O’ZBEKISTON RESPUBLIKASI OLIY VA O’RTA MAXSUS TA’LIM VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI NUKUS FILIALI

«Parallel kompyuterlarning arxitekturasi va dasturlash» fanidan

O’QUV-USLUBIY MAJMUASI

Bilim sohasi: 300000 - Ishlab chiqarish texnik soha

Ta`lim sohasi: 330000 - Kompyuter texnologiyalari va informatika

Ta`lim yo`nalishi: 5330500 – Kompyuter injiniringi

NUKUS - 2018

Ushbu o’quv – uslubiy majmua Oliy va o’rta maxsus ta’lim vazirligining 201_ - yil «__» _____ ____ - sonli buyrug’i bilan tasdiqlangan o’quv reja va dastur asosida ishlab chiqildi.

Tayyorladi:

A.B.Orinbaev – Muhammad al-Xorazmiy nomidagi TATU Nukus filiali «Dasturiy injiniring» kafedrasi assistenti.

O’quv-uslubiy kompleks Muhammad al-Xorazmiy nomidagi TATU Nukus filialining 201_ - yil _____ dagi ___ - sonli qarori bilan tasdiqlashga taklif etildi.

1. O’quv materiallari:
    1.1. Ma’ruza mashg’ulotlar
    1.2. Amaliy mashg’ulotlar
    1.3. Laboratoriya ishlari
    1.4. Glossariy
2. Ilova:
    2.1. O’quv dasturi
    2.2. Ishchi o’quv dasturi
    2.3. Tarqatma materiallar
    2.4. Testlar
    2.5. Ishchi fan dasturiga muvofiq baholash mezonlarini qo’llash bo’yicha uslubiy ko’rsatmalar

M U N D A R I J A

MA'RUZA MASHG'ULOTLARI..... 7

1-mavzu: Kirish. Parallel xisoblash uchun mo'ljallangan masalalari. Parallelashtirish – ishlov berish tezligini oshirishning bir yo'li 8

2-mavzu: Xotira turlari va xossalari, xotira ierarxiyasi..... 13

3-mavzu: Dasturiy ta'miynot sathi. Mashina kodi, mnemakod va dasturlash tillari 19

4-mavzu: Unumdorlikni oshirish: konveyer va superskalyar qayta ishlash, VLIW arxitekturasi. Birinchi superkompyuterlar..... 24

5-mavzu: Parallel kompyuterlar: umumiy va ajratilgan xotirali multiprosessorlar va multikompyuterlar..... 30

6-mavzu: Xotiraga bir xil bo'lmagan ruxsatlilik texnologiyasi, NUMA va ccNUMA arxitekturasi, kesh mustahkamligi 34

7-mavzu: Parallel kompyuterlarning dasturiy ta'miynoti, parallel dasturlash, tillarning kengayishi, maxsus tillar, kutubxonalar va interfeyslar takomillashuvi..... 38

8-mavzu: Dasturlash modellari, unumdorlikni baholash. Amdal qonuni. Ma'lumotlarni va buyruqlarni parallelashtrish 42

9-mavzu: Parallelashtirish tizimlarining arxitekturasi, Flinn klasifikatsiyasi, MIMD arxitekturasi..... 44

10-mavzu: Vektor-konveyer arxitekturasi, CRAY 90 kompyuteri (funksional bloklar, yuqori unumdorlik darajasi), umumiy xotirali parallel kompyuterlar, HP Superdome kompyuteri..... 47

11-mavzu: Ajratilgan xotirali kompyuter tizimlari, klaster tizimlar (Beowulf), MBC – 1000M superkompyuteri, klasterlarining kommunikatsion tizimlari 50

12-mavzu: Grid – texnologiyasi va metaxisoblash, unumdorlikni baholash (MIPS, FLOPS) 53

13-mavzu: Parallelashtirish algoritmlarining samaradorlik ko'rsatkichlari, alohida yig'indini xisoblash usullari, yig'indini kaskad chizmasi..... 56

14-mavzu: Vektor va matritsani ko'paytirsh, matritsa elementlarini taqsimlash, vazifani protsessorlarga taqsimlash..... 58

15-mavzu: Ko'p yadroli prosessorlar, xotirani va ichki kommunakatsiyani tashkillashtirish, dasturiy oqimlar, oqimli qayta ishlash texnologiyasi..... 60

16 -mavzu: Oqimli qaya ishlashning standart vositalari, API standarti, OpenMP universal standarti, SSE kengaytirish buyruqlari, Intel ning ko'p yadroli prosessorlari uchun oqimli qayta ishlash 63

AMALIY MASHG'ULOTLAR..... 68

1-Amaliy ish: Parallel xisoblash tahlili va modellashtirish..... 69

2-Amaliy ish: Parallelashtirish usullarini yaratish asoslari 71
3-Amaliy ish: Vektorni matritsaga ko'paytirishning parallel usullari 76
4-Amaliy ish: Matritsali ko'paytirishning parallel usullari 79
5-6-Amaliy ish: Chiziqli tenglamalar tizimini yechish 82
7-Amaliy ish: Parallel saralash usullari 85
8-Amaliy ish: Graflar ustida parallel amallar 90
Glossariy 95
FAN ISHCHI DASTURI ..... Ошибка! Закладка не определена.

«PARALLEL KOMPYUTERLARNING ARXITEKTURASI VA DASTURLASH» FANIDAN

MA’RUZA MASHG’ULOTLARI

1-mavzu: Kirish. Parallel xisoblash uchun mo’ljallangan masalalari. Parallelashtirish – ishlov berish tezligini oshirishning bir yo’li

REJA:

1.1. Parallel xisoblash tizimlari

1.2. Parallel kompyuterlar

1.3. Parallel dasturlash

Eng avvalo kompyuterda parallel dasturlash kerakmi degan savolga javob olish kerak. Lekin bu savol javob olishni istagan yagona savol emas. Aynan shuning uchun ham, parallel hisoblash dunyosini tushunish qiyin bo'lgan sodda, tushunarli, tushunarli dunyodan navbatdagi hisob-kitoblardan nima o'tish kerakligini tushunish ham muhimdir. Parallel hisoblashning afzalliklari nimadan iborat va parallel hisobga yo'naltirilgan dasturlarni yaratishda dasturchi uchun qanday muammolar kutilmoqda. Ushbu savollarga javob berish uchun keling, kompyuterni rivojlantirish tarixini tezroq ko'rib chiqamiz.

Birinchi kompyuterlar Fon Neyman tomonidan ishlab chiqilgan printsiplarga muvofiq qurilgan. Ularning uchta asosiy komponenti bor edi: xotira, protsessor va kirish va chiqish ma'lumotlarini beruvchi tashqi qurilmalar to'plami.

Xotira ko'p darajali va tashqi xotirasi va ichki xotirasi bo'lgan birinchi kompyuterlar uchun - operatsion va ro'yxatga olish xotirasi. Tashqi xotira (magnit lenta, punch karta, disklarda) kompyuterning yoqilgan yoki yoqilmaganligidan qat'iy nazar, dastur va ma'lumotlarni saqlash imkonini berdi. Ichki xotira faqat kompyuter bilan sessiya davri uchun ma'lumot saqlanadi. Kompyuterni o'chirib qo'ysangiz, ichki xotiraning mazmuni g'oyib bo'ldi.

Dastur kompyuterda bajarilishi uchun u RAMga yuklanishi kerak edi. U o'sha dasturda ishlangan ma'lumotlar kabi saqlangan. Xotirada saqlangan dasturning printsipi Von Neumann kompyuterlarining asosiy tamoyillaridan biridir.

Ro'yxatdan o'tish xotirasi hisoblash vaqtida ishlatilgan. Ma'lumotlar bo'yicha ba'zi operatsiyalarni bajarishdan oldin, ma'lumotlar registrlarda joylashtirilishi

kerak. Ushbu tezkor xotira turi ma'lumotlar bo'yicha operatsiyalarni bajarishda zarur tezlikni ta'minladi.

Barcha operatsiyalarni bajarish - hisoblash jarayonini boshqarishda ma'lumotlar va operatsiyalar bo'yicha operatsiyalarni protsessor amalga oshirdi. Kompyuter protsessori o'ziga xos ko'rsatmalarga ega edi. Ushbu to'siq potentsial hisoblash funktsiyasini hisoblash uchun universal edi. Boshqa tomondan, ushbu vosita odamlarning yozish dasturlarining nisbiy soddaligini ta'minladi.

Dastlabki kompyuterlar uchun dasturlar, amaldagi protsessor buyruqlar majmuasiga kiritilgan qator buyruqlarni ifodalaydi. Dasturni kompyuterda ijro etish juda oddiy edi. Har safar kompyuterda bitta dastur bajarilgan. Protsessor, dasturga muvofiq ketma-ket navbatdagi buyruqlar ketma-ketlikda bajarildi. Barcha kompyuter resurslari - xotira, protsessor vaqti, barcha qurilmalar - dasturning to'liq tasarrufida edi va hech narsa uning ishiga aralashmasdi (albatta odamni hisobga olmagan). Parallelizm ko'zga ko'rinmasdi.

Bu idial juda uzoq vaqt davomida juda qimmat bo'lmagan kompyuter resurslari samarasiz ishlatgani tufayli uzoq davom etmadi. Kompyuterlar o'chirilmadi, bitta dastur boshqasini o'zgartirdi.

Yaqin orada kompyuter protsessor bilan birga markaziy protsessor deb nomlanuvchi qo'shimcha protsessorlarga, eng avvalo, sekin komutlarni bajarish uchun mas'ul bo'lgan kirish / chiqish qurilmalarining maxsus protsessorlariga ega edi. Bu esa, bir vaqtning o'zida bir nechta dastur kompyuterda ishlayotgani - dastur natijalarini nashr etishi, ikkinchisi - bajarilishi va uchinchisi - masalan, magnit tasmasi yoki boshqa tashqi vositadan ma'lumotlarni kiritish uchun dasturni bajarishning ommaviy rejimini tashkil etishga imkon berdi.

Inqilobiy qadam 1964 yilda IBM - OS 360 operatsion tizimining paydo bo'lishi bo'ldi. Kompyuterda paydo bo'lgan operatsion tizim uning mutlaq egasi bo'ldi - barcha resurslari menejeri. Endilikda foydalanuvchi dasturi faqat operatsion tizim nazorati ostida bajarilishi mumkin. Operatsion tizim ikkita muhim vazifani hal etishga imkon berdi: bir tomondan, bir vaqtning o'zida kompyuterda ishlashning barcha dasturlariga zarur xizmatni taqdim etish, ikkinchidan, mavjud resurslarni

ushbu resurslarga da'vo qilayotgan dasturlar orasida samarali foydalanish va tarqatish. Operatsion tizimlarning paydo bo'lishi bitta dasturli rejimdan ko'p dasturli rejimga o'tishga olib keldi, bir vaqtning o'zida bir xil dasturda bir nechta dastur mavjud. Ko'p dasturlash parallel dasturiy emas, biroq bu parallel hisoblash uchun bir qadamdir.

Ko'p dasturlash - bir nechta dasturlarni parallel bajarish. Ko'p dasturlash sizga ularni bajarish uchun umumiy vaqtni kamaytirish imkonini beradi.

Parallel hisoblashda bir xil dasturni parallel bajarish nazarda tutiladi. Parallel hisoblash bir dasturning bajarilish vaqtini kamaytirish imkonini beradi.

Ko'p dasturlash uchun kompyuterning bir nechta protsessorlarga ega bo'lishi juda muhim. Ko'p dasturlashni amalga oshirish uchun protsessorlarning o'zaro ishlashini tashkil qiluvchi operatsion tizim mavjudligi etarli. Parallel hisoblash uchun

Dasturning o'zi uchun zarur bo'lgan qo'shimcha talab mavjud - dastur hisoblarni parallellashtirish imkoniyatini yaratishi kerak, chunki operatsion tizimning ko'rinishi kompyuterni apparat (xotira, protsessorlar, boshqa qurilmalar) deb hisoblash mumkin emasligini anglatadi. Endi u ikki qismga ega: qattiq (qattiq) va yumshoq (yumshoq) - bir-birini to'ldiruvchi apparat va dasturiy komponentlar. Yarim asrdan ko'proq vaqt mobaynida komponentlar tez rivojlana boshladi, asbob-uskunalar uchun eksponentsional o'sishni odatiy holga keltirdi, bu Murning taniqli ampirik qonunida aks ettirilgan - barcha muhim belgilar kattalashib ketgan - barcha darajalarda xotira hajmi, xotiraga kirish vaqtini kamaytirish, protsessor tezligi. Murning qonuniga ko'ra (Gordon Moore Intelning asoschilaridan biri), xarakterli qiymatlar har yarim yilda ikki baravarga ko'paydi. Kompyuterga kiritilgan protsessorlarning soni ham ortdi. O'zgarildi va kompyuter arxitekturasi. Ushbu o'zgarishlar ko'p jihatdan hisoblarni parallellashtirishga qaratilgan qadamlar edi. Bu erda parallelizatsiya jarayoni bilan bevosita bog'liq bo'lgan protsessor arxitekturasidagi o'zgarishlarning bir qismi: Buyruqlar chizig'ini qayta ishlash. Protsessor tomonidan buyruqlar oqimini bajarish jarayoni endi buyruq buyrug'i ketma-ket ravishda bajarilmasligi sifatida ko'rilmaydi. Buyruqlar oqimini qayta

ishlash jarayoni quvur liniyasida amalga oshirildi, shuning uchun bir nechta buyruqlar bir vaqtning o'zida bajarishga tayyorlandi. Bir-biriga bog'liq bo'lmagan buyruqlar bir vaqtning o'zida bajarilishi mumkin, bu allaqachon haqiqiy parallelizmdir. "Uzoq buyruqlar". Ba'zi bir kompyuterlarning arxitekturasi bir nechta protsessorlarni o'z ichiga olgan bo'lib, ular mantiqiy va arifmetik operatsiyalarni butun sonlar bo'yicha bajarish imkonini beradi, bir nechta protsessorlar suzuvchi nuqtali raqamlarda operatsiyalarni amalga oshiradi. Uzoq buyruq bitta buyruqda mavjud protsessorlarning har biri bajarishi kerak bo'lgan amallarni ko'rsatishga imkon berdi. Bu esa, apparat darajasida parallelizmni amalga oshirish imkonini berdi Vektorli va matritsali protsessorlar. Ushbu protsessorlarning ko'rsatmalar to'plami vektorlar va matritsalar bo'yicha asosiy operatsiyalarni o'z ichiga oladi. Masalan, bitta guruh ikkita matritsani qo'shishlari mumkin. Bunday buyruq parallel hisoblashlarni amalga oshiradi. Ushbu operatsiyalar ma'lumotni qayta ishlash asoslarini tashkil etuvchi ilovalar keng tarqalgan. Ma`lumotlarning parallel ishlashi ushbu klassdagi ilovalarning samaradorligini sezilarli darajada oshirishi mumkin. Dasturiy ta'minot darajasida parallel ijro etiladigan dasturlarning yana bir muhim turi - grafik tasvirlar bilan intensiv ishlash. Ushbu ishlash grafik ishlovchilar tomonidan amalga oshiriladi. Grafik tasvirni ballar to'plami sifatida ko'rish mumkin. Rasmni qayta ishlash ko'pincha hamma punktlarda bir xil operatsiyani bajarish uchun kamayadi. Ushbu vaziyatda ma'lumotlar parallelizatsiyasi osongina amalga oshiriladi. Shu sababli, grafik protsessorlar avvaldan ko'p yadroli bo'lib, bu jarayonni parallellash va tasvirni samarali ishlash imkonini beradi. Superkompyuterlar hozirgi vaqtda eng yuqori ko'rsatkichlarga ega bo'lgan kompyuterlarni o'z ichiga oladi. Ular yuz minglab protsessorlardan iborat. Superkompyuterlardan samarali foydalanish hisob-kitoblarning eng keng tarqalgan parallelligini o'z ichiga oladi .. Ilmiy tadqiqotlarda va yangi texnologiyalarda mavjud hisoblash tizimlarining barcha kuchini talab qiluvchi vazifalar mavjud. Mamlakatning ilmiy salohiyati ko'p jihatdan o'zining superkompyuterlari mavjudligi bilan belgilanadi. Superkompyuterning kontseptsiyasi nisbatan nuqtai nazardir. O'n yillik superkompyuterning xususiyatlari odatdagi kompyuterning xususiyatlariga

mos keladi. Bugungi superkompyuterlar petafloplarda (1015 dona perimetrli operatsiyalar) o'lchovlarda ishlaydi. 2020 yilga qadar superkompyuterlarning ishlashi 1000 barobarga oshadi va eksaflopslarda o'lchov qilinadi Kompyuterlar tasniflash Kompyuterlar dunyosi miniatyura o'rnatilgan kompyuterlardan individual binolarni ishlaydigan ko'p tonna superkompyuterlarga qadar farq qiladi. Ular turli yo'llar bilan tasniflanishi mumkin. Birinchi va eng sodda tasniflardan biri - Flynn tasniflashini ko'rib chiqing, bu ma'lumotlar kompyuterda qanday ishlashga asoslangan. Ushbu tasnifga ko'ra, barcha kompyuterlar (komp'yuter komplekslari) to'rtta sinfga bo'linadi - arxitekturali kompyuterlar: SISD (Single Instruction stream - yagona ma'lumotlar oqimi) - bitta ma'lumot oqimi - bitta ma'lumot oqimidir. Bu sinf, programma buyruqlar ketma-ket bajarilganda, keyingi ma'lumotlar elementini qayta ishlashda von Neumann arxitekturasiga ega oddiy "ketma-ket" kompyuterlarni o'z ichiga oladi SIMD (bitta yo'riqnoma oqimi - bir nechta ma'lumotlar oqimi) - bitta buyruq xartasi - bir nechta ma'lumotlar oqimi. Vektorli va matritsali protsessorlarga ega kompyuterlar ushbu turga tegishli: MISD (bir nechta yo'riqnoma oqimi - yagona ma'lumotlar oqimi) - bir nechta buyruqlar oqimi - bitta ma'lumot oqimi.

Ushbu turdagi ma'lumotlarni o'tkazishning konveyer turiga ega kompyuterlar bo'lishi mumkin. Biroq, ko'pchilik bunday kompyuterlarning birinchi turiga havola etilishiga va MISD klassi kompyuterlari hali yaratilmaganligiga ishonishadi. Ko'p yo'riqnomalar oqimi (ko'p ma'lumotli oqim) - bir nechta buyruqlar oqimi - ko'p ma'lumotli oqimlar. MIMD klassi juda keng va bugungi kunda juda ko'p turli xil me'morchilikning ko'plab kompyuterlari unga kiradi. Shuning uchun, MIMD klassiga tegishli bo'lgan kompyuterlarni aniqroq tasniflash imkonini beradigan boshqa tasniflashlar taklif etiladi.MIMD sinfidagi kompyuterlarning batafsil tasnifini ko'rib chiqamiz. Biz faqat kompyuterlarni uchta sinfga bo'lishning yana bir usuliga to'xtalamiz: Multiprocessor hisoblash tizimlari - umumiy xotirada ishlaydigan ko'p protsessorli kompyuterlar. Bu sinf bozorda bugungi kunda sotilgan ko'p yadroli kompyuterlarning ko'pchiligini o'z ichiga oladi.Multikompyuterli hisoblash tizimlari yuqori tezlikda aloqa liniyalari orqali ulangan kompyuterlarning ko'pini anglatadi. Har bir kompyuterda o'z xotirasi bor va ma'lumotni uzatish uchun

tizimdagi boshqa kompyuterlar bilan xabarlar almashadi. Bu sinf klasterlarni o'z ichiga oladi. Kümelenme, bir serverning rolini o'ynaydigan bir necha shaxsiy kompyuter bilan butun hisoblangan hisoblash kompleksidir. Klasterga kiradigan kompyuterlar odatiy kompyuter bo'lishi mumkin, klasterlar nisbatan arzon. Yuqori 500 ta superkompyuterlarning aksariyati klasterlar bo'lib, gibrid hisoblash komplekslari ko'plab nodlardan tashkil topgan bo'lib, ularning har biri ko'p yadroli, ko'p protsessor, grafik protsessor yoki vektorli protsessor bo'lishi mumkin. Bunday komplekslar odatda superkompyuterlardir.

Nazorat savollari

1. Kompyuter arxitekturalari
2. Dastlabkli kompyuterlarning ishlash tamoyillari
3. Dastlabki parallel xisoblash tizimlari haqida
4. Parallelashtirishni ma’nosi

2-mavzu: Xotira turlari va xossalari, xotira ierarxiyasi

REJA:

2.1. Komyuter xotirasi turlari

2.2. Tezkor xotira va uning vazifasi

2.3. Registr va kesh xotira

2.4. Xotira ierarxiyasi

Masalan, kompyuter juda katta muammolarni, masalan, maksimal kattalikdagi matritsali chiziqli algebraik tenglamalar tizimini yechadi. Agar Gauss usuli ishlatilsa, elementlar eng xil tartibda chiqarib tashlanishi mumkin. Shu bilan birga, amalga oshirilgan arifmetik operatsiyalarning umumiy soni bir xil bo'lib qoladi yoki 2-3 martadan ortiq o'zgarmaydi. Shu bilan birga, bir xil tizim uchun usulning turli xil versiyalarini ataylab, muammoni hal qilishning umumiy vaqti o'n barobar yoki undan ko'p bo'lishi mumkinligini ta'kidlaymiz. Yana bir haqiqiy holat. Gauss usulining har qanday variantini aniqlasin, ammo hozirda turli o'lchamdagi tizimlar

hal qilinmoqda. Bunday holda, muammoni hal qilishning umumiy vaqti arifmetik operatsiyalar umumiy soniga mutanosib ravishda o'zgaradi. Ammo bu holda kutilmagan hodisalar mavjud. Ba'zan, ayrim buyruqlar uchun muammoni hal qilish vaqti ancha uzoq bo'lishi mumkin.

Muammoni hal qilishning umumiy vaqti ko'pgina omillarga ta'sir qiladi. Ammo ularning asosiylari arifmetik operatsiyalarni bajarish vaqti va xotira bilan o'zaro aloqa qilish vaqti. Gauss tipidagi usullarda, arifmetik operatsiyalarning butun majmui oldindan ma'lum. Shuning uchun, masalani muayyan sabablarga ko'ra hal qilish vaqti keraksiz bo'lsa, unda xotiradan foydalanishda ma'lum bir noto'g'ri ma'lumotlar mavjud. Bu degani, xotira tuzilishi va uni qo'llash tamoyillarini chuqurroq muhokama qilish uchun sabablar mavjud. Ayniqsa, katta muammolarni hal qilish kerak bo'lsa.

Xotira kompyuterda qanday bo'lishidan qat'i nazar, axborotning har bir biti modellashtirilgan ikki davlatni oladi eng oddiy texnik element. Bunday elementlarning zichligi juda yuqori. Misol uchun, elektron xotira qurilmasi holatida, bir yongada elementlarning soni millionlab kishilarga yetishi mumkin. Biroq, bu xotira o'zboshimchalik bilan katta bo'lishi mumkin degani emas.

Xotiraning asosiy talabi - alohida so'zlarga qisqacha kirish vaqti. Umuman olganda, so'zlar bo'yicha operatsiyalarni bajarish uchun zarur bo'lgan vaqtdan, o'ta og'ir holatlarda, unga mos keladigan vaqtdan ancha qisqa bo'lishi kerak. Xarakterli kirish vaqti operatsiyalarni bajarish vaqtidan sezilarli darajada qisqartiriladigan xotiraning bir qismi tez deb nomlanadi, qolganlari sekinlashadi. Katta qismi adreslanadigan xotirani tashkil qiladi, kichikroq qismi esa manzilsiz hisoblanadi. Manzilning xotirasida har bir so'zda manzil mavjud. Xotiraning bu qismi foydalanuvchi uchun mavjud. U bilan ishlashingiz uchun siz ikkala ma'lumotni yozishingiz va o'qishingiz mumkin. U shuningdek, tasodifiy erkin xotira (RAM) deb nomlanadi va tegishli texnik qurilmalar tasodifiy erkin xotira (RAM) deb ataladi.

Noma'lum xotira foydalanuvchilar uchun mavjud emas. U doimiy xotira va ultrafast xotira o'rtasida farq qiladi. Hech narsa doimiy xotiraga yozilmaydi, lekin bundan oldin yozib olingan ma'lumotlarni bir necha bor o'qib chiqish mumkin.

Odatda, kompyuterni ishga tushirish buyruqlar, turli foydali dasturlar, va hokazolarni saqlaydi. Optik texnologiyalardan foydalanish katta doimiy xotira yaratish imkoniyatini ochadi. Bu holatda, masalan, ko'p maqsadli dasturlarning kutubxonalaridagi barcha yuklarni saqlab qolish mumkin bo'ladi.

Ultra tez xotira, operatsion xotiradan sezilarli darajada kam kirish vaqti bilan farq qiladi. Ko'pincha 1-2 darajaga ega. Eng tezkor daraja - bu xotirani saqlash. Birlik yoki o'nlab so'zlar bilan o'lchangan juda oz miqdori bor. Operatsiyani bajarish natijalari, bajariladigan bajarishdan darhol buyruqni bajarish uchun kerak bo'ladi. Aslida, ro'yxatga olish xotirasi ALU ning ajralmas qismi hisoblanadi. Kesh deyarli tez. Bu registr xotirasi va RAM orasida bir turdagi bufer. Zamonaviy kompyuterlarda uning miqdori bir million so'zga yaqinlashadi. Tez orada bajarilgan buyruqlar bajaradigan buyruqlarni bajarish uchun zarur bo'lgan operatsiya natijalarini saqlaydi. Ultrafastli xotira boshqaruvini ishlatishini nazorat qiladi. Dasturlarni kompyuter kodiga aylantirish bosqichida derleyici buyruqlar yordamida ultrafastli xotiradan foydalanish samaradorligini maksimal darajada oshirishi mumkin. Lekin bu har doim ham bajarilmaydi.

RAMga qisqa vaqt ichida erishish uchun ko'plab shartlarni bajarish kerak, xususan, uning elementlariga boshqaruv signallarining o'tishini kamaytirish. Boshqa narsalar bilan bir qatorda, mavjud texnologiyalarda u ulanishlarning uzunligiga bog'liq. Ko'pgina elementlar bilan nazariy jihatdan ulanishning uzunligi bir xil darajada kichik bo'lishi mumkin emas. Bundan tashqari, noaniqlik ko'proq, elementlarning soni ham katta. Bunday holat faqatgina xotiraning unumsizligi va uning ishlatilishiga olib kelishi kerak. Tasodifiy erkin foydalanish xotirasini loyihalashda turli xil texnik echimlar katta kirish vaqtining tarqalishining salbiy ta'sirini kamaytirish uchun qo'llaniladi.

Xususan, RAM ierarxik tarzda amalga oshiriladi, bu xotirani kublar, bloklar, bo'limlar, sahifalar va hokazolarga bo'lishiga mos keladi. Ierarxiya darajasi ancha yuqori bo'lsa, ushbu darajadagi bitta xotira qismining alohida so'zlari uchun kirish vaqtlarining tarqalishi qancha kichikroq. Eng yuqori darajada, bir vaqtning o'zida yoki minimal vaqt farqiga ega bo'lgan so'z guruhlari mavjud. Ko'pincha ular ketma-

ket jismoniy manzillar yoki doimiy qadam bilan o'zgaradigan manzillar bo'lgan so'zlardir.

Shunday qilib, "oddiy" kompyuterning RAMini etarli darajada kengaytirish istagi uning tuzilishining murakkablashishiga olib keladi. O'z navbatida, bu muqarrar ravishda erkin so'zlarni tarqatish vaqtini alohida so'zlar bilan to'ldiradi. RAM bilan ishlashning eng yaxshi usuli odatda ikkala operatsion tizim va kompilyator tomonidan ham quvvatlanadi. Lekin har qanday dastur uchun u har doim amalga oshirilmaydi. Buni ishlatish uchun programmuvchi oldindan ma'lum bo'lishi kerak bo'lgan aniq belgilangan qoidalarga amal qilish kerak.

Agar algoritmni amalga oshirish uchun faqatgina biron-bir dasturni yaratish emas, balki imkon qadar tez ishlashini ta'minlash zarur bo'lsa, ushbu qoidalarga rioya qilish sezilarli ta'sirga ega bo'lishi mumkin. Yuqorida aytib o'tilganidek, ko'plab kompyuterlar tezda jismoniy manzillarga ega bo'lgan so'zlarga tezlik bilan kirishadi. Misol uchun, dastur Fortran tilida yozilgan. Ushbu til nuqtai nazaridan satrlar yoki ustunlar qatori elementlarini qayta ishlash muhim emas. Olingan dasturning murakkabligi jihatidan ahamiyatga ega emas. Barcha tafovutlar faqat bitta holatda biz ba'zi matrisalarning elementlarini konvertatsiya qilish bilan, ikkinchisida esa - ularga transpozitsiya bilan shug'ullanishimiz bilan bog'liq. Xotiradan foydalanishning o'ziga xos xususiyatlarini hisobga olgan holda, Fortran tilidagi kompilyatorlar doimo jismoniy xotirada, manzillar, ustunlar ustuni, chapdan o'ngga va yuqoridan pastgacha ketma-ketlikda bo'ladi. Gauss usulida lineer algebraik tenglamalar tizimlarini yechish uchun algoritmni qo'llashni nazarda tuting. Bunday holda, dasturning vaqti uch o'lchamli tsikllar bilan belgilanadi. Tizimning matrisi ustunlar yoki qatorlar qatorida o'rnatiladigiga qarab, dasturning ichki aylanishi ham kolonlarning elementlarini ustunlar yoki satrlarda ishlov beradi. Ikkala variantni amalga oshirish vaqti juda sezilarli, ba'zan bir necha marta farq qilishi mumkin. Tegmaslik dasturlarni ishlab chiqishda esa bu muhim ahamiyatga ega. Muhokama qilingan masala, asosan, muammolarni hal qilishda tezkor RAMni tezkor ravishda ishlatish bilan chegaralanishi mumkin bo'lgan holatlar bilan bog'liq. Sekin xotiradan foydalanish kerak bo'lsa, vaziyat ancha murakkablashadi.

