Київський національний університет
імені Тараса Шевченка

Дерев'янченко О.В.

ПАРКС-JAVA система для паралельних обчислень на комп'ютерних мережах

Навчальний посібник

Київ
2011

УДК 681.3

Дерев'янченко О.В. ПАРКС-JAVA система для паралельних обчислень на комп'ютерних мережах: Навчальний посібник для студентів факультету кібернетики. – Київ. – 2011. – 60с.

Рецензент:
Завадський І.О., кандидат фізико-математичних наук, доцент факультету кібернетики Київського національного університету імені Тараса Шевченка

Затверджено вченою радою факультету кібернетики, протокол № 4 від 28 листопада 2011 р.

Навчальний посібник призначений для студентів, які вивчають курси «Система ПАРКС», «Паралельне програмування» або споріднені, які читаються на факультеті кібернетики Київського національного університету імені Тараса Шевченка. В ньому розглядаються питання теорії та практики застосування систем паралельних обчислень, а само, системи ПАРКС-JAVA.

В посібнику надаються теоретичні відомості про технологію ПАРКС та систему паралельних обислень ПАРКС-JAVA, що розроблена на факультеті кібернетики Київського національного університету імені Тараса Шевченка. В додатках надаються програмні приклади використання системи ПАРКС-JAVA для паралельних обчислень класичних задач.

ВСТУП

В Україні, як у всьому світі, сьогодні ведуться розробки систем паралельної обробки інформації. Так в Інститут кібернетики ім. Глушкова досліджується застосування кластерних систем та відповідного програмного забезпечення, Інституті програмних систем досліджується проектування систем паралельної обробки інформації, в Київський національний університет імені Тараса Шевченка (ІОЦ та факультет кібернетики) та інші досліджуються GRID-технології, кластерні системи та система паралельних обчислень на комп'ютерній мережі із застосування технології *ПАРКС* (Паралельні Асинхронні Рекурсивно Керовані Системи) .

Основна перевага *системи паралельної обробки інформації* (СПОІ), яка розглядається в роботі та базується на технології *ПАРКС* перед іншими системами розподілених паралельних обчислень (MPI, OpenMP, DVM) полягає в поєднанні ґрунтовної наукової бази та сучасних інформаційних технологій, що дозволять об'єднати розрізнені обчислювальні ресурси будь-якої організації та використати їх для вирішення складних обчислювальних задач. На відміну від аналогів дана технологія дозволяє об'єднувати в єдину систему різноманітні апаратні засоби, що працюють під управлінням різних операційних систем.

Проблемам моделювання та побудови систем паралельної обробки інформації були присвячені роботи [1, 13, 14, 23-26, 27] Глушкова В.М., Анісімова А.В., Дорошенка А.Ю., Летичевського О.А. та ін. Їх дослідження сприяли як математичному вдосконаленню традиційних алгоритмів паралельної обробки інформації, так і появі відповідних програмних систем [2, 3, 15].

На початку 80-х років XX століття виникла концепція керуючого простору (КП) [12]. Це була спроба створити метод опису і керування функціонуванням систем, які складаються з взаємодіючих паралельних

обчислювальних процесів, склад та зв'язки між якими можуть динамічно змінюватись. Протягом останніх двадцяти років розуміння ролі та способів використання КП зазнало еволюційного розвитку. На основі цих досліджень Анісімовим А.В. було запропоновано технологію ПАРКС (Паралельні Асинхронні Рекурсивно Керовані Системи) [1-4], як універсальний засіб побудови СПОІ. Проблематикою створення СПОІ займаються впливові світові дослідницькі наукові центри у різних куточках світу. На сьогодні запропоновано досить багато програмних засобів різного рівня для опису паралельно функціонуючих та взаємодіючих систем. Базовими серед них є MPI [38, 39], MPICH [41], HPF [35], OpenMP [42], DVM [32], PVM[43], mpC [37, 40] та інші[44, 45].

Для дослідження практичного застосування СПОІ була створена система підтримки паралельних обчислень ПАРКС-JAVA [6-9].

Застосування СПОІ є надзвичайно широким і охоплює усі сфери соціально значущого розподіленого інформаційного простору. Це передусім задачі економічного характеру, проблеми технологічного передбачення, задачі економічного прогнозу, що особливо важливо для нашої економіки. Наші екологічні проблеми потребують постійного і всебічного моніторингу. Важливими є задачі розвитку наукових досліджень, питання ефективного використання супутників Землі для розвитку народного господарства, проблеми проектування складних машин (літаків, енергетичних котлів, ракетної техніки), задачі військового комплексу.

1. Система паралельної обробки інформації

1.1.Основні поняття та означення систем паралельної обробки інформації

Паралельні системи (ПС) характеризуються паралельним функціонуванням і взаємодією складових об'єктів. У ПС з динамічно змінюваною структурою об'єкти можуть динамічно створюватися й знищуватися, а зв'язки між ними - динамічно перекомутовуватися.

Для таких систем необхідні програмні засоби адекватного опису динамічного паралелізму. Цікавий набір подібних засобів надає мова паралельного програмування Ада. У мовах паралельного програмування, реалізованих на трансп’ютерних комплексах (OCCAM, Parallel C. Parallel Fortran і т.д.), таких засобів практично немає. Основний наголос у них зроблений на розв’язання задач із статичним паралелізмом. Найпростіші засоби динамічного паралелізму в цих мовах підтримуються апаратом паралельних галузей (Parallel Execution Threads) і процесорних ферм (Processor Farms).

Для підвищення продуктивності, зазвичай, збільшується швидкодія послідовних архітектур Дж. фон Неймана шляхом застосування більш сучасної технології на основі, наприклад, мініатюризації. Але цей підхід нереалістичний, якщо не буде здійснено науковий прорив, так як технологія досягла тієї границі, коли невелике підвищення продуктивності досягається великими затратами, що ілюструють сучасні суперкомп’ютери. Архітектура Дж. фон Неймана з одним процесором має границю, яка визначається швидкістю поширення електричного сигналу по фізичним лініям зв’язку між компонентами.

Альтернативою послідовним процесорам є архітектури, розробка яких базується на паралелізмі: дані та алгоритми розподіляються між процесорами.

Концепція керуючих просторів та система програмування ПАРКС (Паралельні Асинхронні Рекурсивно Керовані Системи) [1-4] створені

спеціально для використання в мультипроцесорних системах. Продуктивність лінійно збільшується при збільшенні кількості процесорів, що підключені.

Для цього в роботі використовується концепція керуючого простору, розроблена на початку 80-х років в роботах Глушкова В.М. та Анісімова А.В. Вона дозволяє описувати архітектуру будь-якої обчислювальної системи, що складається з довільної динамічної множини паралельно асинхронно діючих процесів. Застосування керуючого простору в роботі дозволило побудувати якісну, ефективну модель системи паралельних обчислень, що дозволяє спростити розв'язання багатьох актуальних задач проектування сучасних обчислювальних систем.

Введемо декілька означень. Під *процесом* або *задачею* (*process*) будемо розуміти код, що виконується у захищеній операційною системою області пам'яті. *Потік* (*thread*) відрізняється від процесу тим, що він не має своєї, захищеної за допомогою засобів операційної системи, області пам'яті, а отже значно легше зберегти текучий стан потоку, ніж процесу при перемиканні задач.

Під *віртуальним паралелізмом* (*virtual parallelism*) будемо розуміти максимальну кількість логічно-незалежних процесів, які задача може породити. В той самий час *фізичний паралелізм* (*physical parallelism*) задачі – це реальна кількість процесів, що виконуються незалежно один від одного на конкретній паралельній платформі.

Паралельний комп'ютер – це набір процесорів, що можуть працювати спільно для вирішення обчислювальних задач. Це визначення достатньо загальне, щоб охопити паралельні суперкомп'ютери із сотнями або тисячами процесорів, мережі персональних ЕОМ, багатопроцесорні робочі станції, та інші паралельні системи. Паралельні комп'ютери цікаві тим, що вони дозволяють концентрувати великі обчислювальні ресурси для вирішення важливих обчислювальних задач. Паралелізм зовсім недавно був екзотичною областю комп'ютерної науки. Сьогоднішній напрямок розвитку архітектури

комп'ютерів і мереж показує, що тепер це не так. Паралелізм і паралельне програмування стає центральною ланкою при розробці програм.

Продуктивність найбільш швидких комп'ютерів росте експоненційно з 1945 року до наших днів, у середньому в 10 разів кожні п'ять років. Немає сумнівів, що цей ріст буде продовжуватися. Однак, для забезпечення цього зростання архітектура комп'ютерів радикально змінюється – від послідовної до паралельної.

Важливою тенденцією, що змінює обличчя комп'ютерів, є величезне зростання можливостей комп'ютерних мереж. Лише недавно, високошвидкісні мережі працювали на швидкості 10 Mbit у секунду; зараз пропускна здатність у 100 і 1000 Mbit у секунду стала звичайною справою. Значно зросла і їхня надійність. Ці досягнення дозволяють розробляти додатки, що використовують процесори на множині віддалених комп'ютерів. Ця область паралельних обчислень називається *розподілені обчислення*. Основною задачею розробки програм, що можуть працювати на множині комп'ютерів водночас, є задача паралельних обчислень. У цьому відношенні, різні поняття паралельний і розподілений збігаються.

Тенденції розвитку архітектур і мереж ЕОМ наводить на думку, що паралелізм охоплює не тільки суперкомп'ютери, але і робочі станції, персональні комп'ютери і мережі. Постачальники традиційних комерційних суперкомп’ютерів (SMP, MPP, паралельних векторних) досить швидко поліпшують продуктивність, надійність і простоту використання своїх продуктів. Однак у цих комп'ютерів є один великий недолік - ціна, часом недосяжна для багатьох освітніх і науково-дослідних організацій. Однак потреба в обчислювальних ресурсах у цих організацій велика. Варто мати на увазі, що продуктивність персональних комп'ютерів на базі процесорів Intel в останні роки також значно виросла. Такі комп'ютери стали створювати серйозну конкуренцію робочим станціям на базі RISC, особливо по показнику ціна/продуктивність. Виникла ідея створювати паралельні обчислювальні

системи - *кластери*, що складаються із загальнодоступних комп'ютерів на базі Intel і недорогих Ethernet-мереж. Виявилося, що на багатьох класах задач і при достатньому числі вузлів такі системи дають продуктивність, порівнянну із суперкомп'ютерами.

Основу математичної моделі паралельного алгоритму складає *граф потоків даних* (ГПД). У потоковій моделі паралельна програма представляється набором обчислювальних вузлів - підзадач (далі *задач*), що мають фіксовану кількість інформаційних входів і виходів. Вузли виконують ті самі операції з різною інформацією, що надходить на входи, і видають результати на свої виходи у вигляді числової інформації. Ці входи і виходи називаються відповідно вхідними і вихідними портами. Вузли з'єднані між собою каналами передачі інформації, вихідні порти з'єднані з вхідними. У реальній системі вузли визначаються програмами, а канали – потоками байтів даних.

Дві ці абстракції – *задача* і *канал* – досить добре відповідають вимогам моделі паралельного програмування: механізм, що дозволяє ясно виразити паралелізм і полегшити розробку масштабуємих і модульних програм, абстракції, з якими легко працювати, і які відповідають архітектурній моделі паралельного комп'ютера.

При розгляді ПОС важливо визначити методи доступу до розподілених даних. Найпростішими методами доступу до розподілених даних є взаємовиключення (MUTEX – MUTual EXclusion) та критична секція (critical section). Коли один процес змінює змінну, що характеризує розподіл між процесами, яким необхідно змінити цю змінну в той самий час, необхідно зачекати, а коли записуючий процес закінчить своє звернення до змінної, буде дозволено продовжити роботу одному з процесів, що знаходяться в стадії очікування. Таким чином, кожний процес, що звертається до розподілених даних, виключає можливість для всіх інших процесів одночасного з ним звернення до цих даних. Це зветься *взаємовиключенням*.

Взаємовиключення необхідне лише в тих випадках, коли процеси виконують звернення до загальних даних – якщо ж вони виконують операції, що не призводять до конфліктних ситуацій, то вони повинні мати можливість працювати паралельно. Коли процес виконує звернення до даних, що розподіляються, то кажуть, що він знаходиться в своїй *критичній секції*. Очевидно, що коли один процес знаходиться в своїй критичній секції, необхідно виключити можливість входження для всіх інших процесів (в крайньому разі, для тих, що звертаються до тих же самих даних, що розподіляються) в їх критичні секції.

Якщо процес, що знаходиться в своїй критичній секції, завершується або природнім, або аварійним чином, то операційна система при виконанні службових функцій після завершення процесів повинна відмінити режим взаємовиключення для того, щоб інші процеси мали можливість входити у свої критичні секції.

Семафор (*semaphore*) – це захищена змінна, значення якої можна опитувати та змінювати лише за допомогою спеціальних операцій P , V та операції ініціалізації, яку ми назвемо initSemaphore. *Двійкові семафори* (*binary semaphore*) можуть приймати лише значення 0 і 1. *Семафори, що рахують* (*counting semaphore*) можуть приймати невід'ємні цілі значення.

Операція P над семафором S записується як P(S) і виконується наступним чином:

якщо S > 0

то S := S – 1

інакше (очікувати на S).

Операція V над семафором S записується як V(S) і виконується наступним чином:

якщо (один чи більше процесів очікують на S)

то (дозволити одному з цих процесів продовжити роботу)

інакше S := S + 1.

