,j,k), ხოლო ყველა კოორდინატი მთელი რიცხვია დიაპაზონში (1,1,1)–დან (l,m,n)-მდე. ყოველ კვანძს ჰყავს 6 მეზობელი, ორ–ორი კოორდინატთა ყოველი ღერძის მიმართულებით, ხოლო კიდურა კვანძები დაკავშირებულია ერთმანეთთან, ისე როგორც ორგანზომილებიან ტორში.

კუბი (ნახ. 3.2, ზ) – ის წარმოადგენს სამგანზომილებიან ტოპოლოგიას. ნახაზზე გამოსახულია კუბი განზომილებით 2×2×2, მაგრამ ზოგად შემთხვევაში შეიძლება არსებობდეს კუბი განზომილებით k×k×k. ნახ. 3.2, თ–ზე ნაჩვენებია ოთხგანზომილებიანი კუბი, რომელიც მიღებუ-

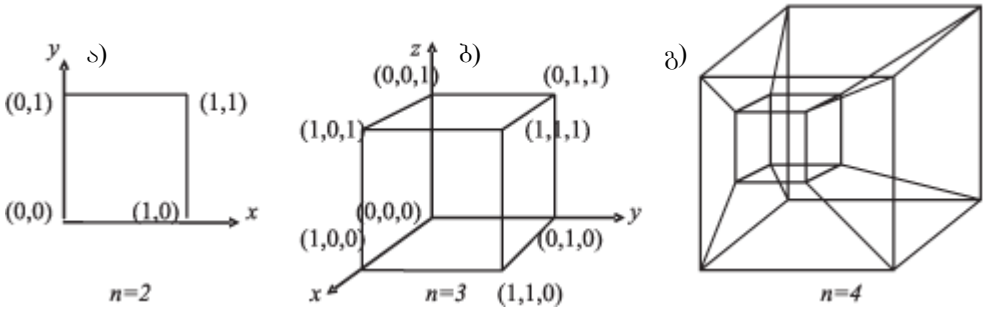
ლია ორი სამგანზომილებიანი კუბის შეერთებით. ოთხგანზომილებიანი კუბების შეერთებით შეიძლება მიღებული იქნას ხუთგანზომილებიანი კუბი.

ჰიპერკუბს უწოდებენ n–განზომილებიან კუბს (ნახ. 3.2, თ). ასეთი ტოპოლოგია გამოიყენება სხვადასხვა პარალელური კომპიუტერების არქიტექტურაში, რადგანაც მისი დიამეტრი წრფივადაა დამოკიდებული მის განზომილებაზე. სხვა სიტყვებით რომ ვთქვათ, მისი დიამეტრი არის ლოგარითმი 2–ის ფუძით მისი კვანძების რაოდენობიდან, ამიტომ 10–განზომილებიან ჰიპერკუბს გააჩნია 1024 კვანძი, მაგრამ მისი დიამეტრი 10–ის ტოლია, რაც უზრუნველჰყოფს მონაცემების გდაცემის ძალიან მცირე შეყოვნებას. აღსანიშნავია, რომ 32×32 განზომილების მქნე მესერი მოიცავს 1024 კვანძს, აქვს დიამეტრი 62, რაც დაახლოებით 6–ჯერ მეტია ჰიპერკუბთან შედარებით. ამიტომ ჰიპერკუბი წარმოადგენს ძირითად ტოპოლოგიას მაღალმწარმოებლური სისტემებისათვის. ამიტომ, უფრო მეტი ყურადღება დავუთმოთ ჰიპერკუბებს.

მულტიკომპიუტერებს გააჩნიათ იმდენად განსხვავებული ფორმები და განზომილებები, რომ მათი კლასიფიკაციის დალაგებული სისტემის ჩამოყალიბება საკმაოდ რთულია. მიუხედავად ამისა, შეიძლება გამოყოფილი იქნას ორი „სტილი“, ესენია პროცესორები მასიური პარალელიზმით და კლასტერები. შემდეგ მათ დაწვრილებით განვიხილავთ.

3.3. ჰიპერკუბები

ჰიპერკუბი არის პროცესორების სიბრტყული მესერის სივრცული განზოგადება, სადაც პროცესორების დამისამართება (შესაბამისად განაწილებული მეხსიერების ბლოკებისაც)დამოკიდებულია მათ შორის არსებულ კავშირებზე. ეს შეიძლება შედარებული იქნასn-განზომილებიან სივრცესთან.



ნახ. 3.3. გამოთვლითი სისტემა „ჰიპერკუბი“

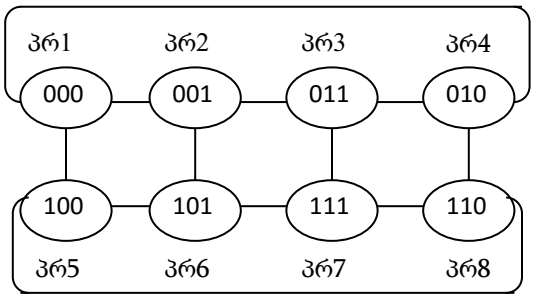
ავიღოთ ერთეულოვანი კვადრატი ორგანზომილებიან სივრცეში წვეროთი კოორდინატთა სათავეში (ნახ. 3.3,ა). ვთქვათ მისი წვეროები შეესაბამება პროცესორებს, ხოლო წიბოები კი მათ შორის კავშირებს. წვეროს კოორდინატებით შექმნილი კოდი იქნება შესაბამისი პროცესორის მისამართი.აქედან ჩანს, რომ კავშირი პროცესორებს შორის იარსებებს იმ შემთხვევაში, თუ მათი მისამართები ერთმანეთისაგან განსხვავდება მხოლოდ ერთ თანრიგში. იგივე სამართლიანი იქნება $n=3$ (ნახ. 3.3,ბ), $n=4$ (ნახ. 3.3,გ) და ა.შ.

ამგვარად, ზოგად შემთხვევაში “ჰიპერკუბის“ ტიპის გამოთვლითი სისტემაფორმირდება შემდეგნაირად. მას წარმოქმნის 2^n პროცესორი, თითოეული მათგანი დაკავშირებულია n რაოდე-

ნობის სხვა პროცესორთან. პროცესორების n სიგრძის მისამართების შემთხვევაში ერთმანეთს უშუალოდ უკავშირდება მხოლოდ ის პროცესორები, რომელთა მისამართებიც ერთმანეთისაგან განსხვავდება ერთ თანრიგში.

„ჰიპერკუბის“ ტიპის სტრუქტურას გააჩნია მნიშვნელოვანი თვისებები.

1. **„ჰიპერკუბის“ სტრუქტურიდან ადვილად მიიღება უფრო მარტივი სტრუქტურები.** მაგალითად, შემთხვევისათვის $n=3$ ადვილად მიიღება მატრიცული გამოთვლითი სისტემა. ამისათვის მოსახერხებელია სიბრტყული მესერი, რომელშიც არსებობს უშუალო (არა ტრანზიტული) კავშირები სამ პროცესორს შორის(სვეტს წარმოქმნის მხოლოდ ორი პროცესორი, ამიტომ არ არისორივე მიმართულებით კავშირების არსებობის აუცილებლობა), როგორც ნაჩვენებია ნახ.3.4-ზე. აქ შესაძლებელია მისამართებს დავუმატოთ კიდევ პროცესორების ნუმერაცია (დროებით) და ეს ნომრები გამოვიყენოთ მათ მისამართებად. პროცესორებმა წარმოქმნეს ბრტყელი მესერის ფრაგმენტი.
2. **გამოთვლითი რესურსებიდან შესაძლებელია გამოიყოს ამოცანებისათვის საჭირო დაკავშირებული არეები.** ეს არეები შეესაბამება სივრცეებს ნაკლები განზომილებით, ჰიპერკუბის მთელ განზომილებასთან შედარებით. ეს ნიშნავს, რომ რადგანაც რომელიმე სამისამართო სივრცეში ხდება პროცესორების დამისამართება, ამიტომ ყოველ ამოსახსნელ ამოცანას შეიძლება გამოვუყოთგამოთვლითი რესურსი რომელიმე მასივის ფარგლებში პროცესორების გამჭოლი ნუმერაციით. n განზომილებიან სივრცეში ეს მასივი ეკუთვნის ქვესიმრავლეს ნაკლები განზომილებით.



ნახ. 3.4. მატრიცული გამოთვლითი სისტემა ორგანზომილებიანი ჰიპერკუბის ბაზაზე

დავუშვათ წინა მაგალითში ჩვენს განკარგულებაში იყო არა 3-განზომილებიანი, არამედ 6-განზომილებიანი ჰიპერკუბი. დავუშვათ, რომ ოპერაციული სისტემის დამგეგმავმა სისტემამ პროცესორების მთელი სამისამართო სივრციდან 000000÷111111 ჩვენს განკარგულებაში გამოყო მასივი 101000÷101111, ანუ 3-განზომილებიანი ქვესივრცე პროცესორების საერთო სივრციდან. ამავდროულად, სხვა ამოცანისათვის შეიძლება გამოყოფილი იყოს მასივი 000000÷001111, ანუ 4-განზომილებიანი ქვესიმრავლე და ა.შ.

ისევე, როგორც სხვა ერთგვაროვანი გამოთვლითი გარემოებისათვის, აქაც იგულისხმება განაწილებული მეხსიერების არსებობა. საერთო სამისამართო სივრცის ფორმირების დროს მივიღოთ, რომ მისამართების უფროსი თანრიგები ემთხვეოდეს პროცესორების მისამართებს.

3.4. პროცესორები მასიური პარალელიზმით

პროცესორები მასიური პარალელიზმით (Massively Parallel Processors, **MPP**) ეს არის დიდი სუპერკომპიუტერები, რომელთა ღირებულებაც განისაზღვრება რამდენიმე ასეული მილიონი დოლარით. ისინი გამოიყენება მეცნიერებისა და ტექნიკის სხვადასხვა სფეროში რთული გამოთვლების ჩასატარებლად, დროის ერთეულში ტრანზაქციების დიდი რაოდენობის ჩასატარებლად, მონაცემთა დიდი ბაზების მართვისათვის. თავდაპირველად ეს კომპიუტერები განკუთვნილი იყო სამეცნიერო გამთვლებისათვის, მაგრამ დღეს ეს კომპიუტერები წარმატებით გამოიყენება კომერციაში. შეიძლება ასეც ითქვას, რომ MPP–კომპიუტერებმა გამოყენებიდან გამოდევნეს SIMD–კომპიუტერები, ვექტორული სუპერკომპიუტერები და მატრიცული პროცესორები.

MPP–კომპიუტერების უმრავლესობაში გამოიყენება სტანდარტული პროცესორები. ეს შეიძლება იყოს Intel Pentium, Sun UltraSPARC, IBM RS/360 და DEC Alpha პროცესორები. მულტი-კომპიუტერებისათვის დამახასიათებელია მაღალმწარმოებლური საკომუნიკაციო ქსელების გამო-ყენება, რომელთა საშუალებითაც შესაძლებელია შეტყობინებების გადაცემა დაგვიანების მცირე დროით და მაღალი გამტარუნარიანობით. ორივე ეს მახასიათებელი ძალიან მნიშვნელოვანია, რადგანაც შეტყობინებები ძირითადად თავისი მოცულობით არაა დიდი (256 ბაიტზე ნაკლები), მაგრამ ძირითად ტრაფიკში დიდი წვლილი შეაქვს დიდ შეტყობინებებს (მოცულობით 8 Kბაიტზე მეტი). მომხმარებლებს MPP–მულტიკომპიუტერები მიეწოდება საკმაოდ ძვირ პროგრამულ უზრუნველყოფასთან და ბიბლიოთეკებთან ერთად.

MPP–მულტიკომპიუტერების კიდევ ერთი დამახასიათებელი თავისებურებაა შეტანა–გამოტანის დიდი მოცულობა. MPP–მულტიკომპიუტერების საშუალებით ხდება მონაცემების ძალიან დიდი მასივების დამუშავება. ეს მონაცემები გადანაწილებული უნდა იყოს დისკების დიდ რაოდენობაზე და დიდი სიჩქარით უნდა მოხდეს მათი მიწოდება კომპიუტერების მოწყობილო-ბებისათვის.

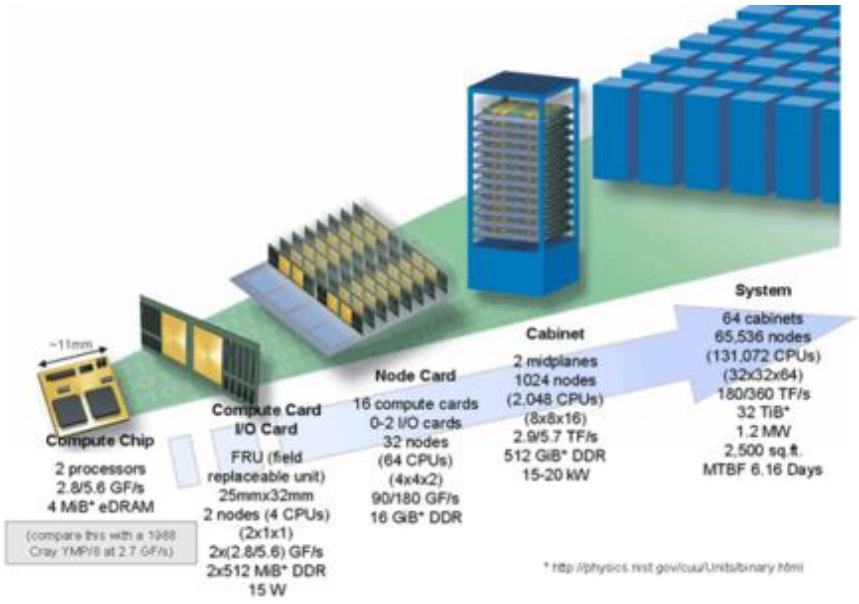
და ბოლოს, MPP–მულტიკომპიუტერებისდადებითი დამახასიათებელი თავისებურებაა მტყუნებამედეგობა. თუ გავითვალისწინებთ იმ ფაქტს, რომ სისტემაში ათასობით პროცესორია, გასაგებია, რომ რამდენიმე პროცესორის მტყუნება მთელი სისტემის მტყუნებას ვერ გამოიწვევს. დიდ MPP–მულტიკომპიუტერებში ყოველთვის არის გათვალისწინებული სისტემის მონიტორინ-გის სპეციალური აპარატული და პროგრამული საშუალებები, რომელთა საშუალებითაც ხდება მტყუნებების აღმოჩენა და აღმოფხვრა.

MPP–მულტიკომპიუტერებს მოქმედების რაიმე განსაკუთრებული პრინციპები არ გააჩნია. მისი დახასიათება მოკლედ ასეთნაირად შეიძლება, რომ ის წარმოადგენს სტანდარტული გამოთვლითი კვანძების სიმრავლეს, რომლებიც ერთმანეთთან დაკავშირებულია მაღალი გამტარუნარიანობის მქონე საკომუნიკაციო ქსელის საშუალებით. შემდეში განხილული იქნება MPP–კომპიუტერების ორი კონკრეტული მაგალითი BlueGene და Red Storm.

3.5. BlueGene

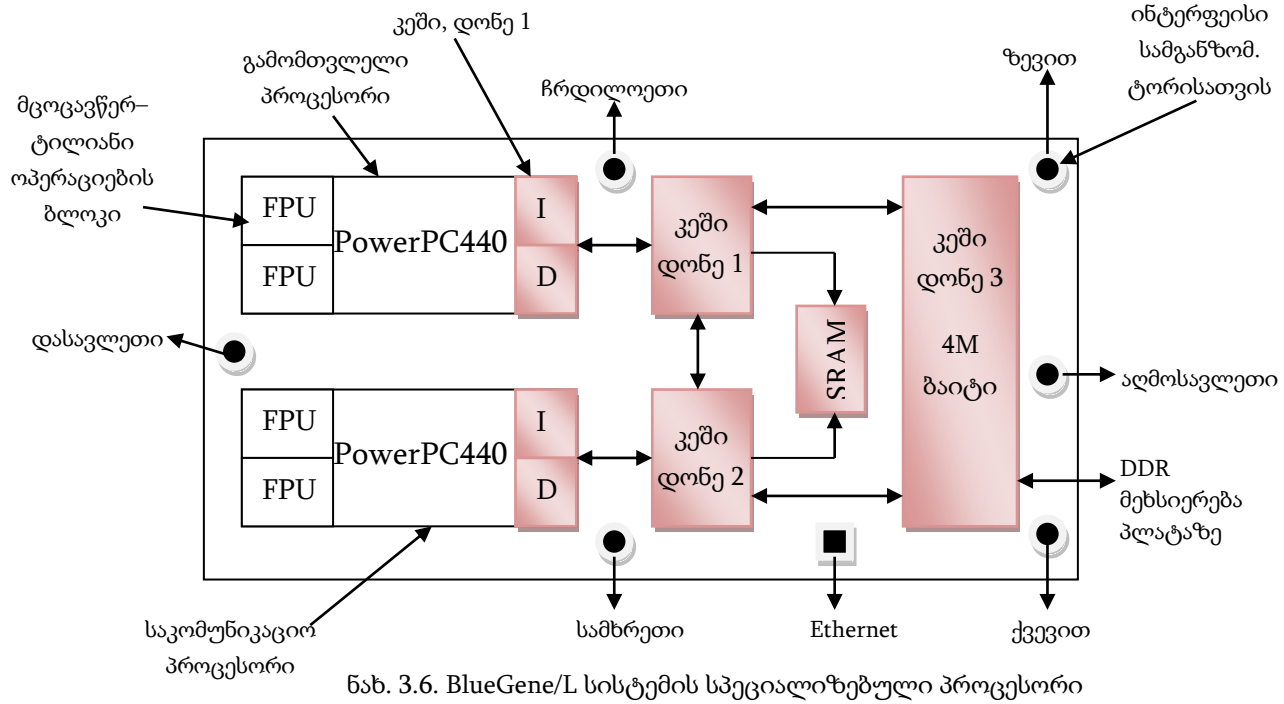
მასიურ პარალელური არქიტექტურის მქონე კომპიუტერების პირველ მაგალითად განვიხილოთ IBM–ის მიერ შექმნილი სისტემა BlueGene (ნახ. 3.5.). ეს პროექტი ჩაფიქრებული იქნა 1999 წელს. ეს უნდა ყოფილიყო სუპერკომპიუტერი რთული სამეცნიერო გამოთვლების ჩასატა-რებლად ბიოლოგიაში. კერძოდ, ბიოლოგები თვლიან, რომ ცილების ფუნქციები წარმოადგენს სამგანზომილებიან სტრუქტურას. ცილის ერთი მოლეკულის ფორმების დადგენისათვის საჭირო გამოთვლებს იმ დროინდელ სუპერკომპიუტერებზე წლები დასჭირდებოდა. ამ დროს, ადამიანის ორგანიზმში ნახევარ მილიონამდე სხვადასხვა ცილაა, რომელთაგანაც ზოგიერთი საკმაოდ რთუ-ლი შემადგენლობისაა. ცხადია, რომ ადამიანის ორგანიზმში შემავალი ცილების სამგანზომილები-ანი სტრუქტურის გათვლას დასჭირდებოდა მაშინდელი გამოთვლითი სისტემების მწარმოებლუ-რობის რამდენიმე რიგით გაზრდა. ასეთივე სირთულის ამოცანები ისმება მოლეკულურ დინამი-კაში, კლიმატის მოდელირების დროს, ასტრონომიაში და სხვ.

ასეთ კომპიუტერებზე რადგანაც გაჩნდა მოთხოვნილება, ამიტომ IBM–მა BlueGene პროექტის განხორციელებაში ჩადო 100 მილიონი დოლარი და 2001 წელს გამოჩნდა მისი პირველი მოდელი BlueGene/L. მისი პირველი დამკვეთი იყო ლივმორის ნაციონალური ლაბორატორია.



ნახ. 3.5. MPP–მულტიკომპიუტერი BlueGene

BlueGene პროექტის მიზანი იყო MPP–სუპერკომპიუტერის შექმნა, რომელიც იქნებოდა არა მარტო ყველაზე სწრაფი, არამედ ეფექტურიც ტერაფლოპი/დოლარი, ტერაფლოპი/ვატი და ტერაფლოპი/მ^3 მაჩვენებლების მიხედვით. ამის გამო, IBM–მა უარი თქვა წინა MPP–კომპიუტერებში გამოყენებულ პრინციპებზე. გადაწყვეტილი იქნა დაემზადებინათ საკუთარი ერთკრისტალიანი კომპონენტი, რომელსაც ექნებოდა მუშაობის საშუალო სიჩქარე და დაბალი ენერგომოხმარება, და მათ საფუძველზე აეგოთ დიდი სისტემა კომპონენტების ეფექტური განლაგებით. პირველი მიკროსქემა დამზადებული იქნა 2003 წელის ივნისში, ხოლო BlueGene/L სისტემის პირველი მეოთხედი, 16384 გამოთვლითი კვანძით, უკვე მუშაობდა 2004 წლის ნოემბერში. სწორედ მაშინ მოხდა მისი, როგორც ყველაზე სწრაფი კომპიუტერის სერტიფიცირება. მისი მწარმოებლურობა იყო 71 ტერაფლოპსი/წმ,

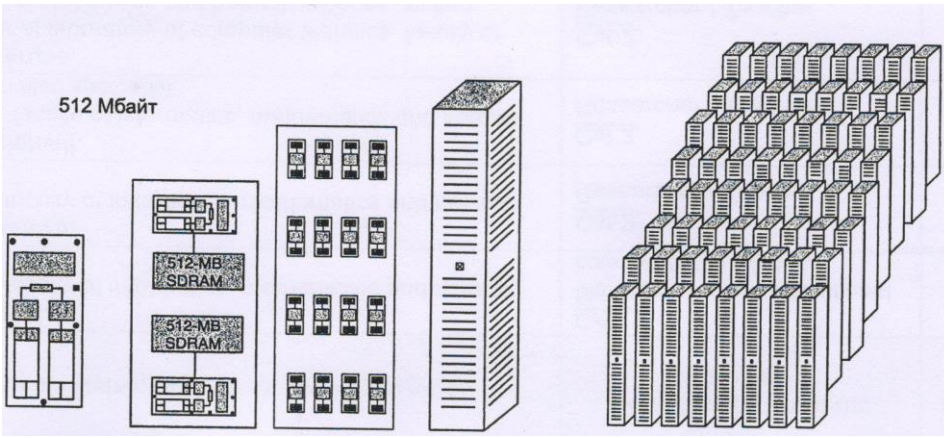


ნახ. 3.6. BlueGene/L სისტემის სპეციალიზებული პროცესორი

მოხმარებული ელექტროენერგია 0,4 მეგავატი. მთლიანობაში სისტემაში უნდა ყოფილიყო 65 536 კვანძი.

BlueGene/L სისტემის გულს წარმოადგენდა სპეციალიზირებული მიკროსქემისაგან შედგენილი კვანძი (ნახ. 3.7). იგი შედგება 700 MHz სიხშირეზე მომუშავე PowerPC 440 ორი კვანძისაგან. PowerPC 440 წარმოადგენს გაორმაგებულ კონვეიერიზებულ სუპერსკალარულ პროცესორს. თითოეულ ბირთვში გამოყენებულია მცოცავწერტილიან მონაცემებზე ოპერაციების შესრულების გაორმაგებული წყვილი ბლოკი. მათში გამოყენებულია SIMD–ბრძანებების მხარდაჭერა, რომელიც ძალიან სასარგებლოა მასივების დამუშავების დროს. თავისი მოკრძალებული შესაძლებლობებით ეს პროცესორი ვერ ჩაითვლება რეკორდსმენ პროცესორად.

ამ მიკროსქემის ორი პროცესორი იდენტურია, მაგრამ დაპროგრამებულია სხვადასხვანაირად. პირველი გამოყენებულია გამოთვლებისათვის, ხოლო მეორე კი დანარჩენ 65535 კვანძთან ურთიერთობისათვის.



ნახ. 3.7. BlueGene/L სისტემა: ა) მიკროსქემა, ბ) პლატა,გ) პანელი, დ) კარადა, ე) მთელი სისტემა

მიკროსქემაში გამოყენებულია კეშირება 3 დონეზე. პირველი დონის კეშ–მეხსიერება გაყოფილია ორ 32Kბაიტიან ნაწილად, პირველი ნაწილი განკუთვნილია ბრძანებებისათვის, ხოლო მეორე კი მონაცემებისათვის. ორ პროცესორში განთავსებული პირველი დონის კეშ–მეხსიერებები ერთმანეთთან შეთანხმებული არ არის, რადგანაც PowerPC 440 სტანდარტულ ბირთვებს ამის შესაძლებლობა არ გააჩნიათ, მათი მოდიფიცირება კი არ მოახდინეს.მეორე დონის გაერთიანებული კეშ–მეხსიერების მოცულობა შეადგენს 2Kბაიტს. სინამდვილეში ეს მეხსიერება კეშ–ზე უფრო მეტად წარმოადგენს წინასწარი ამორჩევის ბუფერს. მეორე დონის კეშ–მეხსიერებაში რეალიზებულია ერთმანეთისათვის თვალყურის მიდევნების მექანიზმი, რის შედეგადაც მიიღწევა მათი შეთანხმებული მუშაობა. მესამე დონე წარმოადგენს გაეთიანებულ კეშ–მეხსიერებას მოცულობით 4Mბაიტი, რომელსაც გამოიყენებს მეორე დონის ორივე კეშ–მეხსიერება. მეხსიერებასთან მიმართვა, რომლის საჭიროებაც წარმოიქმნება კეშ–აცილების პირველ დონეზე და კეშ–მოხვედრების მეორე დონეზე შემთხვევაში, ხორციელდება 11 ტაქტში. შემთხვევას, კეშ–აცილება მეორე დონეზე და კეშ–მოხვედრება მესამე დონეზე, მეხსიერებასთან მიმართვისათვის სჭირდება 28 ტაქტი. და ბოლოს, მესამე დონეზე კეშ–აცილების შემთხვევაში საჭირო ხდება მთავარ ოპერატიულ მეხსიერებასთან (DDR SDRAM) მიმართვა, ამისათვის კი საჭიროა 75 ტაქტი.

მეორე დონის კეშ–მეხსიერებასთან მიერთებულია მცირე ზომის მეხსიერების მოდული (SRAM), რომელიც გამოიყენება სისტემის ჩატვირთვისათვის, გამართვისათვის, მთავარ ჰოსტთან დაკავშირებისათვის.მასში ინახება სისტემური სტეკი, აგრეთვე გამოიყენება სემაფორებისა და სინქრონიზაციის სხვა საშუალებების მისაწოდებლად.

უფრო მაღალი დონისათვის დამუშავებული იქნა სპეციალიზებული პლატა, რომელზეც დაყენებულია ორი ცალი აღწერილი მიკროსქემა და ოპერატიული მეხსიერება მოცულობით 1Gბაიტი. შემდეგ ვერსიებში დაყენებული იქნა ოპერატიული მეხსიერება მოცულობით 4Gბაიტი. მიკროსქემა ნაჩვენებია ნახ. 3.7,ა–ზე, ხოლო პლატა კი ნახ. 3.7,ბ–ზე.

პლატების დამონტაჟება ხდება ჩასადგმელ პანელებზე, 16 პლატა ერთ პანელზე, რაც იძლევა 32 მიკროსქემას (ანუ 32 გამომთვლელ პროცესორს) პანელზე. რადგანაც თითოეულ პლატაზე დაყენებულია SDRAM მეხსიერება მოცულობით 1Gბაიტი, პანელზე მეხსიერების მოცულობა გამოდის 16Gბაიტი (ნახ. 3.7, გ).

შემდეგ დონეზე ხდება 16 ასეთი პანელის ჩადგმა კარადის ზედა ნაწილში. კიდევ 16 პანელი ჩაიდგმება კარადის ქვედა ნაწილში. საერთო ჯამში, კარადაში, რომლის ზომებიც არის 60×90 სმ, დამონტაჟებულია 1024 პროცესორი. პანელების ორი ჯგუფი ერთმანეთისაგან გამოყოფილია გადამრთველით, რომელიც იძლევა შესაძლებლობას პანელების ერთი ჯგუფი მომსახურების მიზნით ამოღებული იქნას კარადიდან და მის ნაცვლად მიერთებული იქნას პანელების სარეზერვო ჯგუფი. კარადა ნაჩვენებია ნახ. 3.7, დ–ზე.

და ბოლოს, მთელი სისტემა, რომელიც შედგება 65 536 გამომთვლელი და 65536 საკომუნიკაციო პროცესორისაგან, გამოსახულია ნახ. 3.7, ე–ზე. სისტემის შემადგენლობაში შედის 131072 გაორმაგებული პროცესორი ფიქსირებულწერტილიანი მონაცემების დამუშავებისათვის, და 262144 გაორმაგებული პროსესორი მცოცავწერტილიანი მონაცემების დამუშავებისათვის, ანუ სისტემას ერთი ციკლის განმავლობაში უნდა შეეძლოს 786432 ბრძანების შესრულება.