Dasturlash tillarining aksariyati xotira hajmining kontseptsiyasiga ega emas. Buning sababi, dasturlarni kompyuter parametrlariga bog'liq bo'lmagan holda qilishdir. Biroq, ma'lum bir kompyuterda foydalanuvchi faqat cheklangan hajmdagi xotira bilan ta'minlanishi mumkin. Uning hajmini maksimal darajada oshirish istagi nafaqat tezkor RAMni, balki xotirani sekinlashtirishni ham talab qiladi. Zamonaviy kompyuterlarda sekin xotira tez-tez qattiq disklarda qo'llaniladi. Uning hajmi RAM hajmidan bir necha barobar katta. Biroq, kirish vaqti bir necha bor ko'proq. Juda katta muammolarni hal qilishda ma'lumotlarning katta qismi muqarrar ravishda sekin xotirada saqlanishi kerak. Ammo har qanday kompyuterda operatsiyalar faqat tez xotirada saqlangan ma'lumotlarga ko'ra amalga oshiriladi. Shu sababli, katta muammolarni hal qilish jarayonida, odatda, ma'lumotlar bir necha marta sekin va tezkor xotiraga uzatilishi kerak. Bu katta muammolarni hal qilishning muddati ikkita komponent tomonidan belgilanadi: algoritm operatsiyalari bajarilish vaqti va sekin va tezkor xotira o'rtasida almashinuvni bajarish vaqti. Sekin xotiradan foydalanishning asosiy muammo - bu ikkinchi guruhning birinchisini o'nlab, yuzlab, hatto minglab marotaba oshib ketishiga olib kelishi mumkin. Eslatib o'tamiz, RAM bilan ishlashning muvaffaqiyatsiz tashkil etilishi algoritmni amalga oshirish vaqtini bir necha bor oshirishi mumkin. Sekin va tezkor xotira o'rtasidagi almashinuvning malakali tashkil etilishi uchta asosiy tamoyilga asoslangan: sekin xotiraga imkon qadar kamroq kirish, almashinuvni iloji boricha ko'proq ma'lumotni uzatish va tezkor xotiraga har bir uzatish uchun iloji boricha ishlov berish. Ushbu printsiplarning maqbul muvozanatini saqlash juda murakkab vazifadir. O'z qarorini o'ylamaslik uchun, foydalanuvchi odatda virtual xotira bilan ishlashga taklif etiladi. Ushbu xotira shartli. Yaxshi belgilangan o'lcham va aniq so'zlashuv tizimi mavjud. Virtual xotira bir hil yoki bir necha tuzilishga ega bo'lishi mumkin. Buning uchun axborotni oldindan joylashtirish uchun qoidalar tizimi ishlab chiqilishi mumkin. Jismoniy xotirada virtual xotira qanday aniqlanadi va sekin va tezkor xotira o'rtasidagi almashinuvlar, operatsion tizimga biriktirilgan algoritmlarga bog'liq. Ushbu algoritmlar haqida kompyuterning hujjatlarida topish mumkin. Shunga

qaramasdan, odatda, ular odatiy bo'lib, shuning uchun ham umuman hisobga olinmaydi yoki ma'lum dasturlarning xususiyatlarini hisobga olmaydilar.

E'tibor bering, M-20 va BESM-6, bu hisoblash texnologiyasi kashshoflari "oddiy" kompyuterlar sinfiga tegishli. Lekin yuqorida muhokama qilingan barcha muammolar bir xil sinfdagi zamonaviy kompyuterlar uchun dolzarbdir. Buni hamma ham osongina ko'rish mumkin. Shaxsiy kompyuteringiz va M-20 ning ish faoliyatini solishtiring. Endi sizning kompyuteringiz Gauss usuli bo'yicha 200-darajali chiziqli algebraik tenglamalar tizimini belgilab oling va uni hal qilish uchun vaqtni o'lchaylik. Keyinchalik, ikkala kompyuter uchun tizimning ishlash muddatini va tizimning ishlash muddatini solishtirish. Va taqqoslash sizning kompyuteringiz foydasiz emasligiga ishonch hosil qiling. Keyin nima uchun bunday bo'lishi mumkinligi haqida savol berish oqilona. Agar qattiq disk yordamida sekin xotira sifatida katta tartibli tizimni tanlasangiz, tizimlar tartibidagi farqlarni hisobga olgan holda taqqoslash natijasi bundan ham yomonroq bo'ladi. Yana o'zingizga shunday savol bering. Bu erda muhokama qilingan narsa, faqat javob izlashga to'g'ri keladi.

Nazorat savollari

1. Birinchi elektron hisoblash mashinalarining tashqi xotirasi sifatida magnit lenta ishlatilganligini bilarmidingiz?

2. 1956-yilda tashqi qattiq disk xotirasi birinchi marta ishlab chiqilganini bilasizmi, har bir diskda faqat bir necha megabayt hajm va stolning o'lchami bor edi?

3. Birinchi marta virtual xotira kontseptsiyasi 1962 yilda ATLAS mashinasida amalga oshirilganini bilasizmi?

4. Agar sizda virtual xotira qurilmasi mavjud bo'lsa, bo'limlardan C, D matritsalarini hisoblang. 2, 3 n ning eng katta qiymatlari uchun, n ning bir xil qiymatlari uchun algoritmlarni amalga oshirish vaqtlarini solishtiring, lekin matritsalarning turli xil tuzilmalarini solishtiring. Vaqtning farqini qanday tushuntira olasiz?

3-mavzu: Dasturiy ta’miynot sathi. Mashina kodi, mnemakod va dasturlash tillari

REJA:

3.1. Dasturiy ta’miynot sathi

3.2. Mashina kodi, mnemakod va avtokod

3.3. Quyi va yuqori darajadagi dasturlash tillari

Biz allaqachon kompyuterda faqat kompyuter buyruqlarida yozilgan dasturlarni bajarishga qodir ekanini aytdik. Biroq, mashina buyruqlar dasturiy juda qiyin. Avvalo, buyruqlar tuzilmasi ko'pincha foydalanuvchi o'z muammolarini hal qilish uchun algoritmlarni tasvirlashni rejalashtirayotgan ishlarning tuzilishiga o'xshamaydi. Agar biz mashina buyruqlarida dastur tuzsak, foydalanuvchimiz muayyan buyruqlar ketma-ketligi bilan har bir harakatini modellashtirishga majbur bo'ladi. Ammo dasturlashning bunday usuli shubhasiz afzalliklarga ega: bu sizga eng samarali dasturlarni yaratishga imkon beradi. Buning sababi shundaki, bu holda kompyuter va vazifaning o'ziga xos xususiyatlarini birgalikda hisobga olish mumkin.

Oddiy foydalanuvchi mashinaning buyruqlar dasturiga ega emas. Samarali dasturlarni yaratish kerak bo'lgan mutaxassislar uchun bunday ehtiyoj tug'ilishi mumkin. Masalan, ko'pincha keng maqsadli standart dasturlarning kutubxonalarini ishlab chiquvchilaridan kelib chiqadi. Bunday mutaxassilarning ishini muayyan darajaga ko'tara olish uchun ular avtomatik kod deb nomlangan ramziy kodlashning ma'lum bir tizimida dasturlashishlari kerak. Aslida, bu mashina buyruqlar tizimi, lekin yanada qulay ramziy tuzilishi. Bundan tashqari, avtotovodga mashina yo'riqnomasining oddiy va eng keng tarqalgan kombinatsiyalarini simulyatsiya qiluvchi qo'shimcha ko'rsatmalar kiritilishi mumkin. Albatta, avtomatik kodda yozilgan dasturlarni kompyuter kodiga tarjima qiladigan kompyuterda kompilyator bor.

Autocode oddiy dasturlash tilidir yoki aksincha, eng past darajadagi til deb ataladi. Bu faqat birinchi kompyuterlarda keng dasturlashtirilgan edi. Avtoto'ldagi

ommaviy dasturiy quyidagi sabablarga ko'ra to'xtatildi. Yuqorida ta'kidlanganidek, dasturlash juda ko'p vaqt talab qiladi. Shu sababli, yuqori darajadagi tillar deb ataladigan turli xil til qo'shimchalari otokode ustida yaratilgan. Ushbu tillarning asosiy vazifasi avtomatik kodning tushunilmaydigan yo'riqnomalarini foydalanuvchiga yaqinroq bo'lgan boshqa yo'riqnomalar bilan almashtirish orqali dasturiy jarayonni osonlashtirishdir. Yuqori darajadagi tillarning yuqori qismiga muammoli bo'lgan tillar kiradi. Ularning ko'rsatmalari foydalanuvchilarga ma'lum mavzudagi sohalarda professional tarzda yozishni ta'minlaydi. Har bir kompyuterda autokod odatda bitta. Yuqori darajadagi tillar, xususan, muammoni hal qiluvchi yo'nalishdagi ko'plab tillar mavjud. Ulardan kompilyatorlar kompleks ierarxik dasturiy tizimga aylandi. Dasturlash tillarini soddalashtirishning salbiy tomoni dasturlarni kompyuter kodiga aylantirish uchun algoritmlarning murakkabligi va natijada dasturchi tomonidan yaratilgan dasturlarning samaradorligini nazorat qilish imkoniyatini yo'qotadi. Bu juda muhim ahamiyatga ega va biz katta muammolarni hal qilishga qaratilgan e'tiborni inobatga olgan holda, unga qayta-qayta qaytib boramiz.

Foydalanuvchilarning nuqtai nazari bo'yicha, avtokode bo'yicha dasturni ishlab chiqish qiyin emas, balki u o'ziga xos kompyuterga yo'naltirilgani yoki boshqa turdagi kompyuterlarga qaram bo'lib qolganligi sababli amaliy emas. Kompyuterning turli xil modellarida autokodlar farq qiladi. Dasturni bitta avtomatik koddan boshqasiga tarjima qiladigan universal tarjimon yo'q. Shuning uchun, kompyuterning bir turidan ikkinchisiga o'tish foydalanuvchi avtomatik kodda yozilgan barcha dasturlarni qayta yozishi kerak. Bu jarayon nafaqat super kompleks emas, balki juda qimmat. Natijada yuqori darajadagi dasturlash tillarini ishlab chiquvchilar oldida turgan muhim vazifalardan biri ma'lum kompyuterlarning xususiyatlaridan qat'iy nazar tillarni yaratish edi. Bunday tillar juda ko'p sonda paydo bo'lgan. Bunga Algol, Fortran, C va boshqalar kiradi, ular mashinadan mustaqil, deyiladi, ular kompyuterlarning mustaqilligini ta'kidlaydi. Ularni universal deb ham atashadi, bu esa juda keng algoritmlarni ifodalash qobiliyatini ko'rsatadi.

Mashinadan mustaqil dasturlash tillarini yaratish o'tgan asrning 50-60-yillari davrida o'z go'zalligida ajoyib g'oyalar paydo bo'lishiga olib keldi. Kitoblarda algoritmlarning an'anaviy ta'riflari oldindan izlanmagan holda biron bir kompyuter tiliga o'tkazilishi mumkin emas. Odatda, ular noaniq bo'lib, ko'plab kamchiliklarni o'z ichiga oladi, sharhning noaniqligi va boshqalarni kiritish mumkin. Xususan, assotsiativlikning taxminiy xususiyatlari, komutativlik va raqamlar bo'yicha operatsiyalarning taqsimlanishi tufayli, formuladagi ko'p formulalar yo'q bo'lib, operatsiyalar tartibini belgilaydi. Shuning uchun, aslida, kitobning tavsiflari alohida xususiyatlarning tarqalishi juda katta bo'lishi mumkin bo'lgan turli xil algoritmlarni o'z ichiga oladi.

Dasturda har qanday algoritm har doim aniq ta'riflanadi. Uning rivojlanishiga ko'plab tadqiqotlar olib borilmoqda. Boshqa mutaxassislarni uni takrorlash zarurligini saqlash uchun bu ishni saqlab qolish kerak. Yuqori darajali kompyuterdan mustaqil tillarda dastur saqlash funktsiyalarini bajarish uchun qulaydir. Ideal rivojlangan algoritmlar va dasturlarning bagajlarini bunday tillarda to'plash g'oyasi va har bir kompyuterda dasturlarni ushbu tillardan dasturiy kodga aylantiradigan kompilyatorlar bo'lishi kerak edi. Bunday holda, bir vaqtning o'zida ikkita muhim muammolarni hal qilish mumkin. Birinchidan, dasturlashdagi ortiqcha narsa kamaydi. Ikkinchidan, dasturlarning buzilmasligi, ya'ni bitta binoning kompyuteridan boshqa binoning kompyuteriga o'tkazilishi haqidagi savol avtomatik ravishda hal qilindi. Kompyuterlar juda tez-tez o'z strukturasini sezilarli darajada o'zgartirganligi sababli, juda muhim bo'lgan savol bo'lib qoldi. Ushbu g'oyani amalga oshirishda kompilyatorlar dasturlarni o'ziga xos kompyuterlarning o'ziga xos xususiyatlariga moslashtirish funktsiyalarini o'z zimmalariga oldi. Natijada, foydalanuvchi kompyuterlar va kompilyatorlarning ishlash tamoyillari va tamoyillarini bilib olishdan ozod etildi.

Dastlabki yillarda g'oya juda tez ishlab chiqilgan va amalga oshirilgan. Ko'p turli sohalardagi yaxshi rivojlangan algoritmlar va dasturlar to'plamlari nashr etildi. Ularning keng miqyosda almashinuvi, shu jumladan, xalqaro miqyosda amalga oshirildi. Dasturlar, albatta, bir kompyuterdan boshqasiga osongina ko'chib keldi.

Asta-sekin foydalanuvchilar yuqori darajadagi kompyuterdan mustaqil tillardagi dasturlarni ishlab chiqish bilan cheklangan kompyuterlar va kompilyatorlarning batafsil o'rganishidan uzoqlashdilar. Biroq, yaqinda muhokama qilinayotgan g'oyani hayotga tadbiq etishdan uzoq muddatli ta'sirga ega bo'lgan yorqin umidlar pasayib ketdi.

Buning sababi sodda bo'lib chiqdi va ko'plab kompilyatorlarning ishi noma'lumligi bilan bog'liq edi. Albatta, biz bu noto'g'ri kodlarni yaratganlik haqida gapirmayapmiz, garchi bu ham ro'y beradi. Boshqa tomondan - kompilyatorlarning hech biri natijada olingan kodlarning samaradorligini kafolatlaydi. Kichkina muammolarni hal qilishda samaradorlik kamdan-kam hollarda yuzaga keladi. Bunday vazifalar juda tez amalga oshiriladi, foydalanuvchi ko'p muammolarni sezmaydi. Dasturlarning samaradorligini turli jihatlari, birinchi navbatda, ularni amalga oshirish tezligi va olingan echimlarning aniqligi kompyuterda katta va ayniqsa juda katta vazifalar qo'yilsa juda dolzarb bo'lib qoladi.

Ajoyib holat yuz berdi. Foydalanuvchilar o'zlarining muammolarini kompyuterda hal qilish samaradorligi bilan to'g'ridan-to'g'ri qiziqishmoqda. U samaradorligini oshirish uchun u katta harakatlarni amalga oshiradi: u matematik modellarni o'zgartiradi, yangi soni usullarni ishlab chiqadi, dasturlarni qayta yozadi va hokazo. Lekin foydalanuvchi kompyuterni dasturiy muhiti orqali, birinchi navbatda, kompilyator va operatsion tizim orqali ko'rib chiqadi. Mashina kodlarini yaratish va bajarish ushbu komponentlar foydalanuvchi dasturlarini amalga oshirish samaradorligiga katta ta'sir ko'rsatadi. Biroq, ularning yordami bilan, yaratilgan kompyuter kodlarining to'siqlarini qaerda ekanligini va samaradorligini oshirish uchun yuqori darajadagi tilda ma'lum bir dasturda qanday o'zgarishlar bo'lishi kerakligini tushunish qiyin yoki oddiygina mumkin emas. Dasturdan so'ng kompilyatordan va operatsion tizimdan olinishi mumkin bo'lgan haqiqiy ma'lumot ko'pincha dasturning o'zi va uning amalga oshiradigan algoritmlarini yangilash yo'llarini tanlashda kamdan-kam hollarda qo'llanilishi mumkinligini anglash qiyin va qiyin.

Foydalanuvchilar uchun eng samarali dasturlarni yaratish yo'llarini topish uchun kompyuter bilan aloqa qilish masalasi o'ta dolzarbdir. Ammo uning qarori deyarli butunlay foydalanuvchining elkasiga o'tadi. Kompyuter dasturiy muhiti hech qanday xato tuzatuvchi dasturlarni, dasturlarning tuzilishini tahlil qilish uchun tizimlarni va hokazolarni o'z ichiga olmaydi. Kompilyatorlarning tavsiflaridan hatto eng kerakli yuqori darajali til konstruktsiyasining mashinalar kodida ishlash samaradorligi kabi zarur ma'lumotlarni ham topish qiyin. Algoritm va dasturlarni ishlab chiquvchilarning manfaatlariga etarlicha e'tibor bermaslik uchun tilni yaratuvchilarni, kompilyatorlarni va operatsion tizimlarni tanqid qilishning ko'pgina sabablari mavjud. Bu haqoratda haqiqat donasi bor, chunki foydalanuvchilarning manfaatlari tizim dasturchilarining ko'pgina boshqa qiziqishlari orasida yo'qolib ketgan va dominant bo'lishni to'xtatgan. Shu bilan birga, foydalanuvchilarning muammolari juda qiyin bo'lganini bilish ham kerak.

Hisoblash jarayonlarining eng muhim xususiyatlaridan biri olingan natijalarning aniqligi. Uzoq vaqt davomida ma'lum bo'lganidek, jarayonlarning umumiy tavsifidan tashqari, raqamlarning vakillik shakli va yaxlitlash jarayonini bajarish usuli ta'sir ko'rsatadi. Bu omillar mashinaga bog'liq. Ular haqida ma'lumot oliy darajadagi tillar tomonidan algılanmıyor. Shuning uchun, albatta, bunday tillar, hatto shu sababli ham, mustaqil ravishda mashinani mustaqil deb hisoblanmaydi, chunki turli xil kompyuterlar raqamlarning turli ko'rinishlarini va yaxlitlashni amalga oshirishning turli yo'llarini berishi mumkin. Uzoq muddatda bu natijaga olib kelishi mumkin va amalda ham xuddi shu dasturni amalga oshirish natijalari juda katta o'zgarishlarga olib keladi.

Nazorat savollari

1. Dunyodagi ilk dasturlar shoir J.Bironning qizi, grafin A. Lovelace qizi ekanligi va u S.Babbage mashinasi uchun qilganini bildingizmi?

2. Fortran, birinchi yuqori darajadagi dasturlash tillaridan biri 1957 yilda amalga oshirilgan va hozirgi kunga qadar faol ishlatilganini bilarmidingiz?

3. Fortran tili boshqa dasturiy tillarni ishlab chiquvchilar tomonidan muntazam ravishda qattiq tanqidlarga uchraganligini bilasizmi? Shunga qaramay, ilmiy va

texnik muammolar sohasida Fortran tilida va uning dialektlarida dasturlarning ulushi boshqa barcha dasturiy tillarda birgalikda qabul qilingan dasturlarning ulushidan ancha yuqoridir. Nima deb o'ylaysiz?

4. Quyidagi vazifalar uchun yuqori darajadagi tilda algoritmlarni yozishning turli shakllarining barqarorligini o'rganing:

4-mavzu: Unumdorlikni oshirish: konveyer va superskalyar qayta ishlash, VLIW arxitekturasi. Birinchi superkompyuterlar

REJA:

4.1. Buyruqlarni parallelashtirish
4.2. Konveyer va superskalyar qayta ishlash haqida
4.3. VLIW arxitekturasi
4.4. Dastlabki superkompyuterlar

Bir superkompyuter bilan birinchi marta uchrashadigan odamning hissiy holati bir necha bosqichdan o'tadi. Birinchidan, u kompyuter haqida reklama ma'lumotlarini o'qib chiqib, uning barcha hisob-kitob muammolarini tezda hal qilishni kutib turganidan so'ng, u eforiya kabi narsalarni boshdan kechiradi. Uning dasturini birinchi marta boshlaganidan so'ng, u noto'g'ri narsa qilgani uchun shubha va shubhaga ega, chunki erishilgan natijalar kutilganidan juda farq qiladi. U yana dasturni boshlaydi, lekin natijasi yaxshi bo'lsa, juda zaif bo'ladi. U mahalliy kompyuter gurasiga boradi va bu erda unga haqiqiy zarba berish kutilmoqda. Avvaliga, u eng yuqori kompyuter ishining 5 foizini olgani bunday yomon natija emasligini aytadi. Keyinchalik, agar u bunday hisoblash tizimidan maksimal darajada "siqish" qilmoqchi bo'lsa, unda barcha ishlar unga boshlanadi. Xotirada yuzaga kelishi mumkin bo'lgan ixtiloflarni bartaraf etish va protsessorlarning hisoblash yukini muvozanatlashtirishga harakat qilish kerak. Biz, op¬timalnom ma'lumotlar tarqatish haqida o'ylab serial algoritm parallel¬nye qismini o'rniga

struk¬turoy dasturi bilan tuzuvchi shartnoma yordam kerak. Ko'p hollarda, odatda, avvalgi dastur dasturchisining ishiga xos emas edi. Va ketma-ket kompyuterlar va an'anaviy ketma-ket dasturlash uchun xos bo'lgan barcha oldingi muammolar, albatta, qolaveradi ...

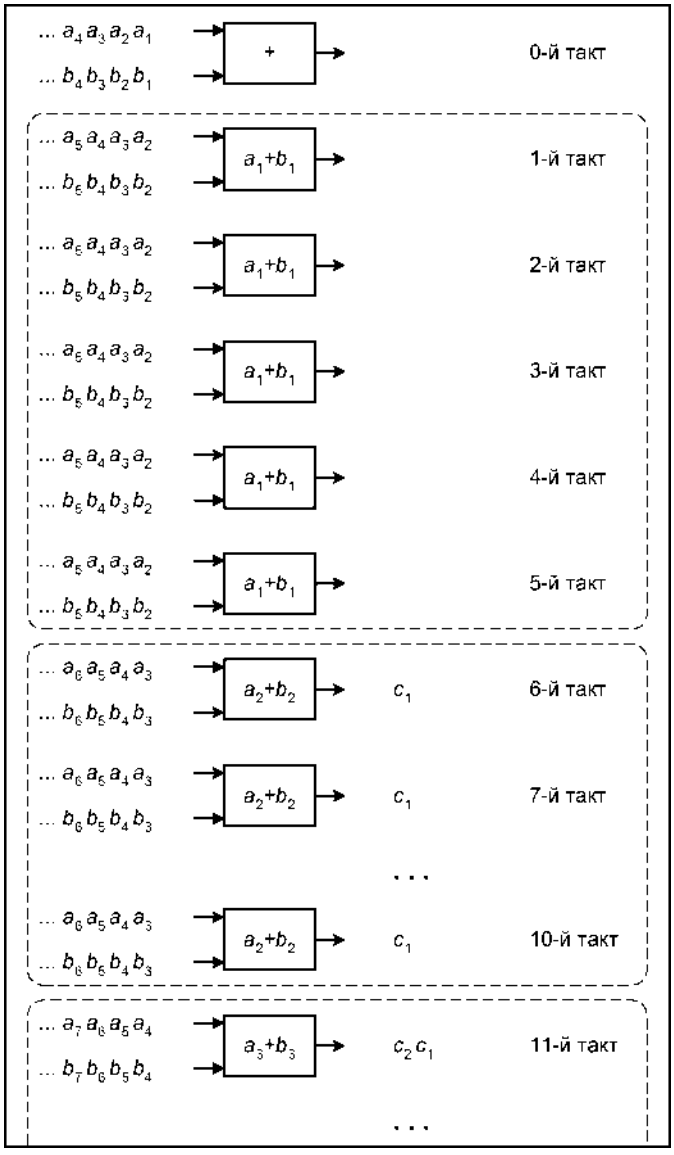
Bu holat uzoq vaqtdan beri kuzatildi va uni hal qilishning turli yo'llarini taklif qildi. Chorshanba kompyuterlar arxitekturasini, ishlab programmirova¬niya o'zgardi, lekin bir narsa sobit qoladi - "bosh og'rig'i" Amaliy pro¬grammistov, jiddiy parallel kompyuter ustida ish boshlashdan qaror qildi. Biz darhol zudlik bilan rezervasyon qilamiz, biz bunday vaziyatda kimni ayblashni o'ylamaymiz. Muhandislar va dasturchilar juda katta ishlarni amalga oshirganlar. Hudud o'z-o'zidan qiyin, va hayot o'z-o'zidan tuzatishlar beradi. Bitta mikro protsessorda yaxshi ishlaydigan kesh xotirasi yaratilganda, hech kim, ko'p protsessorli tizimlarda qanday muammolar yuzaga kelishidan shubha qilmadi. Faqat tillar C va C ++, parallel dastur tuzilishi statik tahlil privnesshie znachi¬telnye qiyinchiliklar bor edi, deb Fortran dasturlarida manba kodi tuzilishini tahlil qilish o'rgandim. Deyarli ni¬kak programcısı tegmasligini faqat g'alaba uzluksiz rivojlanishi, soat chastotasini oshirish: tezroq pro¬gramma, va u qo'shimcha qiladi, bu hech narsa emas.

Har birimiz 100 raqamdan iborat ikkita vektorning yig'indisini topamiz. Bizning ixtiyorimizda kompyuterning beshta tsiklida juftlik sonini yig'ishni amalga oshiradigan qurilma mavjud. Qurilma ushbu operatsiyani bajarish muddati mobaynida bloklanadi va boshqa foydali ishlarni bajarolmaydigan holga keltiriladi. Bunday sharoitda barcha operatsiya 500 tsiklda bajariladi. Vaqtda qayta ishlash jarayonining rivojlanishi shakl. 2.2.

Keling, biz bir vaqtning o'zida va bir-biridan mustaqil ishlashi mumkin bo'lgan ikkita qurilmaga egamiz. Oddiylik uchun kirish ma'lumotlarini qabul qiluvchi va natijalarni saqlash qurilmalari bilan bog'liq qo'shimcha xarajatlar yo'q bo'lganda ideal vaziyatni ko'rib chiqamiz. Har bir uskuna uzatuvchi elementlarning elementlari bilan doimiy ravishda o'rnatilganda siz kerakli miqdori 250 tsikldan (2.3-rasm) olishingiz mumkin - bajarilgan ishni ikki marta tezlashtirasiz. Bunday 10 ta qurilma

uchun natija olish muddati atigi 50 tsikldan iborat bo'ladi va umumiy holda N qurilmalari tizimi 500 / 7V summa summaga sarflanadi.

Parallel ishlov berishning ko'plab misollari kundalik hayotimizda mavjud: supermarketda ko'plab kassa apparatlari, ko'plab avtoyo'llar, ekskavatorlar ekipaji, benzin stantsiyalarida bir necha benzin ustunlari va boshqalar. Ayni paytda, kashshoflar parallel ravishda 1950-yillarning boshlarida yadroviy portlashlarni modellashtirish uchun zarur bo'lgan hisob-kitoblarni amalga oshirgan akademik A. Samarskiy tomonidan qayta ishlangan. Aleksandr Andreyevich bu muammolarni o'nlab yosh ayollarni stollarda mashinalar qo'shish bilan ochib berdi. Yosh ayollar oddiygina so'zlarni bir-biriga uzatdilar va kalkulyatorlarga kerakli raqamlarni ajratishdi.



2.2-rasm. V = A + B vektorlarining summasi beshta davrda bitta operatsiyani amalga oshiradigan ketma-ket qurilmadan foydalaniladi

Har bir raqam juftiga qo'shilishi buyurtmalar bilan taqqoslash, tartiblash buyurtmalarini solishtirish, mantissalarni qo'shish, normalizatsiya qilish va h.k. kabi mikroblar ketma-ketligi sifatida amalga oshiriladi. Shunisi diqqatga sazovordirki: har bir juft ma'lumotni qayta ishlash jarayonida mikroorganizmlar faqat bir marta ishlatiladi va har doim bir xil tartibda: birma-bir. Bu, birinchi navbatda, agar birinchi mikro-operatsiya o'z ishini bajargan va natijalarni ikkinchi darajaga o'tkazgan bo'lsa, u holda joriy juftni qayta ishlashga zarurat bo'lmaydi va bu qurilmani kutish uchun navbatdagi dalillar juftini ishlashni boshlashi mumkin deganidir.