Ми будемо вважати, що черга процесів, що очікують на S, обслуговується за дисципліною FIFO – first in – first out (перший прийшов - перший вийшов). Операції P(S) та V(S) є неподільними.

Семафори, що рахують, використовуються тоді, коли деякий ресурс виділяється із пулу ідентичних ресурсів. При ініціалізації подібного семафора в його лічильнику вказується кількісний показник об'єму ресурсів пулу. Кожна операція P викликає зменшення значення лічильника семафора на 1, показуючи, що деякому процесу для використання виділено ще один ресурс з пулу. кожна операція V викликає збільшення значення лічильника семафора на 1, показуючи, що процес повернув до пулу ресурс, і цей ресурс може бути виділено тепер іншому процесові. Якщо робиться спроба виконати операцію P, коли в лічильнику семафора вже нуль, то відповідному процесу доведеться зачекати до того моменту, коли в пул буде повернуто ресурс, що звільнився, тобто доти, доки не буде виконано операцію V.

Монітор (*monitor*) – це механізм організації паралелізму, який містить як дані, так і процедури, необхідні для реалізації динамічного розподілення конкретного загального ресурсу чи групи загальних ресурсів. Необхідність входу в монітор може виникати в різні моменти часу у багатьох процесів. Але вхід у монітор перебуває під жорстким контролем – тут відбувається взаємовиключення процесів, так що в кожний проміжок часу лише одному процесу дозволяється вхід у монітор.

Якщо процес звертається до деякої процедури монітору та виявляється, що відповідний ресурс вже зайнятий, ця процедура видає команду Wait. Із часом процес, що займав даний ресурс звернеться до монітору щоб повернути цей ресурс системі. В цьому випадку монітор виконує примітив *сповіщення* (*сигналізації*) Signal, щоб один з процесів, що очікує, мав змогу зайняти даний ресурс.

Щоб гарантувати, що процес, що знаходиться в стані очікування деякого ресурсу, міг дійсно із часом отримати цей ресурс, процес, що очікує, має більш

високий пріоритет, ніж новий процес, який ще тільки намагається увійти у монітор.

Іноді у процесів може виникнути бажання та необхідність знаходитись в режимі очікування поза межами монітору. Для цього було введено поняття *змінної-умови* (*conditional variable*). Для кожної окремо взятої причини, з якої процес може бути переведений в стан очікування, призначається своя умова. В зв'язку з цим, команди очікування модифікуються – до них включаються імена умов:

wait (ім'я умови)

signal (ім'я умови)

Коли означається змінна-умова, заводиться черга. Процес, що видав команду очікування, включається в чергу, а процес, що видав команду сигналізації, тим самим дозволяє процесу, що очікує, вийти з черги та ввійти в монітор.

Система паралельної обробки інформації (СПОІ) – це обчислювальна система паралельної дії, що забезпечує одночасне виконання операцій за однією, або кількома програмами. Вона складаються з трьох основних складових: процесорів (процесорних блоків), блоків пам'яті та мережі каналів (комутації). Процесори - це іноді спеціально розроблені спрощені процесори, а іноді ті ж самі процесори, що використовуються в машинах з одним процесором.

Основною метою створення СПОІ є одержання високих показників продуктивності (потужності) обчислень.

Такими показниками продуктивності є:

- *швидкодія*, що вимірюється, як правило, в кількості операцій за секунду і визначає обчислювальну потужність паралельної системи; найбільш важливою одиницею вимірювання продуктивності є 1Мфлопс = 1000000 операцій з плаваючою точкою за секунду;

- *пропускна здатність* – кількість інформації (трансакцій, запитів, команд), оброблюваної мультипроцесорною системою за одиницю часу; може вимірюватись (Мбайт/с) пропускною здатністю каналів зв'язку мультипроцесорної системи;
- *загальний обсяг операційної пам'яті* паралельної системи (Мбайт).

1.2. Архітектури систем паралельних обчислень

При розгляді класифікацій типів архітектур паралельних обчислювальних систем [22] головними характеристиками архітектури є:

- *Кількість і якість процесорів ПОС*; системи з кількістю процесорів більше ста прийнято називати масово паралельними. ПОС, що складаються з однакових процесорів, називаються однорідними (гомогенними), а інші – різнорідними (гетерогенними) ;
- *Вид основної пам'яті* – характеризує доступ до основної пам'яті з боку процесорів: спільна пам'ять для всієї ПОС і рівнодоступною для всіх процесорів; розподілена пам'ять розбита на окремі ділянки (за кількістю процесорів), доступ до кожної з яких має тільки один процесор, тому така пам'ять є локальною;
- *Система зв'язку між процесорами* – характеризує топологію зв'язку процесорів ПОС, прикладами якої є : шинний зв'язок; статична топологія (лінійна, кільцева, зіркова, матрична, у вигляді тору, повнозв'язна, гіперкуб – 2^n процесорів мають попарні зв'язки довжиною n), динамічна топологія (із змінними зв'язками), комутатори;
- *Спосіб керування*: синхронний (централізований), коли команди у всіх процесорах ПОС виконуються за єдиним сигналом від центрального блоку керування, і асинхронний (розподілений), коли виконання команди в процесорах не синхронізуються.

Паралельні обчислювальні системи будемо класифікувати за типами потоків команд та даних у центральній частині обчислювальної системи.

З точки зору потоку команд будемо розрізняти чотири класи обчислювальних систем, які були наведені Флінном [33, 34]:

SISD – Single Instruction Single Data

SIMD – Single Instruction Multiple Data

MISD – Multiple Instruction Single Data

MIMD – Multiple Instruction Multiple Data

Перший клас систем – це SISD (Одна Інструкція Одні Дані), класичні послідовні ЕОМ з одним процесором. За принципом Дж. фон Неймана: вони виконують операцію, що подана на шину команд, над операндами, що знаходяться на шинах даних.

Інший важливий клас паралельних систем – це SIMD (Одна Інструкція Багато Даних), векторні й матричні ЕОМ. В цих системах кожний процесор виконує один потік інструкцій, але розповсюджує ці інструкції для виконання процесорами даних. Підмножини процесорних модулів можуть пропускати виконання команд, що визначається за допомогою операцій маскування. В системах такого типу реальна швидкість обробки інформації залежить від можливості навантажити процесорні модулі.

Вони складаються з тисяч процесорних елементів, що можуть виконати деяку спільну інструкцію над даними, що в них містяться. Дуже часто виникають ситуації, коли треба виконати якусь операцію над багатьма даними (наприклад, скласти дві матриці великої розмірності); у таких випадках SIMD-система дає суттєвий приріст у швидкості.

У системах класу MISD (Багато Інструкцій Одні Дані) – конвеєрні ЕОМ – процес обробки розбивається на декілька етапів, кожному з яких відповідає один з процесорних модулів. Ці модулі складають у сукупності магістраль обробки – конвеєр процесорів. Реальна швидкість обробки залежить від можливості заповнення магістралі. Найбільш висока швидкість обробки

досягається при виконанні довгих великих ланок програм з однорідними операціями. При частих припиненнях лінійних ланок програм командами розгалуження швидкість обробки сповільнюється.

Ці системи виконують кілька операцій над даними: наприклад, скласти два числа, зсунути результат на один розряд праворуч та помножити на третє число. Такі системи дозволяють отримати деякий виграш у виконанні, але є досить спеціалізованими системами та не отримали значного поширення. Деякі підходи до таких систем використовуються при конвеєризації SISD-структур.

Системи четвертого класу MIMD (Багато Інструкцій Багато Даних) – багатопроцесорні ЕОМ – є найбільш розповсюдженими і лише ними ми будемо займатись у подальшому. Ці системи є цілком паралельними, тобто декілька керуючих пристроїв одночасно керують виконанням декількох ланок програми.

MIMD-системи представляють подальший розвиток обчислювальної техніки: вони складаються з декількох процесорів, кожен з яких функціонує за власною програмою. Така структура призначена для серверних завдань (дозволяє кільком користувачам одночасно підключатися до системи та запускати на виконання кожен власну програму), складних задач, коли потрібне розпаралелювання частин програми, або для використання її як SIMD-системи.

З розвитком обчислювальної техніки з метою підвищення потужності найбільш розповсюджених серед звичайних користувачів SISD-систем до них починають використовувати підходи конвеєризації та розпаралелювання, характерні для MIMD-систем. Нові системи складаються з кількох (найчастіше – двох) процесорів та забезпечуються швидкодіючими механізмами внутрішньої комунікації між системними компонентами (пам’ять, накопичувачі, пристрої вводу/виводу).

Але все ж самі системи SIMD та MIMD з тисячами процесорів мають значно більшу обчислювальну потужність та находяться поза межами конкуренції у складних наукових розрахунках.

Системи SIMD та MIMD за типом зв'язку між процесорами поділяються на два підкласи:

MIMD

- тісний зв'язок через загальну пам'ять – багатопроцесорна ЕОМ;
- розподілена обчислювальна система з не тісним зв'язком через мережу комутацій та обміном повідомленнями.

SIMD

- векторна ЕОМ без безпосереднього зв'язку між процесорними елементами;
- матрична ЕОМ зі зв'язком між елементами через комутаційну мережу.

Класифікація Флінна корисна, але не повна. Вона не включає, наприклад, так звані систолічні архітектури – набори зв'язаних між собою однорідних процесорів (часом дуже простих), що синхронно виконують різні програми.

Система ПАРКС-JAVA, що розглядається в даній роботі, належить до класу MIMD за класифікацією Флінна.

1.3 Моделі паралельних обчислень

Паралельне програмування представляє додаткові джерела складності - необхідно явно керувати роботою тисяч процесорів, координувати мільйони міжпроцесорних взаємодій. Для того вирішити задачу на паралельному комп'ютері, необхідно розподілити обчислення між процесорами системи так, щоб кожен процесор був зайнятий вирішенням частини задачі. Крім того, бажано, щоб як можна менший обсяг даних пересилався між процесорами, оскільки комунікації є значно більш повільні операції, ніж обчислення. Часто,

виникають конфлікти між ступенем розпаралелювання й об'ємом комунікацій, тобто між чим більшим числом процесорів розподілена задача, тим більший обсяг даних необхідно пересилати між ними. Середовище паралельного програмування повинне забезпечувати адекватне керування розподілом і комунікаціями даних.

Розглянемо основні моделі паралельного програмування:

Процес/канал (*Process/Channel*)

У цій моделі програми складаються з одного чи більш процесів, розподілених по процесорах. Процеси виконуються одночасно, їхнє число може змінитися протягом часу виконання програми. Процеси обмінюються даними через канали, що являють собою односпрямовані комунікаційні лінії, що з'єднують тільки два процеси. Канали можна створювати й видаляти.

Обмін повідомленнями (*Message Passing*)

У цій моделі програми, можливо різні, написані на традиційній послідовній мові виконуються процесорами комп'ютера. Кожна програма мають доступ до пам'яті виконуючого її процесора. Програми обмінюються між собою даними, використовуючи підпрограми прийому/передачі даних деякої комунікаційної системи. Програми, що використовують обмін повідомленнями, можуть виконуватися тільки на MIMD комп'ютерах.

Паралелізм даних (*Data Parallel*)

У цій моделі єдина програма задає розподіл даних між усіма процесорами комп'ютера й операції над ними. Даними, що розподіляються зазвичай є масиви. Як правило, мови програмування, що підтримують дану модель, допускають операції над масивами, дозволяють використовувати у виразах цілі масиви, вирізки з масивів. Розпаралелювання операцій над масивами, циклів обробки масивів дозволяє збільшити продуктивність програми. Компілятор відповідає за генерацію коду, що здійснює розподіл елементів масивів і обчислень між процесорами. Кожен процесор відповідає за ту

підмножину елементів масиву, що розташована в його локальній пам'яті. Програми з паралелізмом даних можуть бути відтрансльовані й виконані як на MIMD, так і на SIMD комп'ютерах.

Спільної пам'яті (*Shared Memory*)

У цій моделі всі процеси спільно використовують спільний адресний простір. Процеси асинхронно звертаються до спільної пам'яті як із запитами на читання, так і із запитами на запис, що створює проблеми при виборі моменту, коли можна буде помістити дані в пам'ять, коли можна буде видалити їх. Для керування доступом до спільної пам'яті використовуються стандартні механізми синхронізації - семафори і блокування процесів.

До моделі процес/канал відноситься і технологія ПАРКС, розроблена на факультеті кібернетики ак. Глушковим В.М., проф. Анісімовим А.В. та доц. Кулябко П.П., доц. Дерев'янченком О.В.

1.4. Порівняння сучасних системи паралельної обробки інформації

Сьогодні проблематикою створення СПОІ займаються впливові світові дослідницькі наукові центри у різних куточках світу. На сьогодні запропоновано досить багато програмних засобів різного рівня для опису паралельно функціонуючих та взаємодіючих систем. Базовими серед них є MPI [38, 39], MPICH [41], HPF [35], OpenMP [42], DVM [32], mpC [37, 40] та інші[57].

Переваги використання того або іншого підходу будемо визначати за наступними факторами:

- легкість програмування;
- ефективність розроблених програм;
- можливість переносу і повторного використання програм;
- якість засобів налаштування програм.

Розглянемо та порівняємо деякі з них, що найбільш поширені сьогодні.

