Дәріс 1. Көпағындық программалауды қолдау. Үрдістерді басқару. Жадының ұйымдастырылуы.

Параллель компьютерлердің ең танымал классификациясын Флинн ұсынған [13] және компьютерлер жүзеге асыратын параллелизм формасын көрсетеді. Жіктеудің негізгі ұғымдары «командалық ағын» және «мәліметтер ағыны» болып табылады. Командалар ағыны бір бағдарламаның командалар ретін білдіру үшін жеңілдетілген. Деректер ағыны - бұл бір программа өңдейтін мәліметтер тізбегі. Бұл классификацияға сәйкес компьютерлердің төрт үлкен класы бар:

1) OKOD (бір командалық ағын - бірыңғай мәліметтер ағыны) немесе SISD (Single Instruction - Single Data). Бұл тізбектелген компьютерлер, онда бір программа орындалады, яғни тек бір нұсқау санауышы болады.

2) SMD (бір командалық ағын - бірнеше мәліметтер ағыны) немесе SIMD (жалғыз нұсқаулық - бірнеше мәліметтер). Мұндай компьютерлерде бір программа орындалады, бірақ әр команда мәліметтер массивін өңдейді. Бұл параллелизмнің векторлық түріне сәйкес келеді.

3) MKOD (Multiple Instruction - Single Data Stream) немесе MISD (Multiple Instruction - Single Data). Бұл сыныпта бір уақытта бірнеше команда бір мәліметтермен жұмыс істейді деп болжануда, алайда Флинн классификациясының бұл позициясы іс жүзінде қолданылмаған.

4) MIMD (бірнеше командалық ағын - бірнеше мәліметтер ағыны) немесе MIMD (Multiple Instruction - Multiple Data). Мұндай компьютерлерде белгілі бір аралықта мәліметтер алмасып, бір-бірінен тәуелсіз бірнеше бағдарламалық филиалдар орындалады. Мұндай жүйелерді әдетте көппроцессорлы жүйелер деп атайды.

Амдал заңы.

Параллель жүйелердің негізгі сипаттамаларының бірі - өрнекпен анықталатын параллель жүйенің R үдеуі:

R = T1 / Tn,

Мұндағы T1 - бірпроцессорлы жүйеде есепті шығаруға уақыт, ал Tn - n-процессорлық жүйеде сол мәселені шешуге уақыт.

W = Wsc + Wпр болсын, мұндағы W - есептегі амалдардың жалпы саны, Wпр - параллель орындалатын амалдар саны, Wск - скалярлық (параллелденбейтін) амалдар саны. Сонымен қатар біз бір операцияның орындалу уақытын t арқылы белгілейміз. Сонда біз белгілі Амдаль заңын аламыз [13]:

R = W t / (Wsc + Wpr / n) t = 1 / (a + (1 - a) / n) → n → ∞ 1 / a (1.3)

Мұндағы a = Wsc / W - скалярлық амалдардың үлесі. Амдал заңы параллель есептеулер үшін принципиалды маңызды позицияларды анықтайды:

1. Үдеу тапсырманың потенциалды параллелдігіне (мәні 1 - a) және аппараттық параметрлерге (процессорлар саны n) байланысты.

2. Шектік үдеу есептің қасиеттерімен анықталады. Мысалы, a = 0,2 болсын (бұл нақты мән), онда үдеу кез-келген процессор саны үшін 5-тен аспауы керек, яғни максималды үдеу тапсырманың потенциалды параллелизмімен анықталады. A мәнінің өзгеруіне үдеудің өте жоғары сезімталдығы айқын.

Амдалдың желілік заңы.

Amdahl заңының негізгі нұсқасы процессорлармен хабарлама алмасу уақытының жоғалуын көрсетпейді. Бұл шығындар есептеудің үдеуін азайтып қана қоймай, сонымен бірге есептеуді бір процессорлық нұсқаға қарағанда баяулатуы мүмкін. Сондықтан экспрессияны (1.3) біраз модернизациялау қажет. Біз (1.3) келесідей жазамыз:

$$Rc = W\,t / ((Wsc + Wpr / n)\,t + Wc\,tc)$$

$$= 1 / (a + (1- a) / n + Wc\,tc / W\,t) = 1 / (a + (1- a) / n + c) \quad (1.4)$$

Мұнда Wc - деректерді беру саны, tc - бір деректерді беру уақыты. Өрнек

$$Rc = 1 / (a + (1- a) / n + c) \quad (1.5)$$

және бұл Amdal желісі туралы заң. Бұл заң көппроцессорлы есептеудің келесі екі ерекшелігін анықтайды:

1. Есептеу желісінің деградация коэффициенті:

$$c = Wc\,tc / W\,t = cA\,cT, \quad (1.6)$$

деректерді беру үшін есептеу мөлшерін анықтайды (жұмсалған уақыт бойынша). Бұл жағдайда cA алгоритмнің қасиеттеріне байланысты деградация коэффициентінің алгоритмдік компонентін анықтайды, ал cT - техникалық компонент, бұл процессор мен желілік жабдықтың техникалық жылдамдығының арақатынасына байланысты. Осылайша, есептеу жылдамдығын арттыру үшін деградация коэффициентінің екі компонентіне де әсер ету қажет. Көптеген есептер мен желілер үшін cA және cT коэффициенттерін көптеген факторлармен анықталса да, оларды аналитикалық және алдын-ала есептеуге болады: есептің алгоритмі [14,15], мәліметтер мөлшері, MPI кітапханасының орындалуы алмасу функциялары, ортақ жадыны пайдалану және, әрине, коммуникациялық орталардың техникалық сипаттамалары және олардың хаттамалары.

2. Тапсырма мінсіз параллелизмге ие болса да, желілік үдеу мәнмен анықталады

$$Rc = 1 / (1 / n + c) = n / (1 + c\,n) \rightarrow c \rightarrow 0\ n \quad (1.7)$$

және процессорлар санының өсуімен азаяды. Демек, Amdahl желілік заңы көппроцессорлы компьютерлерде шешуге арналған есептер алгоритмін және бағдарламалауын оңтайлы құруға негіз болуы керек.

Кейбір жағдайларда есептеулердің тиімділігін өлшеу үшін тағы бір параметр қолданылады - пайдалану коэффициенті z:

$$z = Rc / n = 1 / (1 + c\,n) \rightarrow c \rightarrow 0\ 1.$$

Орталық процессор бірнеше процестермен пайдаланылуы мүмкін. ОП-ны жоспарлау нәтижесінде біз ОП-ны пайдалануды және компьютердің оның пайдаланушыларына реакция жылдамдығын жақсарта аламыз. Алайда, өнімділікті арттыруды іске асыру үшін біз көптеген процестерді жадымызда сақтауымыз керек, яғни біз жадымызды бөлісуіміз керек. Бұл тарауда жадты басқарудың түрлі тәсілдері туралы баяндалады. Жадты басқару алгоритмдері компьютерсіз жұмысқа қарапайым көзқарастан пейджингті пайдаланатын стратегияға дейін өзгереді. Әрбір тәсіл өз артықшылықтары мен кемшіліктеріне ие. Нақты жүйе үшін жадты басқару әдісін таңдау көптеген факторларға, әсіресе жүйенің аппараттық дизайнына байланысты. Біз көріп отырғанымыздай, алгоритмдердің көпшілігі аппараттық қолдауды талап етеді, бұл көптеген жүйелердің аппараттық қамтамасыз ету жады мен операциялық жүйенің тығыз интеграцияланған басқаруына алып келеді.

Негізгі жад операциялық жүйені де, пайдаланушының әртүрлі процестерін де қамтуы керек. Сондықтан, біз негізгі жадты тиімді түрде бөлуіміз керек. Бұл бөлімде жадыны іргелес бөлудің бір әдісі түсіндіріледі. Жад әдетте екі бөлімге бөлінеді: біреуі амалдық жүйеге және біреуі пайдаланушы процестеріне арналған. Операциялық жүйені төмен жад адрестерінде де, жадтың адрестерінде де орналастыра аламыз. Бұл шешім көптеген факторларға байланысты, мысалы, үзіліс векторының орналасуы. Алайда, көптеген операциялық жүйелер (соның ішінде Linux және Windows) амалдық жүйені үлкен көлемде жадқа қояды, сондықтан біз тек осы жағдайды талқылаймыз. Біз әдетте бірнеше қолданушы процедуралары бір уақытта жадта болғанын қалаймыз. Сондықтан, біз жадқа кіруді күткен процестерге қол жетімді жадыны қалай бөлу туралы ойлануымыз керек. Жадыны сабақтастыра отырып, әр процесс келесі процесті қамтитын бөлімге іргелес жадының бір бөлімінде болады. Осы жадыны бөлу схемасын талқылауды жалғастырмас бұрын, жадыны қорғау мәселесін қарастырған жөн.