Ushbu dalillarga asoslanib biz qurilmani quyidagicha quramiz: Har bir mikro-operatsiya qurilmaning alohida qismiga bo'linadi va ularni bajarish tartibiga qo'yadi. Vaqtning dastlabki vaqtida kirish ma'lumotlari dastlabki qismga ishlov berish uchun qabul qilinadi. Birinchi mikromoliyalash tugallangandan so'ng, birinchi qism o'z ishining natijalarini ikkinchi qismga o'tkazadi, o'zi esa yangi juftlikni oladi. Kirish argumentlari barcha jurnali bosqichlarida o'tganda, operatsiya chiqishi qurilmaning chiqishida paydo bo'ladi.

Hisob-kitoblarni tashkil etishning bu usuli "quvurlash" deb ataladi. Qurilmaning har bir qismi konveyer bosqichi deb ataladi va qadamlarning umumiy soni konveyer uzunligi deb ataladi. Tasavvur qilaylik, har bir bosqichda tetiklenen besh bosqichdan iborat bo'lgan konveyerning haqiqiy sonlarini ishlab chiqarishni amalga oshirish. Bir konveyer apparati tomonidan bitta operatsiyani bajarish muddati konveyerning barcha qadamlarining javob vaqtlari yig'indisiga teng. Buning ma'nosi shuki, ikkita sonni qo'shib qo'yish jarayoni beshta davrda amalga oshiriladi, ya'ni avvalgi holatda ketma-ket qurilmada bir xil operatsiya qilingan vaqtga teng.

Bugungi kunda kompyuterlar arxitekturasidagi o'zaro kelishuv juda kam odamni hayron qoldirdi. Barcha zamonaviy mikroprotsessorlar, ALPHA 21264, Itanium, RA-8700 yoki Power 4 bo'lsin, bitta kristalning miniatyura ramkalarida

qo'llaniladigan bir yoki boshqa turdagi parallel jarayonlardan foydalaning. Shu bilan birga, bu g'oyalar uzoq vaqt oldin paydo bo'lgan. Dastlab, ular eng ilg'or va shuning uchun yagona, o'z vaqtidagi kompyuterlarida tanishtirildi. Keyinchalik texnologiyani va arzonroq ishlab chiqarishni o'zlashtirishdan so'ng, ular o'rta sinf kompyuterlarda qo'llanila boshlandi va nihoyat, bugungi kunda bularning barchasi ish stantsiyalarida va shaxsiy kompyuterlarda amalga oshdi.

Zamonaviy hisoblash tizimlarining arxitekturasidagi barcha yirik yangiliklarning na mikroprotsessorlar, na superkompyuterlar konsepsiyasi mavjud bo'lmagan kunlardan buyon foydalanilganligiga ishonch hosil qilish uchun, birinchi kompyuterlarning tug'ilish paytidan boshlab, tarixni bir oz ekskursiya bilan o'taylik.

Birinchi kompyuterlar (EDSAC, EDVAC, UNIVAC, XX asr boshining 50-yillari) saqlangan dastur printsipini amalga oshirish uchun simob kechiktiruvchi liniyalaridan foydalanganlar. Xotirani tashkil qilishning bu usuli bilan, keyingi so'zlashuvlarni keyingi ketma-ketlikda ketma-ketlikda bajarish uchun so'zlar olinadi. Aritmetik operatsiyalar ham bittagina usulda amalga oshirilgani tabiiy, shuning uchun 32 ta bittadan 32 bitli raqamni qo'shib qo'yish 32 ta mashina davrlarini o'z ichiga olgan. Hisoblash jarayonida bit-ketma-ketlikdagi xotira va bit-ketma-ket arifmetikaning vaqti.

Tasodifiy ma'lumotlarni o'z ichiga olgan xotira qurilmalari ixtirosi bilan ikkala bit parallel xotira va bit parallel arifmetik ham amalga oshirildi. Barcha so'zlar bir vaqtning o'zida xotiradan o'qiladi va arifmetik mantiqiy qurilma bilan operatsiyani bajarishda ishtirok etadi. 1953 yilda IBM 701 mashinasi paydo bo'ldi - bu printsipga asoslangan birinchi savdo kompyuter. Biroq, ushbu sinfning eng muvaffaqiyatli mashinasi 1955-yilda chiqarilgan IBM 704 kompyuteri bo'lib, unda ferrit yadrolari va suzuvchi nuqtali apparatlardan foydalanilgan. Ushbu vaqtlar uchun ajoyib, IBM 704 savdo muvaffaqiyatlari ushbu mashinaning 150 ta nusxasini sotish bilan aniqlandi.

Har ikkisi ham IBM 704 mashinasida va o'sha davrdagi boshqa kompyuterlarda barcha G / Ç operatsiyalari arifmetik mantiqiy birlik orqali amalga oshirildi. Bir nechta tashqi asboblar bor edi, lekin ularning eng tezkori, tarmoqli qurilmasi

soniyada 15000 belgigacha tezlikda ishladi va bu protsessor tomonidan ma'lumotlarni qayta ishlash tezligidan bir necha marta kamroq edi. Kompyuterlarning bunday tashkiloti axborot olish va chiqish vaqtida samaradorlikning sezilarli pasayishiga olib keldi. Ushbu muammoning dastlabki echimlaridan biri, I / U qurilmalari bilan parallel ravishda ishlashga ruxsat beruvchi I / U deb nomlangan maxsus kompyuterni joriy etish edi. 1958 yilda IBM 709 kompyuterini yaratish uchun asos bo'lgan IBM 704 mashinasiga 6 ta I / O kanallari qo'shildi.

VLIW-protsessorlari (juda katta buyruqli so'z) deyarli Fon-Neymann kompyuteri qoidalariga muvofiq ishlaydi. Bitta farq, protsessorga har bir tsikldagi berilgan buyruqlar bir operatsiyani emas, balki bir vaqtning o'zida bir necha aniqlaydi. VLIW protsessori buyrug'i, ularning har biri o'z vazifalari uchun mas'ul bo'lgan, masalan, funktsional qurilmalarni faollashtirish, xotira bilan ishlaydigan, ro'yxatga olish operatsiyalari va boshqalarni o'z ichiga olgan bir qator maydonlardan iborat. Bu bosqichda protsessorning biron bir qismi mavjud bo'lsa gramm talab qilinmaydi, tegishli buyruqlar maydoni faollashtirilmaydi.

Shunga o'xshash me'morchiligi bo'lgan kompyuter misoli Floating Point Systems-dan olingan AP-120B kompyuteridir. Birinchi etkazib berish 1976 yilda boshlangan va 1980 yilga kelib butun dunyo bo'ylab 1600 dan ortiq nusxa o'rnatilgan. AP-120V kompyuter buyrug'i 64 bitdan iborat va mashinaning barcha qurilmalari ishlashini nazorat qiladi. Har bir tsikl (167 ta emas) bitta buyruq beriladi, bu soniyada 6 mln. Har bir jamoa bir vaqtning o'zida ko'plab operatsiyalarni nazorat qilganligi sababli, amalda ishlash yanada yuqori bo'lishi mumkin. AR-120B guruhining barcha 64 bitlari o'zlarining operatsion to'plamlari uchun oltita guruhga bo'linadi: 16 bitli ma'lumotlar va registrlar bo'yicha operatsiyalar, haqiqiy raqamlarni qo'shish, kirish / chiqish nazorati, o'tish buyruqlar, haqiqiy raqamlar va ishchi komandalarni ko'paytirish Asosiy xotira bilan.

VLIW protsessor dasturi har doim parallelizm haqida aniq ma'lumotlarni o'z ichiga oladi. Bu erda derleyici dasturda parallelizmni aniqlaydi va operatsiyalar bir-biridan mustaqil bo'lgan apparatni ochiq-oshkor qiladi. VLIW protsessorlarining

kodlari jarayonning qanday bajarilishini aniq rejasini o'z ichiga oladi: har bir operatsiya bajarilganda, qaysi funktsional qurilmalar ishlay olishi, qaysi operatorlar bo'lishi kerakligini qayd qiladi va hokazo. VLIW kompilyatori bunday dasturni to'liq tushunishga ega bo'ladi. Umuman aytganda, superscalar mashinalari uchun kompilyatorlardan foydalanish mumkin bo'lmagan maqsadli VLIW protsessori.

Ikkala yondashuv ham o'z afzalliklari va kamchiliklariga ega va VLIW arxitekturasining soddaligi va cheklangan imkoniyatlarini supersqalar tizimlarining murakkabligi va dinamik qobiliyatlariga qarshi turish kerak emas. Kompilyatsiya vaqtida operatsiyalarni bajarish uchun reja tuzish, yuqori ral tizimlariga nisbatan yuqori darajadagi parallellashni ta'minlash uchun muhimdir. Derleme vaqtida, faqat dasturni bajarish vaqtida supersqalar me'morchiligiga xos bo'lgan dinamik mexanizmlar yordamida echilishi mumkin bo'lgan noaniqlik mavjudligi aniq.

Nazorat savollari

1. Buyruqlarni parallelashtirish haqida tushunchalar
2. Prosessorlar tuzilishi
3. Komveyer qayta ishlash
4. Superskalyar qayta ishlash
5. VLIW arxitekturasi

5-mavzu: Parallel kompyuterlar: umumiy va ajratilgan xotirali multiprosessorlar va multikompyuterlar

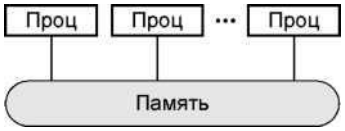
REJA:

5.1. Umumiy xotirali tizimlar
5.2. Ajratilgan xotirali tizimlar
5.3. Multikompyuter va multiprosessorlar

Kompyuter industriyasi mavjud bo'lganda parallel kompyuterlar arxitekturasi juda tez sur'atda va turli yo'nalishlarda ishlab chiqildi. Buni tekshirish uchun hech qanday qiyinchilik yo'q, masalan, mukammal bir umumiy qarashga e'tibor qaratish

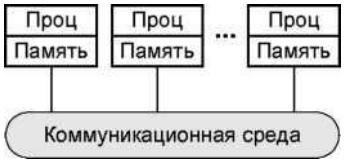
etarli. Biroq, zamonaviy parallel hisoblash tizimlarining aksariyat qismini yaratish g'oyalarini belgilash va umumiy fikrni ta'kidlasangiz, faqatgina ikkita sinf qoladi.

Birinchi sinf - xotira kompyuterlari bilan birgalikda. Ushbu tamoyilga asoslangan tizimlar ba'zan ko'p protsessor tizimlari yoki oddiygina ko'p protsessorlar deb ataladi. Tizimda bir xotiraga bir xil kirish imkoniyatiga ega bo'lgan bir necha peer protsessorlari mavjud (2.8-rasm). Barcha protsessorlar umumiy xotirani o'zlarida birlashtiradi, shuning uchun bu sinf kompyuterlari uchun boshqa xotira - birgalikda xotirasi bo'lgan kompyuterlar. Barcha protsessorlar yagona manzillar maydoni bilan ishlaydi: agar bitta protsessor qiymati 1024 da so'zni 79 ga yozsa, boshqa protsessor 1024 da joylashgan so'zni o'qiydi, 79 qiymatini oladi.



2.8-rasm. Umumiy xotiraga ega parallel kompyuterlar

Ikkinchi sinf - ajiratilgan xotirali kompyuterlar bo'lib, ular oldingi sinfga o'xshash tarzda ba'zan ko'p-komputerli tizimlar deb ataladi (2.9-rasm). Aslida, har bir hisoblash tugunlari o'zlarining protsessorlari, xotirasi, G / Ç quyi tizimi va operatsion tizimiga ega to'liq huquqli kompyuter. Bunday vaziyatda, agar bitta protsessor 1024 da 79 qiymatini yozsa, u har bir kishi o'z manzilgohida ishlayotgani uchun, u boshqa odamning o'sha manzilda o'qigan ma'lumotlariga ta'sir qilmaydi.



2.9-rasm. Ajiratilgan xotiraga ega parallel kompyuterlar

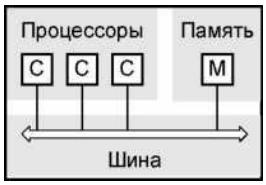
Birgalikda ishlatiladigan xotira kompyuterlari barcha Simmetrik Multi Processors (SMP) sinf tizimlarini o'z ichiga oladi. SMP-da bir nechta protsessorlar bitta nusxada: bitta xotira, bitta operatsion tizim, bitta G / Ç quyi tizimi. Arxitektura nomidagi «nosimmetrik» so'zi har bir protsessor boshqa narsalarni bajarishi mumkinligini anglatadi. Aytgancha, SMP ko'pincha umumiy xotiraga ega bo'lgan kompyuterlar uchun muqobil nom sifatida qaraladi, bu esa bu qisqartani tushunish uchun ikkita mumkin bo'lgan variantlar: simmetrik ko'p ishlaydigan va umumiy xotira protsessorlari.

Ushbu ikkita darslik, birgalikda va tarqatilgan xotirasi bo'lgan kompyuterlar tasodifan ko'rinmadi. Ular parallel hisoblashning ikki asosiy vazifasini aks ettiradi. Birinchi vazifa maksimal ishlashi bilan hisoblash tizimlarini qurishdir. Bu esa, tarqalgan xotirasi bo'lgan kompyuterlarni osonlashtirishga imkon beradi. Bugungi kunda yagona aloqa muhitida minglab hisoblash nodlarini birlashtiradigan qurilmalar mavjud. Lekin, nima demoq-da, Internetni tarqatilgan xotirasi bo'lgan millionlab hisoblash nodlarini birlashtirgan eng katta parallel kompyuter hisoblanishi mumkin. Ammo bunday tizimlarni qanday qilib samarali ishlatish kerak? Parallel protsessorlarning o'zaro ta'siriga tushadigan katta xarajatlar qanday ketadi? Parallel dasturlarni ishlab chiqishni qanday soddalashtirish kerak? Bunday tizimlarni amalda qo'llashning yagona usuli - har doim ham oson bo'lmagan xabarlar tizimini ishlatish, masalan, PVM yoki MPI. Bu erda ikkinchi vazifa - parallel hisoblash tizimlarining samarali dasturiy ta'minotini ishlab chiqish usullarini izlash. Ushbu vazifa birgalikda xotirasi bo'lgan kompyuterlar uchun biroz osonroqdir. Protsessorlar o'rtasida umumiy xotira orqali ma'lumotlar almashinuvi minimal va bunday tizimlarning dasturlash metodlari odatda soddalashtiriladi. Muammo bu erda boshqacha. Texnologik sabablarga ko'ra, juda ko'p sonli protsessorlarni bitta RAM bilan birlashtirish mumkin emas, shuning uchun bugungi kunda bunday tizimlarda juda yuqori ishlashga erishish mumkin emas.

Shunga e'tibor beringki, har ikki holatda ham protsessorlarni xotira modullari yoki protsessorlar bilan o'zaro bog'laydigan almashtirish tizimi juda ko'p muammolarga olib keladi. 32 ta protsessor bir RAMga teng kirish huquqiga ega,

yoki 1024 protsessor ularning har biri bilan bog'lanishi mumkinligini va buning amalda qanday amalga oshirilishini aytish osonmi? Kompyuterlarda aloqa tizimlarini tashkil qilishning ba'zi usullarini ko'rib chiqing.

Keyinchalik kuchli tizimlarni yaratish uchun boshqa yondashuvlarga ehtiyoj bor. Ulardan biri xotiraning mustaqil modullarga bo'linishi va turli protsessorlarni bir vaqtning o'zida turli modullarga kirish imkoniyati. Ko'pgina mumkin echimlar, xususan, matritsali kalitlarni ishlatish mumkin. Protsessorlar va xotira modullari shakl. 2.11. Yo'nalishlarni kesishda, elementar nuqta kalitlari joylashgan bo'lib, ular protsessorlar va xotira modullari o'rtasida ma'lumotlarni uzatish imkonini beradi yoki taqiqlaydi. Bunday tashkilotning shubhali afzalligi - turli xotira modullariga ega bo'lgan protsessorlarning bir vaqtda ishlashi. Tabiiyki, ikkita protsessor bitta xotira moduli bilan ishlashni xohlaydigan bo'lsa, ulardan biri bloklanadi. Matritsali kalitlarning ahvoliga katta hajmdagi zarur uskunalar kiradi, chunki n 2-tugmachalari n protsessorlarni n xotira modullariga ulash uchun kerak. Ko'pgina hollarda, bu juda qimmat qaror bo'lib, ishlab chiquvchilarni boshqa yo'llar izlashga majbur qiladi.



2.10-rasm. Umumiy avtobusga ega ko'p protsessor tizimi

Nazorat savollari

1. Ajratilgan xotirali xisoblash tizimlarini ishlash tamoyillari
2. Umumiy xotirali xisoblash tizimlarini ishlash tamoyillari
3. Dastlabki multiprosessor va multikompyuterlar

6-mavzu: Xotiraga bir xil bo’lmagan ruxsatlilik texnologiyasi, NUMA va ccNUMA arxitekturasi, kesh mustahkamligi

REJA:

6.1. Umumiy va ajratilgan xotirali xisoblash tizimlarini birlashtirish

6.2. NUMA va ccNUMA arxitekturasi

6.3. Kesh mustahkamligi

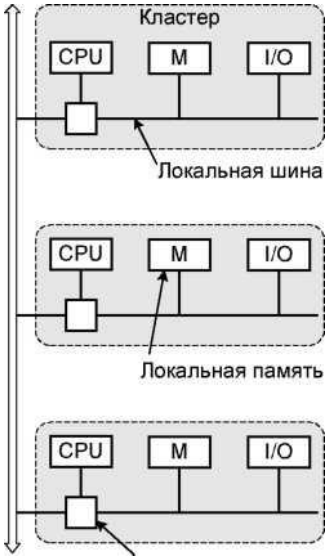
Keling, birgalikda va tarqatilgan xotirasi bo'lgan kompyuterlarning xususiyatlarini muhokama qilamiz. Yuqorida aytib o'tganimizdek, har ikkala sinf ham o'zlarining afzalliklariga ega, biroq ularning zaifliklarini darhol o'zlarining kamchiliklariga aylantiradi. Parallel dasturlarni birgalikda ishlatiladigan xotira kompyuterlari uchun yaratish oson, biroq ularning maksimal ishlashi juda kam sonli protsessorlar tomonidan cheklangan. Va tarqatilgan xotira bilan jihozlangan kompyuterlar uchun buning aksi rost. Bu ikki sinfning mahsullarini birlashtirish mumkinmi? Mumkin bo'lgan yo'nalish - NUMA (Yagona xotiraga ega bo'lmagan) arxitekturasi bilan jihozlangan kompyuterlarning dizayni.

Umumiy xotirasi bo'lgan kompyuterlar uchun parallel dasturlarni yozishni nima uchun osonlashtiradi? Bitta manzillar maydoni bo'lgani uchun va foydalanuvchining ma'lumotlar almashinuviga ma'lumotlar almashinishini tashkil qilishning hojati yo'q. Agar foydalanuvchi dasturlarining kompyuterning barcha kommuttal jismoniy xotirasini bitta adreslangan xotira sifatida ko'rish imkonini beradigan mexanizmni yaratadigan bo'lsangiz, hamma narsa juda oson bo'ladi.

1970-yillarning oxirlarida birinchi NUMA kompyuterini yaratgan Cm * tizimini ishlab chiquvchilari bu yo'ldan ketishdi. Ushbu kompyuter interkuster avtobusi orqali bir-biriga ulangan klasterlardan iborat. Har bir klaster bir protsessorni, xotira tekshirgichni, xotira modulini va, ehtimol, mahalliy avtobus orqali bir-biriga ulangan ba'zi I / U asboblarini birlashtiradi (2.15-rasm). Protsessor o'qish yoki yozish operatsiyalarini bajarishi kerak bo'lsa, u to'g'ri manzilni o'z xotirasi tekshirgichiga yuboradi. Nazoratchi manzilning yuqori darajadagi bitlarini

tahlil qiladi, unda qaysi modul kerakli ma'lumotlarning saqlanishini aniqlaydi. Agar manzil mahalliy bo'lsa, talab mahalliy avtobusda joylashtiriladi, aks holda uzoqdan klasterning so'rovi interlustust avtobus orqali yuboriladi. Ushbu rejimda, bitta xotira modulida saqlangan dasturni tizimning har qanday protsessori bajarishi mumkin. Bitta farq - bu bajarilish tezligi. Barcha mahalliy havolalar masofaviy havolalardan ko'ra tezroq qayta ishlanadi. Shuning uchun, dastur saqlanadigan guruhning protsessori uni boshqa har qanday o'lchamdan tezroq tartibga soladi.

Ushbu funksiyadan kompyuter sinfining nomi - yagona xotiraga ega bo'lmagan kompyuterlar keladi. Shu ma'noda, klassik SMP kompyuterlarning har qanday protsessorning har qanday xotira moduliga bir xil kirishini ta'minlaydigan UMA (Uniform Memory Access) me'morchiligi mavjud.



2.15-rasm. Cm* kompyuter tizimining sxemasi

NUMA kompyuterining yana bir misoli, maksimal konfiguratsiyasida 256 protsessorni birlashtiradigan BBN Butterfly kompyuteri edi (2.16-rasm). Har bir kompyuter hisoblash tugunida protsessor, mahalliy xotira va xotira tekshiruvi mavjud bo'lib, xotira so'rovining mahalliy yoki yo'qligini yoki Butterfly kalitidan uzoq tugunga uzatilishi kerakligini aniqlaydi. Dasturchi nuqtai nazaridan, xotira - yagona umumiy xotira, masofaviy ulanish mahalliy dasturlardan bir oz ko'proq vaqtni tashkil qiladi (masofadan boshqariladiganlar uchun taxminan 6 mil, mahalliy bo'lganlar uchun 2 millimetrgacha).

Katta NUMA kompyuteri qurish yo'lida, agar kutilmagan bir muammo bo'lmasa - shaxsiy protsessorlarning kesh xotirasi bo'lmasa, xavfsiz tarzda davom etishi mumkin. Ko'p protsessorli tizimlar uchun individual protsessorlarning ishini sezilarli darajada tezlashishiga yordam beradigan kesh xotira buzilishdir. Birinchi NUMA kompyuterlarning protsessorlarida kesh yo'q edi, shuning uchun bunday muammo yo'q edi. Ammo zamonaviy mikroprotsessorlar uchun kesh ajralmas qismdir. Bizning tashvishlarimiz sababini tushuntirish juda oson. Protsessor P \ hujayra q ning .v qiymatini saqlab qo'ygan deb taxmin qilamiz va keyin P \ protsessori bir xil hujayraning tarkibini o'qishni istaydi. Protsessor nimani oladi P (1) Albatta, har kim x ning qiymatini olishni xohlaydi, lekin x protsessorning keshiga kirsa, uni qanday qilib oladi? P (1) Bu muammo kesh xotirasining mazmunini , muvofiqlik keshlari muammosi). Ushbu muammo zamonaviy SMP kompyuteri uchun ham dolzarb bo'lib, ularning protsessorlari keshlari ma'lumotlardan foydalanishda nomuvofiqlikka olib kelishi mumkin.

Ushbu muammoni bartaraf qilish uchun NUMA arxitekturasining maxsus modifikatsiyasi - ccNUMA (kesh mos keluvchi NUMA) ishlab chiqildi. Endi u barcha keshlarni tarkibi bilan ta'minlashni ta'minlaydigan protokol majmui texnik tafsilotlariga kiradi. Ushbu muammoni hal qilish va foydalanuvchilarning qo'llariga tushmasligi muhimdir. Foydalanuvchilar uchun yana bir savol muhimroq: NUMA me'morchiligi qanday "heterojen"? Agar boshqa tugunning xotirasiga kirish xotiraga kirmasdan 5-10 foiz ko'proq vaqt talab etsa, bu hech qanday savol tug'dirishi mumkin emas. Foydalanuvchilarning aksariyati UMA (SMP) kabi tizim bilan bog'liq va SMP uchun ishlab chiqilgan deyarli barcha dasturlar juda yaxshi ishlaydi. Biroq, bu zamonaviy NUMA tizimlarida mavjud emas va mahalliy va masofaviy kirish vaqtidagi farq 200-700 oralig'ida. Foydalanishda bo'lgan tezlikda bunday farq bilan, dasturlarning to'g'ri ishlashini ta'minlash uchun kerakli ma'lumotlarni to'g'ri joylashishini ta'minlash uchun ehtiyot bo'lish kerak.

CcNUMA arxitekturasiga asoslanib, an'anaviy kompyuterlarning imkoniyatlarini umumiy xotiraga ega kengaytiradigan ko'plab real tizimlar mavjud.

Ko'p hollarda amaliyot dasturchilari hech qachon aniq parallel tuzilmalarni ishlatishmaydi, vaqtni tanqidiy hisoblash qismlarida parallel ob'ekt kutubxonalarining subrutinlari va funktsiyalariga ishora qiladilar. Barcha parallellik va barcha optimallashtirish chaqiruvlarda yashiringan va foydalanuvchi faqat uning dasturining tashqi qismini yozishi va standart bloklarni to'g'ri ishlatishi mumkin. Bunday kutubxonalarga Lapack, ScaLapack, Cray Scientific Library, HP Matematik kutubxonasi, PETSc va boshqalar kiradi.

Va nihoyat, eslatib o'tish kerak bo'lgan oxirgi yo'nalish - bu maxsus paketlar va dasturiy ta'minot paketlaridan foydalanish. Odatda, bu holatda, foydalanuvchi dasturiy ta'minotga ega emas. Asosiy vazifa - barcha kerakli kirish ma'lumotlarini to'g'ri belgilash va paketning funksiyasidan to'g'ri foydalanish. Shunday qilib, ko'plab kimyogarlar GAMESS paketini parallel kompyuterlar bo'yicha kvant kimyoviy hisob-kitoblarini bajarish uchun ishlatadilar, bu ma'lumotlarning parallel ishlashi paketning o'zi qanday amalga oshirilishini o'ylamaydi.

Nazorat savollari

1. Ikkita kompyuter tizimlari faqat o'rnatilgan operatsion tizimlar versiyalarida farqlanadi. Ushbu kompyuterlar bir xil dastur uchun turli xil ish vaqtlariga ega bo'ladimi?

2. Bunday vaziyat shunday bo'lishi mumkinmi: 10 000 satr manba matnli dasturda faqat bir belgi o'zgartirildi va uning ishlash vaqti 10 barobar ko'payganmi?

3. Bunday vaziyat shunday bo'lishi mumkinmi: 10 000 satr manba matnli dasturda faqat bir belgi o'zgartirildi, undan keyin kompyuterning ishlashi 10 marta oshdi?

4. Maxsus protsessor sinxronli bir bitli protsessorlarning matritsasi deb taxmin qiling. Bunday maxsus protsessorlarda qanday muammolar samarali echilishi mumkin?

7. Ma'lumki, dastur o'z-o'zidan ishlab funktsional qurilmalar to'plamiga ega bo'lgan superssalar protsessorida yaxshi ishlaydi. Bu shuni anglatadiki, ayni dastur VLIW protsessorida bir xil qurilmalar bilan ishlaydi. Aksincha haqiqatmi?

7-mavzu: Parallel kompyuterlarning dasturiy ta’miynoti, parallel dasturlash, tillarning kengayishi, maxsus tillar, kutubxonalar va interfeyslar takomillashuvi

REJA:

7.1. Parallel kompyuterlarning dasturiy ta’miynoti

7.2. Parallel dasturlash tillari va texnologiyalari

7.3. Maxsus tillar va kutubxonalar

An'anaviy dasturlash tillari va maxsus izohlardan foydalanish asosida umumiy xotira kompyuterlari uchun eng mashhur dasturiy vositalaridan biri hozirda OpenMP texnologiyasidir. Vaqtinchalik dastur asos sifatida olinadi va parallel versiyasini yaratadi, foydalanuvchi bir qator dir direktivalari, funktsiyalari va atrof-muhit o'zgaruvchilari bilan ta'minlanadi. Yaratilgan parallel dastur OpenMP API-ni qo'llab-quvvatlaydigan turli xil umumiy xotira kompyuterlari o'rtasida ko'chma bo'ladi deb taxmin qilinadi.

OpenMP texnologiyasi foydalanuvchi dasturning parallel va ketma-ket ijro etilishi uchun dasturning bitta versiyasiga ega bo'lishini ta'minlashni maqsad qiladi. Biroq, faqat parallel rejimda to'g'ri ishlashi yoki ketma-ket rejimda boshqa natija beradigan dasturlarni yaratish mumkin. Bundan tashqari, yaxlitlash xatolarining to'planishi tufayli turli xil ish zarrachalaridan foydalangan holda hisoblash natijalari ayrim hollarda farq qilishi mumkin.

Ushbu standart OpenMP ARB notijorat tashkiloti tomonidan ishlab chiqilgan (Arxitektura tadqiqoti kengashi) [1], bu SMP-arxitektura va dasturiy ta'minotni ishlab chiqaruvchi yirik kompaniyalarning vakillari hisoblanadi. OpenMP FORTRAN va C / C ++ tillari bilan ishlashni qo'llab-quvvatlaydi. Fortran tilining birinchi spetsifikatsiyasi oktyabr 1997-yilda va C / C ++ tilining 1998 yil oktyabrida paydo bo'lishi. Hozirgi vaqtda eng so'nggi rasmiy standart spetsifikatsiya OpenMP 3.0 [3] (2008 yil may oyida qabul qilingan).