На підставі проведеного аналізу різних підходів до розробки паралельних програм для обчислювальних кластерів і мереж ЕОМ (MPI, HPF, OpenMP, DVM й mpC) можна зробити наступні висновки:

1. З погляду простоти розробки паралельних програм та їхнього повторного використання явну перевагу мають підходи HPF, OpenMP, DVM та mpC. Для оцінки складності програмування цілком підходять дані [56] про співвідношення кількості додаткових операторів при розпаралелювання тестів NPB 2.3. Слід зазначити при цьому, що додаткові оператори DVM-програми є простими спецкоментарями, що не залежать від розмірів масивів і числа процесорів. Додатковий код MPI-програми являє собою складну систему програм керування передачею повідомлень, що залежать від розмірів масивів і числа процесорів. Дані для HPF й OpenMP повинні бути близькі по цьому показнику до DVM-підходу.
2. По ефективності виконання програм HPF помітно відстає від інших підходів.
3. OpenMP обмежує можливість переносу програм, але є білш простим у використанні.
4. Гібридні підходи OpenMP+MPI чи MPICH+MPI, як самої MPI, не може забезпечити ефективного виконання програм на неоднорідних кластерах і мережах ЕОМ.
5. Найбільш гнучким, що може переноситися, і загальноприйнятим інтерфейсом програмування є MPI (інтерфейс передачі повідомлень). Однак модель передачі повідомлень по-перше недостатньо ефективна на SMP-системах; по-друге відносно складна в опануванні, тому що вимагає мислення в "специфічних" термінах.

Таким чином, жоден із підходів (MPI, HPF, OpenMP, mpC) не може розглядатися в цей час, як той, що цілком підходить для розробки паралельних

програм для обчислювальних кластерів і мереж ЕОМ. Цю точку зору розділяють багато спеціалістів в області паралельних обчислень [55].

Розглянута в роботі у розділах 3 та 4 мова ПАРКС-JAVA в чомусь схожа на мови системи DVM та втілює кращі засоби інших технологій.

Освоєння підходу, що пропонується у розділах 3 та 4 може істотно скоротити час написання, налагодження й супроводу програм. А у випадку широкого розповсюдження та накопичення великої кількості різних алгоритмічних модулів значно спростити роботу програмістам.

2. Засоби побудови паралельних обчислювальних систем за технологією ПАРКС

2.1. Керуючий простір та його властивості

Модель асинхронних обчислень, що запропонована в [12, 17] можна розглядати, як мережу алгоритмічних модулів певної природи, в якій уточнені закони взаємодії між модулями та їх структурне розміщення.

Базисний алгоритмічний модуль - це послідовний скінченний дискретний перетворювач (можливо недетермінований), що має керуючу та інформаційну частини, а також засоби взаємодії та синхронізації з іншими модулями.

Для опису дискретного перетворювача необхідно описати його керуючу частину (скінченний автомат, програма на алгоритмічній мові і т.п.), а також початковий стан обробляючого інформаційного середовища. Конкретний опис залежить від вибору реалізації. З базисних алгоритмічних модулів шляхом визначених композицій можна отримувати складені модулі. В якості засобу взаємодії та синхронізації оберемо спосіб комунікації, запропонований Хоаром. Модуль, що розпочав передачу, чекає, поки його повідомлення не буде прийнято. Аналогічно модуль чекає данні, поки не отримує відповідну інформацію. Такий обмін (типу телефонних викликів) дозволяє, окрім обміну даними, досягти бажаної синхронізації. Команди обміну можна виразити наступним чином:

ВВЕДЕННЯ ::= < джерело >?< цільова змінна >

ВИВІД ::= < місце призначення >!< вираз >

< джерело > ::= < ім’я модуля >

< місце призначення > ::= < ім’я модуля >

< ім’я модуля > ::= < ідентифікатор >

< інший спосіб локалізації модуля >

Не будемо уточнювати поняття "інший спосіб локалізації модуля", яке залежить від орієнтації на конкретну систему програмування. Наприклад, Хоар допускав масиви процесів.

В моделі обчислень, що розглядається, модуль, який має ім'я, може перейти одразу в декілька точок керуючого простору, через те не достатньо знати одне ім'я для однозначної передачі даних. В такому випадку необхідно в момент передачі використовувати координати точки. Це відноситься і до "іншого способу локалізації модуля".

При взаємодіях подібного типу опис обміну великими об'ємами інформації стає громіздким та досить важко доступним для огляду. Через це зручно буде ввести додаткові засоби глобального обміну пам'яттю та керуванням. Зробимо це після уточнення поняття керуючого простору.

Керуючий простір (КП) - це множина точок C, що задані разом зі своєю пів-групою переходів Π, $\pi \in \Pi, \pi: C \to C.$ Вважаємо, що в Π виділена система базисних переходів - що утворюють пів-групи Π. Перехід із точки в точку простора C здійснюється тільки за допомогою базисних переходів. Пів-група П називається пів-групою координат. В кожний момент часу в точках керуючого простору виконуються деякі алгоритмічні модулі, базисні чи складені. Таким чином в кожний момент часу процес характеризується функцією s, що відображає C в множину поточних станів обчислень в точках. Модулі можуть мати імена і таким чином здійснювати обмін даними по командам вводу-виводу незалежно від їх розташування в точках простору. При цьому, якщо модулі можуть динамічно породжуватися та передаватися одночасно в декілька точок простору, то для одночасної локалізації процесів при вводі-виводі необхідно використовувати координати точок. Окрім обмінів подібного типу, вважаємо, що повинні бути передачі інформаційного середовища та керування з точки в точку керуючого простору C, тобто обчислення, локалізоване в точці c, може передаватися в іншу точку $(c)g$, де $g \in \Pi$. Таким чином, алгоритмічні модулі можуть змінювати своє

місцезнаходження в керуючому просторі. При одночасній передачі декількох обчислень або середовищ в одну й ту ж точку c керуючого простору здійснюється вибір інформації по закону, що залежить від конкретної реалізації. Іншими словами, кожна точка керуючого простору має свій буфер черг, де зосереджуються дані, що підлягають прийому. Вважається також, що інформаційне середовище алгоритмічного модуля належить точці простору, де знаходиться цей модуль, та залишається там після того, як модуль перейде в іншу точку або закінчить свою роботу. Вся інформація передається по програмним каналам (ПК), що з'єднують відповідні точки КП.

Розглянемо на неформальному рівні декілька прикладів:

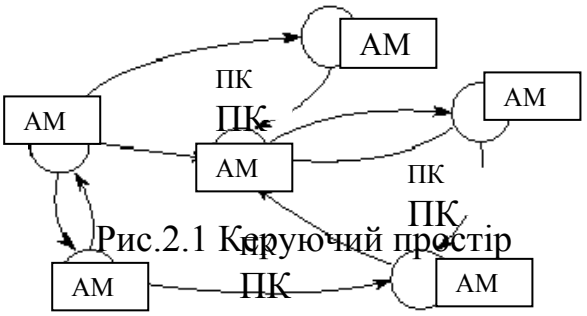


Рис.2.1 Керуючий простір

1. *Блок-схеми.* Звичайне представлення блок-схемою послідовного детермінованого алгоритму задає керуючий простір, що співпадає з графом блок-схеми. Можна вважати, що в кожній точці зосереджена керуюча частина алгоритмічного модуля, а пам’ять передається з точки в точку після обробки відповідного модуля.
2. *Модель Дейкстри-Хоара.* В цьому випадку керуючим простором є сукупність ізольованих точок, що мають імена. Допускаються тільки прості обміни, а модулі не змінюють свого місцезнаходження.
3. *Матричні обчислення.* Керуючий простір співпадає з решіткою із цілих чисел. Пів-група координат - $Z \times Z$, де Z - група цілих чисел.

2.2. Композиції алгоритмічних модулів

Розглянемо тепер композиції алгоритмічних модулів та операції взаємодії їх із керуючим простором. Поняття базисного алгоритмічного модуля уточнювати не будемо, так як воно залежить від орієнтації на конкретну реалізацію. Необхідно пам'ятати, що базисний алгоритмічний модуль - це послідовний (можливо недетермінований) алгоритм, в якому виділено керування та оброблюване середовище. Крім того, в керуючій частині допускаються команди вводу-виводу, що розглядалися вище.

Шляхом певних композицій з базисних алгоритмічних модулів можна побудувати більш складні складені алгоритмічні модулі.

Послідовне виконання - $A_1; A_2$.

Виконується модуль A_1. Після того як A_1 закінчить свою роботу, починає виконуватись модуль A_2 над інформаційним середовищем, яке залишилося після A_1.

Налаштовування середовища - $A_1.A_2$.

В цьому випадку після виконання роботи модуля A_1 модулю A_2 передається тільки деяка необхідна частина інформації від середовища, яке залишилось після A_1, тобто виконується передача параметрів від A_1 до A_2. Ці деталі фіксуються в описі конкретної мови програмування.

Паралельне функціонування - $A_1 \oplus A_2$.

Модулі A_1 та A_2 незалежно функціонують, перетворюючи свої власні інформаційні середовища. При цьому якщо A_1 та A_2 мають імена або < інший спосіб локалізації модуля >, то A_1 та A_2 можуть обмінюватися даними та сигналами при допомозі команд вводу-виводу.

Умовна конструкція. ЯКЩО $p_1 \to A_1 \diamond \ldots \diamond p_k \to A_k$ КІНЕЦЬ ЯКЩО.

Усі модулі $A_1, A_2 \ldots A_k$ повинні бути інформаційно сумісними, тобто можуть обробляти одне й те ж середовище. Отримавши інформаційне

середовище, умовна конструкція оброблює її за допомогою модуля A_i такого, що умова p_i на цьому середовищі вірна. Вибір A_i недетермінований.

Циклічна конструкція. ЦИКЛ $p_1 \to A_1 \Diamond .. \Diamond p_k \to A_k$ КІНЕЦЬ ЦИКЛУ.

Циклічну конструкцію можна визначити як циклічне виконання конструкції ЯКЩО $p_1 \to A_1 \Diamond .. \Diamond p_k \to A_k$ КІНЕЦЬ ЯКЩО до тих пір, поки всі умови $p_i, p_2, ..., p_k$ не стануть "брехня".

Крім вказаних простих композицій, складений алгоритмічний модуль у своїй керуючій частині може містити команди взаємодії з керуючим простором.

В кожний момент часу в точках керуючого простору виконується алгоритмічний модуль. Наступна група команд визначає зміни структурного взаємного розміщення модулів у керуючому просторі.

Нехай g - елемент пів-групи координат π керуючого простору C. Тоді якщо в деякій точці c модуль A оброблює інформаційне середовище в стані S, то виконання команди взаємодій з керуючим простором, яка локалізована в A, викликає наступні зміни в обчислювальній обстановці.

ПЕРЕХІД ПО g. В точку $(c)g$ передається для виконання весь модуль A разом із середовищем S. Після вибору його із буфера очікування точки $(c)g$, в ній виконується модуль A над середовищем S, починаючи з команди, що потрапляє після команди виконаного переходу. Після передачі інформаційного середовища S старе середовище в точці $(c)g$ зникає. В точці c зберігається середовище, але точка стає вільною, та в ній може почати виконуватися модуль, взятий із буферу очікування.

ПЕРЕДАТИ КЕРУВАННЯ ПО g. По цій команді алгоритмічний модуль передає в точку $(c)g$ (в її буфер очікування) тільки свою керуючу частину, яка починає виконуватись, починаючи із наступної команди після виконаної команди передачі керування над інформаційним середовищем S', що залишилось в $(c)g$ від попереднього модуля. Середовище S в точці c зберігається і точка c стає вільною.

ПЕРЕДАТИ ПАМ'ЯТЬ ПО g. Все інформаційне середовище в c передається в точку $(c)g$ (буфер очікування). Після вибору із буфера очікування пам'ять у точці $(c)g$ замінюється на пам'ять S. В точці c продовжує виконуватись модуль A.

Іноді бажано передавати не весь модуль, а деяку групу команд. Тому введемо операцію:

ПЕРЕДАТИ КЕРУВАННЯ (A) ПО g, де A - керуюча частина. По цій команді, виконаній в точці c, в $(c)g$ передається керування A. В точці c продовжує виконуватись модуль, що викликав команду ПЕРЕДАТИ КЕРУВАННЯ (A) ПО g, починаючи з наступної команди.

Окрім команд, пов'язаних із пів-групою координат, введемо команди взаємодії з керуючим простором: ПРИПИСАТИ (A), ПОЧАТИ, ЧЕКАТИ, ВІДНОВИТИ, СТЕРТИ.

Команда ПРИПИСАТИ (A), застосована в точці c, приписує їй алгоритмічний модуль A.

Команда ПОЧАТИ активізує точку, в ній починає виконуватись приписаний їй алгоритмічний модуль. Ініціалізація точки може здійснюватись по команді ПЕРЕХІД ПО g та ПЕРЕДАТИ КЕРУВАННЯ (A) ПО g.

Команда ЧЕКАТИ призупиняє роботу модуля в точці. Буфер звернень продовжує накопичувати інформацію, що передається.

Команда ВІДНОВИТИ скасовує дію оператора ЧЕКАТИ.

Оператор СТЕРТИ знищує пам'ять та керування, що приписані точці.

Так як можливе одночасне застосування в одній точці декількох команд взаємодії з керуючим простором, то будемо вимагати в цьому випадку виконання наступного правила узгодження: усі команди повинні мати однаковий ефект відносно точки, в якій вони застосовуються.