სისტემა წარმოადგენს მულტიკომიუტერს იმ გაგებით რომ არცერთ პროცესორს არ გააჩნია მეხსიერებასთან პირდაპირი წვდომა, თუ არ ჩავთვლით 512 Mბაიტ მეხსიერებას თვითონ პლატაზე. პროცესორების არცერთ წყვილს არ გააჩნია საერთო მეხსიერება. გარდა ამისა, არ არსებობს ფურცლების გამოძახების მხარდაჭერა, რადგანაც არ არსებობს ლოკალური დისკები ფურცლების განთავსებისათვის. სამაგიეროდ სისტემაში არსებობს შეტანა–გამოტანის 1024 კვანძი, რომლებთანაც მიერთებულია დისკები და სხვა პერიფერიული მოწყობილობები.

მიუხედავად სისტემის დიდი ზომისა, ის საკმაოდ მარტივია და მასში არა არის გამოყენებული რაიმე განსაკუთრებული ტექნოლოგიური სიახლეები, თუ არ ჩავთვლით იმას, რომ მისი კვანძები ძალიან მჭიდროდ არის განლაგებული. ეს შემთხვევით არაა, პროექტის ძირითადი მიზანი საიმედოობა იყო. ძალიან კარგადაა დამუშავებული სისტემის ელექტროკვება, გაცივება, საკაბელო სისტემა და სხვ. ყველაფერმა ამან გაზარდა სისტემის საიმედოობა.

ყველა მიკროსქემის მიერთებისავის საჭიროა კარგად გათვლილი და ეფექტური მიერთების სიტემა. სისტემის ტოპოლოგია არის სამგანზომილებიანი ტორი ზომებით 64×32×32. ყოველ მიკროსქემას მეზობლებთან კავშირებისათვის სჭირდება კავშირის 6 ხაზი: 2 ხაზი ზევითა და ქვევითა მეზობლებთან კავშირისათვის, 2 ხაზი მეზობლებისათვის ჩრდილოეთით და სამხრეთით

და 2–ც მეზობლებისათვის აღმოსავლეთით და დასავლეთით. ეს მიერთებები ნაჩვენებია ნახ. 3.6–ზე. კონსტრუქციულად ყოველი კარადა, რომელიც 1024 კვანძზეა გათვლილი, წარმოადგენს სამგანზომილებიან ტორს ზომებით 8×8×16. მეზობელი კარადების წყვილი ქმნის ტორს ზომებით 8×8×32. კარადების 4 წყვილი ქმნის ტორს ზომებით 8×32×32, და ბოლოს 8 რიგით კი წარმოიქმნება ტორი ზომებით 64×32×32.

ამგვარად, სისტემაში ყველა მიერთება არის ორმხრივი და მუშაობს სიჩქარით 1,4 Gბაიტი/წმ. ყოველი კვანძიდან (მათი სრული რაოდენობაა 65536) მეზობლებისაკენ გადის კავშირის 3 ხაზი, თითო ხაზი თითო განზომილებისათვის, რაც საერთო ჯამში ქმნის სისტემის გამტარუნარიანობას 275 Tბაიტი/წმ. ეს ძალიან მაღალი მაჩვენებელია.

სამგანზომილებიან კუბში თანამოქმედების უზრუნველყოფა ხდება **ვირტუალური გამჭოლი მარშრუტიზაციის** (virtual cut through routing) ფორმით. მონაცემების გადაცემა ხდება პაკეტების სახით. შუალედურ კვანძებში გავლის დროს მათი მთლიანი შენახვა არ ხდება. შუალედურ კვანძში როგორც კი პაკეტის მორიგი ბაიტი შემოვა, ის მაშინვე გადაეცემა მითითებული მარშრუტით ისე, რომ არ ხდება მოცდა მთელი პაკეტის შემოსვლისათვის. შესაძლებელია როგორც დინამიური (ადაპტიური), ასევე სტატიკური (ფიქსირებული) მარშრუტიზაცია. ვირტუალური გამჭოლი მარშრუტიზაციის რეალიზებისათვის მიკროსქემაში გათვალისწინებულია რამდენიმე სპეციალიზებული მოწყობილობა.

სისტემაში კომუნიკაციის ძირითადი ფორმაა სამგანზომილებიანი ტორი, მაგრამ მასთან ერთად გამოიყენება კომუნიკაციის კიდევ რამდენიმე სისტემა. მეორე სისტემაა იერარქიული (ხე) სტრუქტურა. სისტემაში რეალიზებულია პარალელიზმის ძალიან მაღალი მაჩვენებელი, ბევრი ოპერაციის დროს საჭირო ხდება სისტემის ყველა კვანის ერთდროულად მუშაობა. მაგალითის სახით განვიხილოთ მინიმალური ელემენტის მოძებნა 65536 მნიშვნელობიდან იმ პირობით, რომ ეს ელემენტები გადანაწილებულია სისტემის კვანძებზე. სისტემაში თუ არსებობს ხისმაგვარი სტრუქტურა, მაშინ ყოველი ორი კვანძი უგზავნის თავის მნიშვნელობას ზემდგომ კვანძს. ზემდგომი მათგან აარჩევს უმცირესს და გაგზავნის ზევით და ა.შ. ასეთი მიდგომით ხის წვეროში მოხვდება ინფორმაციის მხოლოდ მინიმუმი. წარმოვიდგინოთ რა მოხდებოდა, რომ შესადარებლად 65536 კვანძიდან ყოველს გაეგზავნა თავისი ელემენტები ერთ რომელიმე კვანძში.

მესამე საკომუნიკაციო ქსელი გამოიყენება გლობალური შეჩერებებისათვის და წყვეტებისათვის. ზოგიერთი ალგორითმი მოითხოვს ამოცანის ეტაპობრივად შესრულებას, როდესაც ყოველი კვანძი რომ დაამთავრებს თავის ეტაპს, შემდეგ ეტაპზე კი არ გადადის, არამედ იცდის ვიდრე ეს ეტაპი ყველა კვანძზე არ დასრულდება. განსაკუთრებული ბარიერული ქსელი უზრუნველყოფს ამ ეტაპების პროგრამულად განსაზღვრას და ამ ეტაპების მიხედვით აჩერებს გამოთვლებს ყველა პროცესორზე. როდესაც ყველა პროცესორი დაასრულებს თავის ეტაპს, გამოთვლები გაგრძელდება. ასეთივე ბარიერული ქსელი გამოიყენება წყვეტების ორგანიზებისათვის.

მეოთხე და მეხუთე ქსელები აგებულია Gigabit Ethernet ტექნოლოგიის საფუძველზე. ერთი მათგანი შეტანა–გამოტანის კვანძებს აერთებს ინტერნეტთან და ფაილების სერვერთან, რომელიც BlueGene/L სისტემის შემადგენლობაში არ შედის. მეორე ქსელი კი გამოიყენება სისტემის გამართვისათვის.

ყოველ გამომთვლელ და საკომუნიკაციო კვანძზე მუშაობს სპეციალიზებული მცირე ოპერაციული სისტემა, რომელიც ახდენს ერთი მომხმარებლის და ერთი პროცესის უზრუნველყოფას. პროცესს შეიძლება გააჩნდეს ორი პროგრამული ნაკადი, თითო ნაკადი კვანძის თითოეული პროცესორისათვის. ასეთი სტრუქტურა უზრუნველყოფს მაღალ მწარმოებლურობას და საიმედოობას.

საიმედობის გაზრდის მიზნით გამოყენებით პროგრამაში შეიძლება **შენახვის წერტილების** შექმნა. მას შემდეგ რაც სისტემაში დასრულდება შეტყობინებების გადაცემის პროცესი, შეიძლება შექმნილი იქნას შენახვის გლობალური წერტილი, იმისათვის, რომ სისტემის მტყუნების შემთხვევაში საჭირო არ გახდეს გამოთვლების თავიდან დაწყება და შესაძლებელი იყოს გამოთვლების გაგრძელება შენახვის წერტილიდან. შეტანა–გამოტანის კვანძები იმართება ტრადიციული OS Linux–ის საშუალებით, ანუ უზრუნველყოფილია მრავალამოცანიანი მუშაობის პრინციპი.

3.6. Red Storm

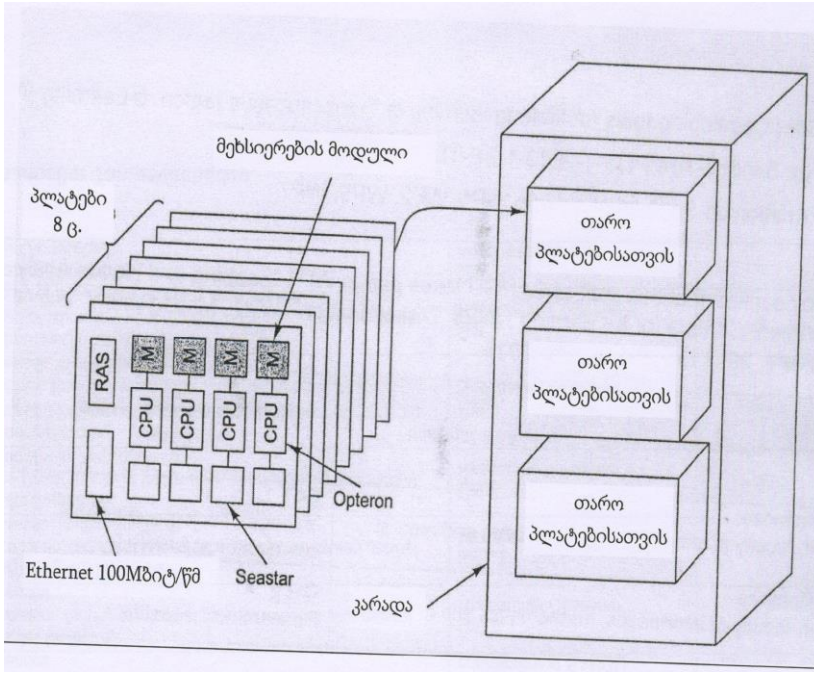
MPP სისტემების მეორე მაგალითის სახით განვიხილოთ ამერიკის ნაციონალურ ლებორატორია Sanda–ში შექმნილი სისტემა Red Storm (ე.წ. Thor's hammer) – ნახ. 3.8. ლაბორატორია Sanda ასრულბეს ამერიკის ენერგეტიკის დეპარტამენტის საიდუმლო და არასაიდუმლო დავალებებს. საიდუმლო დავალებებში შეიძლება აღნიშნული იქნას ატომური აფეთქების მოდელირება, რასაც სჭირდება ძალიან ინტენსიური გამოთვლები.

ნახ. 3.8. მულტიკომპიუტერული სისტემა Red Storm

ლაბორატორია Sanda დიდი ხანია ეწევა ამ საქმიანობას და გააჩნია ძლიერი სუპერკომპიუტერები. ათეული წლების განმავლობაში უპირატესობა ენიჭებოდა ვექტორულ სუპერკომპიუტერებს, მაგრამ ტექნოლოგიის განვითარების და ეკონომიკური ცვლილებების შედეგად წინა პლანზე გამოვიდა MPP სისტემები. 2002 წელს ძალიან პოპულარული იყო MPP სისტემა ASCI Red, მაგრამ მის მუშაობაში თავი იჩინა პრობლემებმა. მიუხედავად იმისა, რომ მის შემადგენლობაში იყო 9460 კვანძი და ერთობლიობაში ჰქონდათ 1,2 Tბაიტი ოპერატიული მეხსიერება და 13,5 Tბაიტი დისკური მეხსიერება, სისტემის საერთო მწარმოებლურობა გაჭირვებით აღწევდა 3 ტერაფლოპს/წმ. ამიტომ, 2002 წელს მიღებული იქნა მისი შეცვლის გადაწყვეტილება და ამისათვის დაუკავშირდნენ ძალიან ავტორიტეტულ კომპანიას Cray Research.

ახალი სისტემის დამუშავება დამთავრებული იქნა უკვე 2004 წლის აგვისტოში. ასეთი ძლიერი კომპიუტერისათვის ეს არის ძალიან მოკლე ვადა. ასეთი სწრაფი დაპროექტება შესაძლებელი გახდა იმის გამო, რომ კომპიუტერი Red Storm აგებული იქნა იმ დროს არსებული კომპონენტების საშუალებით. გამონაკლისს წარმოადგენდა მხოლოდ მარშრუტიზაციისათვის გამოყენებული სპეციალიზებული მიკროსქემა.

Red Storm სისტემისათვის შერჩეული იქნა AMD კომპანიის მიერ წარმოებული პროცესორი Opteron. ეს არჩევანი განაპირობა პროცესორის რამდენიმე მახასიათებალმა. პირველი – გააჩნია მუშაობის სამი რეჟიმი. ამ პროცესორს მემკვიდრეობით რეჟიმში ყოველგვარი მოდიფიკაციის გარეშე შეუძლია შეასრულოს ჩვეულებრივი Pentium კომპიუტერებისთვის დაწერილი პროგრამები. თავსებადობის რეჟიმში ოპერაციული სისტემა მუშაობს როგორც 64–თანრიგიანი და შეუძლია 2^{64} ბაიტი მეხსიერების დამისამართება. და ბოლოს, 64–თანრიგიან რეჟიმში სრულად ხდება 64–თანრიგიანი პროცესორი და შეუძლია 64–თანრიგიანი სამისამართო სივრცის სრულად დამისამართება. თანაც, 64–თანრიგიან რეჟიმში ერთდროულად შეიძლება სრულდებოდეს 32–თანრიგიანი და 64–თანრიგიანი პროგრამები, რაც ამარტივებს სისტემის განახლებას.



ნახ. 3.9. კომპონენტების განთავსება სისტემაში Red Storm.

Opteron პროცესორის კიდევ ერთ არსებით მახასიათებელს წარმოადგენს ის, რომ მასში ზედმიწევნითაა გათვლილი მეხსიერების გამტარუნარიანობა. ბოლო დროს საგრძნობლად გაიზარდა პროცესორების სწრაფქმედება და ბევრად გაუსწრო მეხსიერების ელემენტების სწრაფქმედებას. შედეგად მივიღეთ ის, რომ მეორე დონის კეშ–მეხსიერებაში კეშ–აცილების შემთხვევაში ძალიან გაიზარდა ოპერატიულ მეხსიერებასთან მიმართვის დრო. AMD–ს ინჟინრებმა Opteron პროცესორში დააყენეს მეხსიერების კონტროლერი, რომელიც მუშაობს პროცესორის სიხშირეზე და არა სალტის სიხშირეზე, რამაც გაზარდა მეხსიერების სწრაფქმედება. კონტროლერს შეუძლია მეხსიერების 8 DIMM მოდულთან მუშაობა, რომელთაგან თითოეულის მოცულობაა 4Gბაიტი, ანუ საერთო ჯამში 32Gბაიტი. სისტემის მწარმოებლურობის გაზრდის შესაძლებლობაა აგრეთვე მრავალბირთვიანი პროცესორების გამოყენება.

ყოველი Opteron პროცესორისათვის გამოყოფილია IBM–ს მიერ დამუშავებული Seastar სპეციალიზებული საქსელო პროცესორი. ეს სისტემის ძალიან მნიშვნელოვანი ელემენტია, რადგანაც პროცესორებს შორის ინფორმაციის გაცვლა სწორედ Seastar ქსელის საშუალებით ხდება. სწრაფი საკომუნიკაციო ქსელის გარეშე, რომლის ფუნქციონირებაც ამ მიკროსქემით არის უზრუნველყოფილი, სისტემა „ჩაიძირებოდა“ მონაცემებში.

მიუხედავად იმისა, რომ Opteron პროცესორები გაყიდვაში არსებული ჩვეულებრივი პროცესორებია, ისინი დაყენებულია სპეციალიზებულ პლატებზე (ნახ. 3.9). ყოველ ასეთ პლატაზე განლაგებულია 4 Opteron პროცესორი, 4Gბაიტი ოპერატიული მეხსიერება, 4 Seastar პროცესორი,

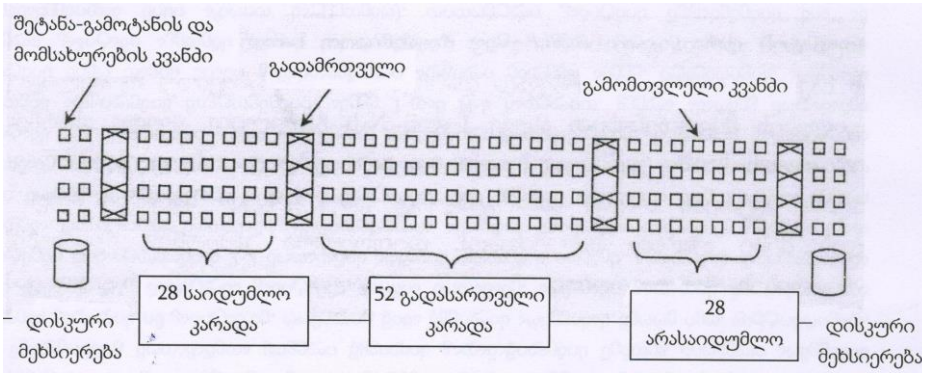
პროცესორი RAS (Reliability, Availibility and Service - საიმედობა, მუშაობისუნარიანობა და ექსპლუატაციის მოხერხებულობა), 100 Mბიტ/წმ Ethernet–ის მიკროსქემა.

8 პლატისაგან შემდგარი ნაკრები დაყენებულია პანელზე, რომელიც იდგმება კარადაში თაროში. ყოველ კარადაში 3 ასეთი თაროა. საერთო ჯამში ერთ კარადაში განთავსებულია 96 Opteron პროცესორი, საჭირო კვების წყარო და გაგრილების სისტემა. მთელი სისტემა შედგება 108 კარადისაგან, რაც საერთო ჯამში იძლევა 10368 პროცესორს SDRAM მეხსიერების მოდულებით მოცულობით 10T ბაიტი. ყოველ პროცესორს გააჩნია წვდომა მხოლოდ მეხსიერების საკუთარ SDRAM მოდულთან. სისტემაში საერთო მეხსიერება არ არსებობს. სისტემის თეორიული გამოთვლითი სიმძლავრე არის 41 ტერაფოპსი/წმ.

ცალკეული Opteron პროცესორების თანამოქმედებისათვის გამოიყენება სპეციალიზებული Seastar მარშრუტიზატორები, ერთი მარშრუტიზატორი ერთ პროცესორზე. ისინი ერთმანეთთან დაკავშირებულია სამგანზომილებიანი ტორით, რომლის ზომებიცაა 27×16×24, ყოველ კვანძში მოთავსებულია ერთი მარშრუტიზატორი. ყოველ მარშრუტიზატორს გააჩნია 7 ორმხრივი მაღალსიჩქარიანი (24 Gბიტი/წმ) კავშირის ხაზი. მათგან 6 დაკავშირებულია მეზობლებთან: ჩრდილოეთით, აღმოსავლეთით, სამხრეთით, დასავლეთით, ზევით, ქვევით და მეშვიდე ხაზი კი მარშრუტიზატორს აკავშირებს თავის Opteron პროცესორთან. მეზობელ კვანძებს შორის მონაცემების გადაცემის დროა 2 მკწმ. სისტემაში გამოიყენება კიდევ ერთი ქსელი, რომელიც აგებულია 100 Mბიტ/წმ Ethernet ტექნოლოგიის საფუძველზე და გამოიყენება სისტემის მომსახურებისა და უზრუნველყოფისათვის.

სისტემაში გამოთვლებისათვის განკუთვნილ 108 კარადას ემატება კიდევ 16 კარადა, რომლებიც განკუთვნილია შეტანა–გამოტანის პროცესორებისათვის და მომსახურებისათვის. ყოველ მათგანში განთავსებულია 32 Opteron პროცესორი. ამ 512 პროცესორიდან 256 განკუთვნილია შეტანა–გამოტანისათვის, 256 კიდევ მომსახურებისთვის. სისტემაში გამოყენებულია აგრეთვე დისკური მეხსიერება, რომელშიც იქმნება RAID–მასივები. დისკური მეხსიერების ჯამური მოცულობაა 240 Tბაიტი.

მექანიკური გადამრთველების საშუალებით სისტემა შეიძლება გაიყოს ორ ნაწილად: საიდუმლო და არასაიდუმლო ნაწილებად. პროცესორების საერთო რაოდენობიდან 2688 პროცესორი ყოველთვის იმყოფება საიდუმლო სექციაში, 2688 პროცესორი კი არასაიდუმლო სექციაში. დანარჩენი 4992 გამომთვლელი პროცესორი შეიძლება გადართული იქნას სექციებს შორის როგორც ეს ნაჩვენებია ნახ. 3.10–ზე.



ნახ. 3.10. Red Storm სისტემა, ზედხედი

გამოთვლითი კვანძები იმართება გამარტივებული ბირთვით Catamount (“ველური კატა”). შეტანა–გამოტანის კვანძები იმართება ჩვეულებრივი OS Linux–ის საშუალებით, RAS კვანძებში კი გამოიყენება Linux–ის შეკვეცილი ვერსია.

ასეთი დიდი სისტემისათვის განსაკუთრებული მნიშვნელობა ენიჭება საიმედოობის საკითხებს. ყველა პლატაზე დაყენებულია სისტემის მომსახურებისათვის განკუთვნილი RAS–პროცესორი და სპეციალიზებული აპარატული საშუალებები.

მთელი სისტემა შენობაში იკავებს 2000მ^2. სისტემის ენერგომოხმარებაა 1,6 Მვატი, დისკური სისტემა მოიხმარს კიდევ 1 Მვატს, გაგრილების, ჰაერის კონდიციონირებისა და ვენტილაციის გათვალისწინებით სისტემის სრული ენერგომოხმარება შეადგენს 3,5 Მვატს. სისტემის აპარატული და პროგრამული უზრუნველყოვის ფასი შეადგენს 90 მილიონ დოლარს.

3.7. BlueGene/L დაRed Storm სისტემების შედარება

BlueGene/L და Red Storm სისტემები ერთმანეთს ბევრი რამითაც ჰგავს და ბევრი რამითაც განსხვაცდევა, ამიტომ საინტერესოა მათი ძირითადი პარამეტრების ერთმანეთთან შედარება. ეს შედარება ნაჩვენებია 3.1 ცხრილში.

ცხრილი 3.1. BlueGene/L და Red Storm სისტემების პარამეტრების შედარება

პარამეტრი	**BlueGene/L**	**Red Storm**
ცენტრალური პროცესორი	32–თანრიგიანი,Power PC	64–თანრიგიანი, Opteron
სიხშირე	700 MHz	2 GHz
გამომთვლელი პროცესორების რაოდენობა	65536	10368
პროცესორების რაოდენობა პლატაზე	32	4
პროცესორების რაოდენობა კარადაში	1024	96
გამომთვლელი კარადების რაოდენობა	64	108
მწარმოებლურობა ტერაფლოპსი/წმ	71	41
მეხსიერების მოცულობა ერთ პროცესორზე	512 Მბაიტი	2–4 Გბაიტი
მეხსიერების საერთო მოცულობა	32 Tბაიტი	10 Tბაიტი
მარშრუტიზატორი	Power PC	Seastar
მარშრუტიზატორების რაოდენობა	65536	10368
ტოპოლოგია	ტორი ზომებით 64×32×32	ტორი ზომებით 27×16×24
დამატებითი ქსელები	Gigabit Ethernet	Fast Ethernet
სექციებად დაყოფის შესაძლებლობა	არა	კი
ოპერაციული სისტემა გამოთვლითი	სპეციალიზებული	სპეციალიზებული

კვანძებისათვის		
ოპერაციული სისტემა შეტანა–გამოტანის კვანძებისათვის	Linux	Linux
დამპროექტებელი	IBM	Cray Research
ფასი	მაღალი	მაღალი

ეს ორი სისტემა შექმნილი იქნა დაახლოებით ერთიდაიგივე დროს და ამიტომ, განსხვავებები მათ პარამეტრებში გამოწვეულია არა განსხვავებული ტექნოლოგიებით, არამედ განსხვავებული დასმული ამოცანებით და დამპროექტებლის განწყობილებით. სისტემა BlueGene/L თავიდანვე შიქმნა როგორც კომერციული კომპიუტერი, ხოლო Red Storm კი შექმნილი იქნა ლაბორატორია Sandia–ს ინდივიდუალური დაკვეთით.

IBM–ის მიდგომა ძალიან მარტივია: არსებული ბირთვებისგან ააგოს სპეციალიზირებული, მაგრამ დაბალსიჩქარიანი და მასობრივი წარმოებისათვის იაფი მიკროსქემა და შემდეგ ამ მიკროსქემების ძალიან დიდი რაოდენობა გააერთიანოს არცთუ მაღალი სიჩქარის ქსელით. სხვა, მაგრამ მსგავსი მიდგომა აირჩია კომპანიამ Sandia: აღებული იქნა გაყიდვაში არსებული პროცესორებიდან ყველაზე ძლიერი 64–თანრიგიანი პროცესორი, აღჭურვილი იქნა მაღალმწარმოებლური მარშრუტიზატორით და მეხსიერების დიდი მოცულობით, რის შედეგადაც მიღებული იქნა BlueGene კვანძებთან შედარებით გაცილებით უფრო ძლიერი გამოთვლითი კვანძები. ასეთი კვანძების საჭიროება გაცილებით უფრო ნაკლებია და შესაბამისად მათ შორის ინაფორმაციის გაცვლა განხორციელდება უფრო სწრაფად.

ორივე გადაწყვეტილებამ ელემენტების განთავსებაზე მოახდინა თავისი გავლენა. იმის გამო, რომ IBM–მა დაამუშავა სპეციალიზირებული მიკროსქემა, რომელშიც გაერთიანებული იქნა პროცესორი და მარშრუტიზატორი, მიღწეული იქნა კომპაქტურობის უფრო მაღალი მაჩვენებელი – კარადაში განთავსებული იქნა 1024 პროცესორი. სისტემაში Sansia ყოველ კვანძში განთავსებული იქნა ჩვეულებრივი მასიური პროცესორი და მეხსიერება მოცულობით 2–4 Gბაიტი, ამიტომ კარადაში განთავსებული იქნა 96 გამომთვლელი პროცესორი. შედეგად, სისტემა Red Stormიკავებს უფრო მეტ ფართს და მოიხმარს უფრო მეტ ენერგიას ვიდრე სისტემა BlueGene/L.

გამოთვლითი სისტემებისათვის ძალიან მნიშვნელოვან პარამეტრად ითვლება მწარმოებლურობა. ამ პარამეტრის მიხედვით იგებს სისტემა BlueGene/L, რადგანაც მისი მწარმოებლურობა არის 71 ტერაფლოპსი წამში, ხოლო Red Storm სისტემის მწარმოებლურობა კი არის 41 ტერაფლოპსი წამში. აქ გასათვალისწინებელია ისიც, რომ სისტემა Red Storm გაფართოებადია და კიდევ დამატებით 10 368 Opteron პროცესორის დამატებიათ მისი მწარმოებლურობა შეიძლება გაიზარდოს 82 ტერაფლოპს/წმ–მდე. IBM–ის სისტემაში ამის საპასუხოდ შეიძლება გაზრდილი იქნას ტაქტური სიხშირე (დღეისათვის 700 MHz სიხშირე დიდი არაფერია). მოკლედ, დასკვნის გაკეთება რთულია, შედეგი კი არის ის, რომ MPP–სისტემების განვითარება კიდევ დიდ ხანს გაგრძელდება.

3.8. კლასტერული გამოთვლები

მულტიკომპიუტერების მეორე ვარიანტია **კლასტერული კომპიუტერები**. როგორც წესი, კლასტერი შედგება ასობით ან ათასობით ქსელის საშუალებით ერთმანეთთან დაკავშირებული

კომპიუტერებისაგან. MPP სისტემებსა და კლასტერებს შორის ისეთივე განსხვავებაა, როგორც მაინფრეიმებსა და პერსონალურ კომპიუტერებს შორის. მათგან ორივეს გააჩნია პროცესორი, ოპერატიული მეხსიერება, დისკები, ოპერაციული სისტემა და ა.შ. მაგრამ, მაინფრეიმში ყველაფერი ეს მუშაობს გაცილებით უფრო სწრაფად. იგივე შეიძლება ითქვას MPP სისტემებისა და კლასტერების შესახებაც.

რამდენიმე წლის წინ MPP სისტემის ელემენტებს შორის ურთიერთქმედება ხდებოდა გაცილებით უფრო სწრაფად, ვიდრე კომპიუტერებს შორის, მაგრამ, მაღალსიჩქარიანი ქსელების შექმნის შემდეგ ეს უპირატესობა შემცირდა. უკვე გამოიკვეთა კლასტერების მიერ MMP სისტემების შევიწროების ტენდენცია, ისევე როგორც პერსონალურმა კომპიუტერებმა შეავიწროვეს მაინფრეიმები, რომლებიც ამჟამად გამოიყენება სპეციალური დანიშნულებით. ამჟამად MMP სისტემების გამოყენება ხდება იმ ამოცანებისთვის, რომელთათვისაც მთავარი მიზანია მაღალი მწარმოებლურობა, ხოლო სისტემის ფასს გადამწყვეტი მნიშვნელობა არ გააჩნია.