OpenMP interfeysi umumiy xotira modelida ölçeklenebilir SMP tizimlarida (SSMP, ccNUMA va boshqalar) dasturiy uchun standart sifatida mo'ljallangan. OpenMP standarti kompilyator direktiflari, yordamchi funktsiyalar va atrof-muhit o'zgaruvchilari uchun spetsifikatsiyalarni o'z ichiga oladi. OpenMP "master" (master) ish zarrachalar majmuasini "qul" (thread) toifalarini hosil qiladigan va ular orasidagi vazifa taqsimlangan multithreading yordamida parallel hisoblashni amalga oshiradi. Bir vaqtning o'zida bir nechta protsessorli mashinada parallel ishlaydi va protsessorlarning soni ish zarrachalar soniga teng yoki teng bo'lmasligi kerak.

POSIX interfeysi (Pthreads) deyarli barcha UNIX tizimlarida qo'llab-quvvatlanadi, biroq ko'plab sabablarga ko'ra amaliy parallel dasturlash uchun mos emas: Fortranni qo'llab-quvvatlamaydi, dasturlash darajasi juda past, parallelizmga mos kelmaydi. va iplar mexanizmi aslida parallelizmni tashkil qilish uchun mo'ljallangan emas edi. OpenMP'ni Pthreads (yoki shunga o'xshash yozuvlar kutubxonalari) orqali yuqori darajadagi qo'shimcha sifatida o'ylash mumkin; OpenMP terminologiyani va Pthreads-ga yaqin bo'lgan dasturiy modelini, masalan, dinamik ravishda yaratilgan ish zarrachalarini, birgalikda va birgalikdagi ma'lumotlarni va sinxronlashtirish uchun "qulflarning" mexanizmidan foydalanadi.

OpenMP texnologiyasining muhim ustunligi dasturiy vosita dasturda parallellik manbasini o'z ichiga olgan bo'limlarni asta-sekinlik bilan topib, taqdim etilgan mexanizmlardan foydalangan holda ketma-ket dasturlashni amalga oshirish imkoniyati bo'lib, ularni parallel qiladi va keyinchalik quyidagi bo'limlarni tahlil qilishga o'tadi. Shunday qilib, dasturda parallel bo'lak asta-sekin kamayadi. Ushbu yondashuv ketma-ketlikdagi dasturlarni parallel kompyuterlarga moslashtirish jarayonini, shuningdek disk raskadrovka va optimallashni osonlashtiradi.

Ushbu qo'llanmada OpenMP funksiyasining tavsifi ko'plab misollar bilan ta'minlangan. Barcha misollar M.V. Moskva Davlat Universitetining Tadqiqot Kompyuterlari Markazining Parallel Axborot Texnologiyalari Laboratoriyasi xodimlari tomonidan sinovdan o'tgan. Lomonosov "SKIF" SKU MSU "CHEBYSHEV" da Intel Fortran / C ++ 11.0 kompyuteri yordamida ishlaydi.

OpenMP mexanizmlarini ishlatish uchun OpenMP derivatori bilan mos kalit bilan kompilyatsiya qilishingiz kerak (masalan, icc / ifort -openmp derleyici kaliti, gcc / gfortran -f openmp, Sun Studio -xopenmp, Visual C ++ - / openmp, PGI - mp). Derleyici OpenMP direktivalarini sharhlaydi va parallel kod yaratadi. OpenMP-ni qo'llamaydigan kompilyatorlardan foydalanilganda OpenMP direktivlari qo'shimcha xabarlarsiz e'tiborsiz qilinadi.

OpenMP yordamiga ega bo'lgan kompilyator, dasturning parallel versiyasi uchun odatiy bo'lgan alohida bloklarni shartli ravishda kompilyatsiya qilish uchun ishlatilishi mumkin bo'lgan _OPENMP so'lini belgilaydi. Bu so'l yyyym formatida aniqlanadi, bu erda yyyy va mm - qo'llab-quvvatlanadigan OpenMP standarti qabul qilingan yil va oy uchun raqamlar. Masalan, OpenMP 3.0 standartini qo'llab-quvvatlaydigan derleyici 2008/05 da _OPENMP ni belgilaydi.

Derivatning OpenMP ning har qanday versiyasini qo'llab-quvvatlashini tekshirish uchun, shartli derleme ko'rsatmalarini #i fdef yoki #i f ndef yozib olish kifoya. C va Fortran dasturlarida shartli kompilyatsiya qilishning eng oddiy misollari 1-misolda keltirilgan.

```
#i ncl ude <st di o. h>
int m a i n ( ) {
#i f def _OPENMP
    p r i n t f ( " O p e n l V P  i s  s u p p o r t e d ! \  n " ) ;
#endi f
}
```

OpenMP da parallelizatsiya dasturiy matnga maxsus ko'rsatmalar kiritilishi va yordamchi funktsiyalarni chaqirish orqali aniq amalga oshiriladi. OpenMP dan foydalanilganda, parallel dasturlashtirilgan SPMD (Single Program Multiple Data) parallel dasturiy modeli qabul qilinadi, uning ichida bir xil kod barcha parallel iplar uchun ishlatiladi.

Dastur navbatdagi maydon bilan boshlanadi - birinchi navbatda, bir jarayon (parrak) ishi, parallel maydonga kirgandan keyin yana bir necha protsedura hosil

bo'ladi, bu kodning ayrim qismlari o'rtasida taqsimlanadi. Parallel mintaqa tugagandan so'ng, bitta (ustunli master) tashqari barcha iplar tugatiladi va navbatdagi hudud boshlanadi. Dastur parallel va navbatdagi hududlarning har qanday sonini bo'lishi mumkin. Bundan tashqari, parallel joylar ham ichki bo'lishi mumkin. Tugallanmagan jarayonlardan farqli o'laroq, bug'doy hosilasi nisbatan tez operatsiya hisoblanadi, shuning uchun tez-tez o'sib chiqadigan va iplarning bekor qilinishi dasturning ish vaqtiga juda ta'sir qilmaydi.

Samarali parallel dasturni yozish uchun, dasturni qayta ishlash bilan shug'ullanadigan barcha mavzular foydali ish bilan bir xil tarzda o'rnatilgan bo'lishi kerak. Bunga turli xil OpenMP mexanizmlari mo'ljallangan yukni diqqat bilan balanslash orqali erishiladi.

Muhim nuqta ham birgalikda ma'lumotlarga kirishni sinxronlashtirishning zarurati hisoblanadi. Bir nechta tarmoqlar uchun umumiy bo'lgan ma'lumotlarning mavjudligi bir vaqtning o'zida muvofiqlashtirilmagan kirish bilan nizolar keltirib chiqaradi. Shuning uchun, OpenMP funksionalligining muhim qismi ishchi iplarni sinxronlashtirishning turli turlarini amalga oshirish uchun mo'ljallangan.

OpenMP turli xil ish zarralarini bir xil fayllarga sinxronlashtirmaydi. Agar dasturning to'g'riligi uchun zarur bo'lsa, foydalanuvchi sinxronlik ko'rsatmalarini yoki tegishli kutubxona vazifalarini aniq ishlatishi kerak. Har bir ish zarrachasiga faylga kirganda, sinxronizatsiya talab qilinmaydi.

Nazorat savollari

1. OpenMP standartining qaysi versiyasini mavjud tizimda qo'llab-quvvatlanishini aniqlang
2. OpenMP texnologiyasi uchun qo'llab-quvvatlash kompilyatorlarini kiritib, ishlatish bilan istalgan ketma-ket dasturni tuzing
3. OpenMP dasturi faqat parallel hududlardan iborat bo'lishi mumkinmi? Faqat ketma-ket joylardanmi?
4. Bir thread ustasi boshqa barcha ish zarralaridan qanday farq qiladi?

8-mavzu: Dasturlash modellari, unumdorlikni baholash. Amdal qonuni. Ma’lumotlarni va buyruqlarni parallelashtrish

REJA:

8.1. Unumdorlikni baholash

8.2. Amdal qonuni

8.3. Ma’lumotlarni parallelashtirish

8.4. Buyruqlarni parallelashtirish

Vaqt mos yozuvlar tizimini joriy etsin va uning birliklari, masalan, ikkinchi marta o'rnatilsin. Operatsiyaning davomiyligi birlikning fraksiyonlarında o'lchanadi deb taxmin qilamiz. Haqiqiy yoki hipotetik, asosiy yoki yordamchi qurilmalarning har qanday javob vaqtlari bo'lishi mumkin. Faqatgina muhim cheklov bir xil FUning barcha amaliyotlari muddatda bir xil bo'lishi kerak. Baribir, har qanday maxsus FU to'plamining ishi bizni qiziqtiradi. Odatiy bo'lib, ushbu silsilasini ishlash jarayonini ta'minlash uchun zarur bo'lgan barcha boshqa FUlar darhol ishlaydi deb taxmin qilamiz. Shuning uchun, agar maxsus buyurtmalar berilmasa, bunday holatlarda ularning haqiqiy mavjudligini hisobga olmaymiz. O'rganilayotgan FUlarning javob vaqtlari nolga teng deb hisoblanadi.

Agar oldingi operatsiyadan keyingi operatsiyani amalga oshirishni boshlamasangiz, funktsional qurilmani oddiy deb ataylik. Oddiy FU bir xil turdagi yoki turli operatsiyalarni bajarishi mumkin. Turli xil FU har xil vaqtda operatsiyalarni bajarishi mumkin. Oddiy FUning misoli konveyer bo'lmagan summa yoki ko'paytirgich bo'lishi mumkin. Ushbu FU faqat bitta turdagi operatsiyalarni amalga oshiradi. Oddiy qurilma bir vaqtning o'zida turli xil operatsiyalarni amalga oshirolmasa, juda ko'p funksiyali protsessor deb hisoblanishi mumkin va operatsiyalar vaqtidagi farqlarni hisobga olmaymiz. Oddiy FUning asosiy xususiyati faqat bittadir: har bir alohida operatsiyani bajarish uchun faqatgina uning uskunasidan foydalaniladi.

Amdalning 1-qonuni. Bir-biriga ulangan qurilmalardan tashkil topgan hisoblash tizimining ishlashi, odatda, eng ko'p ishlab chiqarish qurilmasi tomonidan aniqlanmaydi.

2.4-bayonnomada tizimning ishlash jarayoni darboğazlaridan biri ko'rsatilgan. Adabiyotda tasvirlangan kompyuter tizimlarining ayrim qismlari kompyuter texnologiyalari sohasida amerikalik Amdal ismiga bog'liq. Turli dalillarni tan olishni yo'qotmaslik uchun biz ushbu an'anani buzmaymiz va hatto soddagina bo'lsa ham, ularga tegishli dalillarni qoldiramiz. Ehtimol, matnni biroz o'zgartirib, ularni materialning hozirgi taqdimotiga moslashtirishi mumkin. Bu sabablarga ko'ra, Amdalning so'nggi qonunini 2.4-sonli bayonotning so'nggi natijalari deb atashgan.

Biz realizmning eng yuqori ko'rsatkichi haqida gapirganda, tizimning ishlashi buyruq berishni taqsimlash bilan ta'minlanadi, bu esa uzilish vaqtini kamaytiradi. Maksimal ishlash turli rejimlarda bajarilishi mumkin. Ayniqsa, Assertion 2.4 da aytilganidek, sistema oddiy qurilmalardan iborat va tizimning grafikasi bog'liq bo'lsa, sinxron rejimda FUning eng sekin ishlashiga teskari proportsional soat bilan erishiladi. Tizim bir xil ishlashning 5 ta oddiy qurilmasidan iborat bo'lishi kerak. Holbuki, ulangan tizim sharoitida va unchalik aloqador bo'lmagan hollarda, katta operatsiya qilish vaqtlari bilan mumkin bo'lgan eng yuqori haqiqiy ish bir qurilmaning maksimal ishlashi bilan bir xil va tengdir.

Nazorat savollari

1. Nima uchun konveyer funktsional qurilmalarida bir xil tetiklashuvni amalga oshirish davom etadi?

2. Hisoblash tizimining grafigini yo'nalishli halqaga aylansin, ularning barcha yoylari bir yo'nalishda, masalan, soat yo'nalishi bo'yicha yo'naltiriladi. Tizimning ishlashi uchun turli xil vaqt rejimi mavjudligini ko'rsating.

3. Amdal qonuni yozib bering

4. Buyruqlarni parallelashtirish haqida

9-mavzu: Parallelashtirish tizimlarining arxitekturasi, Flinn klasifikatsiyasi, MIMD arxitekturasi

REJA:

9.1. Parallelashtirish tizimlarining arxitekturasi

9.2. Flinn klasifikatsiyasi

9.3. MIMD va SISD arxitekturalari

9.4. Feng klasifikatsiyasi

O’tgan mavzularda allaqachon parallel hisoblash tizimlarini tashkil etishning turli xil usullari aniqlangan. Bu erda vektor konveyerli kompyuterlar, ommaviy parallel va matritsali tizimlar, keng buyruqli so'zlar, maxsus protsessorlar, klasterlar, juda ko'prikli arxitekturali kompyuterlar va hokazo. Kompyuterlarga qo'ng'iroq qilishingiz mumkin. Xuddi shu ro'yxatda hali muhokama qilinmagan mimariler Masalan, sistolik massivlar yoki dataflow kompyuterlar. Shu bilan birga, tavsifning to'liqligi uchun xotira tashkil etilishi, protsessorlar o'rtasidagi muloqot topologiyasi, alohida qurilmalar ishining sinxronlashi yoki operatsiyalarni bajarish usuli kabi muhim parametrlar qo'shiladi, keyin turli xil me'morchiliklarning soni mutlaqo tushunarsiz bo'ladi.

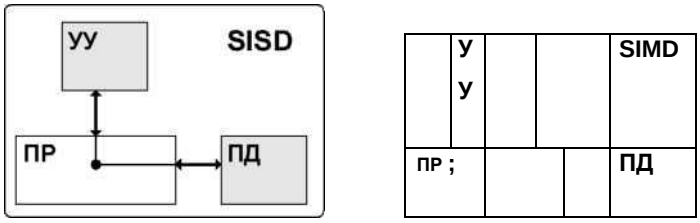
Nima uchun juda ko'p parallel arxitektura mavjud? Ularning o'zaro aloqasi qanday? Har bir me'morlikni tavsiflovchi asosiy omillar nima? Bu kabi savollarga javobni topishning o'zi kompyuter tizimlarining arxitekturasini tasniflash zaruriyatiga olib keladi. Bu yo'nalishdagi faol harakatlar M. Flynn o'tgan asrning 60-yillari oxirida tasnifi birinchi versiyasining nashr etilganidan so'ng boshlangan edi.

Yaxshi tarkibni tasniflash kompyuterni takomillashtirishning mumkin bo'lgan usullarini taklif qilishi mumkinligini inkor etmang. Masalan, qimmatbaho yoki yuqori ko'rsatkichlarga ko'ra tasniflashda noyob bo'lmagan "oq nuqta" ni topish qiyin. Biroq, dasturlarning soddaligi va moslashuvchanligi nuqtai nazaridan mumkin bo'lgan sistematikalar haqidagi aks ettirish yangi arxitekturalarni qidirish yo'nalishini aniqlash uchun juda foydali bo'lishi mumkin.

M. Flinn tasnifi (M.Flynn). Ehtimol, eng erta va eng taniqli 1966 yilda M. Flynn tomonidan taklif etilgan hisoblash tizimlarining arxitekturasi tasnifi. Tasniflash oqim kontseptsiyasiga asoslangan bo'lib, u protsessor tomonidan qayta ishlangan buyruqlar yoki ma'lumotlarni ketma-ketligini anglatadi. Buyruq oqimlari va ma'lumotlar oqimlarining soniga qarab, Flynn to'rtta me'moriy klasslarni belgilaydi.

SISD (Single Instruction stream / Single Data stream) - bitta buyruqlar oqimi va bitta ma'lumot oqimi (3.1-rasm). M. Flynnning tasnifini tasvirlaydigan raqamlarda quyidagi yozuv ishlatiladi:

PR bir yoki bir nechta protsessor elementlari, CU - nazorat qurilmasi, PD - ma'lumotlar xotirasi. SISD klassi, birinchi navbatda, klassik ketma-ketlikdagi mashinalar yoki muqobil ravishda Von Neumann tipidagi mashinalarni, masalan, PDP-11 yoki VAX 11/780 ni o'z ichiga oladi. Bunday mashinalarda faqat bitta buyruq buyrug'i mavjud, barcha buyruqlar ketma-ketlikda bir-biriga qayta ishlanadi va har bir buyruq bir skaler operatsiyani boshlaydi. Buyumlarning ishlash tezligini va arifmetik operatsiyaning tezligini oshirish uchun quvurlarni qayta ishlashni qo'llash muhim emas: bu ikkala CDC 6600 skalar ishlab chiqilgan qurilmalar va CDC 7600 konveyerlar bilan bu sinfga tushadi.



SISD va SIMD sinflar tasnifi. M. Flinn

SIMD (Single Instruction Stream / Multiple Data Stream) - bitta buyruqlar oqimi va bir nechta ma'lumotlar oqimi (3.1-rasm). Ushbu turdagi arxitekturalarda, oldingi sinfdan farqli o'laroq, vektorli buyruqlar bilan bir qatorda buyruqlar oqimi saqlanadi. Bu sizga bir vaqtning o'zida bir nechta ma'lumotlarda, masalan, vektor elementlari bo'yicha bitta arifmetik operatsiya qilish imkonini beradi. Vektorli operatsiyalarni bajarish usuli aniqlanmagan, shuning uchun vektor elementlarini

qayta ishlash, ILLIAC IVda bo'lgani kabi, protsessor matritsasi yoki konveyer yordamida, masalan, SG-1 mashinasida bajarilishi mumkin.

MISD (Multiple Instruction stream / Single Data stream) - bir nechta buyruqlar xaridi va bitta ma'lumotlar oqimi (3.2-rasm). Ta'rif bir xil ma'lumotlar oqimini qayta ishlaydigan ko'plab protsessorlarning arxitekturasida mavjudligini bildiradi. Biroq, Flynn ham, kompyuter arxitekturasi sohasidagi boshqa mutaxassislar ham ushbu printsipga asoslangan real-hayot hisoblash tizimining ishonchli misolini keltira oldilar. Bir qator tadqiqotchilar konveyerlarni bu sinfga havola etmoqdalar, ammo bu ilmiy jamiyatda aniq tan olinmagan.

MIMD (Multiple Instruction stream / Multiple Data stream) - bir nechta buyruqlar oqimi va bir nechta ma'lumotlar oqimi (3.2-rasm). Bu sinf komputer tizimida bir nechta buyruqlar ishlash moslamalari mavjudligini, bir kompleksga birlashtirilganligini va ularning har biri o'zining o'z navbatida buyruqlar va ma'lumotlar oqimlari bilan ishlashini ta'kidlaydi.

T. Feng (T. Feng) tasnifi. 1972 yilda T.Feng ikkita oddiy xususiyatga asoslangan hisoblash tizimlarini tasniflashni taklif qildi. Birinchisi, mashinaning ko'rsatmalarini bajarishda parallel ravishda ishlov beriladigan kompyuter so'zidagi bitlarning soni. Deyarli barcha zamonaviy kompyuterlarda bu raqam mashina so'zining uzunligiga to'g'ri keladi. Ikkinchi xarakteristika ushbu hisoblash tizimi tomonidan bir vaqtning o'zida qayta ishlangan so'zlar soniga teng. Terminani biroz o'zgartirib, har qanday kompyuterning ishlashi bit qatlamlarining parallel ishlashi sifatida ifodalanishi mumkin, ularning har birida m bitlar mustaqil ravishda aylanadi. Ushbu sharhga asoslanib, ikkinchi xarakterga bit qatlamining kengligi deyiladi.

Nazorat savollari

1. Kompyuterlarni o'z xarajatlariga (vazni, hajmi, xato tolerantligi) muvofiq tasniflash amaliyotda foydali bo'lishi mumkinmi? Agar shunday bo'lsa, foydalanuvchilarning qaysi klassi uchun?

2. Kompyuterlarni o'zlarining eng yuqori ko'rsatkichlariga ko'ra tasniflash amaliyotda foydali bo'lishi mumkinmi? Agar shunday bo'lsa, foydalanuvchilarning qaysi klassi uchun?

3. Flinn klassifikasiyasining qaysi klasi, HP Superdome kabi umumiy xotirasi bo'lgan zamonaviy kompyuterlardir?

4. Flinn tasnifi bo'yicha qanday saboqlarni superskalyar va VLIW-protsessorlari bilan bog'lash mumkin?

5. SIMD zamonaviy kompyuter sinflari misollarini keltiring.

6. Feng metriori yordamida Tor500 ro'yxatidagi dastlabki o'nta kompyuterni tavsiflang. Hendler, Schneider, Skillikorn yondoshuvlari bilan ham xuddi shunday qiling.

7. Hendler va Flinn tasniflari bir-birlari bilan qanday bog'liq?

10-mavzu: Vektor-konveyer arxitekturasi, CRAY 90 kompyuteri (funksional bloklar, yuqori unumdorlik darajasi), umumiy xotirali parallel kompyuterlar, HP Superdome kompyuteri

REJA:

10.1. Vektor-konveyer arxitekturasi

10.2. CRAY 90 kompyuteri

10.3. Klaster va funksional bloklar

10.4. HP Superdome kompyuteri

Kompyuterning arxitekturasini bilish uchun uning eng yuqori ko'rsatkichini hisoblash oson. Biz birinchi navbatda real raqamlar bo'yicha operatsiyalarni bajarish tezligidan manfaatdor ekanligimiz sababli, biz haqiqiy arifmetika uchun funktsional qurilmalarni imkon qadar ko'proq yuklashimiz kerak. Balandlikning teskari tomonidagi operatsiya kamdan-kam qo'llaniladi va bo'linish jarayonida qo'shimcha ravishda ko'paytirish ishi talab etiladi. Shuning uchun, kompyuterning eng yuqori ko'rsatkichini aniqlash uchun faqat ko'paytirish va qo'shimcha qurilmalarni ishlatamiz. Maksimal ishlash uchun ular birlashtirilgan rejimda ishlatilishi kerak. D-

= B, + Cr-x d shakllarining ishlashini amalga oshirishga kelganimizda shunga o'xshash narsalarni qildik. Agar qo'shimcha ravishda har bir bunday qurilma vektorli ishni bajarish uchun ikkita ichki konveyerdan foydalanishni hisoblasak, ikkita qurilmaning tizimi soat bo'yicha to'rtta operatsiya natijasini beradi. Kompyuterning aylanish vaqti 4.1 n, shuning uchun bitta Cray C90 protsessorining eng yuqori ko'rsatkichi deyarli 1 gflops yoki sekundiga 109 operatsiya bo'ladi. Agar kompyuterning barcha 16 ta protsessori bir vaqtning o'zida ishlayotgan bo'lsa, unda eng yuqori ko'rsatkich 16 gflopsgacha ko'tariladi.

Biz ushbu kompyuterning arxitekturasining asosiy xususiyatlarini buzib tashladik, undan nega bu qadar tez o'ylab topilgani aniqlandi. Biroq, u uchun samarali dasturlarni qanday yozishni bilish uchun siz uning boshqa tomonini o'rganishingiz kerak. Haqiqiy dasturlarda ishlashini kamaytiradigan omillarni ta'kidlash kerak. Ushbu qadam holda, hosildorlikni oshirish uchun dasturda nimani o'zgartirish kerakligini tushunish qiyin bo'ladi. Paragrafning qolgan qismi ushbu kompyuterda dasturlarni bajarish samaradorligini tahlil qilishga bag'ishlanadi.

Birinchidan, biz terminologiya haqida qaror qabul qilishimiz kerak. Kompyuterda vektor-konveyer arxitekturasi mavjud. Vektorli ishlov berish rejimi yordamida vaqtning asosiy daromadini olish mumkin. Kompyuterning buyruq tizimidagi vektor buyruqlari uni bajarish uchun ishlatilsa, ba'zi dastur qismlari vektor rejimida ishlov berilishi mumkin. Dasturning barcha qismini vektorli buyruqlar bilan almashtirsak, u holda uning to'liq vektorizatsiyasi haqida gapiramiz. Aks holda, biz qisman vektorizatsiya yoki umuman bir qismni vektor qilish imkonsizligi bilan ishlaymiz. Dasturda tegishli qismlarni topish va ularni vektorli buyruqlar bilan almashtirish jarayoni dasturning vektorizatsiyasi deb ataladi.

Nazariy jihatdan, spinning chuqurligi oshgani sayin, unumdorlik, chegarada ma'lum bir qiymatga yaqinlashadi. Biroq, amalda, maksimal ta'sir birinchi qadamlarda bir joyga yetib boriladi va natijada ishlash deyarli bir xil yoki kamayib boradi. Ushbu nazariya va amaliyot o'rtasidagi farqning asosiy sababi Cray C90 kompyuterlarning juda cheklangan vektor registriga ega bo'lishidir: ularning har biri 128 so'zdan iborat 8 ta registr. Odatda, spinning chuqurligini oshirish kirish

vektorlarining sonini ko'payishiga olib keladi. Demak, bizda ham shunday bo'ldi. Asl shakldagi fragment tashqi aylananing har bir itarishida uchta kirish vektorini talab qildi. Chuqurlik 2 ning targ'iboti to'rtta vektorni o'rnatish zaruriyatini tug'dirdi, 3 5 vektor chuqurlikka ko'tarish uchun talab qilinadi va hokazo. Har bir qo'shimcha vektor ilgarigi rag'batlantiruvchi chuqurlikni oshirib, tor joyga aylanadi.

Hewlett-Packard Superdome hisoblash tizimi misolida bu sinf kompyuterlarining arxitekturasini o'rganamiz. Kompyuter 2000-yilda paydo bo'lgan va Tor500 2001-yil noyabr nashrlarida ular 147 vazifani egallagan.

HP Superdome kompyuteri standart komplektda 2 dan 64 gacha bo'lgan protsessorlarni birlashtirishi va keyinchalik tizimning kengayishi mumkin. Barcha protsessorlar ccNUMA arxitekturasiga muvofiq tashkil etilgan umumiy xotiraga ega. Bu, birinchi navbatda, barcha jarayonlarning bir manzil maydonida ishlashi, odatdagi o'qish / yozish operatsiyalari orqali xotiraning har qanday baytiga murojaat qilish deganidir. Ikkinchidan, tizimdagi mahalliy xotiraga kirish masofaviy xotiraga kirishdan ko'ra biroz tezroq bo'ladi. Uchinchidan, protsessorlarning kesh xotirasi sababli yuzaga kelishi mumkin bo'lgan ma'lumotlar kelishmovchiligi muammolari apparat darajasida hal qilinadi.

Nazorat savollari

1. Ikkala tizimning afzalliklari va kamchiliklari: bir xil protsessorlar asosida qurilgan 16 protsessorli xotira va 16 ta protsessor hisoblash klasteri.

2. RA-8700/750 MGts protsessor asosida ishlaydigan 16 ta protsessor HP Superdome xotira kompyuterini va 16 protsessor hisoblash klasterining eng yuqori ko'rsatkichi nima? Har bir yechimning narxini hisoblab ko'ring. Yana qimmatroq echimning afzalliklari nimada?

3. Zamonaviy SMP-serverlarda umumiy xotiraga kirishni tashkil qilish usullarini o'rganish.

4. IA-64 arxitekturasining asosiy xususiyatlari va xususiyatlarini ajratib ko'rsatish. Ushbu arxitektura asosida qanday zamonaviy protsessorlar qurilgan?

11-mavzu: Ajratilgan xotirali kompyuter tizimlari, klaster tizimlar (Beowulf), MBC – 1000M superkompyuteri, klasterlarining kommunikatsion tizimlari

REJA:

11.1. Ajiratilgan xotiralari tizimlar

11.2. Klaster tizimlari

11.3. Beowulf superkompyuteri

11.4. Klasterlarning kommunikatsion tizimlari

NASAning Goddard Space Flight Center (GSFC) tizimida parallel tizimlar - Beowulf-klasterlar deb nomlangan birinchi loyihalardan biri paydo bo'ldi. Beowulf loyihasi 1994-yilning yozida ishga tushirildi va tez orada Intel-486DX4 / 100 MGts protsessorlarida 16 ta protsessorlar to'plandi. Har bir tugunda muntazam chekilgan tarmoq uchun 16 MB RAM va 3 ta tarmoq kartasi o'rnatildi. Ushbu konfiguratsiyani amalga oshirish uchun mavjud tarmoq kartalari orasidagi trafikni tarqatadigan maxsus haydovchilar ishlab chiqildi.

Keyinchalik GSFCda TheIVIVE klasteri yig'ildi - uning strukturasi anjirda ko'rsatilgan Yuqori parallel o'rnatilgan virtual muhit. 3.17. Ushbu klaster E2, B, G va DL kichik birikmalaridan iborat bo'lib, 332 protsessor va ikkita xost kompyuterni birlashtiradi. Ushbu klasterdagi barcha tugunlar Red Hat Linux operatsion tizimida ishlaydi.

