УДК 681.3

ПРО ПІДКЛЮЧЕННЯ ДО МОВИ САА/Д ДЕЯКИХ ЗАСОБІВ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ ПАКЕТУ МРІСН

Петрушенко А.М., Хохлов В.А., Петрушенко І.А.

Робота присв'ячена подальшому розвитку інструментальних засобів алгебро-граматичних методів представлення і розробки знань в різноманітних предметних областях - діалогової трансформаційної машини (ДТМ) і її вхідної мови САА/Д – мови моделей алгоритмів і відповідних їм програм і визначених класів апаратури [1,2], у зв'язку з можливістю використання цього інструментарія для синтезу розподілених додатків для кластерів, які останнім часом почали розробляти і використовувати для виконання значних обсягів трудомістких обчислень, не лише за кордрном, а і в Україні. З цією метою необхідно:

1) проаналізувати деякі апаратні і мовні особливості кластерів в порівнянні з іншими розподіленими обчислювальними системами (ОБС);

2) мову САА/Д розширити засобами паралельних обчислень;

3) до мови САА/Д підключити відповідні засоби паралельного програмування;

4) нові можливості ДТМ експерементально продемострувати на прикладі синтезу в ДТМ відповідного розроділеного додатку.

Кластери та їх місце в класі розподілених обчислювальних систем

Кластери відносяться до розподілених (або децентралізованих) ОБС [3,4]. Оскільки основною ознакою розподілених ОБС є наявність декількох центрів обробки даних, то, поряд з кластерами, до розподілених ОБС відносяться також мультипроцесори і комп'ютерні мережі (КМ).

В мультипроцесорах є декілька процесорів, кожен із яких може відносно незалежно виконувати свою програму. В мультипроцесорі існує спільна для всіх процесорів операційна система (ОС), яка оперативно розподіляє обчислювальне навантаження між процесорами. Взаємодія між окремими процесами організується найбільш простим способом – через спільну оперативну пам'ять (ОП). Сам по собі процесорний блок не є закінченим комп'ютером і тому не може виконувати програми без інших блоків – пам'яті і перефірійних пристроїв, які суть для всіх процесорів мультипроцесора спільними.

Основні переваги мультипроцесора – його висока продуктивність і відмовостійкість. Продуктивність досягається за рахунок паралельної роботи декількох процесорів. Оскільки при наявності спільної ОП взаємодія процесорів здійснюється дуже швидко, мультипроцесори можуть ефективно виконувати навіть додатки з високим ступенем зв'язків по даним. Відмовостійкість призводить, в загальному випадку, до зниження продуктивності (але не до нуля, як в звичайних системах), оскільки для того, щоб мультипроцесор міг продовжити роботу при відмовах деяких елементів, наприклад одного з процесорів чи блоку пам'яті, необхідне спеціальне програмне забезпечення.

Основна мета створення КМ - розподіл локальних ресурсів кожного комп'ютера між всіми користувачами мережі. В КМ програмні і апаратні зв'язки є набагато слабшими, ніж в мультипроцесорах, а автономність оброблюючих блоків проявляється значно сильніше – основними елементами КМ є стандартні комп'ютери, що не мають ні спільних блоків пам'яті, ні дискових накопичувачів, що керуються сумісно. Кожен комп'ютер працює під управлінням власної ОС, а "спільня" ОС відсутня. Взаємодія між комп'ютерами КМ відбувається через мережеві адаптери і канали зв'язку, які, власне, вирішують в КМ досить просту задачу - передають повідомлення від одиного комп'ютера до іншого, як правило - запит і відповідь про можливість доступу до локальних ресурсів

один одного, а основну роботу по організації сумісного використання ресурсів виконують клієнтські і серверні частини ОС.

Кластери володіють проміжними по відношенню до мультипроцесорів і КМ властивостями. Кластер (багатомашинна система) – це обчислювальний комплекс, що складається з декількох комп'ютерів, кожен з яких працює під керуванням власної ОС, а також програмні й апаратні засоби зв'язку комп'ютерів, що забезпечують роботу всіх комп'ютерів комплексу як єдиного цілого. На відміну від мультипроцесорів, що реалізовані на рівні процесорних блоків, кластер складається з декількох, здатних працювати автономно, як правило стандартних, комп'ютерів, кожен з яких має звичайну структуру, що включає один або кілька процесорних блоків, ОП і периферійні пристрої. Однак, завдяки спеціальному програмному й апаратному забезпеченню міжкомпьютерних зв'язків, для користувача кластер виглядає як єдиний комп'ютер. При цьому кожен комп'ютер, який називають також вузлом кластера, може бути як однопроцесорним, так і мультипроцесорним – на організацію кластера це не впливає.