არსებობს კლასტერების მრავალი ნაირსახეობა, რომელთაგან დომინირებს ორი: **ცენტრალიზებული** და **დეცენტრალიზებული**. ცენტრალიზებულ კლასტერში გაერთიანებული სამუშაო სადგურები და პერსონალური კომპიუტერები ჩაშენებულია დიდ კონსტრუქციებში, რომლებმაც შეიძლება დაიკავოს საკმაოდ დიდი ოთახი. მათში მინიმუმამდეა დაყვანილი ფიზიკური ზომები და კაბელების სიგრძე. ცენტრალიზებულ კლასტერში შემავალი ყოველი კომპიუტერი ჰომოგენურია და არ გააჩნია არანაირი პერიფერიული მოწყობილობა, გარდა საქსელო პლატებისა და შეიძლება გააჩნდეთ დისკები.

დეცენტრალიზებული კლასტერები კი შედგება შენობის მასშტაბით გაბნეული სამუშაო სადგურებისა და პერსონალური კომპიუტერებისაგან. მათგან უმეტესობა მთელი დღის განმავლობაში საკმაოდ დიდი ხანია უქმად, განსაკუთრებით ღამის საათებში. ეს კომპიუტერები ერთმანეთთან დაკავშირებულია ლოკალური ქსელის საშუალებით. ყოველი კომპიუტერი ჰეტეროგენულია გააჩნია პერიფერიული მოწყობილობების სრული კომპლექტი. განსაკუთრებით მნიშვნელოვანია, რომ ყოველ კომპიუტერს ჰყავს პატრონი, რომელსაც თვითონაც შეიძლება დასჭირდეს თავისი კომპიუტერი. ამიტომ, კლასტერის შექმნა ხდება მოცემული მომენტისათვის თავისუფალი კომპიუტერებისაგან. დეცენტრალიზებულ კლასტერში აუცილებლად უნდა იყოს გათვალისწინებული დავალებების მიგრაციის მექანიზმი, რათა გაანთავისუფლოს კომპიუტერი, როდესათ თავის პატრონს დასჭირდება და მისი დავალება გადასცეს სხვა განთავისუფლებულ პროცესორს. მიგრაციის პრობლემის გადაწყვეტა ხდება პროგრამული უზრუნველყოფის გართულების ხარჯზე.

კლასტერების უმეტესობა მოიცავს 500–მდე კომპიუტერს. მაგრამ, შესაძლებელია ძალიან დიდ ზომის კლასტერების აგებაც. კომპანია Google–ს მიერ შემოთავაზებული იქნა ძალიან საინტერესო იდეა.

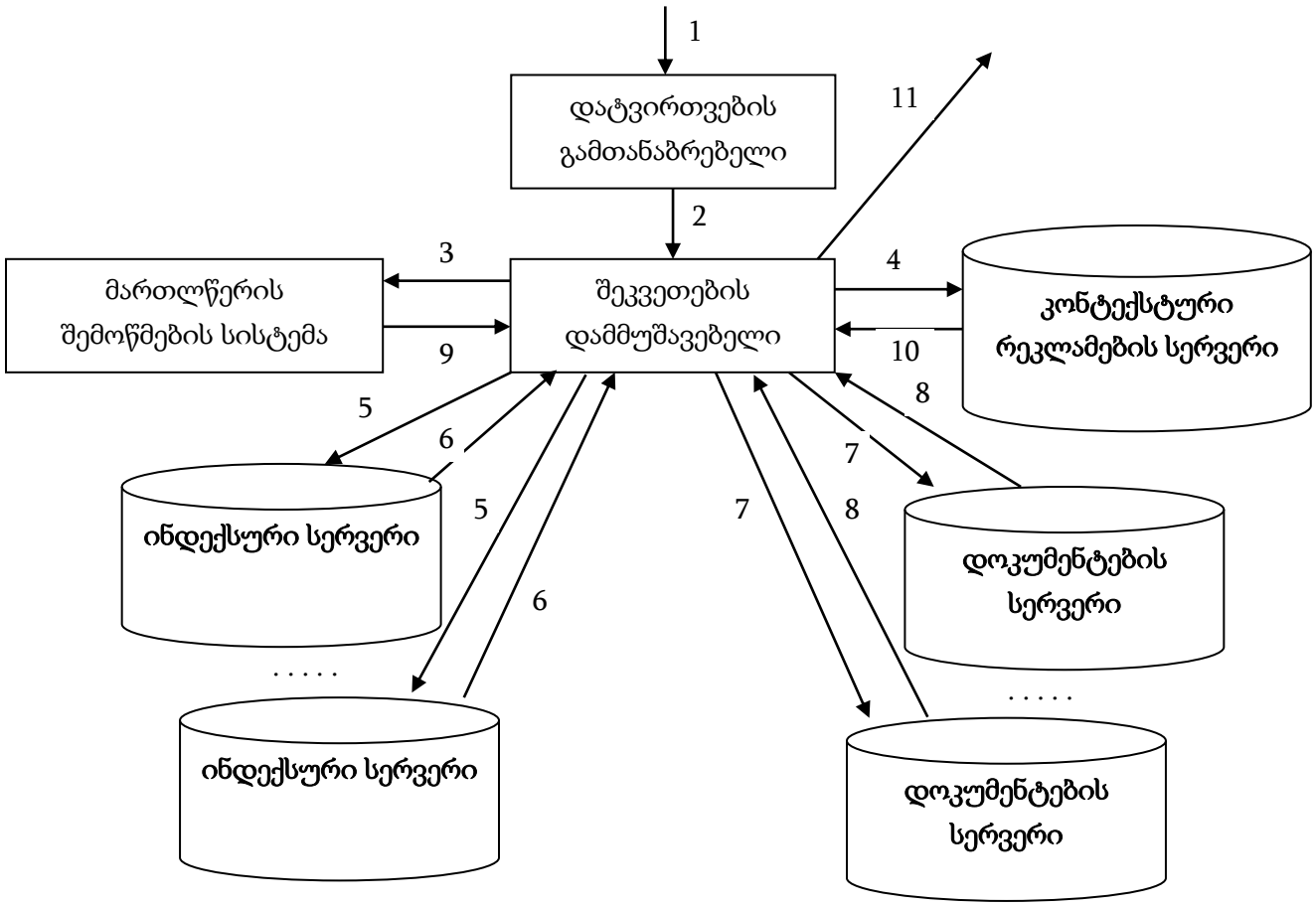
3.9. Google

Google არის ძალიან პოპულარული სისტემა ინტერნეტში ინფორმაციის მოძიებისათვის. ეს პოპულარობა განპირობებულია ინტერფეისის სიმარტივით და რეაქციის ძალიან მცირე დროით, რაც მიღწეულია მისი ორგანიზებისათვის ძალიან ძლიერი აპარატული და პროგრამული

უზრუნველყოფის გამოყენებით. საძიებო სისტემის თალთახედვით ამოცანა მდგომარეობს იმაში, რომ ინტერნეტში არსებული ყოველგვარი ინფორმაცია უნდა იქნას ინდექსირებული და შენახული. ინტერნეტში არსებული ინფორმაცია კი არის მილიარდობით გვერდი და სურათი. Google ამ ზღვა ინფორმაციაში საჭირო გვერდის მოძიებას ახერხებს 0,5 წამში, ამუშავებს წამში ათასზე მეტ შეკვეთას. Google–ის სისტემა არ უნდა გაითიშოს ბუნებრივი კატაკლიზმების დროსაც კი.

როგორ ახერხებს Google ყოველივე ამას? Google–ს ფუნქციონირებას ახორციელებს მთელ მსოფლიოში განაწილებული უამრავი საინფორმაციო ცენტრი. www.google.com მისამართთან მიმართვის დროს ხდება შეკვეთის გამგზავნის IP მისამართის ანალიზი, რის შემდეგაც ბრაუზერი მიმართავს მის უახლოეს საინფორმაციო ცენტრს.

ყოველი საინფორმაციო ცენტრი ინტენეტთან მიერთებულია მინიმუმ ერთი OC-48 (2,488 Gბაიტი/წმ) ოპტიკურ ბოჭკოვანი ხაზით, რომლითაც მოედინება შეკვეთები და ხდება მათზე პასუხის გაცემა. გარდა ამისა, სერვისების სარეზერვო მომწოდებელთან არსებობს დამატებითი ხაზი OC-12 (622 Mბაიტი/წმ), იმ შემთხვევისათვის თუ მოხდება ძირითადი ხაზის მტყუნება. იმ შემთხვევაში, თუ ელექტრო მომარაგების სისტემის მტყუნების შედეგად მოხდება ძირითადი ხაზის მტყუნება, ყოველ ცენტრში არსებობს უწყვეტი კვების წყარო და საავარიო დიზელის გენერატორები. ამგვარად, ბუნებრივი კატაკლიზმების შედეგად Google სისტემის ავარიული გამორთვა არ ხდება, თუმცა კი მისი მწარმოებლურობა მცირდება.



ნახ. 3.11 შეტყობინებების დამუშავება Google სისტემაში

იმისათვის, რომ უკეთ გავიგოთ, თუ რატომ გახდა Google–ს არქიტექტურა ისეთი, როგორი სახეც მას დღეს აქვს, საჭიროა გავეცნოთ შეკვეთების დამუშავების მექანიზმს, რომელიც ხდება ინფორმაციის დამუშავების ნებისმიერ ცენტრში. ინფორმაციის დამუშავების ცენტრში შეღწევის შემდეგ (ნახ. 3.11 ბიჯი 1) დატვირთვის გამთანაბრებლის მიერ ის გადაეგზავნება შეკვეთის დამუშავების უამრავიდან ერთ–ერთ ცენტრს (ბიჯი 2). ამის პარალელურად შეტყობინება მიეწოდება მართლწერის შემოწმების ცენტრს (ბიჯი 3) და კონტექსტური რეკლამის ცენტრს (ბიჯი 4). ამის პარალელურად ხდება აკრძალური სიტყვის ძიება ინდექსურ სერვერებზე (ბიჯი 5), რომლებზეც ინახება ჩანაწერები ქსელში არსებული ყოველი სიტყვის შესახებ.

ყოველ ასეთ ჩანაწერში ჩამოთვლილია ამ სიტყვის შემცველი ყოველი დოკუმენტი (ეს შეიძლება იყოს Web–გვერდები, PDF-ფაილები, PowerPoint პრეზენტაციები და ა.შ.). ამ სიებში მიმთითებლები მოთავსებულია გვერდების რეიტინგების მიხედვით, რომლის პარამეტრებიც გამოითვლება საკმაოდ რთული ფორმულის მიხედვით. რეიტინგის გამოთვლის პრინციპი საიდუმლოს წარმოადგენს, მაგრამ ცნობილია, რომ დიდი მნიშვნელობა ენიჭება გვერდზე მითითებების რაოდენობას და ამ გვერდებზე მიმთითებელი გვერდების რეიტინგს.

მწარმოებლურობის გაზრდის მიზნით ინდექსები დაყოფილია ფრაგმენტებად და მათ შორის ძიება წარმოებს პარალელურად. ამ იდეის თანახმად, ფრაგმენტი 1 ინდექსიდან მოიცავს ყველა სიტყვას და ყოველ სიტყვას მისადაგებული აქვს რეიტინგის მიხედვით პირველი n გვერდის იდენტიფიკატორები. ფრაგმენტი 2 მოიცავს გვერდების რეიტინგების მიხედვით შემდეგი n გვერდის სიტყვებსა და იდენტიფიკატორებს და ა.შ. ქსელის ზომების გაზრდის შესაბამისად ამ ფრაგმენტებიდან ყოველი შეძლება დაყოფილი იქნას რამდენიმე ნაწილად ისე, რომ პირველ ნაწილში მოთავსებული იქნება პირველი k სიტყვა, მეორე ნაწილში შემდეგი k სიტყვა და ა.შ. ამის შედეგად იზრდება ძებნის პროცესში პარალელიზმის დონე.

ინდექსური სერვერები აბრუნებენ დოკუმენტების იდენტიფიკატორების ნაკრებებს (6), შემდგომში შეკვეთების ლოგიკის შესაბამისად ხდება მათი კომბინირება. მაგალითად: +digita +kapibara + dance.

ასეთი შეკვეთის შემთხვევაში შემდეგ ბიჯზე მოხვდება მხოლოდ იმ დოკუმენტების იდენტიფიკატორები, რომლებიც არსებობს სამივე ნაკრებში. ამ ნაბიჯზე Google მიმართავს თვითონ დოკუმენტებს (7), ამოიღებს მათგან მათ დასახელებებს, მითითებებს და აგრეთვე სიტყვებით მოთხოვნილი ტექსტების ფრაგმენტებს. ქსელის ბევრი დოკუმენტის ასლი ინახება ყველა საინფორმაციო ცენტრის დოკუმენტების სერვერებზე. მათი მოცულობა ათასობით ტერაბაიტია. პარალელური ძებნის დაჩქარების მიზნით დოკუმენტები დაყოფილია ფრაგმენტებად. შედეგად მიიღება ის, რომ შეკვეთის დამუშავებისათვის არაა საჭირო ქსელში არსებული მთელი ინფორმაციის გადასინჯვა და ასობით ტერაბაიტი ინდექსების გადამუშავება. მაგრამ, რიგითი შეტყობინების დამუშავებისათვის მაინც საჭირო ხდება ასეულობით მეგაბაიტი ინფორმაციის დამუშავება.

ძებნის შედეგები უბრუნდება შეკვეთების დამმუშავებელს (8) და ხდება მათი გაერთიანება გვერდების რეიტინგების შესაბამისად. თუ აღმოჩენილი იქნება მართლწერის შეცდომები (9), დაემეტება ინფორმაცია ამის შესახებ და კონტექსტური რეკლამა (10). შეკვეთის დამუშავების შედეგებში ხდება გასაღები სიტყვების ჩადება (მაგ. „სასტუმრო“), სწორედ მათ ყიდულობენ

რეკლამების მიმწოდებლები და აქედან იქმნება Google–ს შემოსავალი. ბოლოს ხდება შედეგების გაფორმება HTML ფორმატში და გადაეცემა მომხმარებელს ჩვეულებრივი web–გვერდის სახით.

Google–ს საქმიანობის აღწერის შემდეგ, უკვე შესაძლებელია მისი სისტემის არქიტექტურის აღწერა. კომპანიების უმრავლესობა, რომლებსაც სჭირდება ძალიან დიდი და საიმედო მონაცემების ბაზების შექმნა და მათთან მუშაობა, ღებულობს გადაწყვეტილებას და ყიდულობს ბაზარზე არსებულ ყველაზე სწრაფ და საიმედო მოწყობილობებს. Google კი მოიქცა პირდაპირ საპირისპიროდ, შეიძინა საშუალო შესაძლებლობების იაფი პერსონალური კომპიუტერები, მაგრამ დიდი რაოდენობით. მათი გაერთიანებით Google–მა შექმნა მსოფლიოში ყველაზე დიდი კლასტერი. ასეთი გადაწყვეტილებით მიღწეული იქნა ფასი/მწარმოებლურობა შეფარდების ოპტიმიზირება.

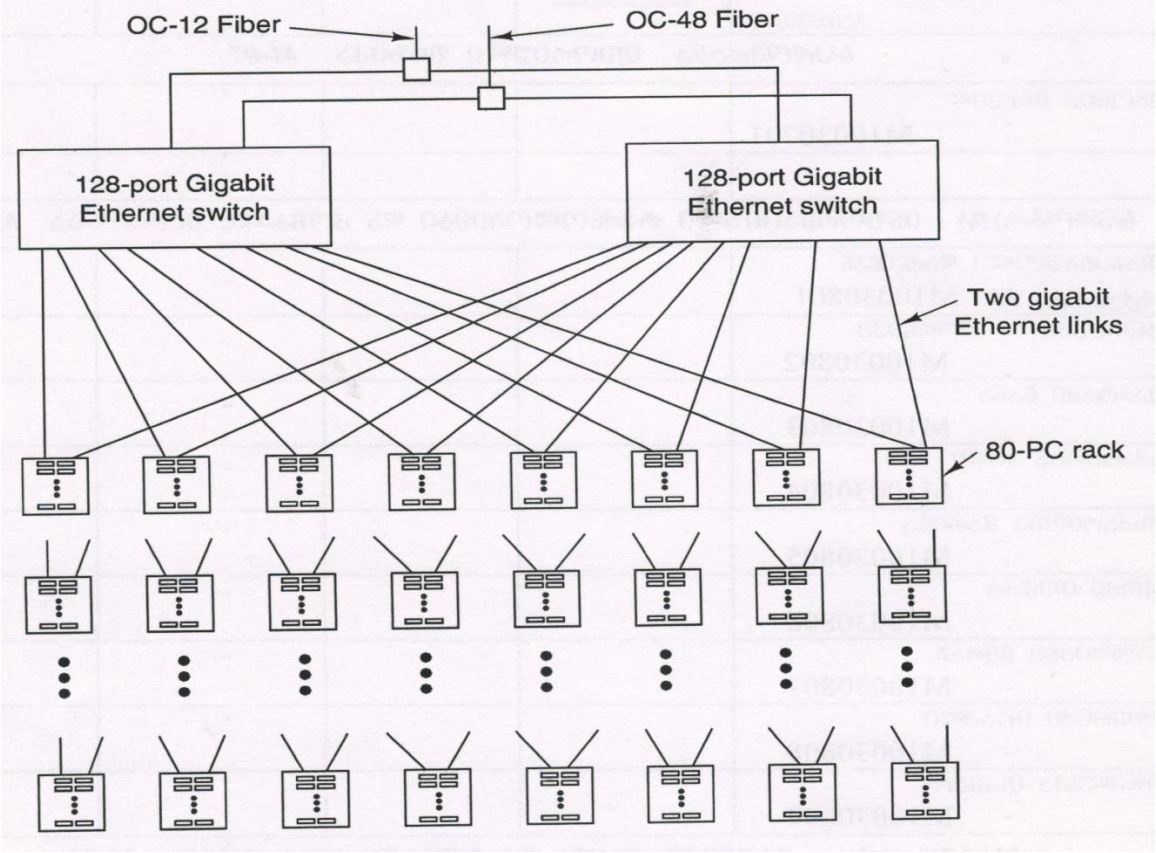
ეს გადაწყვეტილება განპირობებულია ეკონომიკური მოსაზრებით: ჩვეულებრივი პერსონალური კომპიუტერები საკმაოდ იაფია, სერვერები ძვირია, ხოლო მულტიპროცესორები კი ბევრად უფრო ძვირი. მაგალითად, სერვერის მწარმოებლურობა 2–3 ჯერ აღემატება საშუალო პერსონალური კომპიუტერის მწარმოებლურობას, მაგრამ ფასით კი 5–10 ჯერ აღემატება მის ფასს.

ჩვეულებრივი პერსონალური კომპიუტერი საიმედობით ბევრად ჩამოუვარდება სერვერს, მაგრამ მტყუნებები მათშიც ხომაა სესაძლებელი? ამიტომ, Google–ს პროგრამული უზრუნველყოფა დაწერილია ისეთნაირად, რომ მოხდეს ნაკლებადსაიმედო ელემენტებით საიმედო მუშაობა. აქვს რა მტყუნებამედეგი პროგრამული უზრუნველყოფა, უკვე აღარა აქვს მნიშვნელობა მტყუნებების ინტენსივობა წელიწადში იქნება 0,5% თუ 2%. Google–ს გამოცდილება აჩვენებს, რომ წელიწადში კომპიუტერების 2% გამოდის მწყობრიდან. მტყუნებების ნახევარზე მეტი მოდის მაგნიტურ დისკებზე. მტყუნების შემდეგი მიზეზია კვების ბლოკი და შემდეგი მეხსიერების მიკროსქემები. პროცესორების მტყუნება ძალიან იშვიათია. სინამდვილეში მტყუნებების ძირითად მიზეზს წარმოადგენს არა აპარატურა, არამედ პროგრამული უზრუნველყოფა. ამიტომ, მტყუნებაზე პირველი რეაქციაა სისტემის გადატვირთვა, რომელიც უმეტეს შემთხვევებში ამართლებს.

Google თავის სისტემაში იყენებს საშუალო სიმძლავრის პერსონალურ კომპიუტერებს. უფრო მეტ მნიშვნელობას ანიჭებს Ethernet მიკროსქემას, მაგრამ არც ისაა ძალიან ძვირფასი. კომპიუტერები ორრიგადაა მოთავსებულია კონსტრუქციაში, წინა და უკანა რიგებში 40–40 ცალი. ერთ კონსტრუქციაში მოთავსებული 80 კომპიუტერი კარადაში არსებული კომუტატორის საშუალებით მიერთებულია Ethernet–თან. ერთ საინფორმაციო ცენტრში ყველა კარადა გაერთიანებული ასევე კომუტატორით და Ethernet–ით. მაღალი სიცოცხლისუნარიანობის მისაღწევად ყოველ კომუტატორთან გათვალისწინებულია კიდევ 2 სათადარიგო კომუტატორი.

Google–ს ტიპიური საინფორმაციო ცენტრის სტრუქტურა ნაჩვენებია ნახ. 3.12–ზე. მონაცემები ორ 128–პორტიან Ethernet კომუტატორს მიეწოდება ოპტიკურ–ბოჭკოვაკი კავშირის არხით OC-48. კომუტატორებთან ანალოგიურად მიერთებულია სარეზერვო კავშირის არხი OC-12. შემომავალი არხების მიერთებისათვის გამოყენებულია სპეციალური პლატა, რის შედეგადაც არ ხდება კომუტატორის პორტების მოკავება. ყოველი კარადიდან გამოდის ოთხი Ethernet კავშირის არხი, მათგან ორი უერთდება მარცხენა კომუტატორს და მეორე წყვილი კი მარჯვენა კომუტატორს. ამიტომ სისტემის სიცოცხლისუნარიანობა არ დაირღვევა რომელიმე კომუტატორის მწყობრიდან გამოსვლის შემთხვევაში. 4 კავშირის არხის არსებობის შედეგად კარადასთან კავშირი შეიძლება დაიკარგოს მხოლოდ ოთხივივე კავშირის არხის დაზიანების შედეგად, ან თუ ერთდ-

როულად დაზიანდა 2 კავშირის არხი და ერთი კომუტატორი. ორი 128–პორტიანი კომუტატორის და 4 კავშირის არხის მქონე კარადებით სისტემაში შეიძლება 64 კარადის გაერთიანება. თუ გავით-ვალისწინებთ, რომ ყოველ კარადაში 80 კომპიუტერია, საერთო ჯამში ერთ საინფორმაციო ცენტრში გამოდის, რომ გაერთიანებულია 5120 კომპიუტერი. კარადაში არ არის აუცილებელი, რომ მაინცდამაინც 80 კომპიუტერი იყოს, კომუტატორსაც შეიძლება ჰქონდეს 128 პორტზე მეტი, მაგრამ მოყვანილი რიცხვები ტიპიურია Google–ს საინფორმაციო ცენტრებისათვის.



ნახ. 3.12. Google-ს ტიპიური კლასტერი

დიდი მნიშვნელობა ენიჭება ენერგომომარაგების საკითხსაც. ტიპიური პერსონალური კომპიუტერი მოიხმარს დაახლოებით 120 ვატ ენერგიას, ერთ კარადაზე ეს იძლევა 10 კილოვატს. იმისათვის, რომ მომსახურე პერსონალს შეეძლოს კომპიუტერების მომსახურება, ერთ კარადაზე უნდა მოდიოდეს 3 $მ^2$ ფართი. 1 $მ^2$–ზე მოხმარებული ენერგია გამოდის 300 ვატი/$მ^2$. ეს საკმაოდ მაღალი მაჩვენებელია და საჭირო ხდება გაგრილებისა და ვენტილაციის სპეციალური სისტემის გამოყენება.

Google–ში კარგად აითვისეს დიდი web–სერვერების აგებისა და მომსახურების 3 წესი:

1. ყოველი კომპონენტი შეიძლება დაზიანდეს და ამას გათვალისწინება სჭირდება;
2. მაღალი გამტარუნარიანობისა და საიმედობის მისაღწევად საჭიროა ყველაფრის დუბლირება;
3. საჭიროა ფასი/მწარმოებლურობა შეფარდების ოპტიმიზირება.

პირველი წესის თანახმად სისტემის პროგრამული უზრუნველყოფა უნდა იყოს მტყუნებამედეგი. ყველაზე საუკეთესო მოწყობილობებიც კი ადრე თუ გავის გამოვლენ მწყობრიდან და ეს მოვლენა გათვალისწინებული უნდა იქნას პროგრამულ უზრუნველყოფაში. ასეთი დიდი ზომის სისტემამ უნდა აიტანოს მტყუნებები, თუკი ეს მტყუნებები რამდენიმეჯერაც კი ხდება კვირაში.

მეორე წესი მიუთითებს იმაზე, რომ აპარატული უზრუნველყოფაც და პროგრამული უზრუნველყოფაც უნდა იყოს ჭარბი. ეს ზრდის სისტემის არა მარტო საიმედობას, არამედ მის გამტარუნარიანობასაც. Google–ის შემთხვევაში კომპიუტერებიც, მაგნიტური დისკებიც, კავშირის არხებიც, კვების ბლოკებიც მრავალჯერაა დუბლირებული. უფრო მეტიც, ერთი ცენტრის ფარგლებში დუბლირებულია დოკუმენტებისა და ინდექსების ფრაგმენტები. დუბლირებულია თვითონ საინფორმაციო ცენტრებიც.

მესამე წესი წარმოადგენს წინა ორი წესის შედეგს. თუ სისტემა კარგად რეაგირებს მტყუნებებზე, მაშინ აზრი არა აქვს ძვირფასი მოწყობილობების შეძენას, ისეთების, როგორიცაა RAID–მასივები და SCSI–დისკები. როდისღაც ისინიც გამოდიან მწყობრიდან, ამიტომ 10–ჯერ მეტი თანხის გადახდა იმისათვის, რომ 2–ჯერ შევამციროთ მტყუნებების ინტენსივობა, ცუდი იდეაა. Google–მა აირჩია გზა: იყიდა 10–ჯერ მეტი აპარატურა და პროგრამულ უზრუნველყოფაში გაითვალისწინა მათი მტყუნებები. ბოლოსდაბოლოს რაც მეტია მოწყობილობები, მით მეტია მწარმოებლურობაც.

3.10. საკომუნიკაციო პროგრამული უზრუნველყოფა მულტიკომპიუტერებისათვის

მულტიკომპიუტერების დაპროგრამებისათვის საჭიროა სპეციალური პროგრამული უზრუნველყოფა (ჩვეულებრივ, ესაა ბიბლიოთეკები), რომელიც უზრუნველჰყოფს პროცესორების თანამოქმედებას და სინქრონიზაციას. უმეტეს შემთხვევებში პაკეტები განკუთვნილია MPP კომპიუტერებისათვის და კლასტერებისათვის, ამიტომ გამოყენებითი პროგრამები გათვლილია მათ პლატფორმებზე.

მონაცემების გადაცემის სისტემებში ორი და მეტი პროცესი მუშაობს ერთმანეთისაგან დამოუკიდებლად. მაგალითად, ერთი პროცესი შეიძლება ახდენდეს მონაცემების გენერირებას, მეორე (ან სხვები) კი მათ მოიხმარდნენ. მონაცემების გამგზავნს თუ აქვს გადასაცემი მონაცემები, არ არსებობს არანაირი გარანტია, რომ მიმღები (მიმღებები) მზადაა ამ მონაცემების მისაღებად, რადგანაც ყოველი პროცესი მუშაობს დამოუკიდებლად.

მონაცემების გადაცემის სისტემების უმრავლესობაში გამოყენებულია ორი პრიმიტივი send და receive, მაგრამ არსებობს სემანტიკის კიდევ სხვა ვარიანტებიც. სამი ძირითადი ვარიანტია:

- შეტყობინებების სინქრონული გადაცემა;
- შეტყობინებების გადაცემა ბუფერიზებით;
- შეტყობინებების არაბლოკირებადი გადაცემა.

შეტყობინებების სინქრონული გადაცემის დროს გადამცემმა თუ შეასრულა ოპერაცია send, ხოლო მიმღებმა არ შეასრულა ოპერაცია receive, მაშინ ხდება გადამცემის ბლოკირება (შეჩერება) მანამდე, სანამ მიმღები არ შეასრულებს receive ოპერაციას. როდესაც მართვა დაუბრუნდება

გადამცემს, მან უკვე იცის, რომ გაგზავნილი შეტყობინება მიღებულ იქნა. ამ მეთოდს გააჩნია მარტივი სემანტიკა და ბუფერიზაციას არ საჭიროებს. მაგრამ, მას გააჩნია სერიოზული ნაკლიც: შეტყობინების მიღებამდე და ამის დადასტურებამდე გამგზავნი ბლოკირებულია.

ბუფერიზებით შეტყობინებების გადაცემის შემთხვევაში ხდება გაგზავნილი შეტყობინების დროებით სადმე შენახვა (მაგალითად საფოსტო ყუთში), ვიდრე მიმღები მზად იქნება შეტყობინების საფოსტო ყუთიდან წასაღებად. ასეთი მიდგომის დროს გამგზავნს შეუძლია გააგრძელოს მუშაობა send ოპერაციის შესრულების შემდეგ, იმ შემთხვევაშიც კი თუ ამ მომენტში მიმღები დაკავებულია. რადგანაც შეტყობინება უკვე გაგზავნილია, გამგზავნს შეუძლია დაიწყოს ახალი ოპერაცია და ხელახლა დაიკავოს შეტყობინებების ბუფერი. ასეთი სქემა ამცირებს დაყოვნების დროს, მაგრამ არანაირი გარანტია არ არსებობს იმისა, რომ მიმღებმა შეტყობინება მიიღო. საიმედო საკომუნიკაციო ქსელის არსებობის შემთხვევაშიც კი არსებობს იმის ალბათობა, რომ მტყუნების შედეგად დაიკარგოს შეტყობინება.