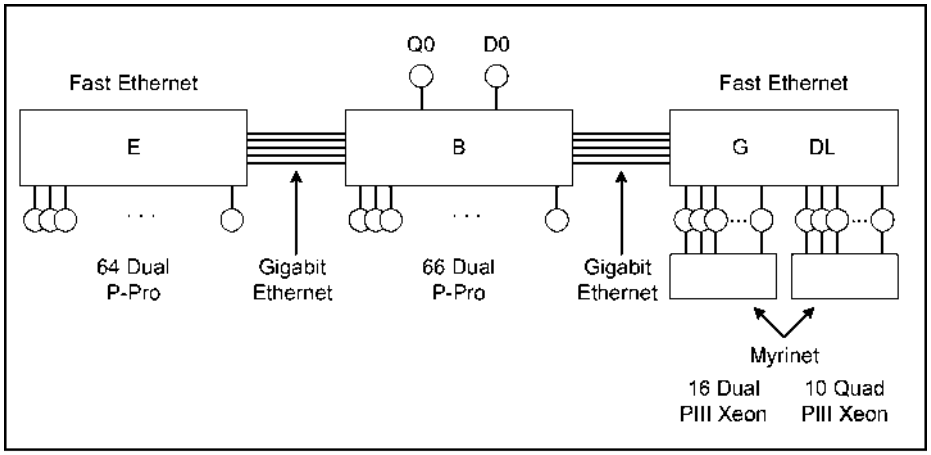


Рис. 3.17. Структура кластера TheHIVE

1998 yilda Avalon Linux klasteri Los Alamos milliy laboratoriyasida 533 MHz soat tezligi bilan ishlaydigan Alpha 21164A protsessorlari asosida yaratilgan. Dastlab Avalon 68 ta protsessordan iborat bo'lib, ularning soni 140 ga ko'tarildi. Har bir tugun 256 MB RAM o'rnatilgan, 3 GB qattiq disk va Fast chekilgan tarmoq adapteri mavjud. Avalon loyihasining umumiy qiymati 313 ming dollarni tashkil etdi. Klaster tomonidan ko'rsatiladigan LINPACK testining ishlashi - 47.7 Gliflops, Tor500 ro'yxatining 12-nashrida 114-o'rinni, 152-protsessor IBM RS / 6000 SP tizimining yonida joylashgan. Bundan tashqari, 1998 yilda Supercomputing'98 yuqori samarali kompyuteri bo'yicha eng nufuzli konferentsiyada Avalon yaratuvchilari "Avalon: Yuqori alfa / Linux klasteri $ 150k uchun $ 10 gfplni qo'lga kiritdi" nomli hisobotni taqdim etib, "Eng yaxshi narxlari / nisbati" nominatsiyasi bo'yicha birinchi sovrinni qo'lga kiritdi. ("1998 Gordon Siyohdondagi narx / ishlash mukofoti").

2000 yil aprel oyida Cornell University PEA loyihasi doirasida biomedikal tadqiqotlarni o'tkazish uchun Velocity + klasteri yaratildi. Har biri to'rtta Intel Pentium III protsessorlari bo'lgan 64 ta tugunlardan iborat. Tugmalar Windows 2000 ishlayotgan va LAN bilan tarmoqqa ulangan.

LoBoS (Shelfes ustida qutilar) loyihasi 1997 yil aprel oyida AQSh Milliy Sog'liqni Saqlash Institutida amalga oshirildi. Gigabit Ethernet aloqa vositasi sifatida foydalanish qiziq. Dastlab, ikkita Intel Pentium Pro / 200 MGts protsessor, 128 MB RAM va har bir tugunda 1,2 Gb disk bo'lgan 47 ta tugunlardan iborat edi. 1998 yilda LoBoS2 loyihasining navbatdagi bosqichi amalga oshirildi, shu vaqt mobaynida tugunlar ish stoli kompyuterlarga aylantirilib, klaster integratsiyasini saqlab qoldi. Hozir LoBoS2 100 protsessorli tugunlardan iborat bo'lib, har biri Pentium II / 450 MHz protsessor, 256 MB RAM va 9 GB disk xotirasi mavjud. Bog'langan klasterga qo'shimcha ravishda

4,2 dona umumiy RAID-quvvati bo'lgan kompyuterlarni boshqarish.

2000 yilda KLAT2 klasteri (Kentukki Linux Athlon Testbed 2) qiziqarli rivojlanish bo'ldi. KLAT2 tizimi AMD Athlon / 700 MGts protsessor va har biriga 128 MB RAMga ega 64 disksiz tugmachadan iborat. Dasturiy ta'minot,

kompilyatorlar va matematikalar kutubxonalari (SCALAPACK, BLACS va ATLAS) 3D-Nowdan samarali foydalanish uchun takomillashtirildi! AMD protsessorlari. Ushbu ish butun tizimning ish faoliyatini sezilarli darajada oshirishga imkon berdi. "Flat Neighborhood Network" (FNN) deb nomlangan foydalanilgan tarmoq hal etish katta qiziqishdir. Har bir tugunning to'rtta Fast Ethernet tarmoq adapteri bor va tugunlar to'qqiz 32-portli kalitlardan foydalanib ulangan. Bundan tashqari, har qanday ikkita tugun uchun har doim kalitlardan biri orqali to'g'ridan-to'g'ri aloqa o'rnatiladi, lekin barcha tugunlarni bitta kalit orqali ulashning hojati yo'q. AMD arxitekturasi va FNN topologiyasi uchun dasturiy ta'minotni optimallashtirish natijasida LINPACK testida rekord narx / ishlash ko'rsatkichi - 1 Gflops uchun 650 dollargacha erishish mumkin edi.

Nazorat savollari

1. tarqalgan xotiraga ega bo'lgan ko'pchilik parallel hisoblash tizimlari tugunlar sifatida SMP kompyuterlarini ishlatadi. Muammoni hal qilish uchun qanday algoritm bo'lishi kerak, bunday arxitekturani ishlatadigan kompyuterni ishlatish kerak?

2. Ikkita tizim mavjud. Ulardan biri tezkor protsessorlarga va sekin ulanishlarga ega, ikkinchisi esa sekin ishlovchilarga va tez ulanishga ega. Har bir tizimning afzalliklari va kamchiliklari qanday? Dasturlar qaysi tizimda yanada kengroq miqyosda bo'ladi?

3. Tarqatilayotgan xotira va ikki o'lchamli torusli topologiyaga ega bo'lgan aloqa tarmog'i bo'lgan haqiqiy hisoblash tizimiga misol keltiring.

4. Tor500 ro'yxatining so'nggi nashriga kiritilgan qancha kompyuterda 1000 dan ortiq protsessor mavjud? Umumiy xotirada qancha kompyuter bor?

12-mavzu: Grid – texnologiyasi va metaxisoblash, unumdorlikni baholash (MIPS, FLOPS)

REJA:

12.1. Grid texnologiyasi

12.2. Metaxisoblash

12.3. Unumdorlikni baholash

GIMPS - Buyuk Internet Mersenne Prime Search. Mepsenning bosh sonlarini, ya'ni, 2 ^ -1 formulalarini toping, bu erda P - asosiy raqam. 2001 yil noyabr oyida, ushbu loyihaning bir qismi sifatida, Mersenning maksimal soni hozircha 213,466,917ni tashkil qildi. Dunyoning o'nlab minglab kompyuterlari hisoblash resurslarini uzaytirib, bu vazifani ikki yarim yil davomida ishlab kelmoqda. Electronic Frontier Foundation, 10 million raqamli Mer raqamini topish uchun $ 100,000 mukofotini taklif qiladi. Loyiha manzili: http://mersenne.org/.

Globus loyihasi dastlab Argon milliy laboratoriyasida tug'ilgan va hozir butun dunyoda keng tan olingan. Loyihaning maqsadi global axborot-hisoblash muhitini tashkil qilish uchun imkoniyat yaratishdir. Loyiha turli xil lokal yuk taqsimlash tizimlari, autentifikatsiya qilish tizimlari, Nexus kommunikatsiya kutubxonasi, monitoring va monitoring vositasi va boshqalar uchun bir nechta dasturiy ta'minot va tizimlar ishlab chiqdi. Ishlab chiqilgan uskunalar erkin tarzda tarqatiladi. manba kodli Globus Toolkit to'plami sifatida. Hozirgi kunda Globus ko'plab boshqa yirik loyihalarda, masalan, Milliy Texnologiya Grid, Axborot Quvvat To'lqinlari va European DataGrid kabi asosda olingan. Qo'shimcha ma'lumotni http://www.globus.org/ saytidan topishingiz mumkin.

So'nggi yillarda tarmoq texnologiyalarining rivojlanishi juda katta. Gigabit aloqa yuzlab kilometr masofadagi kompyuterlar o'rtasidagi aloqalar umumiy haqiqatga aylanmoqda. Turli hisoblash tizimlarini yagona tarmoq ichida birlashtirib, maxsus hisoblash muhiti yaratishingiz mumkin. Ba'zi kompyuterlar ulanishi yoki ulanishi mumkin, ammo foydalanuvchining nuqtai nazaridan ushbu virtual muhit bitta metakomputer hisoblanadi. Bunday muhitda ishlash foydalanuvchi faqat

muammoni hal qilish bo'yicha vazifani qo'yadi, qolganlari esa metacomputerning o'zi tomonidan amalga oshiriladi: mavjud hisoblash resurslarini izlaydi, ularning ish faoliyatini nazorat qiladi, kerak bo'lganda ma'lumotlar uzatadi, topshiriq bajariladigan kompyuter formatiga ma'lumotlar uzatiladi; va hokazo. Foydalanuvchiga qaysi kompyuter resurslari taqdim etilganligini bilish ham mumkin emas. Va sizningcha, qanchalik tez-tez bilishingiz kerak? Muammoni hal qilish uchun hisoblash qobiliyatlari kerak bo'lsa, siz metacomputerga ulangansiz, topshiriqni topshirishingiz va natijani olishingiz mumkin.

Bu erda elektr tarmog'i bilan deyarli to'la o'xshashlik bor. Elektr chovgumni elektr rozetkasiga ulashda qaysi stansiya elektr energiyasi ishlab chiqarishi haqida o'ylamaysiz. Sizga resurs kerak, uni ishlatasiz. Aytgancha, elektr tarmog'iga o'xshab, ingliz tilidagi adabiyotda tarqalgan hisoblash muhiti Grid yoki "hisoblash tarmog'i" deb ataladi. Bundan tashqari, Grid va metacomputer so'zlarini sinonim sifatida ishlatamiz.

Elektr tarmog'i bilan o'xshashlikni davom ettiradigan bo'lsak, men metacomputerga faqatgina nom bilan emas, balki foydalanuvchi bilan o'zaro aloqada bo'lgan oddiy usulni ham etkazmoqchiman. Ammo bu erda metakomputerning o'zini tashkil etish murakkabligi bilan bog'liq bo'lgan asosiy muammolar paydo bo'ladi. An'anaviy kompyuterdan farqli o'laroq, metakomputerda faqat o'ziga xos xususiyatlar to'plami mavjud:

□ metacomputer oddiy kompyuter resurslari bilan tengsiz katta resurslarga ega. Bu deyarli barcha parametrlar uchun amal qiladi: mavjud protsessorlar soni, xotira miqdori, faol ilovalar soni, foydalanuvchilar va boshqalar;

□ metacomputer tabiatda taqsimlanadi. Metakomputerning tarkibiy qismi bir-biridan yuzlab, minglab kilometr uzoqlikda bo'lishi mumkin, bu muqarrar ravishda katta latentlikka olib keladi va natijada ularning o'zaro ta'sirining tezligiga ta'sir qiladi;

□ metacomputer konfiguratsiyani dinamik ravishda o'zgartirishi mumkin. Ba'zi shaxsiy kompyuterlar unga ulangan va ularning resurslaridan foydalanish huquqlarini topshirgan, ba'zilari o'chirib qo'yilgan va mavjud emas. Lekin

metacomputer bilan ishlaydigan foydalanuvchi shaffof bo'lishi kerak. Metakomputerning qo'llab-quvvatlash tizimining vazifasi, metacomputerning hozirgi konfiguratsiyasidan qat'i nazar, tegishli resurslarni izlash, ularning ishlashini tekshirish, kiruvchi vazifalarni taqsimlashdir;

□ Metacomputer heterojen. Vazifalarni tayinlashda unga kiritilgan operatsion tizimlarning o'ziga xos xususiyatlarini hisobga olish kerak. Turli tizimlar turli xil buyruqlar tizimlari va ma'lumotlarni taqdim etish formatlarini qo'llab-quvvatlaydi. Turli xil tizimlarda turli xil yuklamalar bo'lishi mumkin, kompyuter tizimlari bilan aloqa turli tarmoqli kenglikdagi kanallar orqali amalga oshiriladi. Nihoyat, metacomputerda asosiy shaxsiy kompyuterlardan boshlab, Top500 ro'yxatidagi eng kuchli tizimlar bilan yakunlangan turli xil me'moriy tizimlar bo'lishi mumkin;

□ metacomputer turli tashkilotlarning resurslarini birlashtiradi. Muayyan resurslarga kirish va ulardan foydalanish siyosati muayyan tashkilotga aloqadorligiga qarab juda katta farq qilishi mumkin. Meta-kompyuter hech kimga tegishli emas, shuning uchun uni boshqarish siyosati faqat umumiy ma'noda belgilanishi mumkin. Shu bilan birga, metakomputerning ko'plab komponentlari ishining mustahkamligi barcha xizmatlar va xizmatlarning ishlashini majburiy standartlashtirishni nazarda tutadi.

Nazorat savollari

1. Tashkilotingiz tarmog'ida necha kompyuter mavjud? Ularning umumiy ishlashi qanday? Ular bir parallel kompyuter rejimida ishlatilishi uchun nima qilish kerak?

2. Hisoblash klasterini va metakomputerni dastur dasturchisi nuqtai nazaridan farqlash nima?

3. Metakomputerning eng yuqori ko'rsatkichini qanday hisoblash mumkin?

4. Metakomputerning haqiqiy dasturlarda ishlashini kamaytiradigan 10 ta sababni ayting.

5. Metakomputerning faqat SMP nodlaridan iborat klasterlar birlashishini taxmin qilaylik. Dasturni parallel ravishda qanday dasturlash uchun ishlatish mumkin?

13-mavzu: Parallelashtirish algoritmlarining samaradorlik ko’rsatkichlari, alohida yig’indini xisoblash usullari, yig’indini kaskad chizmasi

REJA:

13.1. Parallelashtirish algoritmlari

13.2. Yig’indini ketma-ket va kaskadli xisoblash

13.3. Samaradorlik ko’rsatgichlari

Bir qator ketma-ketlikdagi raqamli qiymatlarni topishning oddiy vazifasini hisoblashning parallel usullarini yaratish va tahlil qilish usullari ko'rib chiqaylik

$$S_k = \sum_{i=1}^{k} x_i, \quad 1 \le k \le n$$

,

Yig’indini ketma-ket xisoblash usuli.

Ushbu muammoni hal qilishning an'anaviy algoritmi raqamli to'plam elementlarini ketma-ket yig'ishdan iborat

$$S = 0,$$
$$S = S + x_1, \ldots$$

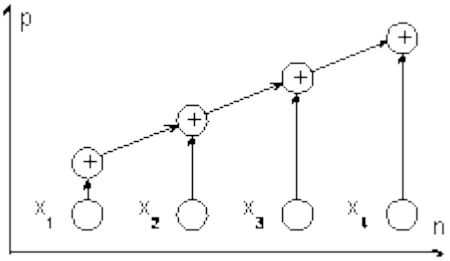
$$G_1 = (V_1, R_1)$$

,

где $V_1 = (v_{01}, \ldots, v_{0n}, v_{11}, \ldots, v_{1n})$ есть множество операций суммирования (вершины $v_{01}, \ldots, v_{0n}$ обозначают операции ввода, каждая вершина v_{1i}, $1 \le i \le n$, соответствует прибавлению значения x_i к накапливаемой сумме S), а

$$R_1 = ((v_{0i}, v_{1i}), (v_{1i}, v_{1i+1}), \quad 1 \le i \le n-1)$$

есть множество дуг, определяющих информационные зависимости операций.



4.1-rasm. Yig’indini ketma-ket hisoblash algoritmi

Kassa summasi

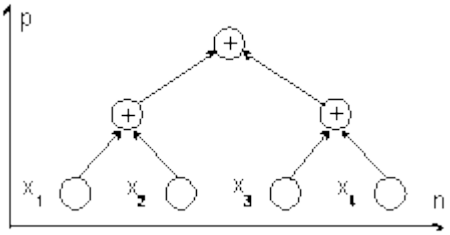
Qo'shish operatsiyasining assotiviyasiga asoslangan holda hisoblash jarayonini qurishning boshqa usullari bilan yig'ish algoritmining parallelligi mumkin bo'ladi. Olingan natija versiyasi (adabiyotda kaskad sxemasi sifatida tanilgan) quyidagilar (4.2-rasm):

kaskad sxemasining birinchi iteratsiyasi bo'yicha barcha dastlabki ma'lumotlar juftlarga bo'linadi va har bir juft uchun qiymatlar yig'iladi,

natijada olingan barcha juftlar juftlarga bo'linadi va juft qiymatlari jamlanmasi qayta bajariladi va hokazo.

Ushbu hisoblash sxemasi grafik sifatida belgilanishi mumkin (let)

$$G_2 = (V_2, R_2),$$



4.2-rasm. Yig’indini xisoblashning kaskad alogoritmi sxemasi

Ketma-ket yig'indini xisoblash algoritmining operatsiyalari soni bilan mos keladi. Kaskad davri alohida yinelemelerinin parallel ravishda amalga oshirilishi bilan, parallel yig'ish operatsiyalari umumiy soni teng

Nazorat savollari

1. Yig’indini xisoblashning ketma-ket usuli

2. Yig’indini xisoblashning kaskad chizmasi
3. Yig’indini parallel xisoblashning samaradorlik ko’rsatgichlari

14-mavzu: Vektor va matritsani ko’paytirsh, matritsa elementlarini taqsimlash, vazifani protsessorlarga taqsimlash

REJA:

14.1. Vektor va matritsani ko’paytirish
14.2. Vektorni va matritsaga ko’paytirishni parallelashtirish
14.3. Vektorni va matritsaga gorizantal satr yoki vertikal ustun bo’ylab ko’paytirish usuli
14.4. Vektorni va matritsaga blokli to’rtburchak bo’yicha ko’paytirish usuli

Matritsa operatsiyalari keng ko`lamli jarayonlarni, hodisalar va tizimlarni matematik modellashtirishda keng ishlatiladi. Matritsa hisob-kitoblari ko'plab ilmiy va muhandislik hisob-kitoblarining asoslarini tashkil etadi - dasturlar, kompyuter matematikasi, fizika, iqtisod va hokazo sohalar orasida.

Matritsalarni hisob-kitoblarni samarali bajarish muhimligini hisobga olgan holda, ko'plab standart dastur kutubxonalari turli matritsali operatsiyalar uchun protseduralarni o'z ichiga oladi. Matritsalarni qayta ishlash uchun dasturiy ta'minot hajmi muntazam ortib bormoqda - maxsus matritsa turlarini (uchburchak, lenta, siyrak va boshqalar) yangi iqtisodiy saqlash tuzilmalari ishlab chiqilmoqda, algoritmlarning har xil yuqori performansli mashinalarga bog'liqligi yaratilmoqda, nazariy tadqiqotlar olib borilmoqda. tezroq matritsa hisoblash usullarini topish.

Matematik hisob-kitoblarga ko'ra, parallel hisoblashning klassik maydoni qo'llaniladi. Bir tomondan, yuqori samarali multi-protsessorli tizimlardan foydalanish, echilishi kerak bo'lgan vazifalarning murakkabligini ancha oshirishi mumkin. Boshqa tarafdan, uning oddiy formulasi tufayli matris operatsiyalari

parallel dasturlashning ko'plab usullari va usullarini namoyish qilish uchun ajoyib imkoniyat yaratadi.

Ushbu bobda matritsa-vektorning ko'payishi uchun parallel hisoblash usullari ko'rib chiqiladi, keyingi bobda matritsalarni ko'paytirishning ishlashini ko'rib chiqamiz. Matritsalarni hisoblashning muhim turi - linear tenglamalar tizimlarini echish - 8-bobda keltirilgan. Yuqorida sanab o'tilgan barcha muammolar bo'yicha umumlashtirilgan matritsalarni bir vaqtning o'zida ajratish bilan ajratish muammosi Bo'lim 6.2da muhokama qilinadi.

Quyidagi materialni taqdim etayotganda, ko'rib chiqilayotgan matrislar matritsa elementlarining umumiy soniga nisbatan nol elementlarning soni ahamiyatsiz bo'lgan zich ekanligini taxmin qilamiz.

Matritsalarni hisoblashning ko'plab usullari uchun matritsaning turli elementlari uchun bir xil hisoblash harakatlarini takrorlash xarakterlidir. Ushbu moment matritsa hisob-kitoblarini bajarishda ma'lumotlardagi parallellik mavjudligini ko'rsatadi va natijada matritsa operatsiyalarining parallelizatsiyasi ko'p hollarda iplar orasidagi qayta ishlangan matritsalarni ajratish uchun kamayadi. Matritsani ajratish usulini tanlash parallel hisoblashning muayyan usulini aniqlashga olib keladi; turli ma'lumotlar tarqatish sxemalari mavjudligi matris hisob-kitoblari uchun bir qator parallel algoritmlarni hosil qiladi.

Matritsalarni ajratish uchun eng keng tarqalgan va keng tarqalgan usullar ma'lumotlarni bantlarga (vertikal yoki gorizontal) yoki to'rtburchaklar qismlarga (bloklar) ajratishdir.

Matritsani tarmoqli ajratish. Ip (blokli chiziqli) bo'linish chog'ida har bir oqim matritsaning bir yoki bir nechta qatorini (rowwise yoki gorizontal ajratish) yoki ustunlar (ustunli yoki vertikal bo'linish) ajratilgan. Satrlar va ustunlarni chiziqlar sifatida ajratish ko'p holatlarda doimiy (ketma-ket) asosda amalga oshiriladi.

Matritsani bloklar bo’yicha bo’lish. Shaxmat taxtasi blokida matritsa elementlarning to'rtburchak shakllariga bo'linadi - bu holatda, odatda, bo'linma doimo qo'llaniladi. Oqimlarning soni p=s*q bo'lsin, matris satrlari soni

s ning ko'pligi va ustunlar soni q ning ko'pligi, ya'ni m = k*s va n = l*q

Ushbu yondashuv yordamida hisoblash tizimining topologiyasi (hech bo'lmaganda mantiqiy darajada) s-satr va q ustunlari panjarasi shaklida foydalidir. Bu holatda, ma'lumot uzluksiz ajratilganda ko'p holatlarda hisob-kitoblar qobiq tuzilishiga yaqin hisoblash elementlari asl matritsiyaning bitişik bloklarini ishlov berishga imkon beradi. Shuni ta'kidlash kerakki, satr va ustunlar davriy o'zgarishi blok tizimiga ham qo'llanilishi mumkin.

Nazorat savollari

1. Vektorni matritsaga ko’paytirishda parallelashtirish usullarini abzalliklari
2. Vektorni matritsaga ko’paytirishni parallel usullari
3. Vektorni matritsaga ko’paytirishni C++ tilida algoritmini yozish
4. OpenMP texnologiyasidan foydalangan holda vektorni matritsaga ko’paytirishni parallel dasturlarini tuzish
5. Vektorni matritsaga ko’paytirishni ketma-ket va parallel dasturlarini tuzish va ularni bir-biri bilan solishtirish

15-mavzu: Ko’p yadroli prosessorlar, xotirani va ichki kommunakatsiyani tashkillashtirish, dasturiy oqimlar, oqimli qayta ishlash texnologiyasi

REJA:

15.1. Ko’p yadroli protsessorlar

15.2. Xotirani va ichki kommunakatsiyani tashkillashtirish

15.3. Dasturiy oqimlar

15.4. Oqimli qayta ishlash texnologiyasi

Ba'zida parallel dasturlash faqat superkompyuterlar uchun katta qiziqishlarga ega bo'lganlar uchun edi. Ammo hozirda, odatiy dasturlar ko'p yadroli protsessorlarda ishlay boshlaganda, parallel dasturlash tezda har qanday professional

dasturiy ta'minotni ishlab chiqaruvchi usta bo'lishi va foydalanishi mumkin bo'lgan texnologiyaga aylanadi.

Parallel dasturlash qiyin bo'lishi mumkin, ammo uni "qiyin" deb hisoblashni bilish osonroq, ammo oddiygina "bir oz boshqacha". Bu an'anaviy, ketma-ket dasturlashning barcha xususiyatlarini o'z ichiga oladi, ammo parallel dasturlashda uch qo'shimcha, aniq belgilangan qadam mavjud.

Parallelizm ta'rifi: bir vaqtning o'zida bajarilishi mumkin bo'lgan subtaskalarni aniqlash uchun muammoni tahlil qilish

O'zaro muvozanatni aniqlash: vazifaning strukturasini o'zgarishlarni samarali bajarish uchun o'zgartirish. Bu ko'pincha subtasklar bilan bog'liqliklarni topish va manba kodini tashkil qilishni talab qiladi, ular samarali tarzda boshqarilishi mumkin.

Parallelizmning ifodasi: parallel dasturiy belgilar yordamida parallel algoritmni manba kodini qo'llash

Ushbu bosqichlarning har biri o'z yo'lida muhimdir. Ularning dastlabki ikkitasi parallel dasturlashdagi tizimli naqshlar haqidagi so'nggi kitobda batafsil tavsiflangan [mattson05]. Ushbu maqolada biz uchinchi bosqichni ko'rib chiqamiz: parallel dasturiy belgilar tizimidan foydalanib, manba kodidagi parallel algoritmni amalga oshirish. Bunday yozuv parallel dasturlash tili, kutubxona interfeysi yordamida amalga oshiriladigan dasturiy dasturiy interfeysi (API) yoki mavjud ketma-ketlikdagi dasturlash tiliga qo'shilgan kengaytma bo'lishi mumkin.

Parallel dasturlash uchun ma'lum bir belgi tanlash juda qiyin vazifa bo'lishi mumkin. Ushbu eslatma tizimlarida vaqtni talab etuvchi keskin o'rganish egri mavjud. Shu sababli, eng munosibni tanlash uchun bir nechta eslatish tizimlarini o'rganish mantiqan yaramaydi. Dasturchilarga kerak bo'lgan barcha narsalar - bu turli eslatmalarni tezda "teginish" va ularning asosiy xususiyatlari bilan batafsilroq tanishish uchun ma'lum bir tizimni ongli ravishda tanlab olish uchun etarli darajada tanishish qobiliyati.

Ushbu maqola bir nechta parallel dasturiy belgilar tizimini umumiy tavsifi bilan ta'minlaydi, ulardan foydalanishning asosiy usullarini, shuningdek, ularning kuchli

va zaif tomonlarini batafsil tavsiflaydi. Xususan, quyidagi markalash tizimlari ko'rib chiqiladi:

OpenMP: oddiy parallel dasturlash uchun kompilyator direktifi

MPI: kutubxonaning yuqori samarali taşınabilirliğini amalga oshirish uchun muntazam

Java: etakchi dasturiy tilida bir vaqtda kelishuv

Muhokamani iloji boricha aniqroq qilish uchun har bir tanlov uchun taniqli dasturning parallel versiyasi taqdim etiladi. Bu to'rtburchak formulasidan foydalanib odatiy raqamli integratsiya va integral funktsiyasi va integral chegaralari tanlanadi, shunda 'n' raqami matematik jihatdan to'g'ri natija hisoblanadi. Ushbu vazifa parallel dasturlashda "salom dunyosi" dasturining analogiyasidir. Maqolaning oxirida ish va o'qish uchun parallel dasturiy ko'rsatmani tanlash bo'yicha qisqacha tushuntirish berilgan.

OpenMP [omp] - umumiy xotira kompyuterlari uchun parallel dasturlarni yaratish uchun sanoat standarti bo'lgan API. OpenMP ning asosiy maqsadi - aylanish yo'naltirilgan dasturlarni yozishni osonlashtirishdir. Bunday dasturlar tez-tez yuqori samarali hisoblash uchun yaratiladi. Bundan tashqari, komponentlar OpenMP tarkibiga SPMD, "master and workflow", quvur liniyasi kabi parallel usullarini qo'llab-quvvatlash uchun kiritilgan.

OpenMP juda muvaffaqiyatli parallel dasturlash tiliga aylandi. U bozorga kiradigan har bir xotira almashadigan kompyuterda mavjud. Bundan tashqari, Intel yaqinda Klasterlarni qo'llab-quvvatlash uchun OpenMP versiyasini yaratdi. OpenMP parallelizmin mavjud ketma-ketlik dasturi parallel holga kelgunicha asta-sekin qo'shilgan dasturlash uslubini qo'llab-quvvatlaydi. Biroq, bu afzallik OpenMP ning eng zaif nuqtasidir. Agar muvozanat asta-sekin qo'shilsa, dasturchi dasturni keng miqyosda qayta qurish qila olmaydi, bu ko'pincha maksimal ishlash uchun zarur bo'ladi.