Команди взаємодії з керуючим простором можна віднести до групи базисних алгоритмічних модулів. Складений алгоритмічний модуль - це або базисний модуль, або модуль, що отриманий з базисних при допомозі вказаних

композицій. При орієнтації на конкретну архітектуру багатопроцесорної мережі можна вводити різні обмеження на використання керуючих команд та тип керуючого простору. Наприклад, в деяких випадках, можна вважати, що є тільки один модуль (керуючий процес), якому дозволено застосовувати команди: ПРИПИСАТИ , ПОЧАТИ, ЧЕКАТИ, ВІДНОВИТИ, СТЕРТИ.

Приклади застосування композиції алгоритмічних модулів розглянуті в роботі[12].

2.3. Особливості ПАРКС-технології

Паралельні системи (ПС) характеризуються паралельним функціонуванням і взаємодією складових об'єктів. У ПС з динамічно змінюваною структурою (ДЗС) об'єкти можуть динамічно створюватися і знищуватися, а зв'язки між ними - динамічно перекомутовуватися.

Для програмування ПС з ДЗС необхідні програмні засоби адекватного опису динамічного паралелізму. Цікавий набір подібних засобів надає мова паралельного програмування Ада. У мовах паралельного програмування, реалізованих на трансп'ютерних комплексах (OCCAM, Parallel C. Parallel Fortran і т.д.), таких засобів практично немає. Основний наголос в них зроблений на розв’язання задач зі статичним паралелізмом. Найпростіші засоби динамічного паралелізму в цих мовах підтримуються апаратом паралельних галузей (Parallel Execution Threads) і процесорних ферм (Processor Farms).

Система програмування ПАРКС, як і система паралельного програмування на трансп’ютерах це сукупність програмних засобів, що підтримують процес розробки і реалізації алгоритмів паралельної обробки інформації за рахунок розширення базових алгоритмічних мов (JAVA, C, Pascal, Fortran, Modula-2). Відмінність системи програмування ПАРКС полягає в тому, що пропоновані нею програмні засоби сполучать найбільш могутні алгоритмічні концепції - рекурсію і паралельність - і призначені для опису

взаємодій зі змінною комутацією зв'язків і рекурсивно-паралельного розвитку процесів, ядра яких описуються на основі конструкцій базової мови. Акцент робиться на програмні засоби опису динамічного паралелізму.

Для CSP-моделі паралельних обчислень (яка лежить в основі мов ОССАМ і Ada) ПАРКС - засоби забезпечують динамічне створення і знищення паралельних процесів (з використанням рекурсії по даним і рекурсії по керуванню), динамічну перекомутацію зв'язків між процесами.

Іншим важливим критерієм класифікації ПС. крім статичності/динамічності, на нашу думку, варто вважати відкритість чи закритість програмного забезпечення. Ряд розробок і технологій (наприклад, Ада, ОССАМ, МАЯК [28] та ін.) пропонують як базу заново розроблену мову (як правило, дуже складну), змушуючи користувачів її освоювати. Це позиція закритих систем і технологій. Сильною стороною такого підходу є можливість досягнення високої ефективності виконання для визначених класів задач.

Альтернативні відкриті системи і технології (наприклад, LINDA, ПАРКС, Parallel Fortran та ін.) надають користувачу можливість працювати в найбільш придатному для його задач середовищі, доповнюючи його тими чи іншими засобами паралелізму. Крім комфортності для користувача у відкритих системах часто спрощуються і семантичні проблеми, тому що вони концентруються винятково на паралелізмі.

Відмінність системи програмування ПАРКС полягає в тому, що пропоновані нею програмні засоби сполучать найбільш могутні алгоритмічні концепції - рекурсію та паралельність і призначені для опису взаємодій зі змінною комутацією зв'язків і рекурсивно-паралельного розвитку процесів, ядра яких описуються на основі конструкцій базової мови. Акцент робиться на програмні засоби опису динамічного паралелізму.

Основними термінами ПАРКС-технології програмування є:

- керуючий простір (КП);
- точка;

- програмний канал (ПК);
- алгоритмічний модуль (АМ).

ПАРКС-технологія програмування базується на концепції керуючого простору рис.2.1. КП — динамічний граф, за допомогою якого описується логічна й комунікаційна структура досліджуваної задачі (системи) і відображаються динамічні зміни в ній.

Завдяки КП ПАРКС - технологія дозволяє:

♦ підтримувати структурне паралельне програмування на основі як статичного, так і динамічного паралелізму;

♦ використовувати розвинуті засоби опису рекурсії за даними і рекурсії по керуванню, розглядати, у залежності від ситуації, керування як дані, а дані - як керування (з урахуванням можливої ефективної реалізації), забезпечувати за допомогою взаємодій не тільки синхронізацію й комунікацію, але і видозміну керування (на основі засобів базової мови);

♦ на логічному рівні в явному вигляді описувати розподіл ресурсів, схеми комутації і перекомутації зв'язків (з урахуванням рекурсивного розгортання алгоритмів) чи одержувати логічну структуру системи, розподіл ресурсів і схему перекомутації неявно, як результат обробки й побудови КП;

♦ будувати систему віртуальних ПАРКС-машин, що забезпечують можливість паралельної обробки інформації на різних рівнях деталізації; ефективна реалізація отриманої системи віртуальних ПАРКС-машин можлива на широкому класі паралельних і спеціалізованих ЕОМ.

2.4. Моделі керуючого простору та топології мереж

ПАРКС - модель визначається в такий спосіб:

а) виділяються кінцеві набори базових АМ, що описують способи розпаралелювання алгоритмів на різних рівнях деталізації (операцій, операторів, процедур, процесів і т.п.);

б) визначаються правила створення, знищення й взаємодії АМ на всіх рівнях деталізації;

в) уточнюються поняття рекурсії за даними й рекурсії по керуванню для заданої базової алгоритмічної мови.

ПАРКС - модель, фактично, і задається відповідною моделлю КП. В свою чергу, модель КП утворює топологію певної мережі.

Кроком підвищення обчислювальних потужностей ЕОМ є їх об'єднання в паралельні комплекси, мережі. На сьогоднішньому етапі розвитку електронно-обчислювальних систем велика увага приділяється створенню програмного забезпечення не тільки для однопроцесорних ЕОМ, а й для мереж великого розміру. Серед них можна виділити мережі безпосереднього зв'язку, багатостанові розподілені мережі. Для великої кількості існуючих топологій мереж розробляються алгоритми для дослідження задач стійкості, маршрутизації [20, 31], надійності та коректності їх роботи. Поширеною задачею є також пошук сприятливої топології, на якій можна отримати розв'язок певної задачі.

Швидкість багатопроцесорного комплексу та ефективність паралельної обробки даних повністю залежить від методів комутації елементів мережі. Схема повного з'єднання є найкращою для комунікації, оскільки для передачі повідомлення між двома довільними процесорними елементами потрібен лише один такт. Але непрактичність цієї топології полягає у її вимірності: для сполучення n процесорних елементів необхідно використати n^2 зв'язків. Тому схему повного з'єднання для великих мереж не використовують. Схема сполучення зіркою [107] має перевагу в тому, що вона має малу кількість зв'язків і для передачі інформаційного пакета досить використати два з'єднання, але вада її полягає в існуванні вузького горла – центрального

процесора, який буде задіяний майже при будь-якій трансакції. Вузьке горло має і деревоподібна мережа, хоча вона має сприятливу структуру для вирішення багатьох задач.

Деякі з наведених мережевих архітектур, реалізовані у вигляді моделей КП за допомоги мови ПАРКС-JAVA і розглядаються у п'ятому розділі цієї роботи.

2.5. ПАРКС - система програмування

ПАРКС - система програмування — сукупність програмних засобів, що забезпечують процес розробки і реалізації алгоритмів паралельної обробки інформації. Ця система може використовуватися при створенні програмного забезпечення паралельних ЕОМ, а також логічної структури паралельних і спеціалізованих ЕОМ, що забезпечують найбільш ефективну реалізацію розробленого програмного забезпечення. Подібні розробки здійснюються користувачами в звичному для них програмному середовищі, природним чином розширеному досить простим набором відповідних ПАРКС - засобів.

ПАРКС - система програмування пропонує набір програмних засобів розширення базових алгоритмічних мов (JAVA, C, Modula-2, Pascal, Fortran і т.п..) і підтримку отриманого мовного середовища, що має назву відповідно ПАРКС - JAVA, ПАРКС - C, ПАРКС - Modula і т.п.

У своїй основі пропоновані програмні засоби сполучають найбільш могутні алгоритмічні концепції — рекурсію й паралельність і призначені для опису взаємодій і рекурсивно-паралельного розвитку процесів, ядра яких описуються на основі конструкцій базової мови.

Можна з упевненістю стверджувати, що в рамках одної, навіть ретельно розробленої мови програмування не вдасться задовольнити всі категорії користувачів і вирішити всі проблеми. Більш розумним, на наш погляд, є використання базового набору програмних засобів, що дозволяють у рамках широкого класу мов працювати на основі визначеної технології. ПАРКС -

система програмування підтримує ПАРКС - технологію програмування, що ґрунтується саме на цій ідеї. Варіюючи базові алгоритмічні мови, можна одержати сукупність програмних засобів, що охоплюють широкий спектр додатків — від мов проектування спец-процесорів до універсальних мов паралельного програмування. У цьому перша оригінальна особливість ПАРКС - технології.

Концепції КП складає іншу оригінальну особливість ПАРКС - технології. Інші особливості є наслідком згаданих двох: можливість паралельної обробки інформації на взаємозалежних рівнях — від рівня операцій до рівня процесів; уніфікована розробка алгоритмів, орієнтованих на реалізацію в сильно і слабко зв'язаних системах з загальною і розподіленою пам'яттю; можливість проектування й розробки операційних систем для паралельних ЕОМ і т.п.

Таким чином, програма на деякій умовній мові алгоритмічних модулів складається з наступних частин:

1.Опис структури чи типу керуючого простору.

2.Програма налаштування керуючого простору. (При виконанні цієї програми відбувається розподілення алгоритмічних модулів по точкам простору.)

3.Опис локальних алгоритмічних модулів та опис функціонування всього керуючого простору.

2.6. Програмні засоби розширення базових алгоритмічних мов

Алгоритмічні моделі, розроблені засобами ПАРКС - системи програмування, будемо називати ПАРКС - моделями. Розробка ПАРКС - моделей здійснюється на базі ієрархічної структури віртуальних ПАРКС - машин. Розглянемо програмні засоби й можливості, надані кожною віртуальною ПАРКС - машиною для розширення базової алгоритмічної мови.

Рівень 1. Програмні засоби, що забезпечують побудову й модифікацію КП.

КП складається з точок і каналів, що з'єднують точки. У кожній точці КП на наступному рівні ієрархічного представлення може розміщуватися новий КП. При побудові й обробці КП припустиме використання рекурсії за структурою КП. Кожна точка має свій тип, що визначає її властивості й алгоритм функціонування (монітор) для рівня 2. Наприклад, точка може моделювати логічний процесор, розподілюваний ресурс, блок пам'яті і т.п. Точка КП однозначно ідентифікується деяким логічним номером. Канали КП призначені для забезпечення взаємозв'язків між точками. Вони також типізовані. За допомогою типів каналів задається, зокрема, ієрархічна структура КП. Для кожного алгоритму існує адекватна йому структура КП, що відображає його логічну структуру.

Програмні засоби рівня 1 повинні забезпечувати наступні можливості:

— створити нову точку КП заданого типу і визначити для неї новий логічний номер;

— з'єднати точки КП каналом заданого типу;

— визначити, які точки КП приєднані до заданого, і за допомогою яких каналів;

— знищити точку КП і всі канали, що з'єднують її з іншими точками;

— знищити канал заданого типу, що з'єднує точки КП;

— визначити логічний номер і тип точки КП.

Рівень 2. Програмні засоби, що забезпечують збереження й передачу інформації за допомогою КП.

Монітор точки КП, що відповідає її типу, визначає правила збереження, прийому і передачі інформації у точці КП. Крім того, монітор визначає її алгоритм функціонування.

На рівні 2 у рамках каналів КП виділяються нумеровані канали, за допомогою яких уточнюється структура взаємозв'язків між точками КП.

Програмні засоби рівня 2 повинні забезпечувати наступні можливості:

—створити скінченне число моніторів (на основі засобів базової алгоритмічної мови, мови асемблера і т.п.)

— створити блок інформації;

— записати в канал із заданим номером (можлива явна вказівка точки КП, до якої приєднаний даний канал) блок інформації з визначеним пріоритетом;

—перевірити, чи маються в каналі із заданим номером блоки інформації;

—прочитати з каналу із заданим номером черговий блок інформації;

—очікувати запису блоку інформації у кожній з каналів із заданими номерами.

Відповідні одна одній операції запису і читання блоків інформації у точках КП повинні посилатися на канал із тим самим номером. Номер каналу однозначно визначає тип каналу КП, в рамках якого він установлений. Установка номерного каналу здійснюється в рамках каналу КП відповідного типу, що з'єднує розглянуті точки КП у момент передачі інформації. При наявності декількох взаємозалежних каналів заданого типу точок КП і посилань на канал із тим самим номером передача інформації здійснюється по встановлених каналах в усі точки, з'єднані з передавальною, якщо тільки явно не зазначена одна з них. Правила прийому інформації до аналогічної ситуації визначаються монітором точки КП.