შეტყობინებების არაბლოკირებადი გადაცემის შემთხვევაში გადამცემი აკეთებს გამოძახებას და აგრძელებს მუშაობას. ბიბლიოთეკის დანიშნულებას წარმოადგენს ის, რომ ოპერაციულ სისტემას შეატყობინოს, რომ მან დაამუშაოს შეტყობინება როდესაც ამის საშუალება ექნება. შედეგად, გადამცემის ბლოკირება საერთოდ არ ხდება. ამ მეთოდის ნაკლი მდგომარეობს იმაში, რომ როდესაც send ოპერაციის შესრულების შემდეგ გამგზავნი აგრძელებს მუშაობას მას აღარ შეუძლია შეტყობინებების ბუფერის დაკავება, რადგანაც არსებობს იმის ალბათობა, რომ შეტყობინება ჯერ გაგზავნილი არ არის. გამგზავნმა რაღაც გზით უნდა შეიტყოს, რომ მას შეუძლია შეტყობინებების ბუფერის გამოყენება. მაგალითად, შეიძლება ჰკითხოს სისტემას. მეორე ვარიანტია – ბუფერის განთავისუფლების შემდეგ შეიქმნას სისტემური წყვეტა. ორივე ვარიანტი მოითხოვს რთულ პროგრამულ გადაწყვეტას.

3.11. შეტყობინებების გადაცემის ინტერფეისი

რამდენიმე წლის წინათ პაკეტი **PVM** (Parallel Virtual Machine - პარალელური ვირტუალური მანქანა) ითვლებოდა ყველაზე პოპულარულ პაკეტად მულტიკომპიუტერებს შორის ინფორმაციის გასაცვლელად. დღეს მის ადგილს იკავებს პაკეტი MPI (Message–Passing Interface – შეტყობინებების გადაცემის ინტერფეისი). MPI პაკეტი გაცილებით რთულია. მას გააჩნია ბიბლიოთეკების გაცილებით მეტი გამოძახებების მხარდაჭერა და გაცილებით მეტი პარამეტრი ყოველი გამოძახებისათვის. არსებობს MPI პაკეტის ორი ვერსია: MPI–1 და MPI–2.

MPI–1 პაკეტი, PVM–საგან განსხვავებით არ არის დაკავშირებული პროცესების შექმნასა და მართვასთან. პროცესები უნდა შექმნას თვითონ მომხმარებელმა ლოკალური სისტემური გამოძახებების საშუალებით. შექმნის შემდეგ პროცესებისაგან ხდება ჯგუფების ორგანიზება, რომელთა ცვლილებაც შემდეგ აღარ ხდება. სწორედ ასეთ ჯგუფებთან მუშაობს MPI.

MPI პაკეტს საფუძვლად უდევს ოთხი კონცეფცია: კომუნიკატორები, გადასაცემი მონაცემების ტიპი, საკომუნიკაციო ოპერაციები და ვირტუალური ტოპოლოგიები. **კომუნიკატორი** ეს არის პროცესების ჯგუფი პლიუს კონტექსტი. კონტექსტს უწოდებენ ჭდეს, რომლითაც ხდება რაიმეს იდენტიფიცირება (მაგალითად, შესრულების ფაზის). შეტყობინების გაგზავნის და მიღე-

ბის პროცესში კონტექსტი შეიძლება გამოყენებული იქნას იმისათვის, რომ არ მოხდეს შეტყობინებების არევა.

შეტყობინებებში გადაცემული მონაცემების ტიპები შეიძლება იყოს სხვადასხვა, მათ შორის სომბოლური, მოკლე, ჩვეულებრივი და გრძელი მთელები, მცოცავწერტილიანი ჩვეულებრივი და ორმაგი სიზუსტის ნამდვილი რიცხვები და ა.შ. გარდა ამისა, არსებული ტიპებიდან შეიძლება აგებული იქნას ახალი ტიპები.

MPI პაკეტს გააჩნია სხვადასხვა საკომუნიკაციო ოპერაციის მხარდაჭერა. ასე გამოიყურება შეტყობინების გაგზავნის ოპერაცია:

MPI_Send (&buffer, count, type, tag, source, comm, &staus)

ამ გამოძახებაში მიმღებს გადაეცემა ბუფერის შიგთავსი, რომელიც მონაცენების რაოდენობაში შეიცავს ელემენტების ტიპს – მონაცემების ტიპი. ტეგის ველი წარმოადგენს შეტყობინების ჭდეს; მიმღებს შეუძლია მიიღოს მხოლოდ ის შეტყობინებები, რომლებშიც არსებობს მოცემული ტეგი. უკანასკნელი ველი (კომუნიკატორი) უჩვენებს, თუ პროცესების რომელ ჯგუფს განეკუთვნება მიმღები. შეტყობინების მიღების დადასტურების შეტყობინებას აქვს შემდეგი სახე:

MPI_Send (&buffer, count, type, tag, source, comm, &staus)

ეს გამოძახება მიუთითებს იმაზე, რომ მიმღები ელოდება გამომგზავნისაგან შეტყობინებას მონაცემების ტიპის შესახებ *მონაცემების ტიპი მოცემული ტეგით.*

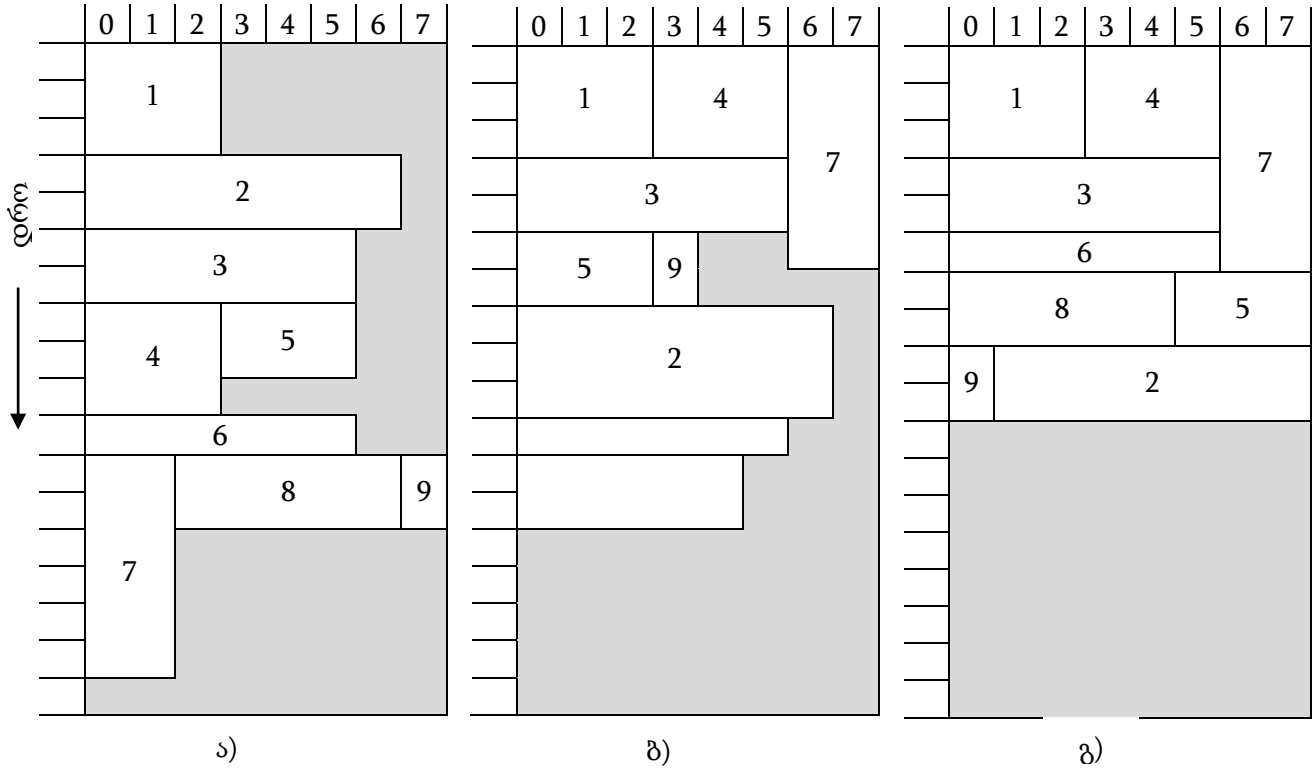
MPI–ს გააჩნია ოთხი საბაზო საკომუნიკაციო რეჟიმის მხარდაჭერა. პირველი რეჟიმი არის **სინქრონული**. მასში გამგზავნს არ შეუძლია მონაცემების გადაცემის დაწყება, ვიდრე მიმღები არ გააკეთებს გამოძახებას MPI_Recv. მეორე რეჟიმია **ბუფერიზაცია**. ამ რეჟიმში პირველი რეჟიმის შეზღუდვა არ მოქმედებს. მესამე რეჟიმია **სტანდარტული**. ის დამოკიდებულია კონკრეტულ რეალიზაციაზე, ანუ შეიძლება რეალიზებული იყოს ან სინქრონული რეჟიმი ან ბუფერიზაციის რეჟიმი. მეოთხე რეჟიმია **მზადყოფნა**. აქაც, ისევე როგორც პირველ რეჟიმში, გამგზავნი მოითხოვს, რომ მიმღებთან იყოს წვდომა. ყველა ამ პრიმიტივს გააჩნია როგორც ბლოკირებადი, ასევე არაბლოკირებადი ვერსიები, რაც ჯამში იძლევა 8 პრიმიტივს.

MPI–2 პაკეტშიდამატებულია მხარდაჭერა დინამიური პროცესების, მეხსიერებასთან დაშორებული წვდომის, არაბლოკირებადი კოლექტიური თანამოქმედების, მასშტაბირებადი შეტანა–გამოტანის, დროის რეალურ მასშაბში მუშაობის და სახვ. სპეციალისტებს შორის დიდხანს გრძელდებოდა კამათი MPI და PVM პაკეტების შესახებ. ორივეს ჰყავდა თავისი მომხრეები. PVM–ის მომხრეები ამტკიცებდნენ, რომ ამ სისტემის შესწავლა უფრო ადვილია და ადვილია მისი გამოყენება. MPI–ს მომხრეების არგუმენტი კი იყო ის, რომ MPI–ს გააჩნია უფრო მეტი ფუნქციების მხარდაჭერა და რომ ეს სისტემა უკვე სტანდარტიზებულია.

3.12. დაგეგმვა

MPI სისტემის გამომყენებელი პროგრამისტებისათვის არაა რთული დავალებების დამუშავება, რომლებშიც შეკვეთები ეგზავნებათ რამდენიმე პროცესორს ერთდროულად მათ შესრულებას სჭირდება საკმაოდ დიდი დრო. უფრო რთული შემთხვევაა როდესაც შეკვეთები მოდის რამდენიმე მომხმარებლისაგან ერთდროულად და მათი დამუშავება უნდა მოხდეს პროცესორების

სხვადასხვა რაოდენობაზე და სხვადასხვა დროში. ამიტომ კლასტერისათვის საჭიროა დამგეგმავი, რომელმაც უნდა განსაზღვროს რომელი დავალება როდის უნდა გაუშვას შესრულებაზე.



3.13. დავალებების შესრულება კლასტერში (რუხი ფერით მონიშნულია უქმად დარჩენილი პროცესორები): FIFO რიგით (ა); რიგის დასაწყისის ბლოკირების გარეშე (ბ); მართკუთხედის შევსებით კოორდინატთა სისტემაში პროცესორები–დრო (გ)

მარტივ მოდელში დავალებების დამგეგმავმა უნდა იცოდეს რომელი დავალების შესრულებას რამდენი პროცესორი სჭირდება. მარტივ მოდელში შეკვეთები იკავებენ რიგს FIFO წესით (ნახ. 3.13, ა). როდესაც იწყება პირველი დავალების შესრულება, დამგეგმავი განსაზღვრავს საკმარისია თუ არა თავისუფალი პროცესორების რაოდენობა რიგის მიხედვით შემდეგი დავალების შესასრულებლად. პროცსორების რაოდენობა თუ საკმარისია, მაშინ იწყება მეორე დავალების შესრულებაც და ა.შ. თავისუფალი პროცსორების რაოდენობა თუ საკმარისი არაა, მაშინ სისტემა იცდის ვიდრე განთავისუფლდება პროცესორების საჭირო რაოდენობა. განხილულ მაგალითში კლასტერი შეიცავს 8 პროცესორს. ასევე შესაძლებელია კლასტერი შეიცავდეს 128 პროცესორს, რომლებიც დაჯგუფებულია ბლოკებად, თითოეულ ბლოკში 16 პროცესორი, ანუ 8 ბლოკი. ორივე შემთხვევაში მსჯელობა ერთნაირი იქნება. შესაძლებელია გამოყენებული იქნას პროცესორების ნებისმიერი კომბინაცია.

დავალებების დაგეგმვის უფრო რთულ ალგორითმში შესაძლებელია რიგის დასაწყისის ბლოკირებების თავიდან აცილება. ამ შემთხვევაში ხდება იმ დავალებების გამოტოვება, რომელთათვისაც არაა თავისუფალი პროცესორების საკმარისი რაოდენობა, ხოლო შესრულება ხდება რიგის მიხედვით იმ დავალებების, რომელთათვისაც არსებობს თავისუფალი პროცესორების საკმარისი რაოდენობა. ყოველი დავალების შესრულების შემდეგ ხდება რიგის შემოწმება FIFO წესით (ნახ. 3.13, ბ).

დავალებების დაგეგმვის ყველაზე რთული ალგორითმი მოითხოვს, რომ წინასწარ იყოს ცნობილი რომელი დავალების შესასრულებლად რამდენი პროცესორია საჭირო და რამდენი დრო დასჭირდება. ამ ინფორმაციის საფუძველზე დავალებების დამგეგმავს შეუძლია დავალებებით შეავსოს მართკუთხედი კოორდინატთა სისტემაში პროცესორები–დრო (ნახ. 3.13, გ). ეს მეთოდი განსაკუთრებით ეფექტურია იმ შემთხვევაში თუ დავალებები მიღებულია დღისით, ხოლო მათი შესრულება უნდა მოხდეს ღამის საათებში. ამ შემთხვევაში დამგეგმავი ფლობს სრულ ინფორმაციას, სამუშაო შეიძლება დაგეგმილი იქნას ოპტიმალურად.

3.13. საერთო მეხსიერება გამოყენებით დონეზე

მოყვანილი მაგალითებიდან ჩანს, რომ მასშტაბირებას მულტიპროცესორებზე უკეთ ექვემდებარება მოლტიკომპიუტერები. ამან გამოიწვია შეტყობინებების გადაცემის სისტემის შექმნა, მაგალითად MPI. ბევრ პროგრამისტს ეს მოდელი არ მოსწონს, მათ ურჩევნიათ საერთო მეხსიერებასთან მუშაობა, თუნდაც რეალურად ასეთი მეხსიერება არ არსებობდეს. ეს თუ შესაძლებელი გახდება, მაშინ: მოხდება გამოყენებული აპარატურის გაიაფება (თუნდაც ყოველი კვანძის დონეზე) და უფრო მოხერხებული გახდება დაპროგრამების პროცესი.

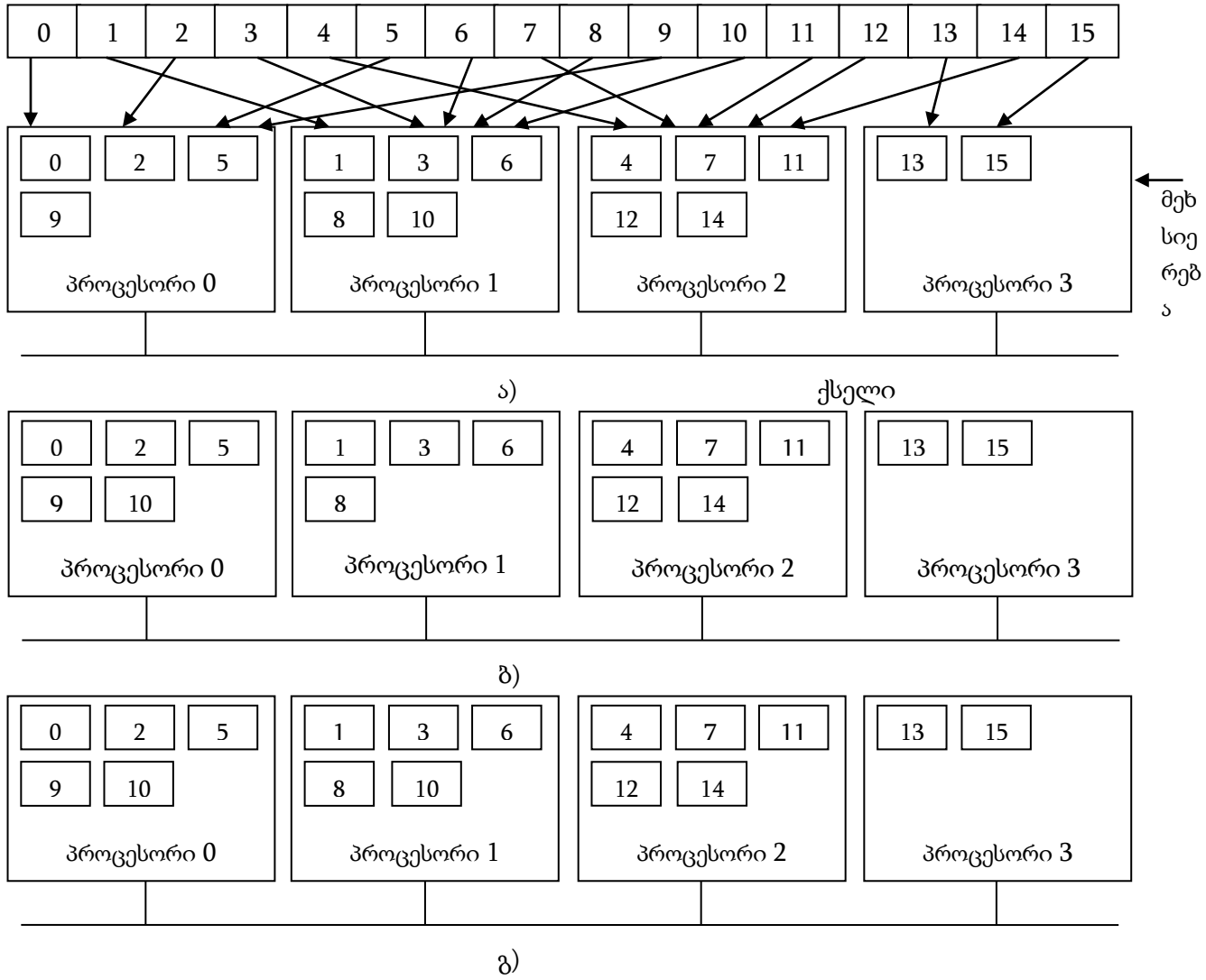
ბევრი მკვლევარი მივიდა იმ დასკვნამდე, რომ საერთო მეხსიერება არ არის აუცილებელი, რომ აგებული იქნას არქიტექტურულ დონეზე, არამედ ამ მიზნის მიღწევა შესაძლებელია სხვა გზებითაც. ნახ. 2.1.3–ზე ჩანს, რომ არსებობს რამდენიმე დონე, რომლებზეც შესაძლებელია საერთო მეხსიერების ორგანიზება.

3.13.1. განაწილებული საერთო მეხსიერება

გამოყენებით დონეზე საერთო მეხსიერების მქონე სისტემების კლასს განეკუთვნება სისტემები მეხსიერების ფუცლოვანი ორგანიზებით. სისტემების ეს კლასი ცნობილია აბრევიატურით **DSM** (Distributed Shared Memory – განაწილებული საერთო მეხსიერება). იდეა მარტივია: მულტიკომპიუტერის პროცესორები ერთობლივად გამოიყენებენ ვირტუალური ორგანიზების მქონე საერთო ვირტუალურ სამისამართო სივრცეს. ყველაზე მარტივი ვარიანტია – ყოველი გვერდი ინახება მხოლოდ ერთი პროცესორის ოპერატიულ მეხსიერებაში. ნახ 3.14,ა–ზე ჩანს საერთო ვირტუალური სამისამართო სივრცე, რომელიც 4 პროცესორს შორის განაწილებული 16 ვირტუალური გვერდისაგან შედგება.

პროცესორი როდესაც მიმართავს თავის ლოკალურ ოპერატიულ მეხსიერებაში გათავსებულ ფურცელს, წაკითხვისა და ჩაწერის პროცედურა ხორციელდება დაყოვნების გარეშე. თუ კი პროცესორი მიმართავს სახვა პროცესორის ოპერატიულ მეხსიერებას, წარმოიქმნება ფურცლის არქონის შეცდომა. ოპერაციული სისტემა, იმის მაგივრად, რომ არ არსებული ფურცელი მოძებნოს მაგნიტურ დისკზე, აგზავნის შეტყობინებას კვანძში, რომელშიც მოცემული ფურცელი იმყოფება, იმისათვის, რომ ამოიღოს ლოკალური სამისამართო სივრციდან და გააგზავნოს დანიშნულების მისამართით. ფურცლის მიღების შემდეგ ის კვლავ აისახება მეხსიერებაში და მოხდება შეჩერებული ბრძანების თავიდან გაშვება, ისევე როგორც ფურცლის არ არსების შემთხვევაში. ნახ. 3.14,ბ–ზე

ვხედავთ სიტუაციას, მას შემდეგ, რაც პროცესორმა 0 მიიღო ფურცლის არ არსებობის შეცდომა 10, რის შემდეგაც ის გადაეცა პროცესორი 1–დან პროცესორს 0.



ნახ. 3.4 მულტიკომპიუტერის 4 კვანძზე გადანაწილებული 16 ფურცლიანი ვირტუალური სამისამართო სივრცე: საწყისი მდგომარეობა (ა); პროცესორი 0–ის ფურცელ 10–თან მიმართვის შედეგად მიღებული მდგომარეობა (ბ); პროცესორი 1–ის მხოლოდ წაკითხვისათვის განკუთვნილ ფურცელ 10–თან მიმართვის შედეგად მიღებული მდგომარეობა (გ)

პირველად ეს იდეა რეალიზებული იქნა სისტემაში IVY. მასში მულტიოკომპიუტერს გააჩნია საერთო მეხსიერების პოლიფუნქციონალური მეხსიერების სეკვენციალური უნარიანობა. მწარმოებლურობის გაზრდის მიზნით შესაძლებელია ოპტიმიზაციის რამდენიმე ვარიანტის გამოყენება. პირველი ოპტიმიზაცია IVY–ში – მხოლოდ წაკითხვისათვის განკუთვნილი ფურც-ლები ერთდროულად შეიძლება ინახებოდეს რამდენიმე კვანძში. ფურცლის არქონის მიზეზით გამოწვეული შეცდომის შემთხვევაში საჭირო კომპიუტერში იგზავნება ამ ფურცლის ასლი, ხოლო ორიგინალი კი რჩება თავის ადგილზე, კონფლიქტის არანაირი საშიშროება არ არსებობს. ნახ. 3.14,გ–ზე ნაჩვენებია სიტუაცია, როდესაც ორი პროცესორი ერთდროულად გამოიყენებს ფურცელ 10–ს, რომელიც განკუთვნილია მხოლოდ წაკითხვისათვის. ასეთი ოპტიმიზაციის გზით ძნელია მაღალი მწარმოებლურობის მიღწევა, განსაკუთრებით ისეთ შემთხვევაში, როდესაც ერთი

პროცესი ფურცლის თავში წერს რამდენიმე სიტყვას, ხოლო მეორე პროცესი კი იგივე ფურცლის ბოლოში წერს რამდენიმე სიტყვას. რადგანაც დაშვებულია ფურცლის მხოლოდ ერთი ასლის არსებობა, საჭირო ხდება ამ ფურცლის წინ და უკან ხშირი გადაგზავნა. ასეთ სიტუაციას უწოდებენ **მოჩვენებით გაყოფას**.

მოჩვენებითი გაყოფის პრობლემის მოგვარება რამოდენიმენაირად შეიძლება. მაგალითად, შეიძლება უარი ვთქვათ სეკვენციალურ უნარიანობაზე და გადავიდეთ თავისუფალ უნარიანობაზე. პოტენციალურად ჩაწერისათვის გამოსადეგი ფურცლების თავისუფალი უნრიანობის შემთხვევაში ეს ფურცლები შეიძლება ერთდროულად არსებობდეს რამდენიმე კვანძში, მაგრამ ჩაწერის წინ პროცესმა განაცხადის გასაკეთებლად აუცილებლად უნდა შეასრულოს ოპერაცია acquire. ამ მომენტში ყველა არსებული ასლი გამოცხადდება არაუნარიანად და release ოპერაციის შესრულებამდე არანაირი ასლის შექმნა დაშვებული არ არის. release ოპერაციის შესრულების შემდეგ ამ ფურცელზე ისევ შესაძლებელი გახდება საყოველთაო წვდომა.

ოპტიმიზაციის მეორე ვარიანტის შემთხვევაში მეხსიერებაში თავდაპირველად ყოველი ფურცელი აისახება მხოლოდ წაკითხვის რეჟიმში. როდესაც ფურცელში ხდება პირველი ჩაწერა, სისტემა ქმნის ამ ფურცლის ასლს და უწოდებს მას **მეწყვილეს**. ამის შემდეგ საწყისი ფურცელი აისახება მეხსიერებაში წაკითხვისა და ჩაწერის დასაშვებ ფორმატში, რის შედეგადაც შემდგომი ჩაწერის ოპერაციები შეუფერხებლად სრულდება. თუ დაშორებულ კვანძში ფურცლის არ ქონის მიზეზით წარმოიშვება შეცდომა, მაშინ იქ უნდა გადაიგზავნოს მოდიფიცირებული გვერდი. ჯერ ხდება მიმდინარე ფურცლის მის მეწყვილესთან შედარება და შემდეგ დაშორებულ კვანძში მთელი ფურცლის მაგივრად იგზავნება მხოლოდ მოდიფიცირებული სიტყვები.

ფურცლის არქონით გამოწვეული შეცდომის შემთხვევაში საჭიროა იმის დადგენა თუ სად უნდა მოინახოს საჭირო ფურცელი. ამისათვის შეიძლება გამოყენებული იქნას რამოდენიმე მიდგომა, მაგალითად გამოყენებული იქნას კატალოგები ისე, როგორც ეს გაკეთებულია NUMA- და COMA–კომპიუტერებში. DSM–ში გამოყენებული ბევრი გადაწყვეტილება შეიძლება ასევე გამოყენებული იქნას NUMA- და COMA–კომპიუტერებში, რადგანაც DSM არის კომპიუტერების პროგრამული რეალიზება, რომლებშიც მეხსიერების ფურცელი გაიგივებულია კეშის სტრიქონთან. DSM დღესაც რჩება აქტიური კვლევების არენად.

3.14. მწარმოებლურობა

პარალელური კომპიუტერის შექმნის მიზანია, რომ მან ერთპროცესორიან კომპიუტერთან შედარებით გაცილებით სწრაფად იმუშაოს. ეს მიზანი თუ არ იქნება მიღწეული, მაშინ პარალელური კომპიუტერის შექმნას აზრი არა აქვს. უფრო მეტიც, პარალელურ კომპიუტერზე, რომელიც ჩვეულებრივ კომპიუტერზე 2–ჯერ უფრო სწრაფად მუშაობს, მაგრამ ღირს 50–ჯერ უფრო ძვირი, მოთხოვნილება ვერ იარსებებს. ამიტომ განვიხილოთ პარალელური კომპიუტერების არქიტექტურის მწარმოებლურობასთან დაკავშირებული რამდენიმე ასპექტი.

3.14.1. აპარატული კრიტერიუმები

აპარატურის პოზიციიდან გამომდინარე, უმთავრეს ინტერესს წარმოადგენს პროცესორების, შეტანა–გამოტანის მოწყობილობების და საკომუნიკაციო ქსელის სწრაფქმედება. პარალელურ კომ-

პიუტერებში პროცესორებისა და შეტანა–გამოტანის მოწყობილობების სწრაფქმედებასთან ერთად დიდი მნიშვნელობა ენიჭება საკომუნიკაციო ქსელის სწრაფქმედებას, მასზე დიდად არის დამოკიდებული მთელი სისტემის სწრაფქმედება. საკომუნიკაციო ქსელთან მიმართბაში ძირითადად განიხილება ორი კრიტერიუმი: დაყოვნების დრო და გამტარუნარიანობა. განვიხილოთ ისინი თანმიმდევრობით.