OpenMP doimiy rivojlanayotgan standartdir. OpenMP Architecture Review Board deb nomlangan sanoat guruhi ushbu tilga yangi kengaytmalar kiritish uchun muntazam uchrashuvlar o'tkazadi. OpenMP ning keyingi versiyasi (3.0 versiyasi)

vazifa navbatini tashkil etish qobiliyatini o'z ichiga oladi. Bu esa, OpenMP-ga yanada keng boshqaruv tuzilmalarini boshqarish va umumiy umumiy recursiv algoritmlardan foydalanish imkonini beradi.

Nazorat savollari

1. Ko’p yadroli protsessorlar
2. Dasturiy oqimlar va oqimlarni qayta ishlash texnologiyasi
3. OpenMP texnologiyasidan foydalanish

16 -mavzu: Oqimli qaya ishlashning standart vositalari, API standarti, OpenMP universal standarti, SSE kengaytirish buyruqlari, Intel ning ko’p yadroli prosessorlari uchun oqimli qayta ishlash

REJA:

16.1. Oqimli qayta ishlashning standart vositlari

16.2. API standari

16.3. OpenMP universal standarti

16.4. SSE kengaytirish buyruqlari

16.5. Intelning ko’p yadroli prosessorlari uchun oqimli qayta ishlash

Kompyuterning operatsion tizimining (OS) ko'rinishi bitta dasturli rejimdan ko'p dasturli (ko'p dasturli) ish rejimiga o'tishga imkon berdi. Operatsion tizim odatda bir nechta vazifani bir vaqtning o'zida bajarayotganiga ishonib, ko'p ishlarni talab qiladi. C # dasturchisi nuqtai nazaridan OS uchun vazifa - bu dastur yoki loyiha. Turli xil operatsion tizimlar bir xil yoki o'xshash tushunchalar uchun turli atamalardan foydalanadi. Bundan tashqari, OS haqida gapirganda, Windows operatsion tizimini yodda tutamiz va biz bu OS operatsion terminologiyasini ishlatamiz.

Ilovamizning har bir bajarilgan loyihasi uchun operatsion tizim bir jarayonni yaratadi. Kompyuter har doim ishlayotgan vaqtda, OS turli jarayonlar bilan ishlaydi,

ularning aksariyati rasmiy hisoblanadi. Antivirus dasturlari kabi ushbu jarayonlarning ba'zilari kompyuterimda doimo mavjud bo'lib, kompyuter yoqilganda boshlanadi.

OpenMP dasturida parallel dastur direktivalar yordamida maxsus tayinlangan dastur - parallel qismlar - dasturiy kodni bir necha alohida buyruqlar oqimlariga (iplar) bo'linishi mumkin bo'lgan dastur hisoblanadi. Umuman olganda, dastur dastur kodining ketma-ket (bir nusxadagi) va parallel (ko'p qismli) qismlari majmuasi sifatida ifodalanadi (4.2-rasmga qarang).

Shuni ta'kidlash joizki, threadlar orasidagi hisob-kitoblarni taqsimlash tegishli OpenMP direktivalari tomonidan boshqariladi. Hisoblagich yukini muvozanatlashning (yukni muvozanatlash) muntazam taqsimoti - parallel dasturni bajarishning mumkin bo'lgan maksimal tezlashuvini olish uchun juda muhim ahamiyatga ega.

Mavzular turli xil protsessorlarda (protsessor yadrosi) bajarilishi mumkin yoki bitta hisoblash elementida bajarilishi uchun guruhlangan bo'lishi mumkin (bu holda ular vaqtni almashish rejimida amalga oshiriladi). Cheklash jarayonida parallel dasturni amalga oshirish uchun bitta protsessor qo'llanilishi mumkin - odatda, bu usul parallel dasturning to'g'riligini dastlabki tekshirish uchun ishlatiladi.

Iplar soni parallel dastur qismlarini bajarish boshida aniqlanadi va odatda tizimdagi mavjud hisoblash elementlarining soniga to'g'ri keladi; Yaratilgan iplar sonini o'zgartirish turli OpenMP vositalaridan foydalangan holda amalga oshirilishi mumkin. Dasturning parallel qismlarida barcha oqimlar navbat bilan 0 dan np - 1 ga qayta numaralandırılır, bu erda np, jami oqim soni. Oqimning raqamini OpenMP funktsiyasidan foydalanib ham olish mumkin.

Parallelizmni tashkil qilish uchun OpenMP ishlarini qo'llash ko'p protsessorli hisoblash tizimlarining afzalliklarini umumiy xotira bilan birgalikda ko'rib chiqishga imkon beradi. Avvalo, bir xil parallel dasturning iplari umumiy manzillar maydonida ishlaydi, bu parallel holda ishlaydigan ish zarrachalar uchun umumiy ma'lumotlardan foydalanishga imkon beradi (tarqalgan xotirali tizimlar uchun MPI texnologiyasidagi jarayonlardan farqli o'laroq). Bundan tashqari, ishlarni boshqarish

(yaratish, to'xtatib qo'yish, faollashtirish, tugatish) jarayonlarga nisbatan OS uchun kamroq ish haqi talab qiladi.

Yuqorida ta'kidlab o'tilganidek, iplar parallel dasturning umumiy manzil maydonida amalga oshiriladi. Natijada, parallel oqimlarning o'zaro ta'siri barcha oqimlar uchun mavjud bo'lgan umumiy ma'lumotlardan foydalanish orqali tashkil qilinishi mumkin. Oddiy vaziyat - faqat o'qiladigan ma'lumotlardan foydalanish. Umumiy ma'lumotlar bir nechta oqimlarda o'zgarishi mumkin bo'lgan hollarda, to'g'ri o'zaro munosabatlarni tartibga solish uchun maxsus harakatlar talab etiladi. Aslida, ikkita iplar bir xil dastur kodini bajarishi kerak

n = n + 1;

umumiy o'zgaruvchan n uchun. Keyinchalik, ijro etilish sharoitlariga qarab, bu operatsiyani (bu to'g'ri natijaga erishishga olib keladigan) amalga oshirish mumkin yoki har ikkala oqim bir vaqtning o'zida o'zgarmaydigan n qiymatini o'qishi mumkin, bir vaqtning o'zida bu o'zgaruvchiga yangi qiymatni oshiradi va yozadi (natijada noto'g'ri qiymat olinadi). Shunga o'xshash vaziyat, agar hisob-kitoblar natijasi oqimlarning bajarilishiga bog'liq bo'lsa, irqi shartlarini (poyga sharoitini) oladi. Rassomni istisno qilish uchun umumiy o'zgaruvchan qiymatlardagi o'zgarishlarni bir vaqtning o'zida bitta bitta oqim bilan amalga oshirilishini ta'minlash kerak - ya'ni umumiy ma'lumot bilan ishlashda oqimlarni o'zaro ajratib olishni ta'minlash kerak. OpenMPda ajralmas (atom) operatsiyalari, tanqidiy bo'linishlar mexanizmi (kritik bo'limlar) yoki semafor qulflarning maxsus turi (qulflar) yordamida o'zaro tenglashtirilishi mumkin.

Shuni ta'kidlash kerakki, o'zaro mutlaqo tashkillashni tashkil etish masalalarni parallel ravishda amalga oshirish imkoniyatini kamaytiradi - umumiy o'zgaruvchanlarga bir vaqtning o'zida kirish imkoniyatini beradi, ulardan faqat bittasi ishlashni davom ettirishi mumkin, boshqa barcha masalalar bloklanadi va umumiy ma'lumotlar tarqalishini kutadi. Ta'kidlash joizki, umumiy ma'lumot bilan ishlashda o'zaro hisobdan chiqarishni tashkil etish majburiy hisoblanadi, biroq ish zarrachalarining paydo bo'lish kechikishi (tiqilib qolishi) o'z vaqtida kam bo'lishi kerak.

OpenMP tuzilishi:

- Direktivlar
- Funksiyalar kutubxonasi
- Bir qator o'zgaruvchilari.

Ushbu tartibda OpenMP texnologiyasining imkoniyatlari ko'rib chiqiladi.

Ushbu standart C90, C99, C ++, Fortran 77, Fortran 90 va Fortran 95 algoritmlari uchun OpenMP foydalanishni ta'minlaydi. OpenMP direktivasining formatini va barcha dasturlarning misollaridan C da taqdim etiladi; Fortran tili uchun OpenMP texnologiyasidan foydalanish xususiyatlari 5.8.1-bandda keltirilgan. Eng umumiy shaklida OpenMP direktivasining formati quyidagicha ifodalanishi mumkin:

#pragma omp <directive_name> [<parametr> [[,] <parametr>] ...]

Direktivning boshlang'ich qismi (#pragma omp) sobit bo'ladi, direktivaning turi uning nomi (Direktiv_yeni) bilan belgilanadi, har bir direktiv o'zboshimchalik bilan parametrlar soni bilan birga bo'lishi mumkin (ingliz tilida, OpenMP direktivasining parametrlari atamalar jumlasidan foydalanadi).

Misol uchun, biz bir ko'rsatmaga misol keltiramiz:

#pragma omp parallel default (shared)
\ Shaxsiy (beta, pi)

Misol, ko'rsatmalarni o'rnatish uchun dasturning bir nechta yo'nalishidan foydalanish mumkinligini ko'rsatadi - davom etish mavjudligini belgisi "\" teskarisi.

Birinchi parallel dastur misoli

Juda muhim nuqta ta'kidlab o'taylik - shuni anglash mumkinki, OpenMP texnologiyasining imkoniyatlarini qisqacha ko'rib chiqish oddiy, ammo parallel dasturlarni ishlab chiqish uchun etarli. Yangi dasturlash tillarini - "Salom Dunyo"

degan salom yo'lini chiqadigan dasturni ishlab chiqishda dastlab amalda standart dastur ishlab chiqaylik. Shunday qilib:

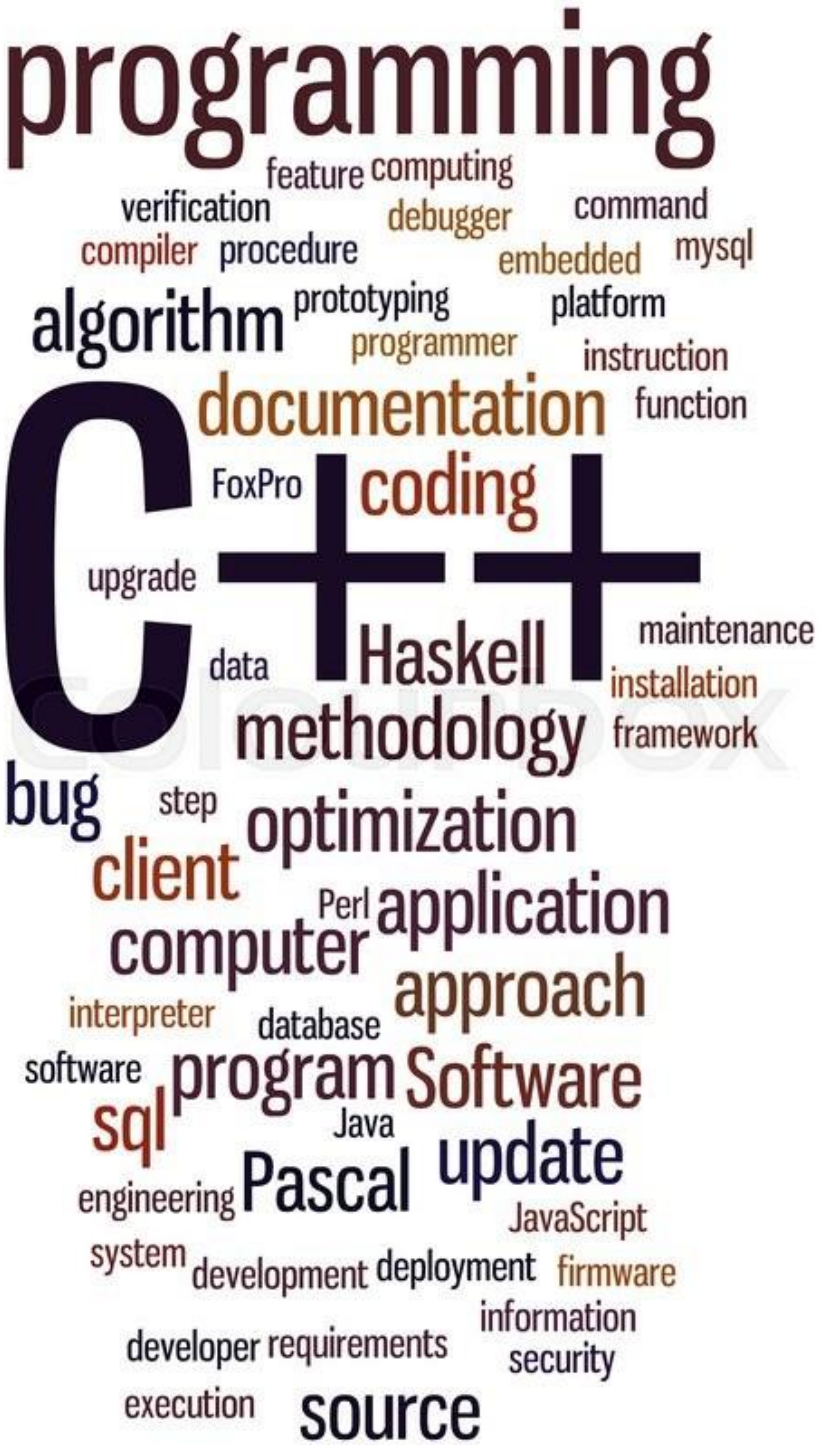
```
#include <omp.h>
main () {
  / * Parallel parchani tanlash * /
#pragma omp parallel
  {printf ("Salom Dunyo! \ n");
  } / * Parallel parchani * /
}
```

Nazorat savollari

1. C++ kompilyatoriga OpenMP ni qo’shish
2. OpenMP dan foydalanib C++ tilida parallel dastur tuzish
3. OpenMP direktivalarini ishlatish
4. Vektor elementlari orasida minimal (maksimal) qiymatni topish uchun dasturni ishlab chiqish.
5. Ikki vektorning skaler mahsulotini hisoblash uchun dastur ishlab chiqish.
6. To'g'ri to'rtburchaklar usulini qo'llash orqali muayyan integralni hisoblash muammosiga dastur ishlab chiqish

«PARALLEL KOMPYUTERLARNING ARXITEKTURASI VA DASTURLASH» FANIDAN

AMALIY MASHG'ULOTLAR



1-Amaliy ish: Parallel xisoblash tahlili va modellashtirish

REJA:

1. Parallel xisoblash tizimlari
2. Parallel dasturlash tillari va texnologiyalari
3. Parallel dasturlar yaratish

Parallel hisoblash dasturlari kompyuter dasturlarini tashkil qilishning bir usuli bo'lib, dasturlarda parallel ishlaydigan hisoblash jarayonlari majmuasi sifatida ishlab chiqiladi (bir vaqtning o'zida). Bu atamalar dasturiy ta'minotning bir vaqtning o'zida bir qator masalalarni qamrab oladi, shuningdek, samarali apparatni qo'llashni yaratadi. Parallel hisoblash nazariyasi algoritmlarning amaliy nazariyasining bir qismi.

Parallel hisoblashni amalga oshirishning turli usullari mavjud. Masalan, har bir hisoblash jarayoni operatsion tizim jarayoni sifatida amalga oshirilishi mumkin yoki hisoblash jarayonlari bir operatsion jarayonida bajariladigan ish zarrachalar majmui bo'lishi mumkin. Parallel dasturlarni har bir hisoblash jarayonini amalga oshirish uchun yoki parallel ravishda har bir hisoblash jarayonida bir yoki bir nechta protsessorni (yaqinda joylashgan yoki kompyuter tarmog'iga bo'linadi) ajratish bilan bir qatorda birma-bir protsessorda jismonan bajarilishi mumkin.

Parallel dasturlarni ishlab chiqishda asosiy qiyinchilik turli hisoblash jarayonlari o'rtasidagi o'zaro ta'sirlarning to'g'ri ketma-ketligini ta'minlash, shuningdek, jarayonlarning o'zaro taqsimlangan resurslarini muvofiqlashtirishdir.

Ayrim parallel dasturiy tizimlarida komponentlar orasidagi ma'lumotlarni uzatuvchi dasturchidan yashiriladi (masalan, so'z mexanizmidan foydalaniladi), boshqalarida esa aniq ko'rsatilishi kerak. Ochiq shovqinlarni ikki turga bo'lish mumkin:

Umumiy xotirasi orqali o'zaro bog'liqlik: ko'p protsessorli tizimning har bir protsessorida bitta jarayonga tegishli ijro etiladigan ish zarrachasi ishga tushiriladi. Ushbu jarayon uchun umumiy xotira joyida ma'lumotlar almashinuvini uzatadi [2]. Iplar soni protsessorlarning soniga mos keladi. Mavzular til vositalari (masalan, Java yoki C #, C ++ (C ++ 11) dan boshlab), C (C11dan boshlab)) yoki aniq kutubxonalar

(masalan, C / C ++ da PThreads yordamida) yoki deklarativ ravishda (Masalan, OpenMP kutubxonasidan foydalanib) yoki avtomatik ravishda o'rnatilgan kompilyator vositalari (masalan, Fortranning yuqori ishlashi). Ushbu turdagi parallel dasturiy ta'minot odatda bir-birlari orasidagi oqimni muvofiqlashtirish uchun boshqarishning ba'zi shakllarini (mutexes, semaforlar, monitorlar) talab qiladi.

Xabar yozish yordami bilan o'zaro ta'siri: ko'p protsessorli tizimning har bir protsessorida xabarlarni ishlatadigan boshqa protsessorlar bilan ishlaydigan boshqa jarayonlar bilan bog'laydigan yagona bitikli jarayon boshlanadi. Jarayonlar operatsion tizimning tegishli funktsiyasini va kutubxonani (masalan, MPI protokolini qo'llash) foydalanib, yoki til vositalarini (masalan, High Performance Fortran, Erlang yoki occam) foydalanib yozish orqali aniq tarzda yaratiladi. Xabarlar doimo mos kelmaydigan shaklda yoki jo'natuvchiga xabar yetkazilguniga qadar taqiqlanadigan topiladigan usul yordamida foydalanish mumkin. Asenkron xabarlar ishonchli bo'lishi mumkin (kafolatlangan etkazib berish bilan) yoki ishonchsiz [3].

Parallel xabarlar tizimlari umumiy xotira tizimlaridan ko'ra ko'proq tushunishadi va ko'pincha ilgari parallel dasturlash usuli hisoblanadi. Xat yozish tizimlarini o'rganish va tahlil qilish uchun matematik nazariyalarning keng tanlovi mavjud, shu jumladan aktyorlar modeli va jarayonlarning har xil turlari. Xabarlar nosimmetrik ko'p protsessorlarda birgalikda izchil xotirasi bo'lgan yoki bo'lmagan holda samarali tarzda amalga oshirilishi mumkin.

Tarqatilayotgan xotira va xabarlarni yuborish bilan parallellik turli ishlash ko'rsatkichlariga ega. Odatda (lekin har doim ham emas), jarayonlar uchun xotira miqdori va xabarlarni uzatuvchi tizimlar uchun vazifalarni almashtirish muddati past bo'ladi, lekin xabarlarni yuborish jarayoni protsedura chaqiruvlaridan ko'ra qimmatroq. Bu farqlar ko'pincha ishlashga ta'sir qiluvchi boshqa omillar bilan bir-biriga o'xshash.

Gibrid usul: tarqatilgan xotirali (DM-MIMD) ko'p protsessorli tizimlarda, bu tizimning har bir tugunini umumiy xotira protsessori (SM-MIMD) bo'lsa, siz gibrid dasturiy usulini qo'llashingiz mumkin [4]. Tizimning har bir tugunida, bu tugun protsessorlari orasidagi iplarni tarqatadigan ko'p tarmoqli jarayon boshlanadi. Bir

tugundagi iplar o'rtasidagi ma'lumot almashinish umumiy xotira orqali amalga oshiriladi va tugunlar orasidagi ma'lumotlar almashinish xabarlar orqali amalga oshiriladi. Bunday holda, jarayonlarning soni tugunlar soniga va har bir tugundagi protsessorlarning soni bo'yicha iplar soniga bog'liq. Gibrid dasturlash usuli yanada murakkab (parallel dasturni maxsus tarzda qayta yozish kerak), biroq juda protsessorli tizimning har bir tugunining apparat-resurslarini ishlatishda eng samarali hisoblanadi.

Albatta, bunday tizimda siz faqat xabarlarni uzatish usulini, ya'ni har bir tugunning har bir protsessorida alohida jarayonni ishlatishingiz mumkin. Bunday holda, jarayonlarning soni (va iplar) barcha tugunlarda protsessorlarning soniga teng bo'ladi. Ushbu usul soddalashtirilgan (parallel dasturda, faqat jarayonlarning sonini oshirish kerak), lekin u samarasiz, chunki bir xil tugun protsessorlari bir-biri bilan xabar almashishadi, xuddi ular turli mashinalardagidek.

Xulosa

1. Parallel kompyuterlardan foydalanib katta hajmdagi masalalarni yechish xisoblash vaqtini qisqartiradi
2. Parallel kompyuterlar uchun parallel dasturlash tillaridan yoki maxsus texnologiyalardan foydalanishga to’g’ri keladi

2-Amaliy ish: Parallelashtirish usullarini yaratish asoslari

REJA:

2.1. Parallel dasturlash tillari va texnologiyalarini yaratilishi

2.2. OpenMP texnologiyasi

2.3. OpenMP tuzilishi (direktivalar, funksiyalar, o’zgaruvchilar)

Ko'p yillar mobaynida protsessorlarning ish faoliyatini oshirishning asosiy usuli bo'lgan "megahertz poygasi" yadro sonini ko'paytirish tendentsiyasiga aylandi. Yaxshiyamki, protsessor ishlab chiqaruvchilari bir nechta protsessorlarni bitta chipga qanday qilib qo'yish kerakligini o'rgandi. Endi deyarli har bir kompyuterda

ko'p yadroli protsessor o'rnatilgan. Hatto kirish stoli ish stoli tizimlarida ikkita yadro mavjud - to'rt va sakkizta yadroli tizimlar mavjud. Agar Mur qonuni kuchini yo'qotmasa, 5 yil ichida o'rtacha kompyuter chip ustida 16 yoki hatto 32 yadroli bo'ladi.

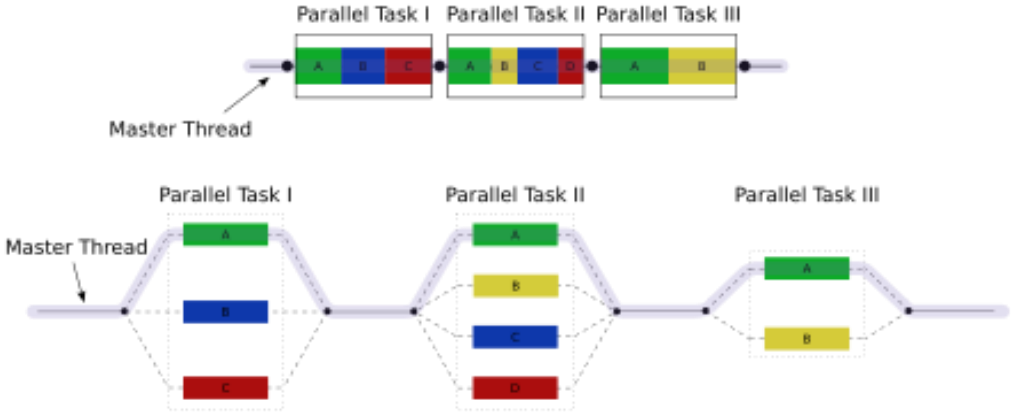
Muammo shundaki, dasturiy ta'minot sohasi apparatni nazorat qila olmaydi va dasturlarning faqat bir qismi ko'p yadroli protsessorlarning resurslaridan samarali foydalanishi mumkin. Har bir dasturda bir asosiy ijro etuvchi bor. Bu ketma-ket ketma-ket bir-birini ketma-ket bajaradigan ko'rsatmalar majmui. Tabiiyki, bu holda protsessorning bir yadrosi shug'ullanadi. Dasturchi qolgan yadrolarni ish bilan ishlashga g'amxo'rlik qilishi kerak, boshqacha qilib aytganda, ba'zi ko'rsatmalar ketma-ket bajarilmasligini, bir vaqtda parallel rejimda bajarilishini ta'minlashi kerak.

Ta'kidlash joizki, bu ko'rsatkich yadro sonining ko'payishi bilan linear ravishda oshmaydi. Ya'ni, to'rtta yadrodan foydalanish hosildorlikning to'rt barobar ko'payishini kafolatlamaydi. Shunga qaramay, o'sish bor va har yili kuchli bo'ladi - ko'p yadroli protsessorlar uchun optimallashtirilgan dasturlar paydo bo'ladi.

Protsessorning to'liq quvvatini ishlatish uchun dasturchilar threadlarni qanday qilib boshqarishi mumkin? Dasturni iloji boricha tezroq bajarish, yadro sonini ko'paytirish bilan o'lchab qo'yish va bunday dasturni yozish dasturchining kabusu emasmi? Bittasi dastur kodida ish zarrachalaridan tuzish, bajarish uchun vazifalarni bajarish, so'ng ularni o'chirishdir. Ammo bu holda, siz juda muhim narsa - sinxronizatsiya haqida g'amxo'rlik qilishingiz kerak. Agar bitta vazifa boshqa topshiriq bilan hisoblangan ma'lumotlarga muhtoj bo'lsa, vaziyat yanada murakkablashadi. Turli bir vaqtning o'zida umumiy o'zgaruvchan qiymatlarni o'zgartirmoqchi bo'lganda nima sodir bo'lishini tushunish qiyin. Ha, men qo'l bilan ish zarrachalari yaratmoqchi emasman va vazifalarni ularga topshirishni xohlamayman. Parallel dasturlash uchun turli kutubxonalar va standartlar qutqarish uchun keladi. C, C ++, Fortran-OpenMP dasturlarini parallellashtirish uchun eng keng tarqalgan standartni ko'rib chiqaylik.

OpenMP - umumiy xotira bilan birga ko'p protsessorli tizimlarda multi-threadli dasturlar dasturlash uchun mo'ljallangan API. OpenMP spesifikatsiyasi bir necha yirik kompyuter uskunalari va dasturiy ta'minot ishlab chiqaruvchilari tomonidan ishlab chiqilgan. OpenMP asosiy kompilyatorlari tomonidan qo'llab-quvvatlanadi.

OpenMP-da siz koddagi oqimlarni "ko'rmaysiz". Buning o'rniga, kompilyatorga #pragma ko'rsatmalariga kod blokini parallel holga keltirishi mumkinligini aytasiz. Ushbu ma'lumotni bilgan holda, kompilyator parallel parol bloklari uchun boshqa bir nechta iplarni yaratadigan bitta asosiy threaddan iborat bo'lgan dasturni ishlab chiqishi mumkin. Ushbu iplar parallel bloklar blokining oxirida sinxronlashtiriladi va biz yana bir xil asosiy threadka qaytamiz.



OpenMP #pragma tomonidan nazorat qilinganligi sababli, C ++ kodi har qanday C ++ kompilyatori tomonidan to'g'ri tuzilgan, chunki qo'llab-quvvatlanmaydigan #pragma e'tiborga olinmasligi kerak. Biroq, OpenMP API-da, bir nechta vazifani o'z ichiga oladi va ulardan foydalanish uchun nom faylini kiritishingiz kerak. Derivatning OpenMP-ni qo'llab-quvvatlab turishini aniqlashning eng oson yo'li omp.h faylini kiritishga urinishdir: #include <omp.h>

Agar OpenMP qo'llab-quvvatlanadigan bo'lsa, uni maxsus kompilyatorlar yordamida yoqishga to’g’ri keladi:

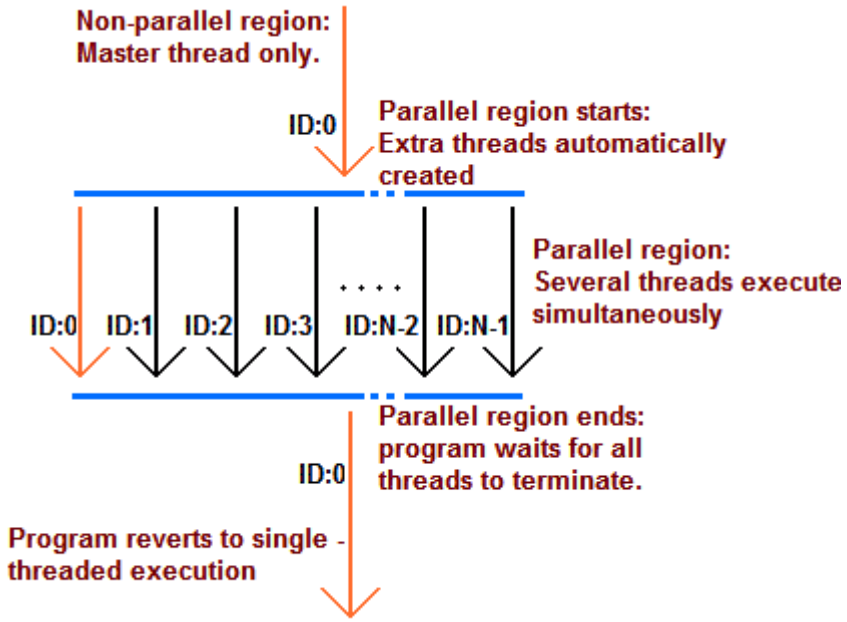
```
gcc -fopenmp
Intel   -openmp (Linux, MacOSX), -Qopenmp (Windows)
```

Microsoft -openmp (Настройки проекта в Visual Studio)

Parallel direktivasi. Ushbu direktiva N guruhining bir guruhini yaratadi. N ish vaqti bilan belgilanadi, odatda protsessor yadrolari soni, lekin siz N qo'lda ham o'rnatishingiz mumkin. Guruhdagi barcha iplar buyruqni bajaradi (yoki {} -shells da belgilangan buyruqlar bloklari). Amalga oshirilgandan keyin, iplar biriga "birlashtiriladi".

```
#pragma omp parallel  {
    // Blok ichidagi kod parallel bajariladi
    printf("Hello!n");
  }
```

Misol, "Hello!" Matnini ko'rsatib turadi. Guruhda hosil bo'lgan ish zarralari qatorida qatorli tanaffus bilan. Ikki yadroli tizimlar uchun matn ikki marta chop etiladi. (Eslatma: "HeHllellolo" kabi bir narsa ko'rsatilishi mumkin, chunki chiqish parallel ravishda sodir bo'ladi.)