Реалізація програмних засобів рівнів 1 і 2 залежать від архітектури машини і її операційної системи. Програмні засоби наступних рівнів і реалізація залежать від базової алгоритмічної мови.

Рівень 3. Програмні засоби, що забезпечують роботу з алгоритмічними модулями.

Поняття алгоритмічного модуля (АМ) призначено для уточнення одиниці паралельної роботи базової алгоритмічної мови. За допомогою АМ здійснюється побудова, модифікація і «наповнення» КП.

Процес функціонування ПАРКС - моделі (що визначена в розділі 2.4.) складається із створення й знищення активних копій АМ, їх частково - децентралізованого асинхронного паралельного функціонування і динамічної

взаємодії (розрізняють опис АМ і активні копії АМ, створені по даному опису). Активні копії АМ можуть створювати (можливо, рекурсивно) нові активні копії АМ, причому припустимі різні зміни, обумовлені ситуаційною обстановкою.

Опис АМ являє собою розширення опису відповідної конструкції базової мови операціями віртуальних рівнів 1-3. Найпростіший опис АМ — це дефініція відповідної конструкції базової мови.

Таким чином, програмні засоби рівня 3 повинні забезпечувати наступні можливості:

— створити, знищити й модифікувати опис АМ;

— створити активну копію АМ і приписати її для виконання в точку КП (логічний номер точки КП визначає динамічне ім'я активної копії АМ; планування виконання активних копій АМ у точках КП здійснюється відповідними моніторами);

— послати повідомлення активної копії АМ (відкрити доступ до розподілених структур даних);

— прийняти повідомлення від активної копії АМ (закрити доступ до розподілених структур даних);

— визначити головну активну копію АМ (із створення якої починається функціонування ПАРКС - моделі);

— здійснити вибіркове очікування й альтернативне виконання в тілі активної копії АМ.

Рекурсія на рівні АМ виявляється у тому, що активні копії АМ можуть створювати нові активні копії по описах АМ (у тому числі по власному зразку), а також нові описи АМ (у тому числі модифікувати власні описи).

Апарат створення, знищення, модифікації, приписування активних копій АМ у точки КП, взаємодії активних копій АМ, обробки даних як керування, а керування — як даних, дозволяє описувати рекурсію за даними, рекурсію по керуванню та їхній комбінації.

Розробка ПАРКС - моделей на рівні 3 передбачає можливість автоматичної побудови КП, що відображує динамічну структуру взаємозв'язків у ПАРКС - моделі, припускаючи наявність необмеженого паралелізму.

Використовуючи програмні засоби рівнів 1 і 2, можна врахувати ряд обмежень, що накладаються на структуру розробленої системи, планувати розподіл ресурсів, прагнути забезпечити ефективне використання обмежених ресурсів і перекомутацію зв'язків.

При розробці моделюючих ПАРКС - комплексів на рівні 3 вводяться також засоби роботи з часом і засоби забезпечення й ведення моделювання.

Рівень 4. Програмні засоби, що забезпечують процедурно - об'єктно - орієнтоване паралельне програмування.

Технологія процедурно та об'єктно-орієнтованого програмування визначається й обмежується базовою алгоритмічною мовою. Якщо взяти за основу об'єктно-орієнтовану Ада-технологію — механізм пакетів і задач, уніфікований виклик процедур пакетів і точок входів задач, поняття рандеву й інші, то на заданій базовій, алгоритмічній мові вона працює наступним чином.

Вибирається програмна конструкція базової мови, що відповідає по структурі поняттю пакета (задачі) у мові Ada. Наприклад, модуль - у мові Modula-2, файл, що містить опис сукупності функцій, — у мові С і т.п. Обрана конструкція розглядається як опис АМ. Розходження між пакетами й задачами визначається на рівні КП при приписці активних копій АМ у точки різних типів (монітори їх задають відповідно правилу обробки звертань до пакетів і задач). Уніфікований виклик процедур і точок входів транслюється на рівень 3, де розписується за допомогою програмних засобів, що забезпечують взаємодію активних копій АМ.

На наступних рівнях розробляються програмні засоби, використання яких забезпечує автоматичну генерацію паралельних програм по їхніх специфікаціях.

Концепції, які були викладені вище, покладені в основу системи паралельної обробки інформації на комп'ютерних мережах ПАРКС-JAVA.

3. Система *ПАРКС-JAVA*

3.1. Особливості реалізації системи ПАРКС-JAVA

Система ПАРКС-JAVA [6-9] реалізується на основі локальної чи глобальної комп'ютерної мережі обробки інформації, надаючи можливість користувачам малопотужних комп’ютерів використовувати ресурси мультипроцесорних комплексів, або ресурси комп’ютерної мережі. В обох випадках програмне забезпечення підтримує процеси взаємодії різних вузлів мережі між собою на основі обміну повідомленнями по мережі.

Застосування мови програмування JAVA надає можливість переносу програм між різними апаратними платформами, на яких встановлена JavaVM – віртуальна машина.

До *позитивних якостей* мови програмування JAVA [63, 91], які вплинули на вибір її в якості базової для проектування системи паралельних обчислень, належать:

- проста об’єктна модель, що дозволяє легко проводити зміни в коди програми, доповнювати її та документувати;
- детермінованість коду – властивість мови, що не дає програмісту робити помилки, наприклад, пов’язані з несанкціонованими діями, такими, як різні виклики;
- система програмування JAVA відповідає сучасним вимогам: підтримка засобів мережевого програмування (сокетів), потоковий ввід/вивід, графічний інтерфейс користувача, системи забезпечення пересилки даних по комп’ютерній мережі, технології, що дозволяють уніфікувати клієнт-серверний (RMI) та інші.

До *недоліків мови* JAVA можна віднести достатню повільність JavaVM.

В середовищі ПАРКС-JAVA паралельні програми реалізуються на мережі, що складається з JavaVM. Це дозволяє використовувати паралельні програми на довільних паралельних системах, як однорідних так і

неоднорідних, а також на системах, що можуть складатися з вільних ресурсів Інтернету, для яких єдиною умовою є наявність JavaVM.

Розглянемо множину операторів паралельного програмування, яка реалізує ПАРКС-технологію програмування та надає можливість розширити базову мову JAVA до паралельної. Система ПАРКС-JAVA має кілька модулів.

Головний модуль системи – *parcs*. Він містить реалізацію функцій системи ПАРКС-JAVA, що беруть участь в обчисленні. Основні компоненти системи:

клас parcs.task - об'єкт задача

public boolean addJarFile(String filename) - додає jar-файл у список файлів, що завантажуються для виконання АМ, де filename – ім'я файлу на диску, повертає true – коли файл ще не був завантажений;

клас parcs.point - об'єкт точка

public point(task curtask) - створення нової точки;

public channel createChannel() - створення каналу, що з'єднує поточну точку (у якій виконується код) із даною точкою, функція повертає посилання на канал;

public channel chan = null - посилання на канал зв'язку з точкою;

public task curTask - задача до якої приписана точка;

public HostInfo host = null - інформація про машину, до якої приписана точка;

public void execute(String AMname) - запускає в точці АМ, який пересилається потрібній точці і запускається на виконання його метод run. Клас, ім'я якого передається як параметр функції, повинен реалізовувати інтерфейс АМ, що містить єдиний метод:

public void run(task curtask, channel parent) - починає виконання АМ, де *curtask* - задача, *parent* - посилання на канал, що зв'язаний з батьківською точкою;

клас parcs.channel - об'єкт канал

public void write(byte x) - відіслати в канал змінну x типу byte;

public void write(int x) - відіслати в канал змінну x типу int;

public void write(long x) - відіслати в канал змінну x типу long;

public void write(double x) - відіслати в канал змінну x типу double;

public void write(Serializable obj) - відіслати об'єкт, який реалізує інтерфейс Serializable, тобто розроблений з можливістю запису в канал (більшість стандартних класів мови Java2, що зберігають дані, реалізують цей інтерфейс);

public void write(byte[] buf) - відіслати в канал масив байтів;

public byte readByte() - отримати з каналу число типу byte;

public int readInt() - отримати з каналу число типу int;

public long readLong() - отримати з каналу число типу long;

public double readDouble() - отримати з каналу число типу double;

public Object readObject() - отримати з каналу об’єкт;

public void read(byte[] buf) - отримати з каналу масив байтів та записати його.

Повний опис класів системи ПАРКС-JAVA наведений на веб сайті: http://www.unicyb.kiev.ua/~der/PARCS

Центральну роль у такій реалізації відіграє процес, що відпрацьовує основні функції КП: створює точки та канали, приписує АМ точкам, підтримує взаємодію між точками у ході роботи. Умовна його назва – *демон* (клас *Daemon*). Спілкування з ним проходить або через локальний файл, або через потік вводу-виводу TCP/IP. *Демон* повинен бути запущений на кожному комп'ютері мережі, який бере участь в обчисленнях. Він постійно знаходиться в пам’яті і приймає з’єднання з мережі на TCP-порт. При надходженні запиту на виконання АМ він породжує новий Java-потік, в якому запускає модуль на виконання.

3.2. Архітектура системи ПАРКС-JAVA

Розглянемо модулі системи ПАРКС-JAVA, які зображені на рис.3.1.

- Бібліотека *parcs* містить класи та методи, що використовуються всіма компонентами системи, в тому числі і алгоритмічними модулями;
- Програма *Daemon* запускається в фоновому режимі на машинах, які підключаються до обчислюваної мережі (хости). Вона виконує запуск алгоритмічних модулів обчислювальних задач, а також тестування продуктивності машин.

[канали]

- Сервер хостів *HostsServer* є централізованою службою для управління та обліку задач та точок. Сервер зберігає інформацію про підключені машини, продуктивність та кількість точок на кожному хосту, і згідно цієї інформації виділяє машини для створення нових точок, а також відмічає видалення точок, початок та завершення задач.
- Обчислювальна *задача*, що поставляється у вигляді одного чи декількох jar-файлів, або окремих класів алгоритмічних модулів. На обчислення запускається початковий алгоритмічний модуль задачі. В процесі виконання початкового АМ він завантажує з фізичного диску програмний код інших алгоритмічних модулів.
- RMI-сервер *ParcsLauncher* використовується для віддаленого запуску програми на виконання. RMI-клієнт для запуску програми повинен передати код обчислювальної задачі, запакований в jar-архів, та може вказати чи потрібно чекати кінець обчислення перш, ніж повернути управління клієнту.

На всіх машинах, що призначені для обчислення, запускається по одному екземпляру програми Daemon, а початковий АМ та HostsServer можна запускати, або на тих самих, або на інших машинах. В окремому випадку навіть всі три елементи системи можна запускати з одного місця. Файл server використовується початковим АМ для знаходження машини, на якій запущений HostsServer. В єдиному його рядку знаходиться ім’я або IP-адреса цього хосту. В файлі даних (одному або декількох) знаходяться вихідні дані

для виконання обчислень, результати записуються в файл(и) результату. При створенні нової точки алгоритмічні модулі звертаються до сервера хостів, щоб отримати адресу машини (найменш завантаженої) для запуску нового АМ. Список доступних машин зберігається в файлі hosts.list.