დაყოვნების სრული დრო, ან დაბრუნების დრო – ეს არის დრო, რომელიც საჭიროა იმისათვის, რომ პროცესორმა გააგზავნოს პაკეტი და მიიღოს სასზე პასუხი. თუ პაკეტი იგზავნება მეხსიერებაში, მაშინ დაყოვნების დრო არის სიტყვის ან სიტყვების ბლოკის მეხსიერებაში ჩაწერის ან წაკითხვისათვის საჭირო დრო. თუ პაკეტი ეგზავნება სხვა პროცესორს, მაშინ დაყოვნების დრო არის მოცემული ზომის პაკეტისათვის პროცესორებს შორის გადაგზავნისათვის საჭირო დრო. დაყოვნების დრო მნიშვნელოვანი მახასიათებელია მინიმალური ზომის პაკეტებისათვის (ერთი სიტყვა ან კეშ–მეხსიერების ერთი სტრიქონი).

დაყოვნების დროის სიდიდეს განსაზღვრავს რამდენიმე ფაქტორი და ეს დრო სხვადასხვაა არხების კომუტაციისათვის, კომუტაციისათვის შენახვით და ვირტუალური გამჭოლი კომუტაციისათვის. არხების კომუტაციის შემთხვევაში დაყოვნების დრო წარმოადგენს კავშირის დამყარებისა და გადაცემის დროების ჯამს. კავშირის დამყარებისათვის წინასწარ იგზავნება სასინჯი პაკეტი, ხდება საჭირო რესურსების დარეზერვება და გამგზავნს ეგზავნება შეტყობინება ანგარიშით. ამის შემდეგ შესაძლებელია მონაცემების პაკეტის შედგენა. როდესაც პაკეტი მზად იქნება, შესაძლებელია მისი სრული სიჩქარით გადაცემა. თუ კავშირის დამყარების დრო არის T_s, პაკეტის ზომა– p ბიტი, ხოლო გამტარუნარიანობა არის b ბიტი/წმ, მაშინ დაყოვნების დრო ერთი მიმართულებით გამოითვლება ფორმულით T_s+p/b. კავშირი თუ დუპლექსურია და საპასუხო შეტყობინებისათვის კავშირის დამყარება უკვე აღარაა საჭირო, მინიმალური დაყოვნების დრო გამოთვლება ფორმულით T_s+2p/b.

პაკეტების კომუტაციის შემთხვევაში ადრესატისათვის წინასწარი სასინჯი პაკეტის გაგზავნა საჭირო არაა, მაგრამ პაკეტის შექმნისათვის მაინც საჭიროა რაღაც T_aდრო. პაკეტის ერთი მიმართულებით გადაცემის დრო არის T_a+p/b, მაგრამ ამ დროის განმავლობაში პაკეტი მიაღწევს მხოლოდ პირველ კომუტატორამდე. პაკეტის კომუტატორში გავლას შემოაქვს კიდევ დამატებითი დაყოვნება T_d, შემდეგ ხდება მეორე კომუტატორზე გადასვლა და ა.შ. დაყოვნების T_d დრო თავის მხრივ წარმოადგენს დამუშავებისა და რიგში დაყოვნების დროების ჯამს. ქსელში თუ გამოიყენება n კომუტატორი, მაშინ ერთი მიმართულებით დაყოვნების დრო გამოითვლება ფორმულით $T_a+n(p/b+T_d)+p/b$, სადაც უკანასკნელი შესაკრები წარმოადგენს უკანასკნელი კომუტატორიდან მიმღებისათვის გაგზავნილი შეტყობინების გადაცემისათვის საჭირო დროს.

ვირტუალური გამჭოლი მარშრუტიზაციის შემთხვევაში დაყოვნების დრო ერთი მიმართულებით უკეთეს შემთხვევაში უახლოვდება მნიშვნელობას $Ta+p/b$, რადგანაც ამ შემთხვევაში კავშირის დამყარებისათვის სასინჯი პაკეტები არ გამოიყენება და ადგილი არა აქვს პაკეტების შუალედური შენახვით გამოწვეულ დროით დაყოვნებებს. დაყოვნების დრო ძირითადად წარმოადგენს პაკეტის შექმნისა და ბიტების გადაცემისათვის საჭირო დროების ჯამს. თეორიულად არსებობს კიდევ კავშირის არხში სიგნალის გავრცელებასთან დაკავშირებული დაყოვნება, მაგრამ ეს იმდენად მიცირე დროა, რომ პრაქტიკულად არაფერს არ ცვლის.

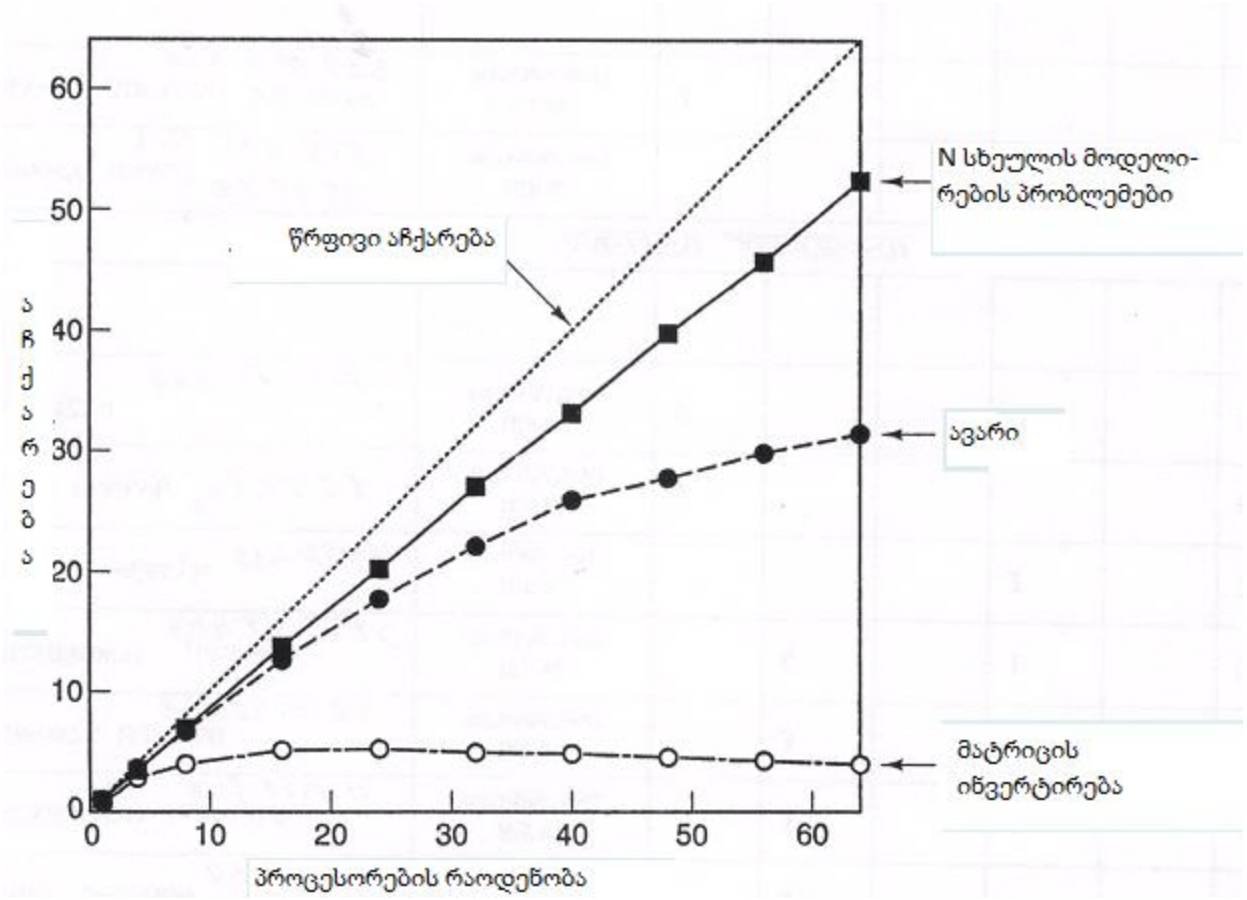
შემდეგი აპარატული კრიტერიუმია **გამტარუნარიანობა**. ბევრი პარალელური პროგრამა, განსაკუთრებით საბუნებისმეტყველი დარგში, ორიენტირებულია მონაცემების დიდი ნაკადების გადაადგილებაზე, ამიტომ სისტემის გამტარუნარიანობა, ანუ წამში რამდენი ბაიტის გატარება შეუძლია, წარმოადგენს მწარმოებლურობის ძალიან მნიშვნელოვან მახასიათებელს. არსებობს მწარმოებლურობის რამდენიმე ნახასიათებელი. მათგან ერთერთია კვეთის გამტარუნარიანობა, რომელიც უკვე განხილული გვაქვს მულტიკომპიუტერების თავში. მეორე მახასიათებელია **ჯამური გამტარუნარიანობა**, რომელიც წამოადგენს კავშირის ყველა არხის გამტარუნარიანობების ჯამს. მისი მნიშვნელობა გვიჩვენებს თუ ერთდროულად რამდენი ბიტის გადაცემაა შესაძლებელი. ასევე ერთერთი მნიშვნელოვანი მახასიათებელია თითოეული პროცესორის საშუალო გამტარუნარიანობა. თუ თითოეულ პროცესორს შეუძლია მონაცემების გადამუშავება სიჩქარით 1 Მბაიტი/წმ, მაშინ საკომუნიკაციო ქსელის გამოყენება გამტარუნარიანობით 100 Gბაიტი/წმ შედეგს ვერ მოგვცემს. თანამოქმედების სიჩქარე ამ შემთხვევაში განისაზღვრება თითოეული პროცესორის შესაძლებლობებით.

პრაქტიკაში თეორიულ შესაძლებლობებთან მიახლოება საკმაოდ რთული საქმეა. ამას შეიძლება სრულიად განსხვავებული მიზეზები ჰქონდეს. მაგალითად, ყოველი პაკეტი შეიძლება მოიცავდეს სამსახურეობრივ მონაცენებს მისი შექმნის, სთაურის შექმნისა და გაგზავნის შესახებ. 4 ბაიტი მოცულობის 1024 პაკეტის გაგზავნის შემთხვევაში ვერ იქნება მიღწეული იგივე გამტარუნარიანობა, რაც შეიძლება მიღწეული იქნას 4096 ბაიტი მოცულობის მქონე ერთი პაკეტის გაგზავნის შემთხვევაში. მაგრამ, მეორეს მხრივ დაყოვნების დროის შემცირების მიზნით უფრო ხელსაყრელია მცირე ზომის მქონე პაკეტები, რადგანაც გრძელი პაკეტები დიდი ხნით ახდენენ კავშირის არხებისა და კომუტატორების ბლოკირებას. გამოდის, რომ ადგლი აქვს კონფლიქტს დაყოვნების მცირე დროის მიღწევასა და მაღალი გამტარუნარიანობის მიღწევის მეთოდებს შორის. ზოგიერთი გამოყენებითი ამოცანისათვის უფრო მნიშვნელოვანია დაყოვნების დროის შემცირება, ხოლო სხვა ამოცანებისათვის კი უფრო მნიშვნელოვანია მაღალი გამტარუნარიანობის მიღწევა. ამ არჩევანის გაკეთების დროს მნიშვნელოვანია იმის გათალისწინება, რომ გამტარუნარიანობის გაზრდა შესაძლებელია დამატებითი მატერიალური დანახარჯების გამოყენების შედეგად (დამატებითი კავშირის არხების გამოყენებით ან სიხშირის გაზრდის შედეგად), ხოლო რაც შეეხება დაყოვნების დროს, აქ ფინანსური დანახარჯების გაზრდით ვერაფერს გავხდებით. ამიტომ, უმჯობესია თავიდან ვიზრუნოთ დაყოვნების დროის შემცირებაზე, ხოლო შემდეგ კი გამტარუნარიანობის გაზრდაზე.

3.14.2. პროგრამული კრიტერიუმები

აპარატული კრიტერიუმებიდაყოვნების დრო და გამტარუნარიანობა მიუთითებს იმაზე, თუ რა შესაძლებლობები გააჩნია პატარულ უზრუნველყოფას. მაგრამ, მომხმარებლებს აინტერესებთ სულ სხვა რამე, მათ უნდათ იცოდნენ რამდენად უფრო სწრაფად იმუშავებს მათი პროგრამები პარალელურ კომპიუტერზე ერთპროცესორიან კომპიუტერთან შედარებით. მათთვის ძირითად მახასიათებელს წასრმოადგენს აჩქარება, ანუ რამდენად უფრო სწრაფად მუშაობს პროგარმა n პროცესორიან სისტემაში ერთპროცესორიანთან შედარებით. შედეგები გრაფიკულად ილუსტრირებულია ნახ. 3.15–ზე.

აქ ჩვენ ვხედავთ 64 პროცესორისაგან შემდგარ მულტიკომპიუტერზე მომუშავე რამდენიმე პარალელურ პროგრამას. ყოველი მრუდი გვიჩვენებს ერთი პროგრამის აჩქარებას k პროცესორის გამოყენების შედეგად. აჩქარება წარმოადგენს k–ს ფუნქციას. წყვეტილი ხაზით აღნიშნულია იდეალური აჩქარება, რომლის დროსაც k რაოდენობის პროცესორების გამოყენება იწვევს პროგრამის k–ჯერ აჩქარებას და ეს ხდება k-ს ნებისმიერი მნიშვნელობისათვის. პრაქტიკაში იდეალური აჩქარების მიღწევა ძალიან იშვიათად ხდება, თუმცა კი არც ისე იშვიათია იდელურ აჩქარებასთან მიახლოებული აჩქარების მიღწევა. N სხეულის მოდელირების ამოცანა იძლევა იდეალურთან მიახლოებულ აჩქარებას. აფრიკული თამაში ავარის ვარიანტების გათვლის დროსაც მიიღება საკმაოდ კარგი აჩქარება, მაგრამ მატრიცის ინვერსიის აჩქარების კოეფიციენტი 5– არ აჭარბებს, და მისი მნიშვნელობა პროცესორების რაოდენობის გაზრდით არ იზრდება.



ნახ. 3.15. პრაქტიკაში პროგრამებს არ შეუძლიათ იდეალური აჩქარების მიღწევა (ნაჩვენების წყვეტილი ხაზით)

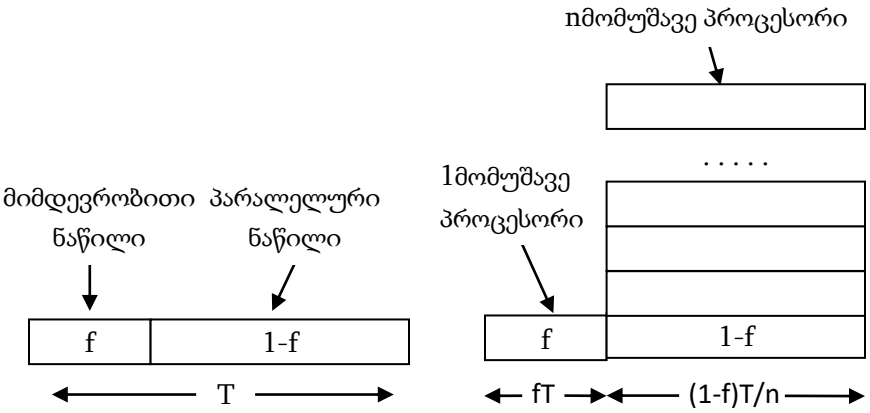
არსებობს მიზეზების მთელი რიგი, რომელთა გამოც ვერ ხერხდება იადელური აჩქარების მიღწევა. პრაქტიკულად ყველა პროგრამაში არსებობს ფრაგმენტები რომელთა გაპარელებაც პრინციპულად ვერ მოხერხდება, მაგალითად, ცვლადების ინიციალიზაცია, საწყისი მონაცემების წაკითხვა და ა.შ. ასეთ შემთხვევებში პროცესორების რაოდენობის გაზრდა საქმეს ვერ ააჩქარება. დავუშვათ ერთპროცესორიან კომპიუტერზე პროგრამის შესრულებას სჭირდება T წამი. ამ დროის f ნაწილის განმავლობაში პროგრამა სრულდება მიმდევრობით, ერთი პროცესორის მიერ, ხოლო (1–f) ნაწილის განმავლობაში კი ხდება სამუშაოების დაპარალელება, რაც ნაჩვენებია ნახ. 3.16,ა–ზე.

პარალელური კოდის შესრულება თუ შესაძლებელია n პროცესორზე, მაშინ ამ კოდის შესრულების დრო (1-f)T მნიშვნელობიდან შემცირდება (1-f)T/n მნიშვნელობამდე. პროგრამის შესრულების ჯამური დრო (მიმდევრობითი და პარალელური ფრაგმენტების) იქნება fT+(1-f)T/n, ხოლო პროგრამის საერთო აჩქარების დრო იქნება

$$\text{აჩქარება} = \frac{n}{1+(n-1)f}.$$

f=0 მნიშვნელობისათვის ვღებულობთ წრფივ აჩქარებას, მაგრამ თუ f>0, მაშინ იდეალური აჩქარება მიუღწეველია, რადგანაც პროგრამაში არსებობს მიმდევრობითი ფრაგმენტი. ამ მოვლენას უწოდებენ **ამდალის კანონს**.

ამდალის კანონი ერთ–ერთი მიზეზია, რის გამოც იდეალური აჩქარების მიღწევა შეუძლებელია. ამაში გარკვეულ როლს თამაშობს აგრეთვე კავშირის არხებში დაყოვნების დროც, გამტარუნარიანობის შეზღუდულობაც და ალგორითმების ხარვეზებიც. 1000 პროცესორიც რომ გვქონდეს, შეუძლებელია ყველა პროგრამა დაიწეროს ისეთნაირად, რომ ერთდროულად ყველა პროცესორი მუშაობდეს. უფრო მეტიც, ბევრი ალგორითმი საერთოდ არ ექვემდებარება გაპარალელებას, ამიტომ ხდება მათი შეცვლა კვაზიოპტიმალური ალგორითმებით. რსებობს ბევრი ისეთი გამოყენებითი პროგრამა, რომელთათვისაც ძალიან სასურველია მწარმოებლურობის n–ჯერ გაზრდა, თუნდაც ამისათვის საჭირო გახდეს 2n პროცესორის გამოყენება. ბოლოსდაბოლოს დღეს პროცესორები უკვე აღარ არის ძვირი.



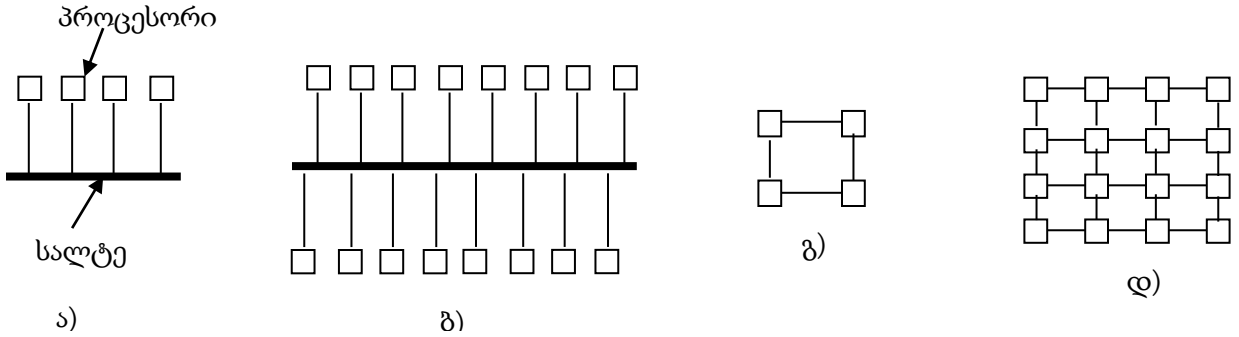
ნახ. 3.16. პროგრამა შეიცავს მიმდევრობით და პარალელურ ნაწილებს (ა); პარალელური ნაწილების ერთდროულად შესრულების შედეგი (ბ)

3.14.3. მწარმოებლურობის გაზრდის ხერხები

სისტემის მწარმოებლურობის გაზრდის ყველაზე მარტივი ხერხია პროცესორების დამატება. მაგრამ, პროცესორების დამატება უნდა მოხდეს ისეთნაირად, რომ სისტემაში არ შეიქმნას სადმე ვიწრო ყელი. სისტემას, რომლის მწარმოებლურობაც იზრდება პროცესორების დამატებით, უწოდებენ **მასშტაბირებადს**.

განვიხილოთ საერთო სალტით ერთმანეთთან დაკავშირებული 4 პროცესორი (ნახ. 4.17,ა). სალტის გამტარუნარიანობა თუ წარმოადგენს bMბაიტი/წმ–ს, მაშინ თითოეულ პროცესორზე გადაანგარიშებით სალტის გამტარუნარიანობა იქნება b/4Mბაიტი/წმ. წარმოვიდგინოთ, რომ სალტეზე პროცესორების რაოდენობა გავზარდეთ 16–მდე (ნახ. 3.17,ბ). სალტეზე პროცესორების რაოდენობის 4–ჯერ გაზრდით თითოეული პროცესორისათვის შევამცირეთ სალტის გამტარუნარიანობადა ერთ პროცესორზე გადაანგარიშებით გახდა b/16Mბაიტი/წმ. ასეთი სისტემა ვერ იქნება მაშტაბირებდი.

იგივე განვიხილოთ საკომუნიკაციო მესერით ერთმანეთთან დაკავშირებული 4 პროცესორისათვის (ნახ. 3.17,გ). ასეთ შემთხვევაში ახალი პროცესორების დამატება ასევე იწვევს კავშირის არხების დამატებას, ამიტომ სისტემის მასშტაბირების დროს თითოეული პროცესორის მწარმოებლურობა არ მცირდება. თუ შემოვიტანთ კავშირის არხების რაოდენობის შეფარდებას რაოდენობასთან, მაშინ ამ შეფარდების მნიშვნელობა 4 პროცესორისათვის არის 1,0 (4 კავშირის არხი, 4 პროცესორი), მასშტაბირების შედეგად კი მისი მნიშვნელობა გახდა 1,5 (16 პროცესორი, 24 კავშირის არხი), ანუ გამოდის, რომ მასშტაბირების შედეგად თითოეული პროცესორის მწარმოებლუ-



ნახ. 3.17. სისტემა საერთო სალტით დაკავშირებული 4 პროცესორით (ა); სისტემა საერთო სალტით დაკავშირებული 16 პროცესორით (ბ); საკომუნიკაციო მესერი 4 პროცესორით (გ); საკომუნიკაციო მესერი 16 პროცესორით (დ)

რობა იზრდება.

ქსელის გამტარუნარიანობა არაა მისი ერთადერთი პარამეტრი. სალტეზე პროცესორის დამატება არ ზრდის არც სისტემის დიამეტრს და არც დაყოვნების დროს. საკომუნიკაციო მესერზე პროცესორების დამატებით კი ეს მაჩვენებლები იზრდება. n×n განზომილების მესერის დიამეტრია 2(n–1), ამიტომ უარეს შემთხვევაში დაყოვნების დრო იზრდება, როგორც კვადრატული ფესვი პროცესორების რაოდენობიდან. 400 პროცესორის შემცველი სისტემის დიამეტრია 38, ხოლო 1600 პროცესორის შემცველი სისტემის დიამეტრი კი 78. ამიტომ, პროცესორების რაოდენობას თუ გავზრდით 4–ჯერ, მაშინ სისტემის დიამეტრი და შესაბამისად დაყოვნების საშუალო დრო გაიზრდება დაახლოებით 2–ჯერ.

მასშტაბირებადმა სისტემამ იდეალურ შემთხვევაში პროცესორების დამატების შედეგად ერთ პროცესორზე გადაანგარიშებით უნდა შეინარჩუნოს მწარმოებლურობის მნიშვნელობა და დაყოვნების საშუალო დრო უნდა იყოს მუდმივი. რეალურად ყოველი პროცესორის მწარმოებლურობის მნიშვნელობის შენარჩუნება ხერხდება, ხოლო პროცესორების რაოდენობის გაზრდით კი დაყოვნების საშუალო დროის მნიშვნელობა იზრდება. ყველაზე უკეთესი შემთხვევაა, როდესაც ამ პარამერტების ზრდა ხდება ლოგარითმულად, ისე, როგორც ეს ხდება ჰიპერკუბების შემთხვევაში.

საქმე იმაშია, რომ დაყოვნების საშუალო დროს გააჩნია გადამწყვეტი მნიშვნელობა მცირემოდულური და საშუალომოდულური გამოყენებითი პროგრამებისათვის. პროგრამას თუ ესაჭიროება მონაცემები, რომლებიც არაა მის ლოკალურ მეხსიერებაში, მათ მიღებაზე იხარჯება საკმაოდ დიდი დრო, და თანაც, რაც უფრო დიდია სისტემა, მით უფრო დიდი ხდება დაყოვნების დრო. ეს პრობლემა დამახასიათებალია მულტიპროცესორებისათვისაც და მულტიკომპიუტერებისათვისაც, რადგანაც ორივე შემთხვევაში ფიზიკური მეხსიერება დაყოფილია ფიქსირებულ განაწილებულ მოდულებად.

სისტემების დამუშავების დროს გამოიყენებენ რამოდენიმე ტექნოლოგიას, რომელთა საშუალებითაც შესაძლებელი ხდება დაყოვნების დროის შემცირება, ან უკიდურეს შემთხვევაში შენიღბვა. პირველ ტექნოლოგიას წარმოადგენს **რეპლიკაცია**. მონაცემების ბლოკების რეპლიკები თუ შენახული იქნება სხვადასხვა ადგილებზე, მაშინ ამ მონაცემებთან წვდომის დრო შეიძლება შემცირებული იქნას. ერთერთი შესაძლებელი ვარიანტია კეშ–მეხსიერების გამოყენება, როდესაც მონაცემების ბლოკის ერთი ან რამდენიმე ასლი ინახება იმ კვანძის მახლობლად, რომელსაც შეიძლება ეს მონაცემები დასჭირდეს. მეორე ვარიანტია – შენახული იქნას მონაცემების ბლოკის რამდენიმე თანაბარუფლებიანი ასლი (არათანაბარუფლებიანი ძირითადი–დაქვემდებარებული მეხსიერებების საპირისპიროდ, როგორც ეს ხდება ძირითად მეხსიერებასა და კეშ–მეხსიერებას შორის). როდესაც შენახულია რამდენიმე ასლი, ძირითადი კითხვებია – ვინ, როდის და სად ათავსებს მათ. აქ შეიძლება არსებობდეს სრულიად განსხვავებული ვარიანტები, მაგალითად, შეკვეთების მიხედვით მონაცემების აპარატული დინამიური განთავსება, კომპილატორის შესაბამისი დირექტივების საშუალებით მათი იძულებითი განთავსება და ა.შ. ყველა შემთხვევაში მთავარია, რომ მიღწეული იქნას შეთანხმებული მართვა.

მეორე ტექნოლოგიაა დაყოვნების დროის შენიღბვა **წინასწარი ამორჩევის** (prefetching) გზით, რომლის დროსაც მონაცემების ელემენტის გამოძახება ხდება, ვიდრე ამის საჭიროება დადგება. ეს საშუალებას იძლევა ერთმანეთთან შეთავსებული იქნას გამოძახებისა და შესრულების პროცესები, შედეგად პროგრამა მოთხოვნილ მონაცემებს ღებულობს დაყოვნების გარეშე. წინასწარი ამორჩევა შეიძლება განხორციელებული იქნას როგორც აპარატულად, ასევე პროგრამულადაც. წინასწარი ამორჩევის შემთხვევაში კეშ–მეხსიერებაში ჩაიტვირთება არა მარტო საჭირო სიტყვა, არამედ მისი შემცველი მთელი სტრიქონი იმ მოსაზრებით, რომ მალე ამ სტრიქონის სხვა სიტყვებიც საჭირო გახდება.

წინასწარი ამორჩევის მართვა შესაძლებელია უშუალოდაც. კომპილატორი, როდესაც დაადგენს, რომ პროგრამას შესრულების დროს დასჭირდება ესა თუ ის მონაცემი, პროგრამის კოდში ჩასვავს მათი მიღების ბრძანებას, თანაც იმ გათვლით, რომ ამ მონაცემების მიღება მოხდება საჭირო დროს. ასეთი სტრატეგია მოითხოვს, რომ კომპილატორს გააჩნდეს სრული მონაცემები სისტემის შესახებ, მისი სინქრონიზაციის მექანიზმი და ასევე აკონტროლებდეს ყველა მონაცემის შენახვის ადგილს. სპეკულაციური ბრძანებები LOAD ყველაზე უკეთ მუშაობენ მაშინ, როდესაც ზუსტად ცნობილია, რომ ჩატვირთული მონაცემები გამოყენებული იქნება. LOAD ბრძანების შესრულების დროს ფურცლის არ არსებობის შეცდომა ნიშნავს, რომ ჩატვირთული მონაცემი გამოყენებული არ იქნება.

დაყოვნების დროის შენიღბვის მესამე ტექნოლოგიაა **რამდენიმე პროგრამული ნაკადის გამოყენება**. პროცესებს შორის გადართვები თუ საკმაოდ ხშირად ხდება, მაშინ როდესაც ხდება ერთი პროცესის ბლოკირება მონაცემების მოლოდინის გამო, გაიშვება შესრულებისათვის მზადმყოფი მეორე პროცესი. ზღვრულ შემთხვევაში პროცესორი ასრულებს პირველი ნაკადის პირველ ბრძანებას, შემდეგ მეორე ნაკადის პირველ ბრძანებას და ა.შ. ამგვარად, ნაკადების დიდი დაყოვნების შემთხვევაშიც კი პროცესორი მუდმივად დასაქმებულია.