3.2.1 Жадыны қорғау Бұрын талқыланған екі идеяны біріктіру арқылы біз өзімізге жатпайтын жадқа қол жеткізе алмауымыз мүмкін. Егер бізде қозғалыс регистрі бар жүйе (3.1.3 бөлімі) шектеу регистрімен бірге болса (3.1.1 бөлімі), біз мақсатымызға жетеміз. Жылжыту регистрі ең кіші физикалық адрес мәнін қамтиды; шекті регистрде логикалық адрестердің диапазоны бар (мысалы, move = 100040 және limit = 74600). Әрбір логикалық адрес лимиттік регистрмен белгіленген ауқымда болуы керек. MMU қозғалыс регистріне мән қосу арқылы логикалық мекен-жайды динамикалық түрде бейнелейді. Бұл салыстырылған адрес жадыға жіберіледі (3.6-сурет). Процессорды жоспарлаушы іске қосылатын процесті таңдағанда, диспетчер жылжу мен шектеу регистрлерін контексттік ауыстырғыштың бөлігі ретінде дұрыс мәндермен жүктейді. Орталық регистрлердің көмегімен жасалған кез-келген мекен-жай тексерілетін болғандықтан, біз операциялық жүйені де, басқа қолданушылардың бағдарламаларын да, деректерін де осы іске қосылу процесінің өзгеруінен қорғай аламыз. Орналасу - бұл амалдық жүйенің көлемін өзгертудің тиімді әдісін динамикалық түрде қамтамасыз ететін регистр. Мұндай икемділік көптеген жағдайларда қажет. Мысалы, амалдық жүйе құрылғы драйверлеріне арналған кодтық және буферлік кеңістіктен тұрады. Егер құрылғының драйвері қазіргі уақытта пайдаланылмаған болса, оны жадта сақтаудың мәні жоқ; оның орнына оны қажет болған жағдайда ғана жадқа жүктеуге болады. Сол сияқты, құрылғы драйвері қажет болмай қалса, оны алып тастауға және оның жадын басқа қажеттіліктерге бөлуге болады.

3.2.2 Жадыны бөлу Енді біз жадыны бөлуді өзгерттік. Жадыны бөлудің қарапайым әдістерінің бірі - әр бөлімде дәл бір процесс болуы мүмкін әр түрлі көлемдегі бөлімдерге процестерді тағайындау. Бұл бөлімнің айнымалы бөлімінде операциялық жүйенің схемасы жадтың қандай бөліктері бар және қайсысы бос тұрғанын көрсетеді. Бастапқыда барлық

жады пайдаланушы процестеріне қол жетімді және қол жетімді жадының бір үлкен блогы, саңылау болып саналады. Ақыр соңында, өздеріңіз көргендей, жадыда әртүрлі көлемдегі көптеген саңылаулар бар. Бастапқыда жад толығымен пайдаланылады, 5, 8 және 2 процестерін қамтиды. 8-процес шыққаннан кейін бір іргелес тесік пайда болады. Кейінірек 9-процесс келіп, жад бөлінеді. Содан кейін 5-процесс жылжиды, нәтижесінде екі іргелес емес тесік пайда болады. Жүйеге процестер енген кезде, амалдық жүйе әр процестің жадқа деген қажеттілігін және жадтың қай процестің бөлінетіндігін анықтағанда бар жадының көлемін ескереді. Процесске орын бөлінген кезде, ол жадқа жүктеледі, сонда ол процессор уақытымен бәсекеге түсе алады. Процесс аяқталған кезде ол жадыны босатады, оны амалдық жүйе кейіннен басқа процеске бере алады.

Алдағы процестің талаптарын қанағаттандыру үшін жад жеткіліксіз болғанда не болады? Опциялардың бірі - процесті тоқтату және қате туралы тиісті хабарлама беру. Сонымен қатар, біз мұндай процестерді кезекке қоюға болады. Жад таусылғанда, амалдық жүйе күту кезегін тексеріп, оның күтіліп отырған процестің жад талаптарына сәйкес келетіндігін анықтайды. Жалпы, айтылғандай, бар жад блоктарында жадқа шашыраған әр түрлі көлемдегі саңылаулар жиынтығы бар. Процесс келіп, есте сақтау қажет болғанда, жүйе процесс үшін жеткілікті үлкен тесік жиынтығын іздейді. Егер тесік тым үлкен болса, ол екіге бөлінеді. Бір бөлігі өту процесіне бағытталады, қалғандары саңылаулар жиынтығына оралады. Процесс аяқталғаннан кейін, ол тесіктерге қайта салынған жадының блогын босатады. Бұл іргелес саңылаулар бірігіп, тағы біреуін құрайды