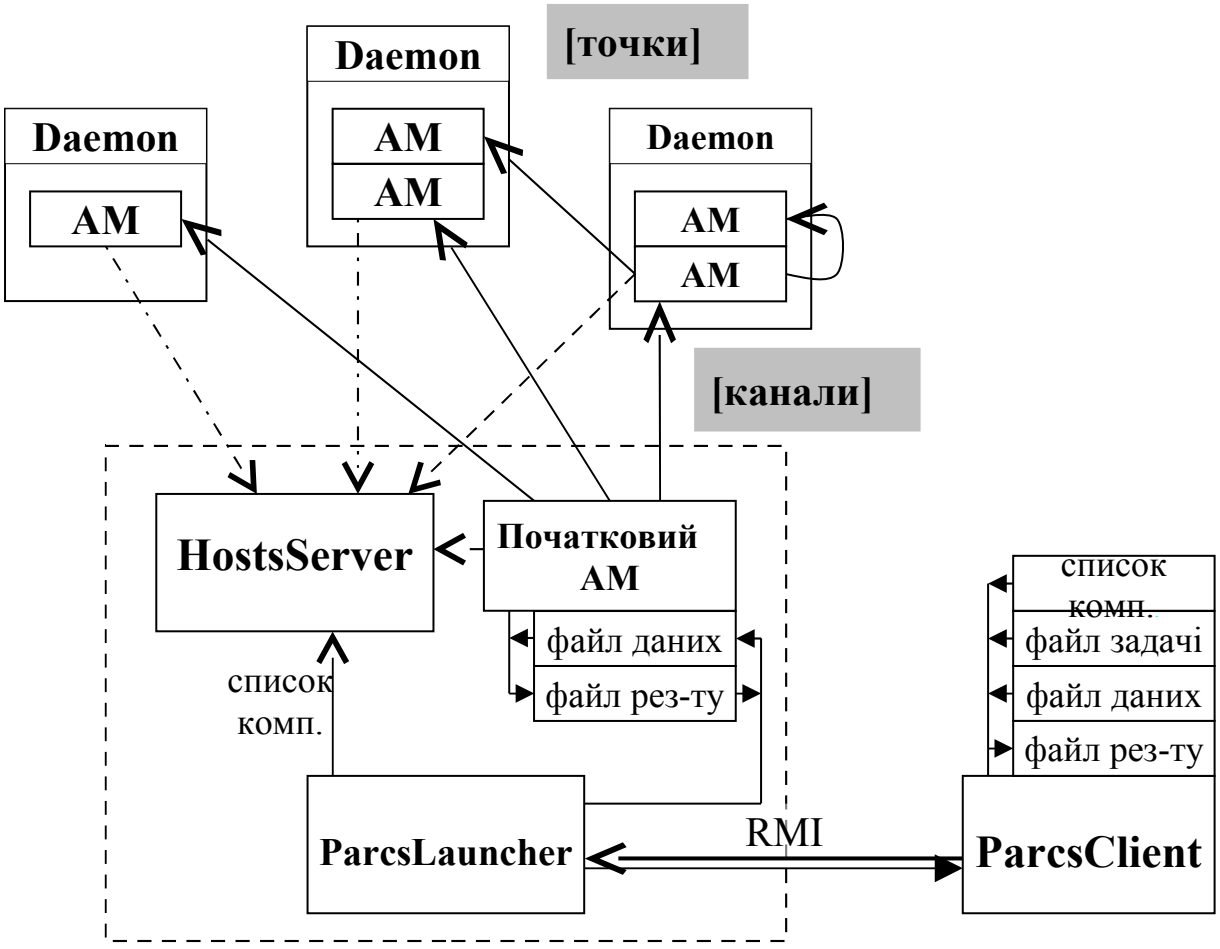


Рисунок 3.1 Архітектура системи *ПАРКС-JAVA*

Remote Method Invocation (RMI) – це технологія, що дозволяє уніфікувати клієнт-серверний зв'язок та представити його як виклик функції. Функції, що надаються сервером, описуються в RMI-інтерфейсі (файл ParcsLauncherInt.java). На початку запускаються демони та RMI-сервер ParcsLauncher, що переходять в режим очікування. RMI-клієнт до запуску може передавати серверу файл даних та список машин, хоча більш зручним є залишити без змін список, що зберігається в файлі hosts.list. Потім виконується виклик функції, що запускає на виконання задачу, код якої попередньо був зчитаний клієнтом з jar-файлу. Для запуску задачі на сервері запускається

HostsServer (якщо не був запущений раніше), а потім запускається початковий АМ задачі. Ім’я початкового АМ вказується RMI-клієнтом, або зчитується з файлу маніфесту (META-INF/manifest.mf) отриманого jar-архіву, як це зазвичай відбувається при запуску jar-архівів. Далі відбувається обчислення задачі при участі демонів, після цього управління повертається RMI-клієнта, і він при потребі може запросити з сервера файл(и) результату.

3.3. Використання протоколу TCP/IP для побудови паралельних розподілених обчислень

Сімейство протоколів TCP/IP було обрано за основу для передачі інформації, оскільки воно широко застосовується в усьому світі і служить для об'єднання комп'ютерів у мережі Internet. Єдина мережа Internet складається з множини мереж різноманітної фізичної природи, від локальних мереж типу Ethernet до глобальних мереж.

Архітектура протоколів TCP/IP призначена для об'єднаної мережі, що складається із поєднаних один з одним шлюзами окремих різнорідних пакетних підмереж, до яких підключаються різнорідні комп'ютери. кожна з підмереж працює у відповідності зі своїми специфічними вимогами і має свою природу засобів зв'язку. проте передбачається, що кожна підмережа може прийняти пакет інформації (дані із відповідним мережевим заголовком) і доставити його по вказаній адресі в цій конкретній підмережі.

3.4. Тестування пам’яті комп’ютерів в системі ПАРКС-JAVA

Тестування пам’яті комп’ютерів в процесі роботи системи ПАРКС-JAVA будемо проводити за декількома параметрами. Для мови JAVA тестування пам’яті є дуже важливим, тому що віртуальна JAVA машина потребує розширення пам’яті для складних обчислювальних задач . Тести проводимо на мережі з трьох комп’ютерів P-IV 2400Mhz. Важливе місце серед цих тестів займає тестування пам’яті, яку використовує клас Daemon, який відповідає за обробку АМ та клас Matrix, що відповідає за формування КП та зборку

підзадач. Для визначення обчислювальної спроможності застосування АМ робимо запуск обчислювальних тестів, які моделюють реальне завантаження машин від одної до трьох. Такий підхід, хоч і має деяку похибку, дозволяє визначати загальну динаміку кількості пам'яті, яку необхідно вказати в опціях для віртуальної Java-машини. Це є дуже важливим, тому що, наприклад, коли не вказати відповідну кількість пам'яті для JavaVM, то в нашому тесті задача за стандартними умовами (64Mb) видає помилку для матриць розмірності вище за 1500*1500. Це легко побачити з Таблиці 3.1, так для розмірності матриць вище за 1500*1500 необхідно пам'яті більше ніж 64Mb.

Таблиця 3.1. Тестування пам'яті

Кількість комп'ютерів / Розмірність матриці	P-IV 2400Mhz 1		P-IV 2400Mhz 2		P-IV 2400Mhz 3	
	Daemon	Matrixs	Daemon	Matrixs	Daemon	Matrixs
500	12 917	8 966	6 850	8 652	5 379	8 198
1000	47 553	41 228	18 684	36 337	17 357	36 301
1500	**101 301**	**76 612**	40 818	**87 040**	37 752	**78 134**
2000	184 684	165 835	91 358	159 180	67 312	145 811

Наведемо графік рис.3.3 залежності використання пам'яті для класів Daemon та Matrixs від розмірності матриць для різної кількості комп'ютерів в локальній мережі. На графіку показана динаміка зміни кількості пам'яті, що потребують класи Daemon і Matrixs.

Коли задача виконується на одному комп'ютері, клас Daemon, який в нашому випадку відповідає за обробку АМ, потребує більше пам'яті ніж, коли задача обчислюється на більшій кількості комп'ютерів. Це пов'язано з тим, що задача виконується паралельно і вона розбивається на підзадачі, які виконуються на різних комп'ютерах (навантаження на один комп'ютер зменшується).

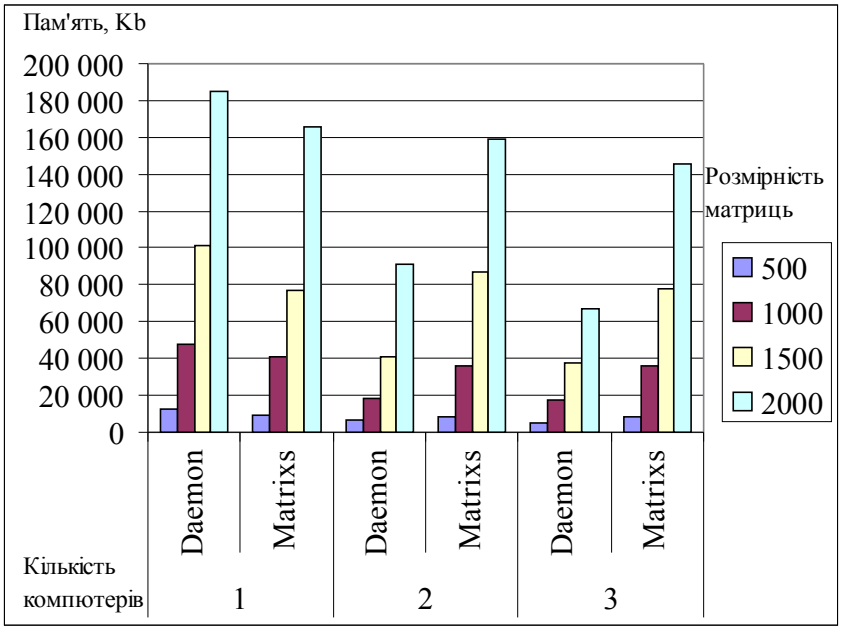


Рис.3.3. Графік тестування пам’яті комп’ютерів для задачі множення матриць великої розмірності.

В свою чергу для класу Matrixs кількість необхідної пам’яті практично не змінюється в залежності від кількості комп’ютерів, тому що він відповідає за формування КП та зборку підзадач. Зрозуміло, що загальна кількість пам’яті, яку необхідно виділити JavaVM для обчислення задачі, зростає зі зростанням розмірності матриць.

Можливо зробити висновок, що *ефективність* роботи паралельної обчислювальної системи ПАРКС-JAVA залежить не тільки від її конкретної структури, а і від того, як реалізується її виконання на конкретній обчислювальній системі, та від багатьох зовнішніх факторів.

4. Застосування системи ПАРКС-JAVA для вирішення задач паралельного програмування

4.1. Побудова моделі КП – "ДЕРЕВО" на прикладі множення матриць великої розмірності

Розглянемо приклади побудови різних моделей КП. Спочатку розглянемо модель КП – "ДЕРЕВО". Для цієї моделі найбільш підходить наступна задача.

Розглянемо можливості побудови паралельної програми на мові ПАРКС-JAVA на прикладі класичної задачі множення матриць великої розмірності. Необхідно помножити дві матриці великої розмірності $A(n \times k)$ и $B(k \times m)$. Необхідно здійснити розбиття задачі на задачі меншої розмірності. Кожен АМ буде здійснювати дії над матрицями розмірності *d,* де $d << n, k, m$. В нашому тесті $n = k = m$, результати обчислень занесені в Таблицю 4.1.

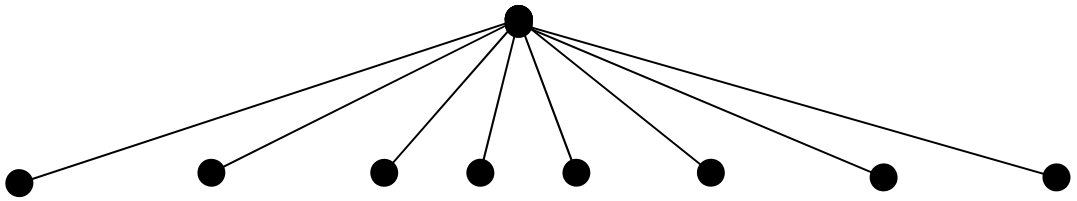
Задачу $C=A \times B$ можна звести до дій з відповідними блоками:

$$C_{11}=A_{11} \times B_{11} + A_{12} \times B_{21};\ C_{12}=A_{11} \times B_{12} + A_{12} \times B_{22},$$
$$C_{21}=A_{21} \times B_{11} + A_{22} \times B_{21};\ C_{22}=A_{21} \times B_{12} + A_{22} \times B_{22}, \qquad (4.1)$$
$$\text{де } A=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},\ B=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},\ C=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Якщо одержані блоки не досягнули потрібної розмірності, то процедуру можна продовжити. В якості КП обираємо 8-дерево, фрагмент якого зображений на рис.4.1. В кожній вершині якого буде виконуватися відповідний АМ(множення матриць у нашому випадку). Для спрощення розглянемо випадок, коли кількість блоків відповідає рис.4.1. Тобто ми будемо розбивати кожну матрицю тільки на чотири частини (фіксуємо значення *d*).

Наприклад, координати блоків матриці $A^{(n \times k)}$ визначаються так :

$$A = \begin{pmatrix} A_{11}(0,0,n/2,k/2) & A_{12}(0,k/2,n/2,k/2+k1) \\ A_{21}(n/2,0,n/2+n1,k/2) & A_{22}(n/2,k/2,n/2+n1,k/2+k1) \end{pmatrix}$$, де $n1=n\%2$, $k1=k\%2$



(1,1)(1,1) (1,2)(2,1) (1,1)(1,2) (1,2)(2,2) (2,1)(1,1) (2,2)(2,1)(2,1)(1,2) (2,2)(2,2)

Рис.4.1 Представлення КП – "ДЕРЕВО" згідно формул (4.1)

При реалізації паралельного алгоритму потрібно створити набір класів (АМ), які реалізують інтерфейс АМ. Кожний АМ запускається в окремій точці методом *point.execute(String)*. Метод *run* запускається на виконання *демоном* коли надійшли відповідні команди., де *curtask* - задача, що буде виконуватися, а *parent* - канал, що з'єднується з батьківською точкою, або *null*, коли АМ початковий. Покажемо частину коду на мові JAVA, яка реалізує паралельні дії нашого алгоритму (не розкриваючи додаткових методів з назви яких можна побачити дії, що вони виконують):

```
public class Matrixs implements AM{
   public static void main(String[] args) {
                Const.parseArgs(args);
                task curtask = new task();
//Завантажуємо всі методи, що реалізують наш алгоритм
                curtask.addJarFile("Matrixes.jar");
                (new Matrixs()).run(curtask, (channel)null);
                curtask.end();
   }
     public void run(task curtask, channel parent) {
                Matrix A = new Matrix(curtask.findFile("Matrix1.data"));
                Matrix B = new Matrix(curtask.findFile("Matrix2.data"));
//Формуємо представлення КП згідно рис.4.1
```

```
point[] p = new point[8];
channel[] c = new channel[8];
for (int i=0; i<8; i++) {
        p[i] = curtask.createPoint();
        c[i] = p[i].createChannel();
        p[i].execute("Matr");
}
```