დაყოვნების დროის შენიღბვის მეოთხე ტექნოლოგიაა **არაბლოკირებადი ჩანაწერების გამოყენება**. STORE ბრძანების შესრულების დროს პროცესორი იცდის მის დასრულებამდე და მხოლოდ ამის შემდეგ აგრძელებს მუშაობას. არაბლოკირებადი ჩანაწერების შემთხვევაში პროგრამა აგრძელებს მუშაობას მეხსიერებასთან მიმართვის ბრძანებების შესრულების შემთხვევა-

შიც. LOAD ბრძანების შესრულების შემთხვევაში მუშაობის გაგრძელება უფრო რთული საქმე, მაგრამ ესეც შესაძლებელია ბრძანებების შესრულების თანმიმდევრობის შეცვლის შედეგად.

3.15. განაწილებული გამოთვლები

თანამედროვე ამოცანები მეცნიერებაში, ტექნიკაში, წარმოებაში, გარემოს დაცვასა და სხვა დარგებში ძალიან მასშტაბურია და დაკავშირებულია ძალიან ბევრ გამოთვლებთან. ამ ამოცანების გადასაწყვეტად საჭიროა გაერთიანებული იქნას ცოდნა, აპარატული და პროგრამული საშუალებები, სხვადასხვა ორგანიზაციების მონაცემები, რომლებიც გაბნეულია მთელ დედამიწაზე. ასეთი ამოცანების რამდენიმე მაგალითია:

- მარსზე მისიის ყველა ასპექტის გამოკვლევა;
- კონსორციუმი, რომელიც ახდენს რთული პროდუქტის დაპროექტებას (თვითმფრინავი, რთული სამშენებლო ნაგებობა);
- ინტერნაციონალური სამაშველო ოპერაცია, რომლის დროსაც ხდება სტიქიური უბედურების ზონაში მოქმედებების კოორდინაცია.

ზოგიერთი ამოცანის გადასაწყვეტად საჭირო ხდება ხანგრძლივი თანამშრომლობა, ზოგიერთი ამოცანის გადაწყვეტა კი უფრო ადვილად ხდება, მაგრამ, მათ გააჩნიათ ერთი საერთო თვისება. რამდენიმე ორგანიზაცია ერთად მუშაობს საერთო მიზნის მისაღწევად, გამოიყენებს რა თავის რესურსებს.

ჯერ კიდევ ცოტა ხნის წინ საკმაოდ რთული იყო სხვადასხვა ორგანიზაციების ერთობლივი მუშაობის უზრუნველყოფა, რომლებიც გამოიყენებენ სხვადასხვა ოპერაციულ სისტემებს, სხვადასხვა მონაცემთა ბაზებს და განაწესებს. ფართომასშტაბიანი თანამშრომლობის მოთხოვნამ დღის წესრიგში დააყენა დიდ ტერიტორიაზე გაბნეული დიდი რაოდენობით კომპიუტერების ერთიან სისტემაში გაერთიანების ტექნოლოგიის დამუშავება. ამ ტექნოლოგიამ მიიღო სახელწოდება **განაწილებული გამოთვლები** (grid computing). განაწილებული გამოთვლების სისტემა შეიძლება განხილული იქნას, როგორც ინტერნაციონალური, სუსტად შეკავშირებული ჰეტეროგენული კლასტერი.

განაწილებული გამოთვლების სისტემის მიზანია ტექნიკური ინფრასტრუქტურის შექმნა, რომელიც შესაძლებელს გახდის ერთი მიზნის მქნე რამდენიმე ორგანიზაციის გაერთიანებას ერთიან ვირტუალურ ორგანიზაციაში. ამ ვირტუალურ ორგანიზაციას უნდა ჰქონდეს მოქნილი სტრუქტურა, მონაწილე წევრების დაოდენობა დინამიურად უნდა იცვლებოდეს, ყოველი წევრი სრულად უნდა აკონტროლებდეს თავის რესურსებს.

განაწილებული გამოთვლების სისტემა თავისია არსით მრავალგანზომილებიანია და მოიცავს მონაწილეების – თანაბარრანგიანი კვანძების ძალიან დიდ რაოდენობას. ის შეიძლება დავუპირისპიროთ გამოთვლების ტრადიციულ მოდელებს. კლიენტ–სერვერის მოდელში ტრანზაქციაში ჩართულია ორი მონაწილე – სერვერი, რომელიც წარმოადგენს სერვისის მიმწოდებელს და კლიენტი, რომელმაც ეს სერვისი უნდა მიიღოს. ტიპიურ მაგალითს წარმოადგენს ინტერნეტი, სადაც დიდი რაოდენობით მომხმარებლები ინფორმაციისათვის მიმართავენ სერვერებს. განაწილებული გამოთვლების სისტემები განსხვავდება ორწერტილიანი სისტემებისაგანაც, რომლებშიც ერთმანეთთან დაკავშირებულია ორი კომპიუტერი მათ შორის ფაილების გაცვლის მიზნით. ასეთი სისტემების ტიპიურ მაგალითს წარმოადგენს ელექტრონული ფოსტა. ამ განსხვავებებიდან

გამომდინარე, განაწილებული გამოთვლების სისტემებისათვის საჭიროა ახალი განაწესებისა და ტექნოლოგიების დამუშავება.

განაწილებული გამოთვლების სისტემებში ორგანიზებული უნდა იყოს წვდომა სრულიად განსხვავებულ რესურსებთან. ყოველ რესურსს ჰყავს თავისი პატრონი ორგანიზაცია, რომელიც წყვეტს თუ მისი რესურსების რა ნაწილთან, რა დროს და ვისგან უნდა განხორციელდეს წვდომა. დეტალებს თუ არ ჩავუღრმავდებით, შეიძლება ითქვას, რომ განაწილებული გამოთვლების სისტემების არსი მდგომარეობს რესურსებთან წვდომის მართვაში.

განაწილებული გამოთვლების სისტემების მოდელირების ერთერთ ვარიანტს წარმოადგენს მისი წარმოდგენა მრავალდონიანი იერარქიული სტრუქტურის სახით (ცხრილი 3. 2). ქვედა დონეს წარმოადგენს **ინფრასტრუქტურების დონე**, მასში გაერთიანებულია კომპონენტები, რომელთა საშუალებითაც აგებულია განაწილებული გამოთვლების სისტემა. აპარატული უზრუნველყოფის სახით მასში შედის პროცესორები, მაგნიტური დისკები, ქსელები და სენსორები. პროგრამული უზრუნველყოფის სახით კი შედის პროგრამები და მონაცემები. ეს არის ის ფიზიკური რესურსები, რომლებთანაც წვომას უზრუნველჰყოფს განაწილებული გამოთვლების სისტემა.

ცხრილი 3.2. განაწილებული გამოთვლების სისტემის იერარქიის დონეები

გამოყენებების დონე	გამოყენებები, რომლებიც ერთად და შეთანხმებულად იყენებენ რესურსებს
კოლექტივების დონე	კვლევები, შუამავლობა, მონიტორინგი, რესურსების ჯგუფების მართვა
რესურსების დონე	ცალკეული რესურსების უსაფრთხოება და წვდომის მართვა
ინფრასტრუქტურების დონე	ფიზიკური რესურსები: კომპიუტერები, დისკები, ქსელები, სენსორები, პროგრამები და მონაცემები

იერარქიის შემდეგი დონეა **რესურსების დონე**. ეს დონე პასუხს აგბს ცალკეული რესურსების მართვაზე. ხშირად განაწილებული გამოთვლების სისტემაში ჩართულ რესურსთან დაკავშირებულია ლოკალური პროცესი, რომელიც მართავს რესურსს და უზრუნველჰყოფს მასთან დაშორებული მომხმარებლების კონტროლირებად წვდომას. ამ დონის დანიშნულება მდგომარეობს იმაში, რომ იერარქიის ზემდგომ დონეებს შეუქმნას ერთგვაროვანი ინტერფეისი, რომლის საშუალებითაც ისინი შეძლებენ ცალკეული რესურსების მახასიათებლების დადგენას, შეასრულებენ მათ მონიტორინგს და შეძლებენ მათ უსაფრთხო გამოყენებას.

იერარქიის შემდეგი დონეა **კოლექტივების დონე**, რომლის საშუალებითაც ხდება რესურსების ჯგუფების მართვა. მის ერთერთ ფუნქციას წარმოადგენს განაწილებული გამოთვლების სისტემის გამოკვლევა და რესურსების განთავსების ადგილების დადგენა. ამ გამოკვლევის შედეგად მომხმარებელს შეუძლია დაადგინოს პროცესორის საჭირო ტაქტები, დისკური სივრცე ან კონკრეტული მონაცემაბი. საჭირო ინფორმაციის მისაწოდებლად კოლექტივების დონეს შეუძლია აწარმოოს კატალოგები და მონაცემების სხვა ბაზები. გარდა ამისა, მას შეუძლია შეასრულოს საშუამავლო ოპერაციები – დააკავშიროს ერთმანეთთან სხვადასხვა სერვისების მიმწოდებლები და მომხმარებლები. გარდა ამისა, შეუძლია გაანაწილოს დეფიციტური რესურსები ერთმანეთის კონკურენტ მომხმარებლებს შორის. კოლექტივების დონე პასუხს აგებს აგრეთვე მონაცემების გამრავლებაზე, განაწილებული გამოთვლების სისტემაში ახალი მონაწილეების და რესურსების ჩართვაზე, წვდომის პოლიტიკის მონაცემთა ბაზის წარმოებაზე, სადაც აღწერილია თუ რომელ მონაწილეს რომელ რესურსთან აქვს წვდომის უფლება.

იერარქიის მწვერვალზე იმყოფება **გამოყენებების დონე**. ამ დონეზე მუშაობენ მომხმარებლების გამოყენებითი პროგრამები. გამოყენებების დონე მიმართავს მის ქვემდებარე დონეებს რათა მიიღოს უფლებები ამა თუ იმ რესურსის გამოყენებაზე, გააგზავნოს მოთხოვნები მათ გამოყენებაზე, გააკონტროლოს შეკვეთების შესრულების პროცესი, დაამუშაოს მტყუნებები, შეატყობინოს მომხმარებელს შედეგები.

განაწილებული გამოთვლების სისტემის წარმატების წინაპირობას წარმოადგენს უსაფრთხოების მაღალი დონე. რესურსების მფლობელები ყოველთვის მოითხოვენ თავისი რესურსების სრული მართვის უფლებას და ამომწურავ მონიტორინგს (ვინ, რამდენს და რამდენი ხნით მოიხმარს მის რესურსებს). კარგი უსაფრთხოების სისტემის გარეშე არცერთი ორგანიზაცია არ დათმობს თავის რესურსებს განაწილებული გამოთვლების ჩასატარებლად. მომხმარებელს, მისთვის საჭირო ყოველ კომპიუტერზე რომ სჭირდებოდეს თავისი სახელისა და პაროლის შეტანა, ეს მოთხოვნა იქნებოდა ძალიან მძიმე პირობა. მაშასადამე, შემუშავებული უნდა იქნას უსაფრთხოების მოდელი, რომელშიც გათვალისწინებული იქნება ეს გარემოება.

უსაფრთხოების მოდელის ძირითადი მოთხოვნაა, რომ სისტემაში მომხმარებლის რეგისტრაცია მოხდეს ერთხელ. განაწილებული გამოთვლების სისტემის გამოყენების პირველი ნაბიჯია რეგისტრაცია და სერტიფიკატის მიღება, ანუ დოკუმენტის მიღება ციფრული ხელმოწერით, რომელიც მიუთითებს თუ ვის ინტერესებში უნდა იქნას შესრულებული გამოთვლები. სერტიფიკატი შეიძლება იქნას დელეგირებული, ასე რომ, გამოთვლების პროცესში თუ საჭირო გახდება დამატებითი გამოთვლების ჩატარება, შვილობილი პროცესები შეიძლება იდენტიფიცირებული იქნას მათი საშუალებით. როდესაც წვდომის სერტიფიკატი მიენიჭება დაშორებულ სისტემას, ის ასახული უნდა იქნას მის ლოკალური დაცვის მექანიზმში. მაგალითად UNIX სისტემაში მომხმარებლების იდენტიფიცირება ხდება 16–თანრიგიანი იდენტიფიკატორების საშუალებით, მაგრამ სხვა სისტემებში შეიძლება გაომოყენებული იყოს სხვა სქემები. და ბოლოს, საჭიროა მექანიზმი, რომლის საშუალებითაც მოხდება წვდომის პოლიტიკის განსაზღვრა და მხარდაჭერა.

სხვადასხვა ორგანიზაციების და კომპიუტერების თანამოქმედების მხარდაჭერისათვის საჭიროა სტანდარტები, როგორც სერვისების შეთავაზებაზე, ასევე მათთან წვდომის განაწესებზე. სტანდარტიზაციის პროცესის მართვისათვის განაწილებული გამოთვლების თანამეგობრობამ შექმნა ორგანიზაცია სახელწოდებით Global Grid Forum. მისი მუშაობის შედეგს წარმოადგენს სხვადასხვა სტანდარტების ფორმირებისა და განვითარებისათვის შაბლონის შექმნა, რომელსაც ეწოდა **OGSA** (Open Grid Services Architecture - **განაწილებული გამოთვლების სამსახურების ღია არქიტექტურა**). დასამუშავებელი სტანდარტები ძირითადად ეყრდნობა არსებულ სტანდარტებს, მაგალითად, ODSA სამსახურების აღწერისათვის გამოიყენება WSDL (Web Services Definition Language - ვებ–სამსახურების აღწერის ენა). ამჟამად სამსახურების სტანდარტიზირება ხვდება რვიდან ერთერთ კატეგორიაში. მომავალში ეს სია აუცილებლად გაფართოებული იქნება.

1. ინფრასტრუქტურის სამსახურები (უზრუნველჰყოფს სერვისებს შორის თანამოქმედებას);
2. რესურსების მართვის სამსახური (რესურსების რეზერვირება და განთავისუფლება);
3. მონაცემების სამსახური (მონაცემების კოპირება და გადააადგილება იქ, სადაც მათზე არსებობს მოთხოვნილება);
4. კონტექსტური სამსახურები (საჭირო რესურსების და მათი გამოყენების პოლიტიკის აღწერა);
5. საინფორმაციო სამსახურები (რესურსების წვდომადობის შესახებ ინფორმაციის მიღება);

6. თვითკონტროლის სამსახურები (მომსახურების მოთხოვნილი ხარისხის უზრუნველყოფა);
7. დაცვის სამსახურები (უსაფრთხოების პოლიტიკის გამოყენება);
8. შესრულების მართვის სამსახურები (ამოცანების ნაკადის მართვა).

განაწილებული გამოთვლების სისტემები ძალიან საინტერესო და პერსპექტიული მიმართულებაა, რომელის განვითრებაზეც მუშაობა დღესაც გრძელდება.

3.16. TOP500

მულტიპროცესორებსა და მულტიკომპიუტერებთან მიმართებაშიგამოყენებაში დამკვიდრდა ტერმინი სუპერკომპიუტერები. სუპერკომპიუტერების შექმნა დაიწყო ჯერ კიდევ მეორე თაობის კომპიუტერებიდან. მათი განვითარების ისტორია ასეთია.

ტერმინი სუპერკომპიუტერი შემოგვთავაზეს 1960–იან წლებში ინჟინრებმა ჯორჯ მაიკლმა და სიდნი ფერნბახმა, რომლებიც მუშაობდნენ ლუვერმორის ნაციონალურ ლაბორატორიაში, შემდეგში კი კომპანიაში CDC. სუპერკომპიუტერები ეწოდება გამომთვლელ სისტემებს, რომლებიც სისწრაფით ბევრად აღემატებიან ტიპიურ კომპიუტერებს.

ნახ. 3.16. ამერიკელი ინჟინერი სეიმურ კრეი

პიონერად თანამედროვე სუპერკომპიუტერიაში ითვლება ამერიკელი ინჟინერი სეიმურ კრეი. მე-20 საუკუნის 70-80-იან წლებში არსებობდა სუპერკომპიუტერის ასეთი სახუმარო განმარტება: სუპერკომპიუტერი ეს არის ნებისმიერი კომპიუტერი, რომელიც დაპროექტებულია კრეის მიერ.

1953 წელს სეიმურ კრეიმ დაიწყო მუშაობა კომპანიაში Engireening Research Associates და შექმნა თავისი პირველი კომპიუტერი ERRA 1103 (ცნობილია ასევე როგორც UNIVAC 1103) მოგვიანებით კომპანიამ რამდენჯერმე შეიცვალა მფლობელი და სახელწოდება. კრეის და მის კოლეგებს ყელში ამოუვიდათ ხშირი ცვლილებები და საკუთარი კომპანია Control Data Corporationი დაარსეს.

ნახ. 3.17. სუპერკომპიუტერი 1604

1960 წელს კრეიმ გერმანიუმის ტრანზისტორების გამოყენებით, ნაცვლად რადიო მილაკებისა, შექმნა ახალი კომპიუტერი CDC 1604, რომელიც მუშაობდა 200 მჰერც სიხშირეზე (1 ტაქტის ხანგძლივობა 5 მიკროწამი). თავისი შესაძლებ-

ლობებით CDC 1604-მა გადააჭარბა ყველა კომპიუტერს, ამიტომ ის შეიძლება ჩაითვალოს პირველ სუპერკომპიუტერად. პირველი 5 ეკზემპლარი მიიღეს მორის უმაღლესმა სკოლამ, ლივერმორის ნაციონალურმა ლაბორატორიამ, ილინოისის შტატის უნივერსიტეტმა და კომპანიებმა: Northrop–მა და Lockheed-მა.

1968 წელს უკვე მოძველებული ერთი CDC 1604 მიიღო სსრკ ბირთვული კვლევების გაერთიანებულმა ინსტიტუტმა. 1965 წელს სეიმურ კრეიმ დაასრულა მუშაობა თავის მეორე სუპერკომპიუტერზე CDC 6600. სილიციუმის ტრანზისტორებზე გადასვლისა და VLIW არქიტექტურის წყალობით სიხშირე გაიზარდა 10 მეგაჰერცამდე (ტაქტის ხანგძლივობა 100 ნანოწამი). CDC 6600-სათვის ჰაერით გაგრილება არასაკმარისი აღმოჩნდა ამიტომ ფრეონით გაგრილების სისტემა გამოიყენეს. CDC 6600 წინამორბედ CDC 1604–თან შედარებით 50 ათასჯერ უფრო სწრაფი აღმოჩნდა და მისმა სისწრაფემ 1 მეგაფლოპსი შეადგინა. (მილიონი ოპერაცია წამში). პირველი ეგზემპლარი დააყენეს ევროპის ბირთვული კვლევების ცენტრში, ცერნში.

ნახ. 3.18. სუპერკომპიუტერი 6600

1969 წელს კრეიმ წარმოადგინა განახლებული სუპერკომპიუერი CDC 7600, რომელიც პროგრამირების ენა fortran–ის წყალობით სრულად შეესაბამებოდა 6600 მოდელს და ფასითაც იგივე ღირდა 7,5 მლნ დოლარი. მუშაობდა 37 მეგაჰერც სიხშირეზე და 10 ჯერ უფრო სწრაფი იყო – 10 მეგაფლოპსი. შემდეგი სუპერკომპიუტერი ოთხპროცესორიანი CDC 8600 დიდ ელექტროენერგიას (24 კვტ) მოიხმარდა და მისი გაგრილებაც რთული იყო. გამაგრილებელი სისტემა 20 ტონას იწონიდა. პროექტი დაიხურა და კრეიმ დატოვა Control Data Corporation.

1972 წელს კრეიმ დაარსა საკუთარი კომპანია Cray Research, სადაც გააგრძელა მუშაობა სუპერკომპიუტერებზე თავდაცვის სააგენტოებისა და სამეცნიერო დაწესებულებებისათვის.
1975 წელს კრეიმ გამოუშვა ახალ კომპანიაში პირველი კომპიუტერი Cray 1, რომელშიც ტრანზისტორების ნაცვლად გამოიყენა ინტეგრალური მიკროსქემები და

ნახ. 3.19. სუპერკომპიუტერი Cray 1

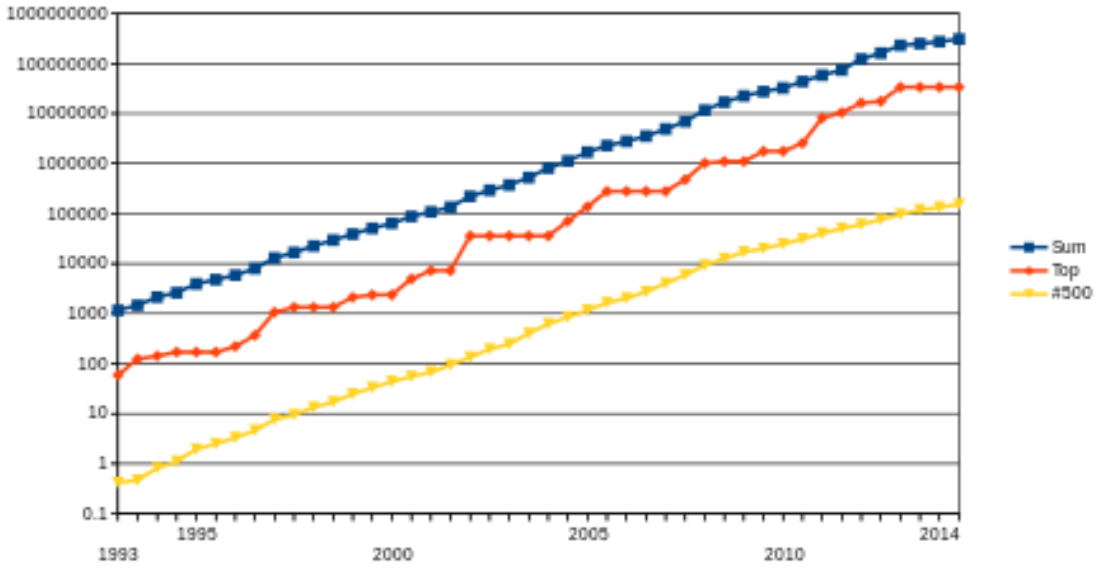
რამდენიმე ჩვეულებრივი პროცესორის ნაცვლად ერთი ვექტორული. ამ კომპიუტერის სისწრაფე 80 მეგაფლოპსამდე იყო.

სუპერკომპიუტერ Cray 2–ზე კომპანიის თანამშრომლები 10 წელი მუშაობდნენ. იგი ეფუძნებოდა იმ დროისათვის ნაკლებად პოპულარულ მასიურპარალელურ გამოთვლებს. იყენებდა 64 სკალარულ პროცესორებს. შედეგმა ყოველგავრ მოლოდინს გადააჭარბა და 1,9 გიგაფლოპსი შეადგინა. 1990 წლამდე Cray 2 ყველაზე მძლავრ სუპერკომპიუტერად ითვლებოდა.

Cray 3–ის დაპტროექტებისას კრეიმ მიზნად დიდი სიმაღლეები დაისახა, ისევე როგორც CDC 8600–ის შემთხვევაში. ამ სუპერკომპიუტერში სილიციუმის პროცესორების ნაცვლად გალიუმის არსენიდზე დამზადებული ჩიპები გამოიყენა. კრეის იმედი ჰქონდა, რომ 500 მეგაჰერც სიხშირეს მიაღწევდა, მაგრამ ასეთი რთული ნახევარგამტარების წარმოება ძალიან ძვირი იყო, ამიტომ პროექტი შეჩერდა. კრეიმ კომპანია დატოვა.

1989 წელს კრეიმ ახალი კომპანია Cray Computer დაარსა და Cray 3-ზე მუშაობა განაგრძო. მან ვერ მოასწრო ლივერმორის ლაბორატორიისგან მიღებულ ვადებში ჩატევა, ამიტომ დამკვეთმა უკვე გამზადებული Cray researchის პროდუქცია ამჯობინა. 1995 წელს Cray Computerი გაკოტრდა. ამის შემდეგ კრეიმ თავისი მესამე კომპანია SRC Computers დაარსა. ერთი წლის შემდეგ სეიმურ კრეი ავტოკატასტროფაში დაიღუპა. მის მიერ შექმნილი Cray research კი დღესაც აწარმოებს სუპერკომპიუტერებს Cray inc–ის სახელწოდებით.

სუპერკომპიუტერების კლასის კომპიუტერების შექმნითა და განვითარებით დაინტერესებულ კორპორაციათა და უნივერსიტეტების რაოდენობამ დაიწყო ზრდა. წლების განმავლობაში დაგროვდა დიდი რაოდენობით სუპერკომპიუტერები. 1993–2014 წლებში მათი შესაძლებლობების ზრდის დინამიკა ნაჩვენებია ნახ. 3.20–ზე. მიმართულების სწრაფი და მიზანმიმართული განვითარებისათვის საჭირო გახდა არსებული სუპერკომპიუტერების ერთმანეთთან შედარება და საუკეთესო სუპერკომპიუტერებისა და არქიტექტურული გადაწყვეტილებების გამოვლენა.



ნახ. 3.20. სუპერკომპიუტერების შესაძლებლობების ზრდის დინამიკა 1993-2014 წლებში

1990-იანი წლების დასაწყისიდან დაიწყეს მუშაობა სუპერკომპიუტერების შესაძლებლობების შეფასების ერთიან მეთოდიკაზე. ამ მეთოდიკაში გათვალისწინებულია როგორც ვექტორული პროცესორების და კომპიუტერების რაოდენობა, ასევე Linpack ტესტის შედეგები. პროცესორებისა თუ კომპიუტერების სწრაქმედება ფასდება FLOPS-ების რაოდენობით, ესაა დროის ერთეულში შესრულებული მცოცავწერტილიანი გამოთვლითი ოპერაციების რაოდენობა. საკმაოდ დიდ ხანს მხოლოდ ამ მაჩვენებლით ხდებოდა სუპერკომპიუტერების სწრაფქმედების შეფასება. დღესაც ხშირად ხდება ამ მაჩვენებლის გამოყენება, მაგრამ მის ნაკლს წარმოადგენს ის, რომ ეს შეფასება არანაირად არაა დაკავშირებული კომპიუტერის პრაქტიკულ გამოყენებასთან. ასეთი შეფასების მისაღებად გაუშვებენ სპეციალურ ტესტ-პროგრამა Linpack-ს, რომლის საშუალებითაც ხდება დიდი განზომილებების მქონე მატრიცების ერთმანეთზე რამდენჯერმე გადამრავლება და ითვლიან ამ ტესტის შესრულების გასაშუალოებულ დროს. ასეთი სატესტო პროგრამების საშუალებით დგინდება, რომ მათი ეფექტურობა იმყოფება თეორიული წარმადობის 90%-ის დონეზე.

1993 წლის ივნისიდან ამ მეთოდიკით დაიწყეს TOP500 სიის გამოქვეყნება, რომელშიც მოცემულია იმ მომენტისათვის 500 საუკეთესო სუპერკომპიუტერის მონაცემები. ამ სიის გამოქვეყნება ხდება წელიწადში ორჯერ: ივნისში და ნოემბერში. შედარებისათვის მოყვანილია პირველი TOP500 სიის პირველი ათეული და უკანასკნელი სიის პირველი ათეული. პირველი სიის პირველი ათეული ასე გამოიყურებოდა (ცხრ. 3.3).

ცხრილი 3.3

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	Los Alamos National Laboratory United States	CM-5/1024 Thinking Machines Corporation	1,024	59.7	131.0	
2	Minnesota Supercomputer Center United States	CM-5/544 Thinking Machines Corporation	544	30.4	69.6	
3	National Security Agency United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
4	NCSA United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
5	NEC Japan	SX-3/44R NEC	4	23.2	25.6	
6	Atmospheric Environment	SX-3/44 NEC	4	20.0	22.0	

	Service (AES) Canada				
7	Naval Research Laboratory (NRL) United States	CM-5/256 Thinking Machines Corporation	256	15.1	32.8
8	Caltech United States	Delta Intel	512	13.9	20.5
9	Cray Research United States	Y-MP C916/16256 Cray Inc.	16	13.7	15.2
10	DOE/Bettis Atomic Power Laboratory United States	Y-MP C916/16256 Cray Inc.	16	13.7	15.2

2015 წლის ნოემბრის TOP500 სიის პირველი ათეული კი ასე გამოიყურება (ცხრილი 3.4).

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom, IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Com-putational Science (AICS), Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laborato-ry, United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom, IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National	Piz Daint - Cray XC30, Xeon E5-	115,984	6,271.0	7,788.9	2,325

	Supercomputing Centre (CSCS) Switzerland	2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.				
7	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P, Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM	393,216	4,293.3	5,033.2	1,972
10	Government United States	Cray XC30, Intel Xeon E5-2697v2 12C 2.7GHz, Aries interconnect , Cray Inc.	225,984	3,143.5	4,881.3	

ნახ. 3.21. სუპერკომპიუტერი Tianhe 2

თანამედროვე სუპერკომპიუტერები წარმოადგენენ გიგანტურ კლასტერებს რომლებიც შედგება სერვერის კვანძებისაგან. სერვერები ყენდება სამონტაჟო კარადებში და თავისი რიგის დროს ირთვება ერთიან ლოკალურ ქსელში ამის წყალობით მთელი კონსტრუქცია ეფექტურად მუშაობს.