Qanday ishlayotganini ko'rib chiqsangiz, GCC maxsus funktsiyani yaratadi va blok kodini bu funktsiyaga o'tkazadi, shuning uchun blok ichidagi barcha o'zgaruvchilar funktsiyaning lokal o'zgaruvchilari (har bir oqimning mahalliy o'zgaruvchilari) bo'ladi. Boshqa tomondan, ICC fork () ga o'xshash mexanizmdan

foydalanadi va maxsus funksiya yaratmaydi. Har ikki dastur ham, albatta, to'g'ri va semantik jihatdan bir xildir.

Agarda `if` dan foydalanilsa, parallelizm shartli bo’lishi mumkin:

extern int parallelism_enabled;

```
#pragma omp parallel for if(parallelism_enabled)
for(int c=0; c<n; ++c)
  handle(c);
```

ushbu holatda parallelism_enabled 0 ga teng va sikl bir marta bajariladi

`for` direktivasi `for` siklini bir nechta oqimlarga ajiratadi:

```
#pragma omp for
 for(int n=0; n<10; ++n)
 {
     printf(" %d", n);
 }
 printf(".n");
```

Ushbu tsikl 0 dan 9 gacha bo'lgan sonlarni aniq bir marta chiqaradi. Biroq, ularni olib chiqish tartibi noma'lum. Masalan, bunday bo'lishi mumkin: 0 5 6 7 1 8 2 3 4 9

Xulosa

1. OpenMP texnologiyasi umumiy xotirali kompyuterlar uchun parallel dasturlar tuzishda samarali texnologiyalardan biri xisoblanadi
2. OpenMP Fortran, C va C++ dasturlash tillari uchun ishlab chiqilgan
3. OpenMP ni C++ dasturlash tilida ishlatish uchun kompilyatorga OpenMP ni qo’shish kerak
4. OpenMP da barcha direktivalar `#pragma omp` kalit so’zidan keyin yoziladi
5. OpenMP da dastur kodi parallel va ketma-ket bo’limlardan turadi.

3-Amaliy ish: Vektorni matritsaga ko’paytirishning parallel usullari

REJA:

3.1. Vektorni matritsaga ko’paytirish algoritmi

3.2. Vektorni matritsaga ko’paytirishning parallel usullari

3.3. OpenMP dan foydalanib vektorni matritsaga ko’paytiruvchi parallel dasturlar yaratish

Matritsa operatsiyalari keng ko`lamli jarayonlarni, hodisalar va tizimlarni matematik modellashtirishda keng ishlatiladi. Matritsa hisob-kitoblari ko'plab ilmiy va muhandislik hisob-kitoblarining asoslarini tashkil etadi - dasturlar, kompyuter matematikasi, fizika, iqtisod va hokazo sohalar orasida.

Matritsalarni hisob-kitoblarni samarali bajarish muhimligini hisobga olgan holda, ko'plab standart dastur kutubxonalari turli matritsali operatsiyalar uchun protseduralarni o'z ichiga oladi. Matritsalarni qayta ishlash uchun dasturiy ta'minot hajmi muntazam ortib bormoqda - maxsus matritsa turlarini (uchburchak, lenta, siyrak va boshqalar) yangi iqtisodiy saqlash tuzilmalari ishlab chiqilmoqda, algoritmlarning har xil yuqori performansli mashinalarga bog'liqligi yaratilmoqda, nazariy tadqiqotlar olib borilmoqda. tezroq matritsa hisoblash usullarini topish.

Matematik hisob-kitoblarga ko'ra, parallel hisoblashning klassik maydoni qo'llaniladi. Bir tomondan, yuqori samarali multi-protsessorli tizimlardan foydalanish, echilishi kerak bo'lgan vazifalarning murakkabligini ancha oshirishi mumkin. Boshqa tarafdan, uning oddiy formulasi tufayli matris operatsiyalari parallel dasturlashning ko'plab usullari va usullarini namoyish qilish uchun ajoyib imkoniyat yaratadi.

Ushbu bobda matritsa-vektorning ko'payishi uchun parallel hisoblash usullari ko'rib chiqiladi, keyingi bobda matritsalarni ko'paytirishning ishlashini ko'rib chiqamiz. Matritsalarni hisoblashning muhim turi - linear tenglamalar tizimlarini echish - 8-bobda keltirilgan. Yuqorida sanab o'tilgan barcha muammolar bo'yicha umumlashtirilgan matritsalarni bir vaqtning o'zida ajratish bilan ajratish muammosi Bo'lim 6.2da muhokama qilinadi.

Quyidagi materialni taqdim etayotganda, ko'rib chiqilayotgan matrislar matritsa elementlarining umumiy soniga nisbatan nol elementlarning soni ahamiyatsiz bo'lgan zich ekanligini taxmin qilamiz.

Matritsalarni hisoblashning ko'plab usullari uchun matritsaning turli elementlari uchun bir xil hisoblash harakatlarini takrorlash xarakterlidir. Ushbu moment matritsa hisob-kitoblarini bajarishda ma'lumotlardagi parallellik mavjudligini ko'rsatadi va natijada matritsa operatsiyalarining parallelizatsiyasi ko'p hollarda iplar orasidagi qayta ishlangan matritsalarni ajratish uchun kamayadi. Matritsani ajratish usulini tanlash parallel hisoblashning muayyan usulini aniqlashga olib keladi; turli ma'lumotlar tarqatish sxemalari mavjudligi matris hisob-kitoblari uchun bir qator parallel algoritmlarni hosil qiladi.

Matritsalarni ajratish uchun eng keng tarqalgan va keng tarqalgan usullar ma'lumotlarni bantlarga (vertikal yoki gorizontal) yoki to'rtburchaklar qismlarga (bloklar) ajratishdir.

Matritsani tarmoqli ajratish. Ip (blokli chiziqli) bo'linish chog'ida har bir oqim matritsaning bir yoki bir nechta qatorini (rowwise yoki gorizontal ajratish) yoki ustunlar (ustunli yoki vertikal bo'linish) ajratilgan. Satrlar va ustunlarni chiziqlar sifatida ajratish ko'p holatlarda doimiy (ketma-ket) asosda amalga oshiriladi.

Vektorni matritsaga ko’paytirishning C++ tilida quyidagicha yoziladi: (ketma-ket algoritmi)

```
for (i = 0; i < m; i++)
{
   c[i] = 0;

   for (j = 0; j < n; j++)
   {
         c[i] += A[i][j]*b[j]
   }
}
```

Vektorni matritsaga gorizantal lenta (satr) bo’ylab ko’paytiruvchi C++ tilidagi funksiya quyidagicha bo’ladi:

```
ParallelResultCalculation(double* pMatrix, double* pVector,
double* pResult, int Size)
{
  int i, j;
  #pragma omp paralell for private (j)
                for (i=0; i<Size; i++)
                { for (j=0; j<Size; j++)
          pResult[i] += pMatrix[i*Size+j]*pVector[j];
  }
}
```

Vektorni matritsaga vertikal lenta (ustun) bo'ylab ko'paytiruvchi C++ tilidagi funksiya quyidagicha bo'ladi:

```
ParallelResultCalculation(double* pMatrix, double* pVector,
double* pResult, int Size)
{ int i, j;
  for (i=0; i<Size; i++) {
  #pragma omp paralell for
    for (j=0; j<Size; j++)
    pResult[i] += pMatrix[i*Size+j]*pVector[j];
  }
}
```

Xulosa

1. Vektorni matritsaga ko'paytirishning odatiy ketma-ket va parallel usullari ko'rib chiqildi
2. Vektorni matritsaga ko'paytirishda matritsa elementlarini taqsimlash usullari ko'rib chiqildi
3. OpenMP dan foydalanib C++ tilida vektorni matritsaga ko'payturuvchi dastur tuzildi

4-Amaliy ish: Matritsali ko’paytirishning parallel usullari
REJA:

4.1. Matrisali ko’paytirish algoritmi

4.2. Matitsali ko’paytirishni parallel usullari

4.3. Matritsali ko’paytirishga C++ tilida dastur tuzish

4.4. Matritsali ko’paytirishda parallelashtirish algoritmlaridan foydalanish

Mxn o'lchamdagi matritsani A va kattalik nxl ning matritsasi B ning kattaligi mxl ning matritsasi C ga keltiriladi, ularning har bir elementi quyidagicha ifodalangan:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj},\ 0 \le i < m,\ 0 \le j < l$$

$$c_{ij} = (a_i, b_j^T),\ a_i = (a_{i0}, a_{i1} \ldots, a_{in-1}),\ b_j^T = (b_{0j}, b_{1j}, \ldots, b_{n-1j})^T.$$

Ushbu algoritm mxnxl ko'paytirish amaliyotlarini va original matritsa elementlarini qo'shish operatsiyalarining bir xil sonini bajarishni o'z ichiga oladi. Nxn hajmidagi kvadrat matritsalarni ko'paytirishda bajarilgan operatsiyalar soni O (n3) buyrug'idir. Yuqorida hisoblash murakkabligi (masalan, Strassen algoritmi) bo'lgan ma'lum matritsalarni ko'payish algoritmlari mavjud, lekin bu usullarni o'zlashtirish uchun ko'p harakat talab etiladi, shuning uchun parallel usullarni ishlab chiqishda yuqorida qayd etilgan ketma-ketlik algoritmi asos sifatida ishlatiladi. Bundan tashqari, barcha matritslarning kvadratik ekanligi va o'lchami nxn bo'lganligini ham hisobga olamiz

Ikki kvadrat matris uchun ketma-ket ko'paytirish algoritmi.

```
// Matritsani ko’paytirishni parallel algoritmi
double MatrixA[Size][Size];
double MatrixB[Size][Size];
double MatrixC[Size][Size];
int i,j,k;
...
```

```
    for (i=0; i<Size; i++){
      for (j=0; j<Size; j++){
        MatrixC[i][j] = 0;
        for (k=0; k<Size; k++){
          MatrixC[i][j]          =          MatrixC[i][j]          +
MatrixA[i][k]*MatrixB[k][j];
        }
      }
    }
```

Olingan matritsaning har bir elementi asl matritsalarning satr va ustunining skaler mahsuloti bo'lgani uchun, nxn o'lchamidagi C matritsasining barcha elementlarini hisoblash uchun siz n2x (2n - 1) skaler operatsiyalarini bajarishingiz va vaqtni sarflashingiz kerak

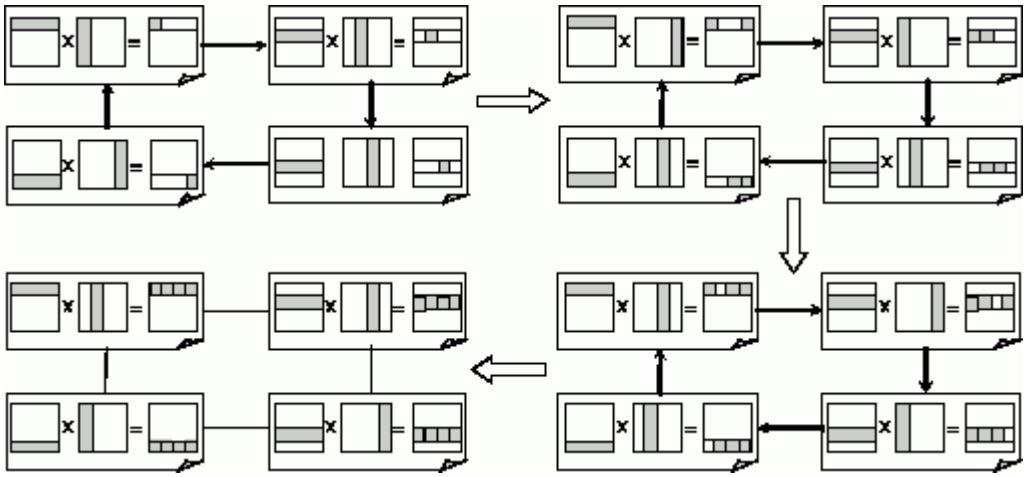
$$T_1 = n^2 \cdot (2n - 1) \cdot \tau,$$

Matritsani lenta taqsimlash matritsalarni ko'paytirish. Matritsalarni ko'paytirishning ishlash ta'rifidan boshlab, C matritsasining barcha elementlarini hisoblash bir-biridan mustaqil ravishda bajarilishi mumkin. Natijada, parallel hisoblashni tashkil qilishning mumkin bo'lgan yondashuvi, natijada paydo bo'ladigan matritsaning bir elementini asosiy subtask sifatida aniqlash uchun foydalanishdan iborat bo'lib, barcha kerakli hisob-kitoblarni bajarish uchun har bir subtaskada bir qator matritsa va bitta matratik B matritsasi bo'lishi kerak. subtask n2 ga teng (matritsaning C elementlari soni bo'yicha).

Taklif etilgan yondashuvni ko'rib chiqib, shuni ta'kidlash joizki, erishilgan parallellik darajasi ko'p hollarda ortiqcha emas. Odatda, amaliy hisob-kitoblarni amalga oshirayotganda, bunday subtaskalar mavjud bo'lgan protsessorlarning sonidan kattaroq va asosiy vazifalarni muqarrar ravishda kengaytirish bosqichini belgilaydi. Shu munosabat bilan, hisob-kitoblarni yig'ish asosiy subtaskalarni tanlash bosqichida allaqachon foydali bo'lishi mumkin. Mumkin bo'lgan eritma bir subtaskada birma-bir emas, balki natijada paydo bo'lgan matritsaning bir necha elementlari bilan birlashtirilishdan iborat bo'lishi mumkin. Keyinchalik ko'rib chiqish uchun biz asosiy matritsani C matrisi satrlaridan birining barcha

elementlarini hisoblash uchun protsedurani aniqlaymiz. Bu yondashuv umumiy sonni kamaytiradi subtasks n qiymatiga teng.

Asosiy subtaskaning barcha kerakli hisob-kitoblarini bajarish uchun matris A satriga va B matritsasining barcha ustunlariga ega bo'lishi kerak.Bu muammoning oddiy echimi - barcha subtasklardagi B matritsasining takrorlanishi - odatda xotira yuki tufayli qabul qilinishi mumkin emas. Shuning uchun hisob-kitoblarni tashkil qilish har bir joriy vaqtning o'zida pastki vazifalar hisob-kitoblarni amalga oshirish uchun zarur bo'lgan ma'lumotlarning faqatgina bir qismini o'z ichiga oladigan tarzda tuzilishi va boshqa ma'lumotlarga kirish protsessorlar o'rtasida ma'lumotlar uzatilishi bilan ta'minlanishi kerak.



Matritsani lentali taqsimlash algoritmi sxemasi

Ushbu amaliy ishda matritsalarni ko'paytirish amaliyotini bajarish uchun uchta parallel usulni nazarda tutadi. Birinchi algoritm protsessorlar orasidagi matritsalarni ajratishga asoslangan. Ushbu algoritmning ikki xil versiyasini taqdim etadi. Algoritmning birinchi varianti ko'paytirilgan matritsaning turli bo'linmalariga asoslanadi - birinchi matris (matritsa) gorizontal chiziqlarga bo'linadi va ikkinchi matritsa (matris B) vertikal chiziqlar bo'lib bo'linadi. Ip algoritmining ikkinchi variantida matritsaning gorizontal chiziqlarga bo'lishini qo'llaydi.

Xulosa

1. Matritsani ko’paytirishda parallel usullardan foydalanish vaqtni tejaydi
2. Matritsa elementlarini taqsimlashni lentali gorizantal, vertical va blokli usullari bor

5-6-Amaliy ish: Chiziqli tenglamalar tizimini yechish
REJA:

5.1. Chiziqli tenglamalarni yechish usullari

5.2. Chiziqli algebrik tenglamalar sistemasini gauss usulida yechish

5.3. CHATS yechish uchun C++ dasturlash tilidan foydalanish

Maqsad algebrik tenglamalar tizimlarini bevosita usullar bilan hal qilish usullarini qo'llashdir. Kompyuterlar orasidagi ma'lumotlarni tarqatishning turli usullari bilan bog'liq Gauss usulida CHATS ning parallel echimi uchun ikkita usul berilgan. Ushbu bo'limda Gauss usulini echish uchun navbatdagi dastur mavjud.

Chiziqli algebraik tenglamalar tizimiga yechim topish kerak:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ \dots\ + a_{1n}x_n &= f_1 \\
a_{21}x_2 + a_{22}x_2 + \ \dots\ + a_{2n}x_n &= f_2 \\
. \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \\
a_{n1}x_1 + a_{n2}x_2 + \ \dots\ + a_{nn}x_n &= f_n
\end{aligned}
$$

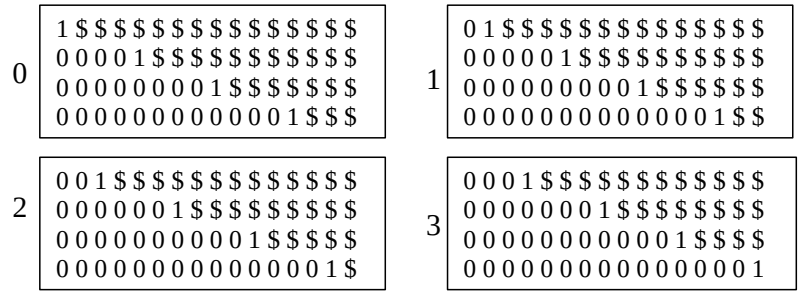
Gauss usuli noaniqlarni ketma-ket olib tashlashga asoslanadi. Gauss usuli yordamida CHATSlarni hal qilish uchun ikkita algoritmni ko'rib chiqamiz. Ular multikompyuterning tarqatilgan xotirasida ma'lumotlarni taqdim qilishning turli yo'llari (koeffitsientlarning matritsalari va o'ng tomonlari) bilan bog'liq. Ikkala misol uchun kompyuterlar uchun ma'lumotlarni tarqatish sxemalari quyida keltirilgan. Ma'lumotlar har bir algoritmda turli-tuman kompyuterlarning xotirasida turli-tuman taqsimlangan bo'lsa-da, lekin ikkalasi ham bir xil kompyuterning topologiyasida - "to'liq grafik" ga kiritilgan.

Bunday ma'lumotlarni taqsimlashda algoritm ushbu tarqatish uchun mos bo'lishi kerak. Boshqa barcha yo'nalishlardan chiqarib olingan chiziq (kerakli koeffitsientlar oldindan bo'linib bo'lgandan keyin) joriy chiziq deb ataladi. Oldinga qadam algoritmi quyidagicha. Birinchidan, joriy chiziq kompyuterdagi 0 0 bilan indikator chizig'i, keyin kompyuterda 1 indeksli chiziq (bu erda matritsadagi barcha qatorlar sonini va har bir kompyuterdagi chiziqlarni indekslashni aralashtirmaslik kerak, har bir kompyuterda qatordagi qatorlarni indekslash noldan boshlanadi) va nihoyat, oxirgi kompyuterda 0 raqamiga ega bo'lgan chiziq. Shundan so'ng kompyuterlardagi tsikl takrorlanadi va joriy chiziq 0da kompyuterda indeks 1 ga ega

chiziq, keyin kompyuterda 1-indeksli liniya 1 va shunga o'xshash chiziqlarga aylanadi. Oldinga o'tishdan keyin har bir kompyuterdagi matritsa barlari shakl 1da ko'rsatilgan shaklga ega bo'ladi. 5.4. Misol to'rtta tugun uchun berilgan; $ - haqiqiy raqamlar.

Shunga o'xshab, tugunlarni ketma-ket ravishda, kompyuterning sonidan boshlab, teskari tartibda amalga oshiriladi.

Ushbu algoritmning o'ziga xos xususiyati, to'g'ridan-to'g'ri va teskarisida, kompyuterlar birinchi usuldan ko'ra bir xil tarzda bir xil tarzda joylashtirilganligi. Shunday qilib, hisoblash yuki birinchi usuldan ko'ra kompyuterlarga nisbatan teng taqsimlanadi. Masalan, bevosita ishlatish paytida chiziqlarni qayta ishlashni yakunlagan nol kompyuter, birinchi kompyuterda bo'lgani kabi, bantlar to'liq ishlov berish o'rniga, boshqa kompyuterlar ulardan bir qatorni qayta ishlashni kutadi.



Gauss usuli bo'yicha CHATS ni yechish algoritmi

Ushbu algoritm asosida C++ dasturlash tilida tuzilgan dastur quyida keltirilgan:

```
#include<stdio.h>
#include<sys/time.h>

#define M 100

double MA[M][M+1], MAD;

int main()
  { int i, j, v, k;


    struct timeval tv1, tv2;
    long int dt1;
```

```
/* Ma'lumotlarni generatsiya qilish */
    for(i = 0; i < M; i++)
      { for(j = 0; j < M; j++)
          {  if(i == j)
              MA[i][j] = 2.0;
            else
              MA[i][j] = 1.0;
          }
        MA[i][M] = 1.0*(M)+1.0;
      }

    gettimeofday(&tv1,NULL);

/* to'gri kod */
        for(k = 0; k < M; k++)
          { MAD = 1.0/MA[k][k];
            for(j = M; j >= k; j--)
              MA[k][j] *= MAD;
            for(i = k+1; i < M; i++)
              for(j = M; j >= k; j--)
               MA[i][j] -= MA[i][k]*MA[k][j];

          }

/* Orqa yuruvchi kod */
        for(k = M-1; k >= 0; k--)
          { for(i = k-1; i >= 0; i--)
              MA[i][M] -= MA[k][M]*MA[i][k];
          }

/* vaqtni aniqlash va chiqarish */
    gettimeofday(&tv2,NULL);
    dt1 = (tv2.tv_sec - tv1.tv_sec)*1000000 +
tv2.tv_usec - tv1.tv_usec;
    printf("Time = %ld\n",dt1);

/* dastlabki 8 ildizni chiqarish */
    printf(" %f %f %f
%f\n",MA[0][M],MA[1][M],MA[2][M],MA[3][M]);
    printf(" %f %f %f
%f\n",MA[4][M],MA[5][M],MA[6][M],MA[7][M]);

    return(0);
  }
```

E’tibor bergan bo’lsangiz vaqtni o'lchash funktsiyasi `gettimeofday (& tv, NULL)` bo’ladi va bu funksiya parallel dasturlarda ham foydalanish mumkin.

Xulosa

1. Chiziqli algebrik tenlamalar sistemasini yechishni gauss usuli keng tarqalgan usullaridan biridir
2. CHATS yechishda gauss usulidan foydalanilganda parallel dasturlash oson va qulay amalga oshirladi

7-Amaliy ish: Parallel saralash usullari
REJA:

7.1. Chiziqli tenglamalarni yechish usullari

7.2. Chiziqli algebrik tenglamalar sistemasini gauss usulida yechish

7.3. CHATS yechish uchun C++ dasturlash tilidan foydalanish

Saralash ma'lumotlarni qayta ishlashning odatiy muammolaridan biri bo'lib, odatda tartibga solinmagan qiymatlar majmuasini joylashtirish vazifasi deb tushuniladi.

$$S = a_1, a_2, \ldots, a_n$$

Monoton o’sish yoki kamayish tartibida:

$$S \sim S' = (a'_1, a'_2, \ldots, a'_n) : a'_1 \leq a'_2 \leq \ldots \leq a'_n$$

(bundan keyin qisqartirish uchun barcha tushuntirishlar faqat ortib borayotgan tartibda ma'lumotlarni buyurtma qilish yo'li bilan beriladi).

Ushbu muammoni echishning yechimlari adabiyotda keng muhokama qilinmoqda; Algoritmlarni saralashning eng keng qamrovli sharhlaridan biri [50] da mavjud, so'nggi nashrlarda [[26]] tavsiya etiladi.

Buyurtma qilish tartibining hisoblash murakkabligi ancha yuqori. Shunday qilib, bir qator mashhur oddiy usullar (pufaklarni ajratish, inklüzyonlarni ajratish va boshqalar) uchun kerakli operatsiyalar soni buyurtma qilinadigan ma'lumotlar soniga kvadratik bog'liqlik bilan aniqlanadi:

$$T \sim n^2.$$

$$T \sim n \log_2 n$$

Bu ifoda, shuningdek, n qiymatlari to'plamini tartibga solish uchun talab qilinadigan operatsiyalar sonining pastligini ham beradi; kamroq murakkablik bilan algoritmlar faqat muayyan muayyan variantlar uchun olinishi mumkin.

Tartibni tezlashtirishi ko'p (p> 1) protsessorlar yordamida amalga oshirilishi mumkin. Bu holda buyurtmaning asl nusxasi protsessorlar o'rtasida taqsimlanadi; Tartiblash jarayonida protsessorlar orasidagi ma'lumotlar bir-biri bilan taqqoslangan. Olingan (buyurtma) naushnik, odatda, protsessorlar orasida bo'linadi; shu bilan birga, protsessorlar uchun bunday ajratishni tartibga solish uchun bir yoki bir nechta ketma-ket raqamlash tizimi joriy qilinadi va odatda saralashning oxirida kichik raqamli protsessorlarda joylashgan qiymatlar katta raqamli protsessor qiymatlaridan oshmasligi kerak.

Ayrim masalalar bo'yicha ajratish masalasini batafsil tahlil qilishdan oldin, biz bu yerda tartiblangan barcha ma'lumotlar kompyuterning asosiy xotirasida to'liq joylashtirilishi mumkin bo'lgan bir qator taniqli ichki saralash usullari uchun parallel bajarish usullarini o'rganishga qaratamiz.

Parallelizatsiya tamoyillari. Algoritmlarni saralashda ishlatiladigan ma'lumotlarni tartibga solish usullarini diqqat bilan o'rganib chiqqandan so'ng, siz ko'plab uslublar bir xil asosiy operatsiyani "taqqoslash va qayta tuzish" (taqqoslash-almashinish) dan foydalanishga asoslanganini ko'rishingiz mumkin. agar ularning buyurtmai tartiblash shartlariga mos kelmasa, bu qiymatlarni almashtirish.

1-misol: "solishtirish va qayta tuzish"

```
if ( A[i] > A[j] ) {
  temp = A[i];
  A[i] = A[j];
  A[j] = temp;}
```

Ushbu operatsiyadan maqsadli foydalanish ma'lumotlaringizni tashkil qilish imkonini beradi; taqqoslash uchun juft juftlarni tanlash usullarida, aslida, algoritmlarni ajratish farqlari namoyon bo'ladi.

Tanlangan asosiy saralash operatsiyalari bilan parallel umumlashtirilishi uchun, dastlab, protsessorlarning soni tartiblangan qiymatlar soniga (ya'ni p = n) to'g'ri kelishini va har bir protsessorning dastlabki ma'lumotlar to'plamining faqat

bitta qiymatini o'z ichiga olgan vaziyatni ko'rib chiqing. Keyinchalik Piy va Pj protsessorlari bo'yicha joylashgan ai va aj qiymatlarini taqqoslash quyidagicha taqsimlanishi mumkin (asosiy saralash operatsiyalari parallel umumiylashtirilishi):

Pi va Pj protsessorlarida mavjud bo'lgan qadriyatlar almashinuvini amalga oshirish (bu protsessorlarda asl elementlarni saqlab qolishda);

har bir protsessor Pi va Pj ga teng qiymatlarni solishtirish (ai, aj); taqqoslama natijalari protsessorlar orasidagi ma'lumotlarni almashish uchun ishlatiladi - bir protsessorda (masalan, Piy) kichik element qoladi, boshqa protsessor (ya'ni, Pj) juftlikning katta qiymatini qayta ishlash uchun eslab qoladi

$$a_i' = \min(a_i, a_j), \quad a_j' = \max(a_i, a_j)$$

Asosiy tartibida ishlashning parallel umumlashtirilishi p ning ishi uchun moslashtirilishi mumkin, agar protsessor soni buyurtma qiymatlari sonidan kam bo'lsa. Bunday holda, har bir protsessor endi bitta qiymatga ega bo'lmaydi, lekin ma'lumotlar majmuasining bir qismi (n / p nusxasi) tartiblanadi.

Biz parallel sortirovka qilish algoritmini ishlab chiqishda, protsessorlarda mavjud bo'lgan ma'lumotlarni buyurtma qilinadigan buyurtma qilingan ma'lumotlar majmuasining bunday holatini aniqlaymiz va protsessorlar orasida blok tarqatish tartibi chiziqli raqamlash tartibiga to'g'ri keladi (P ni protsessoridagi oxirgi elementning qiymati protsessor Piydagi birinchi element qiymatidan kam) +1, bu erda 0 <= i <p-1 yoki unga teng).