// Представлення блоків матриць згідно формул (4.1) та розподіл по каналах

```
c[0].write(A.SubMatrix(0, 0, A.Height() / 2, A.Width() / 2));
c[0].write(B.SubMatrix(0, 0, B.Height() / 2, B.Width() / 2));
c[1].write(A.SubMatrix(0, A.Width() / 2, A.Height() / 2, A.Width() / 2 + A.Width() % 2));
c[1].write(B.SubMatrix(B.Height() / 2, 0, B.Height() / 2 + B.Height() % 2, B.Width() / 2));
c[2].write(A.SubMatrix(0, 0, A.Height() / 2, A.Width() / 2));
c[2].write(B.SubMatrix(0, B.Width() / 2, B.Height() / 2, B.Width() / 2 + B.Width() % 2));
c[3].write(A.SubMatrix(0, A.Width() / 2, A.Height() / 2, A.Width() / 2 + A.Width() % 2));
c[3].write(B.SubMatrix(B.Height() / 2, B.Width() / 2, B.Height() / 2 + B.Height() % 2,
B.Width() / 2 + B.Width() % 2));
c[4].write(A.SubMatrix(A.Height() / 2, 0, A.Height() / 2 + A.Height() % 2, A.Width() / 2));
c[4].write(B.SubMatrix(0, 0, B.Height() / 2, B.Width() / 2));
c[5].write(A.SubMatrix(A.Height() / 2, A.Width() / 2, A.Height() / 2 + A.Height() % 2,
A.Width() / 2 + A.Width() % 2));
c[5].write(B.SubMatrix(B.Height() / 2, 0, B.Height() / 2 + B.Height() % 2, B.Width() / 2));
c[6].write(A.SubMatrix(A.Height() / 2, 0, A.Height() / 2 + A.Height() % 2, A.Width() / 2));
c[6].write(B.SubMatrix(0, B.Width() / 2, B.Height() / 2, B.Width() / 2 + B.Width() % 2));
c[7].write(A.SubMatrix(A.Height() / 2, A.Width() / 2, A.Height() / 2 + A.Height() % 2,
A.Width() / 2 + A.Width() % 2));
c[7].write(B.SubMatrix(B.Height() / 2, B.Width() / 2 , B.Height() / 2 + B.Height() % 2,
B.Width() / 2 + B.Width() % 2));
```

//Додавання відповідних блоків матриць та прийняття даних з каналів

```
Matrix[][] Parts = new Matrix[2][2];
        Parts[0][0]  =  (Matrix)c[0].readObject();
        Parts[0][0].Add((Matrix)c[1].readObject());
        Parts[0][1]  =  (Matrix)c[2].readObject();
        Parts[0][1].Add((Matrix)c[3].readObject());
        Parts[1][0]  =  (Matrix)c[4].readObject();
        Parts[1][0].Add((Matrix)c[5].readObject());
        Parts[1][1]  =  (Matrix)c[6].readObject();
        Parts[1][1].Add((Matrix)c[7].readObject());
```

//Заповнення матриць (збірка)

```
                Matrix Res = new Matrix(A.Height(), B.Width());
                Res.FillSubMatrix(Parts[0][0], 0, 0);
                Res.FillSubMatrix(Parts[0][1], 0, Res.Width() / 2);
                Res.FillSubMatrix(Parts[1][0], Res.Height() / 2, 0);
                Res.FillSubMatrix(Parts[1][1], Res.Height() / 2, Res.Width() / 2);
        System.out.println("\nResult found. Saving to file Matrix.res");
        Res.Save(curtask.addPath("Matrix.res"));
        }
}
public class Matr implements AM{
        public void run(task curtask, channel parent) {
                Matrix m = (Matrix)parent.readObject();
                Matrix m1 = (Matrix)parent.readObject();
                  Matrix res = new Matrix(m.Height(),m1.Width());
//Функції представлення та множення блоків матриць
                  res.Assign(m);
                  res.MultiplyBy(m1);
//Передаємо результат у канал
                  parent.write(res);
        }
}
```

Був проведений експеримент для задачі множення матриць великої розмірності із застосуванням паралельної програми на мові ПАРКС-JAVA, що була запушена на 8 комп'ютерах С-1300Mhz в локальній мережі та на одному комп'ютері PIV-3000Mhz, а також із застосуванням класичної програми множення матриць великої розмірності на одному комп'ютері PIV-3000Mhz на мовах JAVA та С. Одержані результати наведені на графіку 4.1 залежності часу обчислень від розмірності матриць для різної кількості комп'ютерів в локальній мережі.

На графіку можна побачити як змінюється час обчислень для задачі множення матриць великої розмірності в залежності від застосувань різних мов програмування JAVA, С та ПАРКС-JAVA. На одному комп'ютері PIV-3000Mhz на мові С задача вирішується найшвидше. З застосуванням ПАРКС-

JAVA задача вирішується швидше ніж на JAVA, це пов’язано з розбиттям задачі на підзадачі, та їх оптимізації на рівні процесорної обробки.

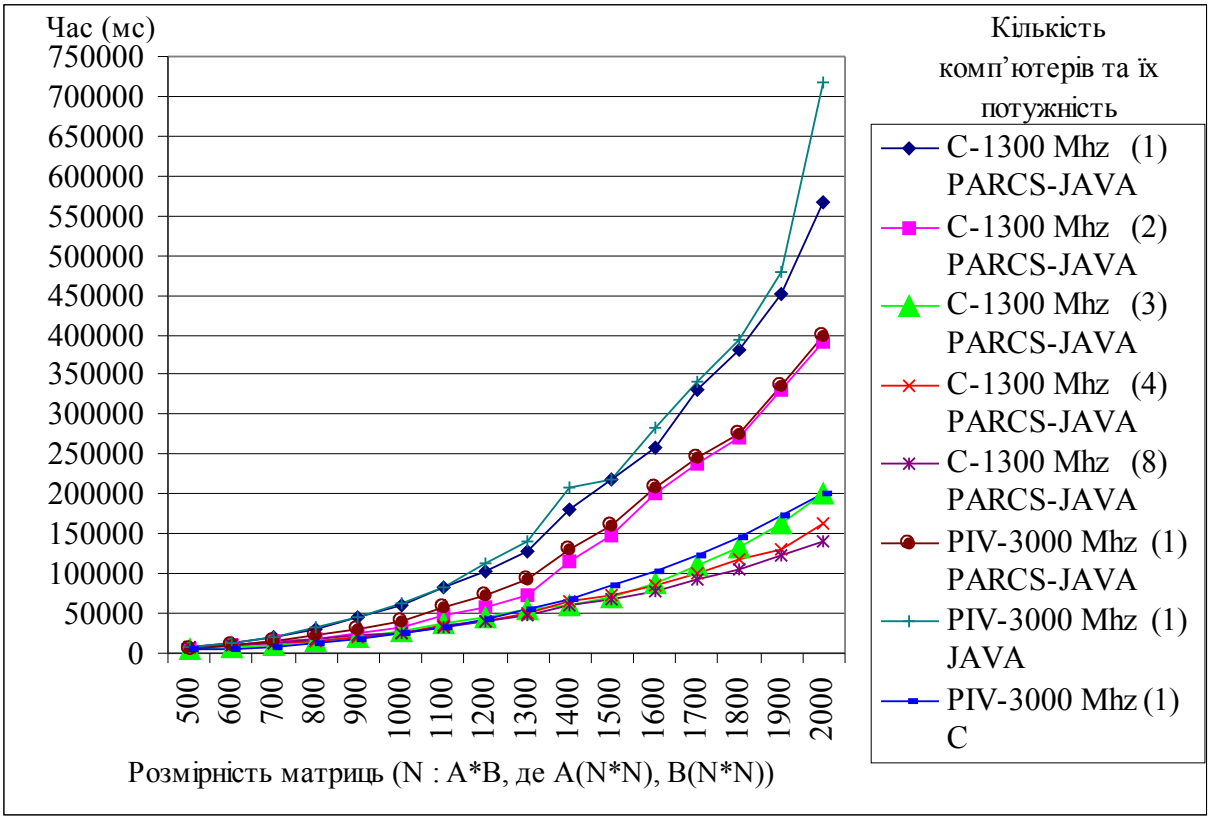


Рис. 4.1. Графік залежності часу обчислень від розмірності матриць для різної кількості комп’ютерів в локальній мережі від 1-8

Видно, що графік для мережі з 8 комп’ютерів C-1300Mhz (не найшвидших), на яких для рішення задачі застосовується ПАРКС-JAVA є найшвидший серед всіх. Цікавим є скачки графіку для мови JAVA та ПАРКС-JAVA на розмірності матриць більше 1400*1400, коли закінчується базова пам'ять 64 Mb , що виділена для JavaVM.

Легко побачити, що значення графіків різко розбігаються для різної кількості комп’ютерів зі зростанням розмірності матриць. Це, звісно, обумовлено зростанням складності обчислень і можливостями JavaVM.

Цікавим є факт, що обчислення на мові ПАРКС-JAVA на 3 комп’ютерах C-1300Mhz швидше, ніж обчислення на мові C на комп’ютері PIV-3000Mhz,

який, насправді, швидший ніж у 3 рази. Це показує певну ефективність паралельного програмування в системі ПАРКС-JAVA.

4.2. Побудова моделей КП – “КІЛЬЦЕ” та “КУБ”

Розглянемо приклади побудови статичних моделей КП. Ці моделі можуть бути різними і формуватися в залежності від типу задачі, що вирішується. Надамо тільки фрагменти коду, що відповідають за утворення КП. Спочатку розглянемо модель – “КІЛЬЦЕ” рис.4.3 , а потім модель – “3-КУБ ” рис.4.4.

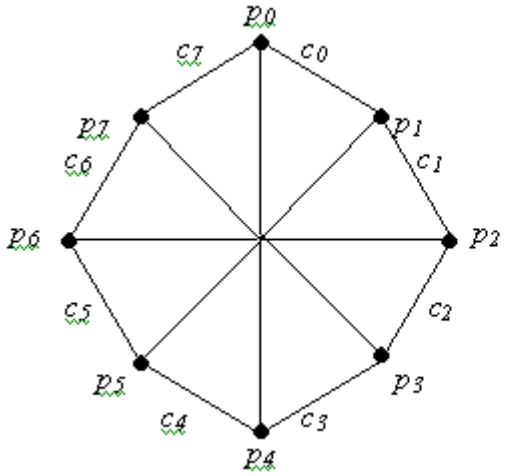


Рис.4.3. Представлення КП моделі “КІЛЬЦЕ”

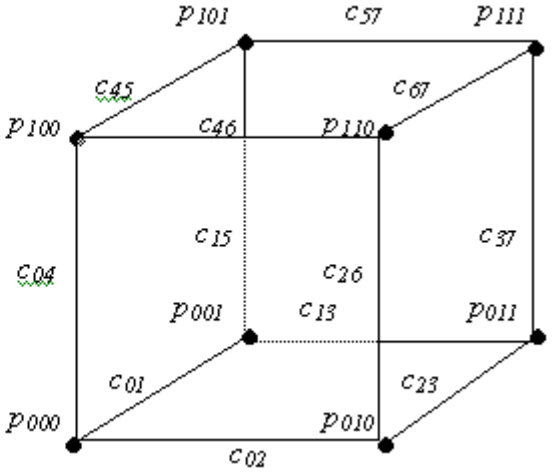


Рис.4.4. Представлення КП моделі “3-КУБ”

//Формуємо представлення КП моделі “КІЛЬЦЕ” рис.4.3

```
point[] p = new point[n];
channel[] c = new channel[n];
for (int i=0; i<n; i++) p[i] = curtask.createPoint();
for (int i=0; i<n-1; i++) c[i] = createChannel(p[i], p[i+1]);
 c[n-1] = createChannel(p[n-1], p[0]);
```

//Запускаємо відповідні АМ

```
for (int i=0; i<n; i++) p[i].execute("TASK_AM"+i);
```

//Формуємо представлення КП моделі “3-КУБ” рис.4.4

```
  point[][][] p = new point[2][2][2];
  channel[][] c = new channel[8][8];
  for (int i=0; i<2; i++)
```

```
 for (int j=0; j<2; j++)
  for (int k=0; k<2; k++)
        p[i][j][k]  = curtask.createPoint();
for (int i=0; i<2; i++)
 for (int j=0; j<2; j++)
  for (int k=0; k<2; k++) {
     int pi = i*4+j*2+k;
     int pj = ((i+1)%2)*4+j*2+k;
     if ( c[pi][pj] == null )
        c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[(i+1)%2][j][k]);
     pj= i*4+((j+1)%2)*2+k;
     if ( c[pi][pj] == null )
        c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i] [(j+1)%2][k]);
    pj= i*4+j*2+(k+1)%2;
     if ( c[pi][pj] == null )
        c[pi][pj] = c[pj][pi] = createChannel(p[i][j][k], p[i][j][(k+1)%2]);
  }
```

//Запускаємо відповідніАМ

```
 for (int i=0; i<2; i++)
  for (int j=0; j<2; j++)
   for (int k=0; k<2; k++)
        p[i] [j] [k].execute("TASK_AM"+i+j+k);
```

В результаті роботи з системою ми можемо моделювати реальні обчислювальні задачі за допомогою побудови динамічної чи статичної структури керуючого простору та різних його конфігурації (дерево, кільце, куб та ін.)[31, 32]. Виникає проблема оптимального керування паралельними обчислювальними процесами, яка є однією з найскладніших в паралельних обчисленнях. Ефективність роботи паралельної обчислювальної системи ПАРКС-JAVA залежить не тільки від її конкретної структури, а і від того, як реалізується її виконання на конкретній обчислювальній системі, і від багатьох зовнішніх факторів, таких як пропускна спроможність мережі та завантаженість комп’ютерів та інших.