ყველაზე მძლავრ სუპერკომპიუტერად მსოფლიოში ითვლება ერთი წლის წინ გამოშვებული ჩინური სუპერკომპიუტერი Tianhe-2 რომლის სისწრაფეც 33,9 პეტაფლოპსია. ამას უნდა უმადლოდეს 3,120,000 პროცესორის ბირთვს და 1,024,000 გიგაბაიტ ოპერატიულ მეხსიერებას. Tianhe-2 იმართება სპეციალურად მისთვის ადაპტირებული ოპერაციული სისტემით Kylin Linux. თითოეული ბლოკი ურთიერთქმედებისა და შეთანხმებული მუშაობისათვის ჩართულია ლოკალურ ქსელში TH Express-2. თითოეული სერვერი შეიცავს ორ ცენტრალურ პროცესორს Xeon E5-2692 (12 ბირთვი 2.2 გიგაჰერცი სიხშირით) და 3 კოპროცესორს Intel Xeon Phi 31S1P(57 გამარტივებული ბირთვი და 8 გიგაბაიტი GDDR5 მეხსიერება) და აგრეთვე 64 გიგაბაიტი ოპერატიული მეხსიერებას. ასეთი სერვერი 16 ათასზე მეტია და მოიხარს 17,8 მეგავატ ელ. ენერგიას.

რეიტინგში მეორე ადგილზეა ამერიკული მონსტრი Titan-ი (ნახ. 3.22). 17,6 პეტაფლოპსი სისწრაფით და 8,2 მეგავატი ელ. ენერგიის მოხმარებით. Titan-ი დააპროექტა „Cray inc-მა“ ოუკ-რიჯის ნაციონალური ლაბორატორიის დაკვეთით. იგი აგებულია AMD Opteron 6274(16 ბირთვი) პროცესორისა და გრაფიკული ამაჩქარებლის NVIDIA Tesla K20X(2688 მიკრობირთვი) ბაზაზე. სულ კომპიუტერი შედგება 560 ათასი პროცესორის ბირთვისა და 710 ათასი გიგაბაიტი ოპერატიული მეხსიერებისგან.

3.22. სუპერკომპიუტერი Titan

ნახ. 3.23. სუპერკომპიუტერები Sequoia და Mira

მე-3 და მე-5 ადგილებს IBM-ის სუპერკომპიუტერები Sequoia (ნახ. 3.22) და Mira (ნახ. 3.22) იკავებენ რომლებიც ბირთვული უსაფრთხოების ნაციონალულ ადმინისტრაციასა და არგონის ნაციანალურ ლაბორატორიაშია დამომონტაჟებული. ამ სუპერკომპიუტერებში Intel-ისა და AMD-ს არქიტექტურის ნაცვლად გამოიყენება IBM Power არქიტექტურის ჩიპები. დამატებითი კოპროცესორები და გრაფიკული ამაჩქარებლები არ გამოიყენება. Sequoia-ს და Mira-ს მახასიათებლები შემდეგნაირად გამოიყურება: 17 და 8,6 პეტაფლოპსი.

3.24. სუპერკომპიუტერი K Computer

ენერგომოხმარება 7,9 და 3,9 მეგავატი.

მე-4 ადგილს იკავებს იაპონური სუპერკომპიუტერი K Computer (ნახ. 3.24), რომელიც კობეს ფიზიკურ-ქიმიური კვლევების ინსტიტუტშია დამონტაჟებული. მასში გამოყენებულია არასტანდარტული პროცესორის არქიტექტურა SPARK64, რომლის ჩიპებიც კომპანია Fujitsu-ს მიერ არის დამზადებული. სისწრაფე 10.5 პეტაფლოპსი. ოპერატიული მეხსიერeბა 1,4 მილიონი გიგაბაიტი. ენერგმოხმარება 12,6 მეგავატი.

მე-6 ადგილს იკავებს ახალი სუპერკომპიუტერი Piz Daint რომელიც განთავსებულია შვეიცარიის ნაციონალურ სუპერკომპიუტერების ცენტში. იგი დაპროექტებული და აწყობილია კომპანია Cray inc-ის მიერ. Piz Daint-ში გამოყენებულია ჰიბრიდული არქიტექტურა - Intel Xeon და Nvidia Tesla. სისწრაფე 6,27 პეტაფლოპსი. ენერგომოხმარება 2,3 მეგავატი.

3.2. სუპერკომპიუტერი Piz Daint

ყველაზე მეტი სუპერკომპუტერები არის აშშ-ში 264, ჩინეთში 115, იაპონიაში 28. საინტერესოა რომ ნახევარ წელიწადში იმ კომპიუტერების რაოდენობა რომელთა სისწრაფეც 1 პეტაფლოპსზე მეტი იყო 26-დან 31-მდე გაიზარდა, ხოლო კომპიუტერი რომელიც ტოპ 500-ში 363 ადგილზე იყო ბოლო ადგილზე აღმოჩნდა. უფრო მძლავრი სუპერკომპიუტერები 50 და 100 პეტაფლოპსამდე სისწრაფით იწყობა აშშ-ში, ჩინეთსა და ინდოეთში. 2025 წელს შესაძლებელი იქნება 1 ექსაპლოპსამდე სისწრაფის სუპერკომპიუტერის შექმნა, რომელსაც შეეძლება გააკეთოს ერთი კვირის ამინდის პროგნოზი მთელ დედამიწაზე. სუპერკომპიუტერების ხელოვნურ ინტელექტს შეუძლია ბევრი რამ, რაც არ შეუძლია ადამიანის ტვინს. ჭადრაკში და ინტელექტუალურ ვიქტორინებში მათ შეძლეს ადამიანის დამარცხება. სუპერკომპიუტერმა IBM Watson-მა დიდი ანგარიშით დაამარცხა ამერიკელი მოაზროვნეები ვიქტორინაში Jeopardy.

4. მატრიცული გამოთვლების ზოგიერთი პარალელური ალგორითმი

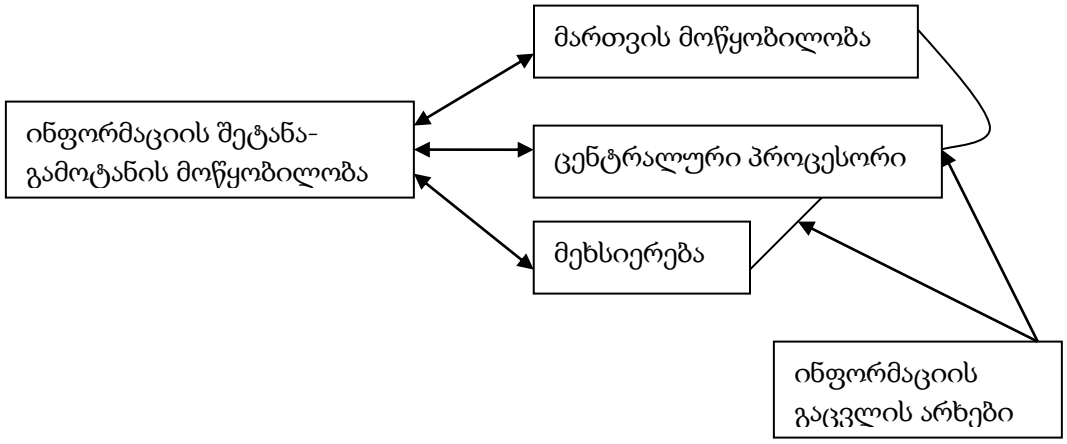
4.1. გამოთვლითი სისტემების არქიტექტურის შესახებ

4.1.1. ერთპროცესორიანისი სტემები

ერთპროცესორიანი გამოთვლითი სისტემების არქიტექტურაში განიხილება შემდეგი მოწყობილობები:

- მართვისმოწყობილობა
- ცენტრალურიპროცესორი
- მეხსიერება
- ინფორმაციისშეტანა-გამოტანისმოწყობილობა
- ინფორმაციისგაცვლისარხები

ამ მოწყობილობების ურთიერთქმედების მექანიზმი ნაჩვენებია შემდეგ ნახაზ 4.1.1-ზე



ნახ. 4.1.1. მოწყობილობების ურთიერთქმედების მექნიზმი ერთპროცესორიან სისტემებში

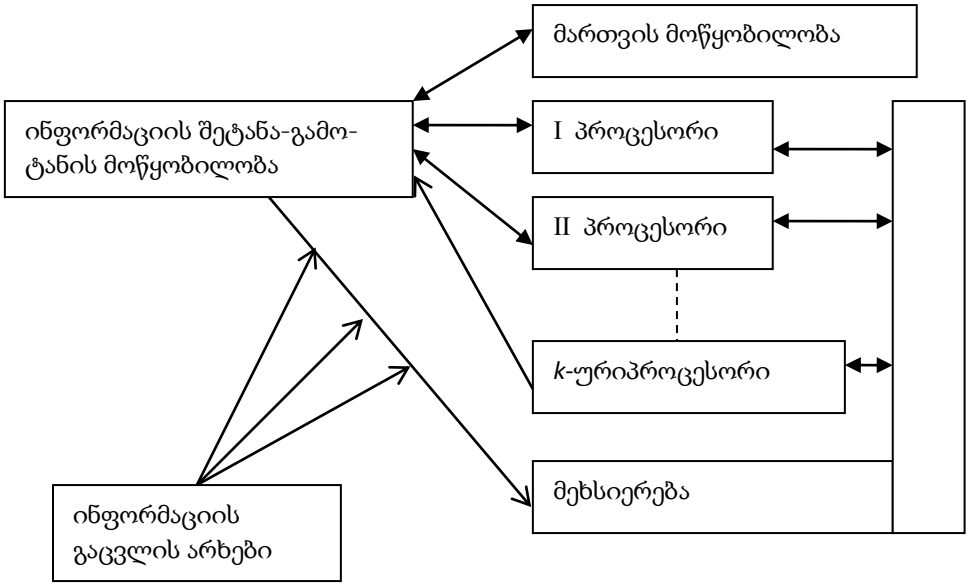
ერთპროცესორიანი გამოთვლითი სისტემის ფუნქციონირების პრინციპია ბრძანებების მიმდევრობითი შესრულება. ამოხსნის ალგორითმის აგების ძირითადი ამოცანაა ალგორითმის წარმოდგენა მიმდევრობითი ბრძანებების საშუალებით. ამ შემთხვევაში ოპტიმიზაციის მთავარი პრობლემა ოპერატორების რაოდენობისა და მეხსიერების საჭირო მოცულობის მინიმიზაციაა.

4.1.2. მრავალპროცესორიანი გამოთვლითი სისტემები

მრავალპროცესორიან გამოთვლით სისტემებს ფორმალურად აქვთ ერთპროცესორიანი სისტემების მსგავსი არქიტექტურა:

_ მართვის მოწყობილობა
_ პირველი პროცესორი
_ მეორე პროცესორი
.....
_ k-ური პროცესორი
_ მეხსიერება
_ ინფორმაციის შეტანა-გამოტანის მოწყობილობა
_ ინფორმაციის გაცვლის არხები

ამ მოწყობილობების ურთიერთქმედების მექანიზმი ნაჩვენებია ნახ.4.1.2-ზე:



ნახ. 4.1.2. მოწყობილობების ურთიერთქმედების მექნიზმი მრავალპროცესორიან სისტემებში

ჩვენი სისტემის სუსტი ადგილი საკომუნიკაციო ქსელია (ინფორმაციის გაცვლის არხები). ქსელის სირთულე ჩვეულებრივ სწრაფად იზრდება პროცესორების რაოდენობის ზრდას თან ერთად. ამჟამად მნიშვნელოვანი პრობლემაა მრავალპროცესორიანი გამოთვლითი სისტემების მოწყობილობებს შორის ეფექტური კავშირის დამყარება.

4.1.3. მრავალპროცესორიანი გამოთვლითი სისტემების გამოყენების სირთულეები

ძირითადი სირთულე არის ის, რომ მრავალპროცესორიანი გამოთვლითი სისტემების მაღალი ეფექტურობა (რაც ახლოსაა მაქსიმალურთან) შეიძლება მიღწეული იქნას მხოლოდ მაშინ, როდესაც ყველა პროცესორი (ან მათი უდიდესი ნაწილი) მთლიანად დატვირთულია. შევნიშნოთ, რომ ერთპროცესორიან სისტემას (შესაბამისი პროგრამის შემთხვევაში) მაქსიმალური ეფექტურობა აქვს.

როდესაც ამოცანას მრავალპროცესორიან გამოთვლით სისტემაზე ვხსნით, ჩვეულებრივ საქმე გვაქვს შემდეგ პრობლემებთან:

_ რომელი ალგორითმების რეალიზაციაა ეფექტურად შესაძლებელი მოცემულ მრავალპროცესორიან სისტემაზე;

_ როგორი მრავალპროცესორიანი გამოთვლითი სისტემები უნდა შეიქმნას მოცემული კლასის ამოცანების ამოსახსნელად;

_ როგორ ავაგოთ ისეთი რიცხვითი მეთოდები, რომელთა რეალიზაციაც მოხერხებული იქნება მრავალპროცესორიან გამოთვლით სისტემაზე.

უნდა აღინიშნოს ისიც, რომ ყოველი ახალი სუპერკომპიუტერის შექმნის შემდეგ ამ სისტემის ეფექტური გამოყენებისთვის აუცილებელი ხდება უმოკლეს დროში ამ სუპერკომპიუტერის მათემატიკური და პროგრამული უზრუნველყოფისა და შესაბამისი რიცხვითი ალგორითმების შექმნა, რათა შევძლოთ სულ უფრო და უფრო მაღალი სირთულის ამოცანების ამოხსნა.

4.1.4. ალგორითმის პარალელური ფორმა

პარალელურ გამოთვლით მანქანაზე ალგორითმის რეალიზაციისთვის საჭიროა, ეს ალგორითმი წარმოვადგინოთ არა ცალკეული ოპერაციების, არამედ ოპერაციების ჯგუფების მიმდევრობის სახით. თითოეულ ჯგუფში ცალკეულ ოპერაციებს უნდა ჰქონდეთ შემდეგი თვისება: მათი შესრულება შესაძლებელი უნდა იყოს ერთდროულად პარალელურ გამოთვლით სისტემაში არსებული ფუნქციონალური მოწყობილობების საშუალებით.

ამრიგად, ვთქვათ, ალგორითმში შემავალი ოპერაციები დაყოფილია ჯგუფებად, ხოლო ჯგუფების სიმრავლე ისეა დალაგებული. რომ თითოელი ჯგუფის ყოველი ოპერაცია დამოკიდებულია ან ამოცანის საწყის მონაცემებზე, ან ამ ჯგუფის წინა ჯგუფებში ჩატარებული ოპერაციების საშუალებით მიღებული შედეგებისგან.

ასეთი სახით ალგორითმის წარმოდგენას ალგორითმების პარალელურ ფორმას უწოდებენ. ოპერაციების თითოეულ ჯგუფს უწოდებენ იარუსს, ხოლო ასეთი იარუსების რაოდენობას _ პარალელური ფორმის სიმაღლეს. იარუსებში ოპერაციების მაქსიმალურ რაოდენობას უწოდებენ პარალელური ფორმის სიგანეს. შევნიშნოთ, რომ ერთსა და იმავე ალგორითმს შეიძლება ჰქონდეს რამდენიმე ფორმა. უპირატესობა ენიჭება მინიმალური სიმაღლის ფორმებს.

განვიხილოთ რამდენიმე მაგალითი.

მაგალითი 1. ვთქვათ, უნდა გამოვთვალოთ შემდეგი

$$(a_1a_2 + a_3a_4)(a_5a_6 + a_7a_8)$$

გამოსახულების მნიშვნელობა. ამასთან უნდა შევინარჩუნოთ მოქმედებების ჩატარების ის რიგი, რომელიც მოცემულია ჩანაწერში.

მოვიყვანოთ ამ გამოსახულების მნიშვნელობის გამოთვლის ალგორითმის რამდენიმე პარალელური ფორმა.

1.1. პირველი ფორმა:

მონაცემები $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

I იარუსი $a_1a_2, a_3a_4\ a_5a_6, a_7a_8$

II იარუსი $a_1a_2 + a_3a_4\ a_5a_6 + a_7a_8$

III იარუსი $(a_1a_2 + a_3a_4)(a_5a_6 + a_7a_8)$

ამ პარალელური ფორმის სიმაღლეა 3, ხოლო სიგანე 4.

ამ პარალელურ ფორმას ახასიათებს ის, რომ 4 პროცესორი დატვირთულია მხოლოდ პირველ ეტაპზე, ხოლო უკანასკნელ ეტაპზე დატვირთულია მხოლოდ ერთი პროცესორი.

1.2. მეორე პარალელურ ფორმას აქვსშემდეგი სახე:

მონაცემები $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

I იარუსი $a_1a_2\ a_3a_4$

II იარუსი $a_5a_6\ a_7a_8$

III იარუსი $a_1a_2 + a_3a_4\ a_5a_6 + a_7a_8$

IV იარუსი $(a_1a_2 + a_3a_4)(a_5a_6 + a_7a_8)$

1.3. მესამე პარალელურ ფორმას აქვს სახე:

მონაცემები $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

I იარუსი $a_1 \cdot a_2\ a_3 \cdot a_4$

II იარუსი $a_1 \cdot a_2 + a_3a_4\ a_5 \cdot a_6$

III იარუსი a_7a_8

IV იარუსი $a_5a_6 + a_7 \cdot a_8$.

V იარუსი $(a_1a_2 + a_3a_4)(a_5a_6 + a_7a_8)$

ცხადია, რომ მეორე პარალელური ფორმის სიმაღლეა 4, მესამე პარალელური ფორმის კი 5, ორივე პარალელური ფორმის სიგანეა 2.

მაგალითი2. გამოვთვალოთ *n* რიცხვის ნამრავლი. ვთქვათ, $n = 8$. უნდა გავამრავლოთ რიცხვები a_1, a_2, a_3, …, a_8

2.1. მიმდევრობითი გამრავლების ჩვეულებრივ სქემას აქვს სახე:

მონაცემები a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8

I იარუსი $a_1 \cdot a_2$

II იარუსი $(a_1a_2) \cdot a_3$

. . . . . .

VII იარუსი $(a_1a_2a_3 \cdots a_7) \cdot a_8$

ცხადია, რომ ამფორმის სიმაღლეა 7, ხოლო სიგანე კი 1.

2.2. იარუსების რაოდენობის შემცირება შესაძლებელია **დაწყვილების ალგორითმის** საშუალებით, რაც შემდეგში მდგომარეობს:

მონაცემები a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8

I იარუსი $a_1 \cdot a_2$, $a_3 \cdot a_4$, $a_5 \cdot a_6$, $a_7 \cdot a_8$

II იარუსი $(a_1a_2)(a_3a_4)$, $(a_5a_6)(a_7a_8)$

III იარუსი $(a_1a_2a_3a_4) \cdot (a_5a_6a_7a_8)$

ამ პარალელური ფორმის სიმაღლეა 3, ხოლო სიგანე კი 4. ასეთი ტიპის პროცესს **დაწყვილების სქემას** უწოდებენ. მისი რეალიზაციისთვის საჭიროა ყოველ იარუსზე განვახორციელოთ იმ რიცხვების წყვილების მაქსიმალურად შესაძლებელი გამრავლება, რომელიც მიღებულია წინა იარუსებზე ან მიღებულია საწყისი მონაცემების სახით.

მაგალითი 3. ვთქვათ, უნდა გამოვთვალოთ $I = \int_a^b f(x)dx$ ინტეგრალის მიახლოებითი მნიშვნელობა

$$I(N) = \sum_{i=1}^{N} c_i f(x_i)$$

კვადრატურული ფორმულის საშუალებით, სადაც x_1 , x_2 , … , x_N კვანძითი წერტილებია, $x_i \in [a,b]$, $i = 1,2,\ldots,N$, ხოლო c_i კვადრატურული ფორმულის კოეფიციენტებია. ჩავთვალოთ, რომ კვადრატურული ფორმულის კოეფიციენტები ცნობილია.

ვთქვათ, $N = 8$.

I იარუსი $f(x_1)$, $f(x_2)$, $f(x_3)$, $f(x_4)$, $f(x_5)$,

$$f(x_6),\ f(x_7),\ f(x_8),$$

(ფუნქციის მნიშვნელობების გამოთვლა კვანძის წერტილებში)

II იარუსი $c_1 f(x_1)$, $c_2 f(x_2)$, $c_3 f(x_3)$, $c_4 f(x_4)$, $c_5 f(x_5)$,

$$c_6 f(x_6),\ c_7 f(x_7),\ c_8 f(x_8),$$

III იარუსი $c_1 f(x_1) + c_2 f(x_2)$, $c_3 f(x_3) + c_4 f(x_4)$, $c_5 f(x_5) + c_6 f(x_6)$,

$$c_7 f(x_7) + c_8 f(x_8),$$

IV იარუსი $\sum_{i=1}^{4} c_i f(x_i)$ $\sum_{i=5}^{8} c_i f(x_i)$

V იარუსი $\sum_{i=1}^{8} c_i f(x_i)$

ცხადია, რომ ამ ალგორითმის პარალელური ფორმის სიმაღლეა 5, ხოლო სიგანე 8. როგორც ვნახეთ, ეს პარალელური ალგორითმი დაწყვილების სქემას იყენებს.

4.1.5. შემოუსაზღვრელი პარალელიზმის კონცეფციის შესახებ.

შეუზღუდავი პარალელიზმის კონცეფცია გულისხმობს შემოუსაზღვრელი პარალელური გამოთვლითი სისტემის გამოყენებას.

სისტემას ეწოდება შემოუსაზღვრელი პარალელური გამოთვლითი სისტემა, თუ მას გააჩნია შემდეგი თვისებები:

1) სისტემა შეიცავს იდენტური პროცესორების ნებისმიერ რაოდენობას.

2) სისტემას აქვს ნებისმიერად დიდი მეხსიერება, რომელთანაც ერთდროულად აქვს წვდომა ყველა პროცესორს.

3) ოპერაციების წინასწარ მოცემული სიმრავლიდან ყოველ პროცესორს დროის დისკრეტულ ერთეულში შეუძლია შეასრულოს ნებისმიერი ოპერაცია. ამ სიმრავლის ოპერაციებს ძირითად ოპერაციებს ვუწოდებთ.

4) უგულებელყოფილია დამხმარე ოპერაციების (ე. ი. იმ ოპერაციების, რომლებიც ძირითადნი არ არიან) გამოთვლით სისტემაზე შესრულების დრო.

5) პროცესორების მეხსიერებასთან მიმართვის დროს არ წარმოიქმნება კონფლიქტები

6) სისტემაზე მუშაობის დაწყების წინ ყველა საჭირო მონაცემი ჩაწერილია მეხსიერებაში.

7) გამოთვლითი პროცესის დამთავრების შემდეგ ყველა მიღებული შედეგი მეხსიერებაში რჩება.

4.1.6. შეწყვილების სქემის შესახებ.

შემდგომში $[a]$ სიმრავლით აღვნიშნავთ a რიცხვის მთელ ნაწილს.

თეორემა 1. n რიცხვის მამრავლების შეწყვილების სქემის შესაბამისი პარალელური ფორმის სიმაღლეა $h = \left[\log_2^n\right]+1.$

დამტკიცება. იმ შემთხვევაში, როდესაც n არ არის 2-ის ხარისხი, მოიძებნება k-ს ისეთი მნიშვნელობა, რომ სამართლიანი იქნება $2^{k-1} < n < 2^k$ უტოლობები. მოცემული რიცხვების ნამრავლს თანამამრავლებად დავუმატოთ 2^{k-n} რაოდენობის რიცხვი, რომელთაგან თითოეული 1-ის ტოლია. შემდეგ ავაგებთ შეწყვილების სქემის შესაბამის პარალელურ ფორმას და ამფორმით გამოვთვლით სიმაღლეს. შევნიშნოთ, რომ შემოუსაზღვრელი გამოთვლითი სისტემის შემთხვევაში ფორმის სიგანეს არსებითი მნიშვნელობა არაქვს.

თეორემა დამტკიცებულია.

ანალოგიურად შეიძლება გამოვიყენოთ შეწყვილების სქემა n რიცხვის ჯამის პარალელური ფორმის აგებისას. მაშინ სამართლიანი იქნება შემდეგი თეორემა:

თეორემა 2. შეწყვილების სქემით n რიცხვის შეკრების პარალელური ფორმის სიმაღლეა $h = \left[\log_2^n\right]+1.$

ამ თეორემის დამტკიცების მეთოდი ანალოგიურია თეორემა 1-ის დამტკიცების მეთოდისა (ამ შემთხვევაში, თუ n არ არის 2-ის ხარისხი, მაშინ ჯამს უნდა დავუმატოთ შესაბამისი რაოდენობის ნულოვანი შესაკრებები).

მაგალითი 4. ვთქვათ, უნდა გამოვთვალოთ n რიცხვის a_1, a_2, a_3, …, a_n ნამრავლი და აგრეთვე ყოველი მომდევნო

$$a_1,\ a_1 \cdot a_2,\ a_1 \cdot a_2 a_3,\ a_1 a_2 a_3 \cdots a_{n-1}$$

ნამრავლი.

შეწყვილების სქემის გამოყენებით ეს ამოცანა შეიძლება ძალიან ეფექტურად ამოიხსნას. მოვახდინოთ ამ ამოცანის ამოხსნის პოვნის დემონსტრაცია $n = 8$ შემთხვევაში. ქვემოთ მოყვანილ სქემაში ყველა საძიებელი ნამრავლი აღნიშნულია ხაზისგასმით.

მონაცემები a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8

I იარუსი $\underline{a_1 \cdot a_2}$, $a_3 \cdot a_4$, $a_5 \cdot a_6$, $a_7 \cdot a_8$

II იარუსი $(a_1 a_2) a_3$, $\underline{(a_1 a_2)(a_3 a_4)}$, $(a_5 a_6) a_7$, $a_5 a_6 a_7 a_8$

III იარუსი $\underline{(a_1 a_2 a_3 a_4) a_5}$ $\underline{(a_1 a_2 a_3 a_4)(a_5 a_6)}$

$$\underline{(a_1 a_2 a_3 a_4)(a_5 a_6 a_7)}\ \underline{(a_1 a_2 a_3 a_4)(a_5 a_6 a_7 a_8)}$$

ცხადია, რომ ამ პარალელური გამოთვლითი ფორმის სიმაღლე $h = 5$, ხოლო სიგანე $w = 4$. ამ ალგორითმის რეალიზაციისას გვაქვს ყველა პროცესორის სრული დატვირთვა. მაგრამ, უნდა შევნიშნოთ, რომ ზოგიერთი შედეგი "ზედმეტია" იმ აზრით, რომ საბოლოო პასუხისთვის ისინი არ გვჭირდება (ეს სიტუაცია საკმარისად ტიპიური სიტუაციაა პარალელური ალგორითმების რეალიზაციისას).

ცხადია, რომ ზემოთ აღნიშნული სქემის გამოყენება შეიძლება იმ შემთხვევაშიც, როდესაც n არ არის 2-ის ხარისხი. ამ შემთხვევაში მონაცემები უნდა შევცვალოთ საჭირო რაოდენობის 1-ით.

აღვნიშნოთ პარალელური ალგორითმების რამდენიმე მნიშვნელოვანი თვისება:

1. ბევრი პარალელური ალგორითმის რეალიზაციისას სრულდება დიდი რაოდენობის ზედმეტი ოპერაცია.

2. ერთი და იმავე ამოცანისთვის სხვადასხვა პარალელური ალგორითმების გამოყენებისას მიღებული შედეგები შეიძლება განსხვავებული იყოს ერთმანეთისგან ("მანქანური არითმეტიკისა" და მასთან დაკავშირებული დამრგვალებების ცდომილების გამო).

3. პროცესორების დიდი რაოდენობის შემთხვევაში პარალელური ალგორითმები უფრო ნაკლებად მდგრადია მიმდევრობით ალგორითმებთან შედარებით.

4.2. მატრიცული გამოთვლების ზოგიერთი პარალელური ალგორითმი შემოუსაზღვრელი პარალელიზმის შემთხვევაში

4.2.1. მატრიცის გამრავლება ვექტორზე

ვთქვათ, მოცემულია *n*-ური რიგის

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

კვადრატული მატრიცი და $\vec{x}=(x_1,x_2,\ldots,x_n)$ ვექტორი. უნდა გამოვთვალოთ $\vec{y}=A\vec{x}$ ვექტორის $\vec{y}=(y_1,y_2,\ldots,y_n)$ კოორდინატები:

$$y_i=\sum_{i=1}^{n}a_{ij}x_j,\ \ i=1,2,\ldots,n.$$

n^2 რაოდენობის პროცესორის შემთხვევაში გამოვთვლით $a_{ij}x_j$ $(i=1,2,\ldots,n,\ j=1,2,\ldots,n)$ ნამრავლებს:

I იარუსი $a_{11}x_1$, $a_{12}x_2$, …, $a_{1n}x_n$, $a_{21}x_1$, $a_{22}x_2$, …, $a_{2n}x_n$,

…, $a_{n1}x_1$, $a_{n2}x_2$, …, $a_{nn}x_n$

შემდეგ გამოვთვლით $\sum_{j=1}^{n}a_{1j}x_j$, $\sum_{j=1}^{n}a_{2j}x_j$, …, $\sum_{j=1}^{n}a_{nj}x_j$ ჯამებს.