Bloklar, odatda, har bir protsessorda alohida algoritm yordamida (parallel saralashning dastlabki bosqichi) alohida tartibda tartibga solinadi. Bundan tashqari, bitta elementli taqqoslash sxemasidan so'ng, Piy va Pi + 1 protsessorlari Ai va Ai + 1 bloklari tarkibini birgalikda tashkil qilish uchun o'zaro ta'sirlar quyidagicha amalga oshirilishi mumkin:

Pi va Pi + 1 protsessorlari o'rtasida bloklar almashinuvini amalga oshirish;

Har bir protsessorga A va Ani + 1 bloklarini birlashtirilgan ikkita o'lchamli blokga (Ai va Ai + 1 bloklari dastlabki buyurtmasi berilganda, ularni birlashtirish jarayoni tartiblangan ma'lumotlar majmui tezkor birlashma operatsiyasiga kamayadi);

hosil bo'lgan er-xotin blokni ikkita teng qismga bo'linib, protsessor Pi-da bu qismlardan (masalan, kichik ma'lumotlar qiymatlari bilan) bir qismini qoldiring va boshqa qismi (katta qiymatlar bilan mos ravishda) protsessor Pi + 1

$$[A_i \cup A_{i+1}]_{copm} = A'_i \cup a'_{i+1} : \forall a'_i, \in A'_i,\ \forall a'_j \in A'_{i+1} \Rightarrow a'_i \leq a'_j.$$

Ushbu amaliyot odatda adabiyotda taqqoslash va taqqoslash operatsiyalari sifatida taqqoslanadi (solishtirish-ajratish). Shuni ta'kidlash kerakki, ushbu protsedura natijasida hosil qilingan bloklar Pi va Pi + 1 protsessorlarida Ai va Ai + 1 bloklari bilan teng bo'lib, protsessor Pi-da joylashgan barcha qiymatlar P + 1 protsessoridagi qiymatlardan oshmaydi.

Yuqorida tavsiflangan "taqqoslash va bo'lish" operatsiyalari parallel hisoblarni tashkil qilish uchun asosiy subtask sifatida ishlatilishi mumkin. Qurilishdan shuni ta'riflaydigan bo'lsak, bunday subtasklarning soni parametrik ravishda mavjud protsessorlarning soniga bog'liq bo'lib, parallel sortirovka qilish algoritmlari uchun hisob-kitoblarni taqqoslash muammosi deyarli yo'q. Shu bilan birga, ajratish jarayonida subtask bilan bog'liq ma'lumotlar bloklari o'zgarganligini ta'kidlash lozim. Oddiy hollarda, subtasklardagi ma'lumotlar blokirovkalari o'zgarmaydi. Keyinchalik murakkab vaziyatlarda (masalan, tezkor tartiblash algoritmida - 9.5-bo'limni ko'ring), protsessorlar bo'yicha mavjud bo'lgan ma'lumotlar o'zgarishi mumkin, bu esa protsessorlarning yagona hisoblash yukini buzilishiga olib kelishi mumkin.

2-Misol: ketma ket algortim Bubble sorting

```
void BubbleSort(double A[], int n) {
  for (int i = 0; i < n - 1; i++)
    for (int j = 0; j < n - i; j++)
      compare_exchange(A[j], A[j + 1]);}
```

3-misol: Toq va juft ketma-ket algoritmi

```
// ketma-ket juftlikda joylashish algoritmi
void OddEvenSort (ikkita A [], int n) {
   for (int i = 1; i <n; i ++) {
     if (i% 2 == 1) {// odd iteratsiya
       (int j = 0; j <n / 2 - 2; j ++)
         compare_exchange (A [2 * j + 1], A [2 * j + 2]);
```

```
            if(n% 2 == 1) // oxirgi juftni odatdagi n bilan
solishtiring
              compare_exchange (A [n - 2], A [n - 1]);
          }
          else // hatto iteratsiya
            for(int j = 1; j <n / 2 - 1; j ++)
              compare_exchange (A [2 * j], A [2 * j + 1]);
       }
    }
```

4-misol: toq va juft parallel algoritmi

```
    ParallelOddEvenSort (ikkita A [], int n) {
      int id = GetProcId (); // jarayon raqami
      int np = GetProcNum (); // jarayonlar soni
     for (int i = 0; i <np; i ++) {
        if (i% 2 == 1) {// odd iteratsiya
          if (id% 2 == 1) {// odd jarayon raqami
            // Proses bilan taqqoslash - almashish - o'ng tomonga
            if (id <np - 1) compare_split_min (id + 1);
          }
         else
            // Process bilan solishtirish - chapga qo'shni
            if (id> 0) compare_split_max (id - 1);
        }
        else {// aks holda iteratsiya
          if (id% 2 == 0) {// hatto protsessing raqami
            // Proses bilan taqqoslash - almashish - o'ng tomonga
            if (id <np - 1) compare_split_min (id + 1);
          }
          else
            // Process bilan solishtirish - chapga qo'shni
            compare_split_max (id - 1);
        }
      }
    }
```

Buble saralash algoritmi asl usulida amaliyotda asosiy ketma-ket ravishda amalga oshirilishi tufayli parallelism emas. Kerakli parallelizmni joriy qilish uchun algoritmning umumlashtirilgan versiyasi - odatdagi permütasyon usuli hisoblanadi. Boshlashning mohiyati shundan iboratki, usulni o'zgartirish uchun ikki xil qoidalar tartiblash sonining paritetiga qarab saralash algoritmiga kiritiladi. Ikkala g'alati permütasyon usulining yinelemelerinde ma'lumotlar majmui qiymatlari juftligini taqqoslash mustaqil bo'lib, parallel ravishda amalga oshirilishi mumkin.

Shell algoritmi uchun parallelizatsiya sxemasi tarmoq topologiyasi hiperküp sifatida ko'rsatilganda ko'rib chiqiladi. Topologiyaning bu namoyishi bilan bir-biridan uzoqda joylashgan protsessorlarning liniyali raqamlash bilan o'zaro aloqasini tashkil etish mumkin bo'ladi. Odatda, bunday hisob-kitoblar tashkiloti amalga oshiriladigan tartiblash algoritmining yineleme sonini kamaytirishga imkon beradi.

Tez saralash algoritmi uchun uchta parallelizatsiya sxemasi berilgan. Birinchi ikkita sxema tarmoq topologiyasining hiperküp ko'rinishiga asoslangan. Hisobotlarning asosiy yinelemasi etakchi elementning protsessorlaridan birini tanlashdan iborat bo'lib, u keyinchalik hiperküpning barcha boshqa protsessorlariga yuboriladi. Asosiy elementni olgandan so'ng, protsessorlar bloklarini ajratib turadi

Xulosa

1. Saralash algortimlarining bir-biridan farqlari
2. Parallel saralash algoritmining ketma-ket saralash algortmlardan asosiy farqlari

8-Amaliy ish: Graflar ustida parallel amallar

REJA:

8.1. Graflar ustida amallar

8.2. Graflar ustida parallel amallar

8.3. Garflar ustida amallar bajarish uchun C++ dasturlash tilidan foydalanish

"Oddiy" kompyuter uchun mo'ljallangan har qanday dastur ma'lum algoritmlar turkumini ta'riflaydi. Uni amalga oshirishda o'ziga xos algoritmni tanlash shartli

operatorlarning ishlashi bilan belgilanadi. Qolgan operatorlarning tarkibi va bajarilishi dasturning o'zi tomonidan qat'iy aniqlanadi. Agar dasturda shartli operator bo'lmasa, dasturda dastlab bitta algoritm tasvirlangan. O'z navbatida, shartli operatorlarning operatsiyalari faqatgina kirish ma'lumotlariga bog'liq. Shuning uchun, "oddiy" kompyuter har doim dastur va ma'lumotlar bilan aniq belgilanadigan ba'zi bir harakatlar ketma-ketligini amalga oshiradi. Bundan tashqari, xuddi shu dastur uchun bu ketma-ket "oddiy" kompyuterning har qanday modellarida bir xil bo'ladi. Shunday qilib, natija aniq aniqlanadi. Bularning barchasi, oxir-oqibat, har qanday "oddiy" kompyuterda har qanday vaqtda faqat bitta operatsiyani amalga oshirish mumkinligi bilan izohlanadi. Hozirgi vaqtda amalga oshiriladigan har qanday operatsiya faqat amalga oshirishga tayyorgarlik bosqichidan o'tishi mumkin.

Vaziyat parallel arxitektura hisoblash tizimlari uchun farq qiladi. Endi, har qanday vaqtda, bir-biridan mustaqil bo'lgan barcha operatsiyalar ansambli amalga oshirilishi mumkin. Har qanday maxsus parallel tizimda dastur va kirish ma'lumotlari ansambllarning tarkibini va ularning ketma-ketligini noyob tarzda aniqlaydi. Ammo turli tizimlarda ansambllar va ketma-ketliklar turli xil bo'lishi mumkin. Shunga qaramay, aniq bir natija olishni kafolatlash uchun, barcha operatsiyalarni bajarish tartibi muayyan shartlarga bo'ysunishi kerak.

Ikkala "oddiy" va parallel kompyuterda ham muammoni hal qilish oddiy operatsiyalarni bajarish natijasida yuzaga keladi. Barcha operatsiyalarda ozgina dalillar mavjud. Odatda ikkitadan ko'p emas. Operatsiya argumentlarining o'ziga xos qiymatlari sifatida kirish ma'lumotlari yoki boshqa operatsiyalarni bajarish natijalari olinadi. Muvofiqlik, bu natijalar dastur ishlab chiquvchi tomonidan belgilanadigan dalillardir.

Ma'lumki, har qanday operatsiya - argumentlarni iste'molchi barcha operatsiyalarni bajarishdan oldin - uning uchun argumentlarni etkazib beruvchilarni bajarishga kirisha olmaydi. Shunday qilib, dasturni ishlab chiquvchi aniq yoki to'liq ravishda barcha operatsiyalar bo'yicha qisman buyurtmani o'rnatadi. Har qanday ikkita operatsiyani bajarish uchun buyurtmanoma imkoniyatlardan birini belgilaydi: operatsiyalarning qaysi birini amalga oshirish kerakligini ko'rsatadi yoki har ikkala

operatsiyalar bir-biridan mustaqil ravishda amalga oshirilishini bildiradi. Xuddi shu qisman buyruqlar bilan, butun operatsiyalarning umumiy temporal tartibi boshqacha bo'lishi mumkin. Keyinchalik, bu ko'rsatmalarning birortasi ham xuddi shunday natijaga erishishini ko'rsatamiz. Shu sababli, dasturda ko'rsatilgan qisman tartibni saqlab qolish, uning bajarilishi natijaning o'ziga xosligini kafolatlaydi. Xuddi shu qisman tartib doirasida biron bir tanlovni tanlash mumkin.

Quyidagi talqinni beramiz. Sababi asosiy ma'lumotlarga ega bo'lgan dasturda ba'zi algoritmlar tasvirlangan. Yo'naltirilgan grafikni yaratish. Masjidlar kabi biz har qanday to'siqni olamiz, masalan, algoritmning barcha operatsiyalarining bir-biriga mos keladigan xaritasi joylashgan arifmetik maydonning nuqtalari to'plami. Har qanday juftlik nuqtasini oling va v. Yuqorida tavsiflangan qisman tartibga ko'ra vertexga mos keladigan operatsiya va vertex v. Keyinchalik vertikadan vertex vga chizamiz. V. Agar mos keladigan operatsiyalar bir-biridan mustaqil ravishda amalga oshirilsa, biz boshqasini qilmaymiz. Jarayonning dastlabki ma'lumotlari dastlabki ma'lumotlar bo'lsa yoki operatsiya natijasi hech qanday joylarda ishlatilmasa, turli kelishuvlar amalga oshirilishi mumkin. Misol uchun, mos keladigan yoylarning yo'qligi hisobga olinishi mumkin. Yoki barcha kirish ma'lumotlari va natijalari kiritilgan va maxsus kirish / chiqish qurilmalari orqali chiqariladi. Bunday holda, bunday operatsiyalarga mos keladigan grafikning tepalari va ularning faqatgina kiruvchi va chiquvchi yoylari bo'lmaydi. Vaziyatga bog'liq holda qilamiz. Shu yo'l bilan tuzilgan grafik, sobit kirish ma'lumotlari bilan algoritmni amalga oshirish uchun axborotga qaramlik grafigi deb atash mumkin. Biroq, bu nom juda og'ir. Shuning uchun uni kelajakda faqatgina algoritmning grafigi deb ataymiz. Yo'naltirilgan grafikani yaratish usuliga qaramay, bitta kirish yoki chiqadigan kamonga ega bo'lgan vertikalarning navbati grafikaning kirish yoki chiqish vertikalari deb ataladi.

Ushbu kontseptsiya ba'zi tushuntirishlarni talab qiladi. Algoritm grafigi deyarli har doim kirish ma'lumotlariga bog'liq. Dasturda hech qanday shartli operator bo'lmasa ham, ular bajarilgan operatsiyalarning umumiy sonini va natijada grafigning umumiy sonlarini aniqlaganligi sababli ular massiv kattaligiga bog'liq

bo'ladi. Shunday qilib, aslida algoritm grafigi deyarli har doim parametrlangan grafildir. Albatta, nafaqat vertices soni, balki butun datchiklar parametrlarining qiymatiga bog'liq. Agar dasturda shartli operatorlar bo'lmasa, biz ham algoritmni, ham u tomonidan tasvirlangan algoritmni deterministik deb ataymiz.

Aks holda ularni noaniq deb hisoblaymiz. Deterministik algoritm grafigi va noanmatik algoritmning grafigi o'rtasida bir asosiy farq mavjud. Deterministik algoritm uchun dasturning barcha amaliyotlari va algoritm grafigining barcha vertikalari o'rtasida doimo bir-biriga moslik mavjud. No-deterministik algoritm uchun parametrlarning barcha qiymatlari, ya'ni kiritish ma'lumotlari uchun hech kimga o'xshashlik yo'q. Faqatgina bu holatda har bir parametr qiymatining to'plami uchun barcha dastur operatsiyalari va grafik tugunlarining ba'zi bir to'plamlari orasida bir-bir bilan yozishma mavjud. Parametrlarning turli xil to'plamlari parametrlarning turli qiymatlariga mos kelishi mumkin.

Kelajakda, qo'shimcha rezervasyonlar qilinmasa, biz deterministik algoritmlarni va dasturlarni ko'rib chiqamiz. Ushbu cheklovni kiritish sabablari juda oz. Avvalo, ular ancha sodda qilib belgilanadi va, tabiiyki, ular tadqiqotlar boshlanishi mumkin. Ikkinchidan, deterministik algoritm va dasturlarning klassi juda keng. Bundan tashqari, ko'pgina no-deterministik algoritmlar aslida deyarli deterministikdir. Misol uchun, filiallar kirish ma'lumotlarining qiymatlaridan qat'iy nazar, sonli operatsiyalarni qamrab olganda. Bunday filiallar katta miqdordagi operatsiyalarga solinishi mumkin, ammo bunga qaramay, son-sanoqsiz dalillar mavjud. Va nihoyat, agar dallanma katta deterministik qismlarni qamrab oladigan bo'lsa, algoritm grafigini o'rganish ushbu qismlarni o'rganishga kamayadi.

Ko'rib chiqilgan algoritm grafigi yo'naltirilgan asiklik multigraph hisoblanadi. O'zgarmasligi har qanday dasturlarda aniq hisob-kitoblarni bajarishdan kelib chiqadi va hech qanday qiymat o'z-o'zidan aniqlanmaydi. Dasturda takroriy iboralar mavjud bo'lsa ham, bu faqat bir turdagi hisoblarni ta'riflash uchun qulay shakldir. O'z-o'zidan ravshanki, har xil operatsiyalar amalda amalga oshiriladi. Umumiy holatda, algoritm grafigi bir nechta grafika, ya'ni ikki vertikal bir necha kamonga ulanishi mumkin. Xuddi shu qiymat bir xil operatsiyadagi turli argumentlar sifatida ishlatilganda

bo'ladi. Biz standart simvolizmni G = (V, E) dan foydalanamiz, bu erda V - vertikalarning to'plamidir, E - G grafining yoylari majmuasi.

Bizning fikrimizcha, algoritm grafigi algoritm yozuvida joylashgan ishlab chiquvchi fikrning eng asosiy yadrosini ifodalaydi. Algoritmni ishlab chiquvchi bu yadroni bilishi mumkin edi. Ehtimol, uni algoritmni ta'riflovchi til yadroni samarali tarzda tasvirlashga imkon bermaganligi sababli uni yashirishga to'g'ri kelgan bo'lishi mumkin. Amaliyot shuni ko'rsatadiki, aksariyat hollarda bu dastur uchun algoritm yadrosi noma'lum. Bundan tashqari, odatda, uning mavjudligi haqida o'ylamaydi. Algoritm grafigi nafaqat algoritmning yadrosini emas, balki uning axborot yadrosini hisobga olsak, hamma narsa aniq bo'ladi. So'nggi paytgacha algoritmlarni amalga oshirish jarayonida axborot qanday tarqalganligi haqida ma'lumot yo'q edi. Shu sababli ularni qabul qilishning hech qanday istagi yo'qligi ajablanarli emas. Parallel hisoblash tizimlarining paydo bo'lishi bu ma'lumotni o'ta zarur holga keltirdi. Shuning uchun ular rivojlana boshladi.

Xulosa

1. Agar qisman buyruqlar operatsiyalar majmuasiga o'zgartirilsa, yangi algoritm asl nusxaga teng keladimi?
2. Algoritmning tuzilishini xarakterlovchi parallel umumiy formadagi qavslar sonining minimal qiymati mumkinmi?
3. Bir xil vazifa algoritmlari turli yuvarlama xatolar bilan echimlarni taqdim etsin. Bunday algoritmlar bir xil parallel shakllarga ega bo'ladimi?
4. Turli muammolarni hal qilish uchun mo'ljallangan turli xil algoritmlardan misollar keltiring, biroq baribir bir xil grafikalar mavjud.

Glossariy

	So’zning o’zbek tilidagi berilishi	So’zning ingliz tilindagi berilishi	So’zning ma’nosi
1	#include	#include	Preprotsessor direktivasi. Programmist tomonidan aniqlangan yoki standart ob’ekt-sarlavhaning programma matnini qo’shish uchun ishlatiladi
2	#include<preprotsessor_leksemalari>	#include<lexemes_preprocessors>	preprotsessor_leksemalari- ob’ekt-sarlavhaning nomi yoki leksemalar seriyasi.
3	adres	address	Operativ xotiradagi obe’kt adresi ko’rinishiga ega bo’ladi. C++ tilide adresli arifmetika xatolarning batcha vaqtda daragi bo’lib hisoblanadi
4	agar	if	Shart instruktsiyasida foydalanadigan C++ tili kalit so’zi.
5	aks holda	else	if instruktsiyasida shart bajarilmaganda foydalanadigan(shart bo’lmagan) C++ tili kalit so’zi.
6	ansi	ansi	Abbreviatura: American National Standards Institute (Amerika Milliy standartlar Instituti). www.ansi.org
7	Belgi	Label	Identifikator bo’lib, goto komandasida foydalaniladi. O’tish ko’rsatilgan joydagi nishondan keyin «:» simvoli qoyiladi.
8	belgiga ega emas	unsigned	Musbat butun sanlar tipin e’lon qilishda foydalanadigan C++ tili kalit so’zi.
9	bool	bool	Berilganlarding mantiqiy tipin e’lon qilishdagi foydalanadigan kalit so’z.
10	C	C	Asosan sistemali programmalashga mo’ljallangan Ritchie, Dennis M/ tomonidan Bell Laboratories korporatsiyasida ishlab chiqilgan programmalash tili
11	C++ dasturlash tili	C++ programming language	Bjarne Stroustrup tomonidan Bell Laboratories korporatsiyasida ishlab chiqilgan programmalash tili
12	cin	cin	C++ tilide standart kirituvchi potokni(standard input stream) ifodalash
13	cout	cout	C++ tilide standart chiquvchi potokni(standard output stream) ifodalash.
14	dastur	program	Qandayda bir programmalash tilida yozilgan komandalar ketme-ketligi yoki

			ma’lum bir masalani echuv uchun yozilgan protsessor komandalari
15	dasturchi	programmer	Programmani ishlab chiqish va tekshirish bilan shug’ullanuvchi mutaxassis. Tizimli va amaliy programmistlar bo’lib ajratiladi.
16	dasturiy ta’minotni yaratish	software development	Bu matematika, informatika va bosqada sohalar bilimidan foydalanish bilan birga programmalash tilin qo’llab programmaviy ta’minotni yaratishga yo’naltirilgan protsess.
17	dasturlash tili	programming language	Sonlar, harflar va so’zlerni ifodalash uchun yaratilgan maxsus tizim
18	dekrement	decrement	-- operatori bilan bog’liq bo’lgan 1 ge orttirish amali.
19	do	do	while da foydalanuvchi tsikl operatori
20	enum	enum	Sanab o’tiluvchi tipni bildirivchi C++ tili kalit so’zi.
21	feof	feof	Faylning oxirini ko’rsatuvchi indikatorning holatini tekshiruvchifunktsiya nomi
22	float	float	Birlik aniqliqdagi haqiqiy sonning tipin e’lon qilishni foydalanadigan C++ tili kalit so’zi.
23	fopen	fopen	Berilgan nomdagi faylni ochish imkaniyotini beruvchi funktsiya nomi.
24	for	for	Tsiklning instruktsiyasini bildiruvchi C++ tili kalit so’zi.
25	freopen	freopen	Berilgan nomdagi faylni ochish imkaniyotini beruvchi funktsiya nomi.
26	Funktsiya prototipi	function prototype	Ko’rsatilgan tipdagi funktsiyalarni e’lon qilishda ishlatiladi.
27	goto	goto	Funktsiya ichida boshqarishni uzatishda (shartsiz o’tishda) foydalanadigan C++ tili kalit so’zi.
28	himoyalangan	protected	Sinf a’zosi bilan faqat sinfning funktsiya a’zolari yoki sinfning do’st sinflari va voris sinflar ishlashish mumkin ekanligini bildiradigan C++ tili kalit so’zi.
29	Initsializatsiya	initialization	Ob’ektlarga boshlang’ich qiymatlarni o’zlashtirish.
30	int	int	Butun sonlar tipini e’lon qilishda foydalanadigan C++ tili kalit so’zi.

31	integrallashgan ishlab chiqish muhiti	ide yoki integrated development environment	Programmalardi yaratish uchun ishlab chiquvchilar foydalanadigan programmaviy vositalar to'plami
32	izohlar	comments	C++ tilida C tilidagi komentariyalar saqlanib qolingan. bunda «/*» simvoli bilan boshlanadi va «*/» simvoli bilan tamomlanadi. C++ tili komentriyalarning yangi stiliga ha ega bo'lib,«//» belgisidan keyin bir satr uchun.
33	ko'rsatkich	pointer	Ob'ektlar adreslari bilan ishlashga mo'ljallangan ob'ekt. Iteratorning xususiy holati.
34	lokal o'zgaruvchi	local variable	Funktsiyaning ichida e'lon qilingan o'zgaruvchi
35	long double	long double	Uzin ikkilik aniqlikdagi haqiqiy sonning tipin e'lon qilishni foydalanadigan C++ tili kalit so'zi.
36	main	main	C/C++ tillaridagi programmaning asosiy bosh funktsiyasining nomi.
37	massiv	array	Bir turga tegishli bir xil nomdagi elementlarning tartiblangan guruhi
38	massivni o'chirish operatori	delete[] operator	Massiv ob'ektlerini o'chirish uchun foydalanadigan delete operatori shakli.
39	namespace	namespace	Funktsiyalar, sinflar h.t.b.-nomlar fozasini e'lon qilish uchun islatiluvchi C++ tili kalit so'zi.
40	new[] operatori	new[] operator	new operatori massiv ob'ektlar uchun xotirani ajratishda foydalanadigan shakli.
41	nom	name	Identifikator bo'lib, ob'ekt, funktsiya, qayta yuklanuvchi funktsiyalar, tip, sanab o'tish, a'zo, shablon, nomlar fazosi, belgilerdi belgilash.
42	NULL	NULL	Maxsus konstanta bo'lib, yo'q adresni ko'rsatadi. Qoyda bo'yicha uning qiymati nolge teng.
43	o'zgarmas	constant	const kalit so'zi bilan e'lon qilingan literal yoki o'zgaruvchi.
44	o'zgaruvchi	variable	Boshqa bir ob'ektting qiymati bo'lishi mu'mkin bo'lgan boshqa bir ob'ekt.
45	ob'ekt	object	Berilganlaning tipiga mos qiymatni saqlovchi xotira oblasti
46	ochiq	public	Sinf a'zosi bilan sinfning iqtiyoriy foydalanuvchisi islashish mumkin ekanligini bildiradigan C++ tili kalit so'zi.

47	OYP	OOP	Ob’ektga-yo’naltirilgan programmalash.
48	parametr	parameter	Funktsiyaga beriladigan o’zgaruvchilar. Funktsiya argumenti.
49	qisqa	short	Qisqa butun sonlar tipin e’lon qilishda foydalanadigan C++ tili kalit so’zi.
50	return	return	Shaqiriliyongan funktsiyaga boshqarishni uzatish komandasi. Agar funktsiyaning tanasida uning tipiga mos qiymatni qaytarishda foydalanilsa return [ifoda] ko’rinishida yoziladi.
51	rost	true	Mantiqiy (bool) tipinining qiymatlaridan biri C++ tili kalit.
52	sarlavha	header	Ob’ekt-sarlavha – bu #include preprotsessor direktivasi yordamida ishlash mumkin bo’lgan ob’ekt.
53	shablon argumenti	template argument	Shablon bilan real sinfni qurishdagi tip.
54	shaxsiy	private	Sinf a’zosi bilan faqat sinfting funktsiya a’zolari yoki sinfning do’st sinflari islashish mumkin ekanligini bildiradigan C++ tili kalit so’zi.
55	sinf	class	Programmaning asosiy qurilgan blogini e’lon qilish uchun ishlatiladigan C++ tili kalit so’zi. Sinfda nimi, a’zolari,boshqaruvga murojat mexanizimi v.h.k. mavjud bo’ladi.
56	sintaksis	syntax	Ifodalarni, instruktsiyalardi, e’lon qilishlarni va boshqada programma qismlarini yaratish qoydalarini to’plami.
57	sizeof	sizeof	Ob’ekt yoki tipning uzunligini o’lshamini baytlarda beruvchi funktsiyani bildiruvchi C++ tili kalit so’zi.
58	Standart ob’ekt -sarlavha	header, standard	Har bir biblioteka protsedurasi mos ob’ekt-sarlavhani e’lon qilishga ega. Standart bibliotekalarning ayrim ob’ekt- sarlavhalari: math.h- matematik funktsiyalar iostream.h- stdlib.h- umumiy belgilangan utilitalar stdio.h- kiritish-shiqarish funktsiyalari iomanip.h –potok manipulyatorlari
59	struktura	struct	Sinf bilan deyarli bir xil ammo, barcha a’zolari public bo’ladi.

60	switch	switch	Bir necha variantlardan bittasini tanlash imkaniyotini beradigan instruktsiyaning C++ tili kalit so’zi.
61	Teskari slesh	backslash	Simvol \ boshqaruvchi ketma-ketlikni formallasstirishda satrli literallar va simvolli konstantalarda ishlatiladi. Masalan: \f –yangi sahifaga o’tish, \n –yangi satrga o’tish
62	this	this	Ob’ektning adresi kerak bo’lganda funktsiya-a’zoda foydalanadigan C++ tili kalit so’zi.
63	throw	throw	Istisno holatlar generatsiyasi uchun foydalanadigan C++ tili kalit so’zi.
64	try	try	Istisno holatlar uchun blokning boshlanishi.
65	tur	type	Nomlarning agregatlari bo’lib, uning ishlash usulini aniqlaydi. Masalan,qandayda bir sinf ob’ekti tipin butun o’zgaruvchiga o’zlashtirib bolmaydi.
66	union	union	Birlashmalar bo’lib, «oddiy» struktura va sinfga uqshash, farqi unda barcha a’zolari bir xotirada joylashadi.
67	ushlab olmoq	catch	Xatoliklarni qayta ishlash uchun foydalanadigan C++ kalit so’zi.
68	uzish	break	for va while tsikllarinda takrorlanishni to’qtatishda va switch instruktsiyasida variantlarni ajiratishda va instruktsiya tanasidan chiqishda foydalanadigan C++ tili kalit so’zi.
69	uzun	long	Uzun butun sanlar tipini e’lon qilishda foydalanadigan C++ tili kalit so’zi.
70	variant	case	switch instruktsiyasida alohida tarmoqni belgilash uchun ishlatiladigan C++ kalit so’zi.
71	void	void	Hesh qanday qiymatga ega bo’lmagan bo’sh tip, C++ tili kalit so’zi.
72	while	while	Tsiklning instruktsiyasin bildiruvchi C++ tili kalit so’zi.
73	yolg’on	false	Mantiqiy (bool) tipinining qiymatlaridan biri C++ tili kalit.