Система ПАРКС-JAVA дає певний механізм для написання паралельних програм. Користувачу потрібно тільки, застосовуючи методи системи, описувати предметне середовище моделюючи його за допомогою КП. Всі

класи (АМ), які були створені при реалізації попередніх паралельних алгоритмів, можливо використовувати при формуванні наступних паралельних програм. Таким чином формується база класів АМ.

На сайті: http://www.unicyb.kiev.ua/~der/ PARCS доступний програмний код системи ПАРКС-JAVA для практичного застосування.

ВИСНОВКИ

Створена на факультеті кібернетики Київського національного університету системи паралельної обробки інформації ПАРКС-JAVA вирішує наступні задачі:

- ♦ можливість підтримки великих за обчисленням задач, що потребують застосування потужних обчислювальних систем та паралельної обробки інформації;
- ♦ можливість контролю та управління інформаційними потоками в системах паралельної обробки інформації;
- ♦ можливість накопичення бази даних алгоритмів (алгоритмічних модулів) паралельного програмування для їх подальшого застосування.

В результаті розробки системи паралельної обробки інформації ПАРКС-JAVA було отримано такі результати:

1. Створено ефективні алгоритми керування паралельними обчислювальними процесами.
2. Запропоновано практичне застосування технології ПАРКС для паралельної обробки інформації на комп'ютерних мережах та кластерах. Розроблено архітектуру ПАРКС для комп'ютерних мереж.
3. Розроблено розширення базової мови програмування JAVA засобами для реалізації паралельних обчислень. Створено програмний продукт ПАРКС-JAVA, що може виконувати реальні обчислювальні задачі на комп'ютерних мережах, дозволяє будувати динамічну чи статичну структуру керуючого простору та різні його конфігурації.
4. Експериментально обґрунтована ефективність алгоритмів для побудови різних моделей керуючого простору та відповідних паралельних програм на мові ПАРКС-JAVA.

Перспективним напрямком розвитку системи ПАРКС-JAVA, яка не залежить від апаратної платформи, є можливість створити сервер у мережі Інтернет, що буде роздавати обчислювальні підзадачі комп'ютерам, власники яких бажають взяти участь в обчислювальних проектах, і при цьому такий сервер зможе працювати з будь-якими клієнтами незалежно від архітектури їх машин, необхідна тільки підтримка клієнтом JavaVM, яка сьогодні знаходить все ширшу підтримку в операційних системах.

Контрольні запитання

1. Які Ви знаєте сучасні технології паралельного програмування?
2. Порівняйте підходи до застосування різних типів паралельних систем в залежності від типу пам’яті, що використовується.
3. Що таке керуючий простір та алгоритмічний модуль?
4. Що таке композиція алгоритмічних модулів?
5. Що таке технологія ПАРКС?
6. Які засоби розширення базової мови програмування до паралельної Ви знаєте?
7. Розкажіть про архітектури системи ПАРКС-JAVA.
8. Як налаштувати систему ПАРКС-JAVA для роботи у локальній мережі?
9. Які можливості системи ПАРКС-JAVA?
10. Моделювання керуючого простору за допомогою технології ПАРКС-JAVA. Наведіть приклади різних моделей КП?
11. Які особливості реалізації паралельного алгоритму класичної задачі множення матриць для системи ПАРКС-JAVA?
12. Порівняйте різні підходи щодо розробки паралельних програм.

Завдання для практичних занять та самостійної робот

На сайті: http://www.unicyb.kiev.ua/~der/ PARCS доступний програмний код системи ПАРКС-JAVA для практичного застосування, який необхідно застосовувати згідно інструкції користувача.

Завдання розділено на 2-а етапи:

1-й етап. Для всіх завдань написати програму на ПАРКС-JAVA та JAVA або C++ та порівняти результати.

2-й етап. Провести тестування на локальному комп'ютері та комп'ютерній мережі.

Результати тестувань *занести в таблицю* та *зробити висновки* щодо застосування різних технологій паралельного програмування для відповідної задачі.

Теми завдань:

1.Написати програму *множення матриць*.

2. Написати програму *обчислення числа Пі* .

3.Написати програму *обчислення інтегралу* (різні методи).

4. Написати програму *сортування* (різні методи).

5. Написати програму *факторизації чисел* (різні методи).

6. Написати програму *розв'язання системи рівнянь* (різні методи).

7. Написати програму *підбору паролю* (перебір).

8. Написати програму *обробка зображень* (різні методи).

9. Написати програму *алгоритми на графах* (різні методи).

10. Написати програму *обробка текстової інформації* (різні методи).

Література

1. Анисимов А.В., Кулябко П.П. Программирование параллельных процессов в управляющих пространствах. // Кибернетика. – 1984. –№3. – С.79-88.
2. Анисимов А.В., Борейша Ю.Е., Кулябко П.П. Технология параллельного программирования «ПАРУС» // Автоматика и телемеханика. – 1990.– №6. – С. 153-160.
3. Анисимов А.В., Борейша Ю.Е., Кулябко П.П. Система программирования ПАРУС // Программирование.–1991.–№6.–С. 91-102.
4. Анисимов А.В., Кулябко П.П. Особенности ПАРУС–технологии. // Кибернетика и системный анализ. – 1993. – №3. – С.128-137.
5. Анисимов А.В., Кулябко П.П. Моделирование сети Петри с помощью ПАРУС–средств. // Проблемы программирования. – 1997. – Вып. 2. – С. 45–56.
6. Анисимов А.В., Деревянченко А.В. Система ПАРУС-JAVA для параллельных вычислений на компьютерных сетях // Кибернетика и системный анализ. – 2005. – №1. – С.25-36.
7. Анісімов А.В., Дерев'янченко О.В. Система ПАРКС-JAVA як засіб вирішення паралельних алгоритмів на комп'ютерній мережі // Материалы четвертой международной научно-практической конференции УкрПРОГ'2004. – Проблемы программирования. – №2-3. – 2004. – С.282-284.
8. Анисимов А.В., Деревянченко А.В., Литвинов Д.В. Архитектура системы для параллельных вычислений ПАРУС-JAVA //Материалы Международной научно-технической конференции Интеллектуальные и многопроцессорные системы – 2004. – Т.1. – С.132-135.
9. Анисимов А.В., Деревянченко А.В. Построение виртуального параллельного пространства с использованием технологии ПАРУС-JAVA //Материалы Международной научно-технической конференции Интеллектуальные и многопроцессорные системы. –2003.– Т.2.– С.18-19.
10. Анисимов А.В. Рекурсивные преобразователи информации. – К. : Вища школа. – 1987. – 392 с.
11. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – К.: БХВ – Петербург. – 2002. – 608 с.
12. Глушков В.М., Анисимов А.В. Управляющие пространства в асинхронных параллельных вычислениях. // Кибернетика. – 1980. – №5. – С. 1-9.
13. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Автоматизация проектирования вычислительных машин. – К.: Наук. Думка, 1975.–232с.

14. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Алгебра алгоритмов и динамические распараллеливание программ // Кибернетика.– 1982. – № 5.– С.4-10.
15. Гороховський С.С., Капитонова Ю.В., Летичевский А.А., Молчанов И.Н., Погребенский С.Б. Алгоритмический язык МАЯК // Кибернетика. – 1984. – № 3. – С. 54-74.
16. Д.Грин, Д.Кнут. Математические методы анализа алгоритмов. – М.: Мир. – 1987. – 119 с.
17. Дал У., Дейкстра Э., Хоар К. Структурное программирование. – М.:Мир. – 1975. – 248 с.
18. Дерев'янченко О.В. Моделювання паралельних програм за допомогою системи ПАРКС-JAVA // Наукові записки НаУКМА, Комп'ютерні науки. – 2005. – Т.36. – С. 32-38.
19. Дерев'янченко О.В. Побудова паралельних програм за допомогою системи ПАРКС-JAVA. // Матеріали Міжнародної конференції Теоретичні та прикладні аспекти побудови програмних систем. – К. –2004. – С.313-320.
20. Деревянченко А.В., Лялецкий А.А. О проблеме распараллеливания вычислений // Математические машины и системы. –2004. –№2.–С.3-14.
21. Дерев'янченко О.В. Застосування технології CUDA в системі паралельних обчислень ПАРКС-Java // Матеріали міжнародної конференції «Штучний інтелект. Інтелектуальні системи ШІ-2011». – Т.1. – С.62-68.
22. Дорошенко А.Ю., Фінін Г.С., Цейтлін Г.О. Алгеброалгоритмічні основи програмування. – К.: Наук. Думка. – 2004. – 457 с.
23. Дорошенко А.Е. Методы повышения производительности параллельных программ с многоуровневой памятью // Кибернетика и системный анализ. – 1994. – № 4. – С. 117-128.
24. Дорошенко А.Е. Динамические модели параллельных вычислений для макроконвеерных программ // Кибернетика и системный анализ. – 1995. – № 6. – С. 45-65.
25. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебраический подход. – К.: Наук. Думка. – 2000. – 177 с.
26. Дорошенко А.Ю. Лекції з паралельних обчислювальних систем: Метод. посіб. – К.: Видавничій дім "КМ Академія", 2003. – 42 с.
27. Капитонова Ю.В., Летичевский А.А. Математическая теория проектирования вычислительных систем. – М.: Наука. – 1988 – 296 с.
28. Системы параллельной обработки / Под ред. Д. Ивенса. – М.: Мир. – 1985. – 416 с.

29. Торбер К.Дж. Архитектура высокопроизводительных вычислительных систем. – М. : Наука. – 1985. – 271с.
30. Яворски Д., Перроун П. Система безопасности Java. Руководство разработчика // ИД Вильямс. – 2001 – 528 с.
31. Communication Aspects of the Star Graph Interconnection Network // *IEEE Transactions on Parallel and Distributed Systems.* – July 1994. – Vol. 5. – №7. – pp. 678-687.
32. DVM-система. – http://www.keldysh.ru/dvm/
33. Flynn M. Very high-speed computing system // Proc. IEEE. – 1966. – N 54. – pp.1901-1909.
34. Flynn M. Some Computer Organisations and Their Effectiveness // IEEE Trans. Computers. – 1972. – V.21. – N 9. – pp.948-960.
35. High Performance Fortran Language Specification. Version 2.0, January 1997. – http://dacnet.rice.edu/Depts/CRPC/HPFF/versions/hpf2/hpf-v20/
36. JAVA 2 SDK. – http://java.sun.com/
37. A.Lastovetsky and R.Reddy On Performance Analysis of Heterogeneous Parallel Algorithms // Parallel Computing 30(11). – 2004. – pp.1195-1216. – http://www.cs.ucd.ie/staff/alexeyl/ParCom2004.pdf
38. Message-Passing Interface Forum, Document for a Standard Message-Passing Interface, 1993. Version 1.0. – http://www.unix.mcs.anl.gov/mpi/
39. Message-Passing Interface Forum, MPI-2: Extensions to the Message-Passing Interface, 1997. – http://www.unix.mcs.anl.gov/mpi/
40. mpC Language Specification. – http://parallel.ru/tech/mpc/
41. MPICH version 1.2.7p1, released on November 4th, 2005. – http://www-unix.mcs.anl.gov/mpi/mpich/
42. OpenMP Version 1.0, October 1997. – http://www.openmp.org/
43. PVM (Parallel Virtual Machine). – http://www.csm.ornl.gov/pvm/
44. Крюков В.А. Разработка параллельных программ для вычислительных кластеров и сетей. – http://www.keldysh.ru/dvm/dvmhtm1107/publishr/clust-141201.htm
45. Технології паралельного програмування. – http://www.parallel.ru/tech.

ЗМІСТ

ВСТУП…… 3
1 Система паралельної обробки інформації…… 5
1.1 Основні поняття та означення систем паралельної обробки інформації…….. 5
1.2 Архітектури систем для паралельних обчислень …… 12
1.3 Моделі паралельних обчислень…….. 15
1.4. Порівняння сучасних системи паралельної обробки інформації….. 17
2. Засоби побудови паралельних обчислювалних систем за технологією ПАРКС…… 20
2.1. Керуючий простір та його властивості……. 20
2.2. Композиції алгоритмічних модулів…….. 23
2.3. Особливості ПАРКС-технології…… 26
2.4. Моделі керуючого простору та топології мереж……. 29
2.5. ПАРКС-система програмування……... 30
2.6. Програмні засоби розширення базових алгоритмічних мов……..… 31
3.Система ПАРКС-Java……. 37
3.1. Особливості реалізації системи ПАРКС-JAVA …….. 37
3.2. Архітектура системи ПАРКС-JAVA……. 40
3.3.Використання протоколу TCP/IP для побудови паралельних розподілених обчислень…… 42
3.4. Тестування пам’яті комп’ютерів в системі ПАРКС-JAVA…… 42
4.Застосування системи ПАРКС-JAVA для вирішення задач паралельного програмування…… 45
4.1. Побудова моделі керуючого простору – “ДЕРЕВО” на прикладі множення матриць великої розмірності…….. 45
4.2. Побудова моделей керуючого простору – “КІЛЬЦЕ” та “КУБ”….. 50
ВИСНОВКИ…….. 53
Контрольні запитання…… 55
Завдання для практичних занять та самостійної роботи…… 56
Література…….. 57
ЗМІСТ……. 60