თითოეული ჯამისთვის გამოვიყენებთ შეწყვილების სქემას. ამრიგად, ამ ჯამების გამოსათვლელად საჭირო იქნება $\lceil\log_2^n\rceil+1$ იარუსი. ამრიგად, ამ ამოცანის ამოსახსნელად n^2 პროცესორის შემთხვევაში დაგვჭირდება $\lceil\log_2^n\rceil+2$ იარუსი.

თეორემა 1. შეუზღუდავი პარალელიზმის კონცეფციის შემთხვევაში $A(n\times n)$ მატრიცისა და $\vec{x}\in R^n$ ვექტორის გამრავლებისთვის შეიძლება აიგოს პარალელური ფორმა, რომლის სიმაღლეა $h=\lceil\log_2^n\rceil+2$, ხოლო სიგანეა $w=n^2$.

4.2.2. მატრიცების გამრავლება

განვიხილოთ ორი

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad B=\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{pmatrix}.$$

n-განზომილებიანი კვადრატული მატრიცის გამრავლების ალგორითმი. *B* მატრიცი წარმოვადგინოთ შემდეგი სახით:

$$B=\left(\vec{b}_1,\vec{b}_2,\ldots,\vec{b}_n\right),$$

სადაც

$$\vec{b}_1=\begin{pmatrix} b_{11} \\ b_{21} \\ \cdots \\ b_{n1} \end{pmatrix}, \quad \vec{b}_2=\begin{pmatrix} b_{12} \\ b_{22} \\ \cdots \\ b_{n2} \end{pmatrix}, \quad \ldots, \quad \vec{b}_n=\begin{pmatrix} b_{1n} \\ b_{2n} \\ \cdots \\ b_{nn} \end{pmatrix}$$

მაშინ მატრიცების AB ნამრავლი შეიძლება წარმოვადგინოთ შემდეგი სახით:

$$AB=\left(A\vec{b}_1,A\vec{b}_2,\ldots,A\vec{b}_n\right).$$

ამრიგად, მატრიცების ნამრავლი შეიძლება ჩაიწეროს როგორც პირველი მატრიცის ნამრავლი მეორემატრიცის სვეტებზე (ასეთი სვეტების რაოდენობაა *n*). *A* მატრიცის გამრავლება თითოეულ სვეტზე შეიძლება განვიხილოთ პარალელური გამოთვლითი ფორმის საშუალებით, რომლის სიმაღლეა $\left\lceil \log_2^n \right\rceil + 2$, ხოლო სიგანეა n^2.

თუ გვექნება n^3 რაოდენობის პროცესორი, მაშინ *A* მატრიცის გამრავლება თითოეულ სვეტზე შეიძლება განვახორციელოთ პარალელურად. ამრიგად მივიღებთ მატრიცების გამრავლების პარალელურ ფორმას, რომლის სიმაღლეა $\left\lceil \log_2^n \right\rceil + 2$, ხოლო სიგანეა n^3.

თეორემა 2. *n*-ური რიგის ორიკვადრატული მატრიცის გამრავლებისთვის შეიძლება აიგოს პარალელური გამოთვლი თიფორმა, რომლის სიმაღლეა $h=\left\lceil \log_2^n \right\rceil + 2$, ხოლო სიგანეა $w=n^3$.

4.2.3. ერთი რეკურსიული პროცესის პარალელური ფორმა.

განვიხილავთ ერთ რეკურსიულ პროცესს, რომელიც ხშირად გვხვდება სხვადასხვა ამოცანების განხილვისას.

ვთქვათ, *r* და *S* ფიქსირებული ნატურალური რიცხვებია, ხოლო

$$\vec{X}_0, \quad \vec{X}_{-1}, \quad \ldots, \quad \vec{X}_{-r+1} \tag{4.1}$$

მოცემული *n*-განზომილებიანი კვადრატული ვექტორებია. უნდა გამოვთვალოთ

$$\vec{X}_1,\ \vec{X}_2,\ \ldots,\ \vec{X}_S \tag{4.2}$$

ვექტორები შემდეგი

$$\vec{X}_i = A_{i1}\vec{X}_{i-1} + A_{2i}\vec{X}_2 + \cdots + A_{ir}\vec{X}_{i-r} + \vec{B}_i \tag{4.3}$$

რეკურენტული დამოკიდებულების საშუალებით, სადაც A_{ij} $(j = 1,2,\ldots,r,\ i = 1,2,\ldots,s)$ მოცემული *n*-ური რიგის კვადრატული მატრიცებია, ხოლო $\vec{B}_i$ – მოცემული ვექტორები.

ამრიგად, (2) ვექტორები გამოითვლებიან შემდეგი

$$\vec{X}_1 = A_{11}\vec{X}_0 + A_{12}\vec{X}_2 + \cdots + A_{1r}\vec{X}_{-r+1} + \vec{B}_1$$

$$\vec{X}_2 = A_{21}\vec{X}_1 + A_{22}\vec{X}_0 + \cdots + A_{2r}\vec{X}_{-r+2} + \vec{B}_2$$

. . . . . .

$$\vec{X}_s = A_{S1}\vec{X}_{S-1} + A_{S2}\vec{X}_{r-2} + \cdots + A_{Sr}\vec{X}_{S-r} + \vec{B}_S$$

ტოლობების საშუალებით. შევნიშნოთ, რომ $A_{SS}\vec{X}_0$ შესაკრები გვხვდება იმ შემთხვევაში, თუ $S \le r$.

თეორემა 3. (4.2) ვექტორების კომპონენტების გამოსათვლელად არსებობს პარალელური ფორმა, რომლის სიმაღლეა $h = S\left\{\left[\log_2^n\right] + \left[\log_2(r+1)\right] + 3\right\}$, ხოლო სიგანეა $w = n^2 r$.

დამტკიცება. ჯერ დავაფიქსიროთ $i \in \{1,2,\ldots,S\}$ და განვიხილოთ

$$\vec{X}_i = A_{i1}\vec{X}_{i-1} + A_2\vec{X}_{i-2} + \cdots + A_{ir}\vec{X}_{i-r} + \vec{B}_i \tag{4.4}$$

ტოლობა, სადაც $\vec{X}_{i-1}$, $\vec{X}_{i-2}$, $\vec{X}_{i-r}$ უკვე გამოთვლილი და ამდენად ცნობილი ვექტორებია.

თითოეული $A_{ij}\vec{X}_{i-j}$ გამოსათვლელად თეორემა 1-ის თანახმად შეიძლება ავაგოთ პარალელური ფორმა, რომლის სიმაღლეა $\left[\log_2^n\right] + 2$, ხოლო სიგანეა n^2.

რადგან ყველა ეს თანამამრავლი შეიძლება გამოვთვალოთ პარალელურად (ეს თანამამრავლები ერთმანეთისგან დამოუკიდებელია), ამიტომ, საკმარისია პარალელური ფორმის სიგანე გავზარდოთ *r*-ჯერ, მაშინ ფორმის სიმაღლე უცვლელი დარჩება. (4.4) გამოსახულების გამოსათვლელად საკმარისია, შევკრიბოთ ვექტორები, რომელთა რაოდენობაა $(r+1)$, რაც მოითხოვს პარალელურ ფორმას, რომლის სიმაღლეა $\left[\log_2(r+1)\right] + 1$, ხოლო სიგანეა $n\left\{\left[\frac{r+1}{2}\right] + 1\right\}$. გამოთვლების ეს ეტაპი მხოლოდ დაამატებს პარალელური ფორმის იარუსებს, ამიტომ არ მოით-

ხოვს სიგანის გაზრდას, რადგან $n\left\{\left[\frac{r+1}{2}\right]+1\right\} \le n^2 r$. ამრიგად, *i*-ურ ბიჯზე ($\vec{X}_i$ ვექტორის გამოსათვლელად) მივიღებთ პარალელურ ფორმას, რომლის სიმაღლეა

$$h = \left\{\left[\log_2^n\right]+2+\left[\log_2(r+1)\right]+1\right\},$$

ხოლო სიგანეა $w = n^2 r$.

(4.4) ფორმულის საშუალებით გამოთვლები უნდა ჩატარდეს *S*-ჯერ. ყოველი $\vec{X}_i$ ვექტორი დამოკიდებულია ადრე გამოთვლილ ვექტორებზე (ან საწყის მონაცემებზე), ამიტომ იარუსების რაოდენობა *n*-ჯერ უნდა გავზარდოთ.

ამრიგად, (4.3) რეკურენტული სქემის რეალიზაციის თვის შეიძლება ავაგოთ პარალელური ფორმა, რომლის სიმაღლეა

$$h = S\left\{\left[\log_2^n\right]+\left[\log_2(r+1)\right]+3\right\},$$

ხოლო სიგანეა $w = n^2 r$.

თეორემა დამტკიცებულია

შევნიშნოთ, რომ ამ ამოცანისთვის შეიძლება აიგოს პარალელური ფორმა, რომელსაც ექნება უფრო ნაკლები სიმაღლე და ამას შეიძლება მივაღწიოთ პარალელური ფორმის სიგანის გაზრდის ხარჯზე.

თეორემა 4. (4.2) ვექტორების კომპონენტების გამოსათვლელად არსებობს პარალელური ფორმა, რომლის სიმაღლეა

$$h = S\left\{\left[\log_2(S+1)\right]+1\right\}\cdot\left\{\left[\log_2(rn+1)\right]+2\right\}, \qquad (4.5)$$

ხოლო სიგანეა

$$w = \left\{\left[\frac{S+1}{2}\right]+1\right\}(nr+1)^3 . \qquad (4.6)$$

დამტკიცება. (4.3) რეკურენტული დამოკიდებულებები წარმოვადგინოთ ექვივალენტური ფორმით, რისთვისაც გამოვიყენებთ

$$\begin{pmatrix} \vec{X}_i \\ \vec{X}_{i-1} \\ \vec{X}_{i-2} \\ \vdots \\ \vec{X}_{i-r+2} \\ \vec{X}_{i-r+1} \\ 1 \end{pmatrix} = \begin{pmatrix} A_{i1} & A_{i2} & \cdots & A_{i\,r-2} & A_{i\,r-1} & A_{ir} & B_i \\ E & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & E & \cdots & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & E & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & E & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{X}_{i-1} \\ \vec{X}_{i-2} \\ \vec{X}_{i-3} \\ \vdots \\ \vec{X}_{i-r+1} \\ \vec{X}_{i-r} \\ 1 \end{pmatrix} \tag{4.7}$$

ბლოკურ მატრიცებსა და ვექტორებს. იმისთვის, რომ დავრწმუნდეთ (4.3) და (4.7) ტოლობების ექვივალენტობაში, (4.7) ტოლობის მარჯვენა მხარეში ბლოკური მატრიცი გავამრავლოთ ბლოკურ ვექტორზე, რის შედეგადაც მივიღებთ r ვექტორულ და ერთ (უკანასკნელ)

$$\vec{X}_i = A_{i1}\vec{X}_{i-1} + A_{i2}\vec{X}_{i-2} + \cdots + A_{i\,r-2}\vec{X}_{i-r+2} + A_{i\,r-1}\vec{X}_{i-r+2} + A_{ir}\vec{X}_{i-r} + B_i$$

$$\vec{X}_{i-1} = E\vec{X}_{i-1}$$

$$\vec{X}_{i-2} = E\vec{X}_{i-2}$$

. . . . . .

$$\vec{X}_{i-r+1} = E\vec{X}_{i-1+1}$$

$$1 = 1$$

სკალარულ ტოლობას.

ამ ტოლობებიდან ჩანს, რომ (4.7) ტოლობები (4.3) რეკურენტული დამოკიდებულებების ექვივალენტურია. შემოვიღოთ აღნიშვნა

$$Q_i = \begin{pmatrix} A_{i1} & A_{i2} & \cdots & A_{i\,r-2} & A_{i\,r-1} & A_{ir} & B_i \\ E & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & E & \cdots & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & E & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & E & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \vec{y}^{\,i} = \begin{pmatrix} \vec{X}_i \\ \vec{X}_{i-1} \\ \vec{X}_{i-2} \\ \vdots \\ \vec{X}_{i-r+2} \\ \vec{X}_{i-r+1} \\ 1 \end{pmatrix}.$$

ცხადია, რომ Q არის კვადრატული მატრიცი, რომლის განზომილებაა $nr+1$.

ამ აღნიშვნის შემდეგ (4.7) ტოლობა მიიღებს სახეს:

$$\vec{y}_i = Q_i\vec{y}_{i1} \tag{4.8}$$

თუ გამოვიყენებთ (4.8) ტოლობას, მივიღებთ

$$\vec{y}_i = (Q_i Q_{i-1} \cdots Q_1)\vec{y}_0, \ \ i = 1,2,\ldots,S .$$

კერძოდ,

$$\vec{y}_1 = Q_1\vec{y}_0$$

$$\vec{y}_2 = Q_2 Q_1\vec{y}_0$$

$$\vec{y}_S = (Q_S Q_{S-1} \cdots Q_1)\vec{y}_0$$

გამოვიყენოთ შეწყვილების ალგორითმი, რომელიც გამოთვლის არა მარტო ყველა $(S+1)$ რაოდენობის მატრიცის ნამრავლს, არამედ ყოველი მომდევნო თანამამრავლების ნამრავლსაც (იხ. 4.1.6, მაგალითი 4). განვიხილოთ $S = 7$ შემთხვევა:

I მაკროიარუსი [ორი $(n \in nL)$ რიგის მატრიცის გამრავლება]

$$\underline{Q_1\vec{y}_0}\ \ Q_3Q_2\ \ Q_5Q_4\ \ Q_7Q_6$$

II მაკროიარუსი

$$Q_2(Q_1\vec{y}_0)\ \underline{Q_3Q_2(Q_1\vec{y}_0)}\ \ Q_6(Q_5Q_4)\ \ Q_7Q_6(Q_5Q_4)$$

III მაკროიარუსი

$$\underline{Q_4(Q_3Q_2Q_1\vec{y}_0)}\ \ \underline{(Q_5Q_4)(Q_3Q_2Q_1\vec{y}_0)}\ \ \underline{(Q_6Q_5Q_4)(Q_3Q_2Q_1\vec{y}_0)}$$

$$\underline{Q_7Q_6(Q_5Q_4Q_3Q_2Q_1\vec{y}_0)}$$

ცხადია, რომ ზემოთ აღწერილი სქემის გამოყენება შეიძლება S-ის ნებისმიერი მნიშვნელობის შემთხვევაში. ამ შემთხვევაში საჭირო იქნება $\lceil \log_2(S+1) \rceil + 1$ მაკროკონტაქტი (მაკროიარუსი), თუ გამოვიყენებთ $\left\lceil \frac{S+1}{2} \right\rceil + 1$ მაკროპროცესორს, რომელიც მაკროოპერაციის საშუალებით გადაამრავლებს ორ $(nr+1)$ რიგის მატრიცს, ე. ი. მაკროოპერაციას ვუწოდებთ ორი მატრიცის გამრავლების ალგორითმს.

თეორემა 2-ის თანახმად $(nr+1)$ რიგის ორი კვადრატული მატრიცის გადასამრავლებლად (ერთი მაკროოპერაციის განსახორციელებლად) შეიძლება ავაგოთ პარალელური ფორმა, რომლის სიმაღლეა $\lceil \log_2(nr+1) \rceil + 2$, ხოლო სიგანეა $(nr+1)^3$. ამრიგად, მთლიანი ალგორითმის განსახორციელებლად აგებული იქნება პარალელური ფორმა, რომლის სიმაღლე იქნება

$$h = \{[\log_2(S+1)]+1\} \cdot \{[\log(nr+1)]+2\},$$

ხოლო სიგანე

$$w = \left\{\left[\frac{S+1}{2}\right]+1\right\}(nr+1)^3.$$

თეორემა დამტკიცებულია.

შენიშვნა (4.3) რეკურენტული დამოკიდებულებანი ფართოდ გამოიყენება რიცხვით ანალიზში იტერაციული პროცესების განხილვისას. მათთვის პარალელური ფორმა მიიღება (4.7) – (4.8) ტოლობების საშუალებით, რომლებიც (4.3) რეკურენტული დამოკიდებულებების ექვივალენტურია მხოლოდ ზუსტი გამოთვლების შემთხვევაში. მაგრამ გამოთვლების პროცესში წარმოშობილმა გამოთვლების ცდომილებებმა შეიძლება მნიშვნელოვნად იმოქმედონ საბოლოო შედეგზე გამოთვლითი პროცესის არამდგრადობის გამო. ამიტომ, პარალელური ფორმების გამოყენებისას ამ გარემოებას განსაკუთრებული ყურადღება უნდა მიექცეს.

4.2.4. პარალელური ალგორითმები განტოლებათა სისტემებისთვის სამდიაგონალური მატრიცით.

განვიხილოთ სამდიაგონალური მატრიცი და წარმოვადგინოთ ის $A = LV$ ნამრავლის სახით:

$$A\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & & a_{k-1,kr} & a_{k-1,knL} & a_{k-1,k} \\ & & & & & 0 & a_{k\,k-1} & a_{kk} \end{pmatrix}$$

ვგულისხმობთ, რომ A მატრიცის ყველა მთავარი მინორი განსხვავებულია 0-ისგან. A მატრიცის LV სახით გაშლის შედეგად მივიღებთ

$$A = BC,$$

სადაც

$$B = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ b_{21} & 1 & 0 & \ldots & 0 & 0 \\ 0 & b_{32} & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & b_{k\,k-1} & 1 \end{pmatrix}$$

$$C=\begin{pmatrix} c_{11} & a_{12} & 0 & \dots & 0 & 0 \\ 0 & c_{12} & c_{23} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & c_{k-1,\,k-1} & c_{\,k-1,k} \\ 0 & 0 & 0 & \dots & 0 & c_{k12} \end{pmatrix}$$

გამოვიყვანოთ ფორმულები B და C მატრიცების ელემენტებისთვის. გადმოცემის სიმარტივისთვის განვიხილოთ შემთხვევა, როდესაც $k=4$.

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$C=\begin{pmatrix} 1 & 0 & 0 & 0 \\ b_{21} & 1 & 0 & 0 \\ 0 & b_{32} & 1 & 0 \\ 0 & 0 & b_{43} & 1 \end{pmatrix} C=\begin{pmatrix} c_{11} & c_{12} & 0 & 0 \\ 0 & c_{22} & c_{23} & 0 \\ 0 & 0 & c_{33} & c_{34} \\ 0 & 0 & 0 & c_{44} \end{pmatrix}$$

$$AB=\begin{pmatrix} c_{11} & c_{12} & 0 & 0 \\ b_{21}c_{11} & b_{21}c_{12}+c_{22} & c_{23} & 0 \\ 0 & b_{32}c_{22} & b_{32}c_{23}+c_{33} & c_{34} \\ 0 & 0 & b_{43}c_{33} & b_{43}c_{34}+c_{44} \end{pmatrix}$$

$A=BC$ ტოლობის გათვალისწინებით მივიღებთ:

$$c_{11}=a_{11},\ c_{12}=a_{12},$$

$$b_{21}=\frac{a_{11}}{c_{11}},\ c_{22}=a_{22}-b_{21}c_{12},\ c_{23}=a_{23},$$

$$b_{32}=\frac{a_{32}}{c_{22}},\ c_{22}=a_{33}-b_{32}c_{23},\ c_{34}=a_{34},$$

$$b_{43}=\frac{a_{43}}{c_{33}},\ b_{44}=a_{44}-b_{43}c_{34}.$$

შეიძლება დამტკიცდეს, რომ ზოგად შემთხვევაში (A მატრიცის რიგია k) სამართლიანია შემდეგი ფორმულები:

$$c_{11}=a_{11}$$

$$c_{i-1,i} = a_{i-1,i}\,,\ \ i = 2,3,\ldots,k\,, \tag{4.9}$$

$$b_{i-1,i} = \frac{a_{i-1,i}}{c_{i-1,i}}\,,\ \ i = 2,3,\ldots,k\,, \tag{4.10}$$

$$c_{i,i} = a_{i,i} - b_{i,i-1}a_{i-1,i}\,,\ \ i = 2,3,\ldots,k\,, \tag{4.11}$$

(11) ტოლობის ორივე მხარე გავამრავლოთ $c_{i-1,i-1}$-ზე, მივიღებთ

$$c_{i,i}c_{i-1,i-1} = a_{i,i}c_{i-1,i-1} - b_{i,i-1}a_{i-1,i}c_{i-1,i-1}\,.$$

ამ უკანასკნელ ტოლობაში ვისარგებლოთ (4.10) ტოლობით:

$$c_{i,i}c_{i-1,i-1} = a_{i,i}c_{i-1,i-1} - a_{i,i-1}a_{i-1,i}\,.$$

შემოვიღოთ აღნიშვნა:

$$q_i = c_{i,i}c_{i-1,i-1}\cdots c_{2,2}c_{1,1}\,,\ \ i = 2,3,\ldots,k\,,$$

(12) ტოლობის ორივე მხარე გავამრავლოთ $q_{i-2} = c_{i,i}c_{i-1,i-1}\cdots c_{2,2}c_{1,1}$ -ზე, როდესაც $i \geq 3$, მივიღებთ

$$c_{i,i}c_{i-1,i-1}q_{i-2} = a_{i,i}c_{i-1,i-1}q_{i-2} - a_{i,i-1}a_{i-1,i}q_{i-2}\,,\ \ i = 2,3,\ldots,k\,,$$

ე. ი.

$$q_i = a_{ii}q_{i-1} - a_{i-1,i}a_{i,i-1}q_{i-2}\,,\ \ i = 3,4,\ldots,k\,.$$

აღვნიშნოთ $i = j+2$, მაშინ მივიღებთ

$$q_{j+2} = a_{j+2,j+2}q_{j+1} - a_{j+1,j+2}a_{j+2,j+1}q_j\,,\ \ j = 1,2,\ldots,k-2\,. \tag{4.12}$$

თუ გავიხსენებთ (3) რეკურენტულ დამოკიდებულებას

$$\vec{X}_j = A_{j1}\vec{X}_{j-1}\cdots + A_{jr}\vec{X}_{j-r} + \vec{B}_j\,,\ \ j = 1,2,\ldots,S\,, \tag{4.3}$$

მაშინ (4.12) ტოლობა შეიძლება გადავწეროთ (4.3) სახით, თუ დავუშვებთ, რომ

$$n = 1,\ \text{ე. ი.}\ X_j\,,\ \ A_{Sj}\,,\ \ \vec{B}_j\ \ \text{რაიმე რიცხვებია}$$

$$\vec{X}_j = q_{j+2}\,,\ \ A_{j1} = a_{j+1,j+2}\,,\ \ A_{j2} = -a_{j+1,j+2}\cdot a_{j+2,j+1}\,,\ \ \vec{B}_j = 0\,,$$

$$S = k-2\,,\ \ r = 2\,.$$

ჩვენს მიერ (4.3) რეკურენტული დამოკიდებულებისთვის აგებული იქნება პარალელური ფორმა, რომლის სიმაღლეა

$$h = \{[\log_2(S+1)]+1\}\{[\log_2(rn+1)]+2\},$$

ხოლო სიგანეა

$$w = \left\{\left[\frac{S+1}{2}\right]+1\right\}(nr+1)^3.$$

თუ ამ გამოსახულებაში შევიტანთ $n=1$, $S=k-2$, $r=2$ და გავითვალისწინებთ, რომ

$$[\log_2(rn+1)]+2 = [\log_2 3]+1 = 3,$$

მაშინ დავინახავთ, რომ სამართლიანია შემდეგი თეორემა:

თეორემა 5. *k*-ური რიგის სამდიაგონალური მატრიცის LV დაშლისთვის არსებობს კვადრატული ფორმა, რომლის სიმაღლეა

$$h = 3\left\{\left[\log\frac{k-1}{2})\right]+1\right\},$$

ხოლო სიგანეა

$$w = 27\left\{\left[\frac{k-1}{2}\right]+1\right\}.$$

ახლა განვიხილოთ პარალელური ალგორითმი განტოლებათა სისტემისთვის სამდიაგონალური მატრიცით.

თეორემა 6. $A\vec{x}=\vec{y}$ განტოლებათა სისტემის ამოხსნისთვის, თუ A *k*-ურირიგის სამდიაგონალური მატრიცია, შეიძლება აიგოს პარალელური ფორმა, რომლის სიმაღლეა $0(\log_2 k)$, ხოლოს იგანეა $0(k)$.

დამტკიცება. ვთქვათ, აგებულია A მატრიცის LV დაშლა $A=BC$, მაშინ $A\vec{x}=\vec{y}$ განტოლებათა სისტემის ამონახსნი შეიძლება მიღებული იქნას ორი

$$B\vec{z}=\vec{y},\ C\vec{x}=\vec{Z}$$

განტოლებათასისტემის მიმდევრობითი ამოხსნით. შევნიშნოთ, რომ B და C ორდიაგონალური მატრიცებია.

ამ განტოლებათა სისტემების ამოხსნა დაიყვანება (3) რეკურენტულ დამოკიდებულებებამდე. მართლაც, პირველი სისტემიდან გვექნება

$$\begin{cases} Z_1 = y_1 \\ b_{21}Z_1 + Z_2 = y_2 \\ b_{21}Z_2 + Z_3 = y_3 \\ \ldots\ldots\ldots\ldots\ldots \\ b_{ii-1}Z_{i-1} + Z_i = y_i \\ \ldots\ldots\ldots\ldots\ldots \\ b_{k\,k-1}Z_{k-1} + Z_k = y_k \end{cases}$$

საიდანაც მივიღებთ

$$\begin{cases} Z_1 = y_1 \\ Z_2 = y_2 - b_{21}Z_1 \\ Z_3 = y_3 - b_{32}Z_2 \\ \ldots\ldots\ldots\ldots\ldots \\ Z_i = y_i - b_{i\,i-1}Z_{i-1} \\ \ldots\ldots\ldots\ldots\ldots \\ Z_k = y_k - b_{k,k-1}Z_{k-1} \end{cases}$$

გამოვიყენოთ (3) რეკურენტული პროცესი, სადაც $n=1$, $\vec{X}_0 = 0$, $\vec{X}_j = Z_j$, $B_j = y_j$, $j = 1,2,\ldots,k$, $S = k$, $r = 1$, $A_{11} = 0$, $A_{i1} = -b_{i,r-1}$, $i = 2,3,\ldots,k$.

თუ გამოვიყენებთ თეორემა 4-ს, მაშინ მივიღებთ, რომ Z_i $(i = 1,2,\ldots,k)$ რიცხვების მოსაძებნად შეიძლება ავაგოთ პარალელური ფორმა, რომლის სიმაღლეა $h = 3\{[\log(r+1)]+1\}$, ხოლო სიგანეა $w = 8\left\{\left[\frac{k+1}{2}\right]+1\right\}$.

ანალოგიური შედეგი მიიღება მეორე განტოლებისთვის. თუ გავითვალისწინებთ თეორემა 5-ს, მივიღებთ, რომ *k*-ური რიგის განტოლებათა სისტემის, რომლის მატრიცი სამდიაგონალურია, ამოხსნისთვის შეიძლება აიგოს პარალელური ფორმა, რომლის სიმაღლეა $0(\log_2 k)$, ხოლო სიგანეა $0(k)$.

თეორემა დამტკიცებულია.

გამოყენებული ლიტერატურა:

1. Tanenbaum A. Structured Computer Organization. Prentice Hall, 2012. – 800 p. – 6th ed. – ISBN: 0132916525, 9780132916523. http://www.twirpx.com/file/1079632/

   (რუსული თარგმანი - Таненбаум Э., Остин Т. Архитектура компьютера, 6–е изд. – СПб.:Питер. 2013.816 с. — ISBN 978-5-496-00337-7.) http://www.twirpx.com/file/1356902/
2. ფრანგიშვილი ა, ქურდაძე მ, გასიტაშვილი ზ, ფაილოძე ნ. კომპიუტერული ქსელების ინჟინირინგის საფუძვლები, საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", 2008.
3. Барский А. Архитектура параллельных вычислительных систем. http://www.intuit.ru/studies/courses/80/80/info
4. Богданов А. Архитектуры и топологии многопроцессорных вычислительных систем. http://www.intuit.ru/studies/courses/45/45/info
5. Fayez Gebali. Algorithms and Parallel Computing, University of Victoria, BC. Viley.2011. http://www.rulit.net/books/algorithms-and-parallel-computing-download-free-205613.html
6. Сухорослов О. Параллельные и распределенные вычисления. http://numeralis.ru/parallelnyie-i-raspredelennyie-vyichisleniya-raspredelennyie-hranilishha-dannyih-suhoroslov-oleg/
7. Гергель В. Основы параллельных вычислений. http://www.intuit.ru/studies/courses/1091/293/info
8. Проектирование высоконагруженных систем. http://www.intuit.ru/studies/courses/3493/735/info
9. В.В.Воеводин, Вл.В.Воеводин «Параллельные вычисления»-СПб.: БХВ-Петербург,2002.-608 с.
10. Олифер В.Г., Олифер Н.А.. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 3-е изд.- СПБ: Издат. «Питер», 2006. http://www.alleng.ru/d/comp/comp28.html