Кластери застосовують для підвищення надійності і продуктивності ОБС. Надійність підвищується за рахунок того, що при відмовленні одного з вузлів кластера обчислювальне навантаження (або частина його) переноситься на інший вузол. Для виконання цієї операції в кластері використовуються два типи зв'язків між вузлами: міжпроцесорні зв'язки і зв'язки за рахунок поділюваних дисків. Міжпроцесорні зв'язки використовуються вузлами для обміну службовою інформацією. Зокрема, за допомогою цих зв'язків кожен вузол кластера періодично перевіряє стан інших вузлів і виконуваних ними обчислювальних задач. Якщо який-небудь вузол або одна з його задач, що входить у набір задач, які захищаються від відмовлень, змінили свій стан на непрацездатний, то починається процедура переміщення (реконфігурації) навантаження на один із працюючих вузлів. У цій процедурі важливу роль грають поділювані диски. Задача, що захищається, повинна зберігати свої дані на одному з таких дисків, щоб новий вузол зміг продовжити їхнє використання після відмовлення основного. Оскільки надійність дискових накопичувачів досить висока (її можна підвищити за рахунок додаткових заходів, наприклад задзеркалювання поділюваного диску), то істотно підвищується і надійність кластера в порівнянні з окремим комп'ютером. Час перекладу навантаження на інший вузол кластера при відмовленні значно більший, ніж час переходу на інший процесор у мультипроцесорі, тому що він пов'язаний з активізацією нової копії програмного процесу на іншому вузлі. При цьому також можлива втрата частини даних, що знаходилися в ОП вузла, що відмовив, але для визначених типів обчислювального навантаження, наприклад систем керування базами даних або web-серверів, ці втрати не позначаються на можливості продовжувати обчислення.

Якщо кластер застосовується для підвищення продуктивності, то кожна задача розпаралелюється на декілька гілок, що виконуються одночасно на декількох вузлах кластера. Синхронізація роботи декількох копій задачі або їхніх гілок, а також синхронізація вироблених ними даних, здійснюється як за рахунок міжпроцесорних зв'язків, так і за рахунок поділюваної дискової пам'яті. Менш тісні і менш швидкісні зв'язки між вузлами кластера в порівнянні зі зв'язками процесорів у мультипроцесорі диктують область застосування кластерів – це задачі, досить незалежні за даними.

Для організації міжпроцесорних зв'язків у кластерах часто використовуються спеціалізовані технології, пристосовані до рішення специфічних задач взаємодії комп'ютерів у кластері. Однак останнім часом усе частіше для цієї мети застосовують стандартні технології локальних мереж, наприклад Fast Ethernet і Gigabit Ethernet. Спільний доступ до дисків також може здійснюватися різними способами. Найбільш популярними варіантами є застосування інтерфейсу SCSI і технології Fibre Channel. Тенденція переходу на стандартні технології локальних мереж у цій області поки виражена не так чітко, як в області межпроцессорных зв'язків, але вона також міняється.

 ISBN 7-776-8361-7

Дана робота була обумовлена створенням в Інституті кібернетикі ім. В.М.Глушкова НАН України кластеру МОС ІК-ЮСТАР1, який складається з чотирьох вузлів по два процесори на кожному. Кожен вузол кластера містить: системну плату з двома процесорами AMD Duron (950 Мгц), 256 Мбайт оперативної пам'яті (DDR PC 2100), твердий диск IDE ATA 40 Гбайт, відеоадаптер (32 Мбайта, рознімання AGP), мережний адаптер (SCI чи Fast Eternet), роз’єми PCI (64 біта, 33 Мгц), блок живлення 250 Ут, вентилятори. Усі вузли однакові, робота програми користувача починається з спільного каталогу мережної файлової системи (він монтується на усіх вузлах однаково). Критерій призначення ресурсів - вузол. Програмні й апаратні засоби дозволяють вирішувати тільки одну задачу з використанням всього обчислювального ресурсу, ефективний поділ вирішального поля (сукупність усіх вузлів кластера) на частини необхідного розміру і надання їх декільком користувачам не передбачається.

Вузли кластера можуть бути зв'язані між собою високошвидкісною мережею SCI (пропускна здатність каналу дорівнює 2000 Мбит/с) і/чи мережею Fast Ethernet (пропускна здатність каналу до 100 Мбит/с). Мережа SCI призначена для високошвидкісного обміну між вузлами в ході обчислень. Мережа Fast Ethernet призначена для початкового завантаження програм і даних у вузли, а також для передачі службової інформації про хід обчислювального процесу при діючій мережі SCI. Якщо мережа SCI відсутня, то мережа Fast Ethernet реалізує й обміни даними між вузлами в ході обчислень. При обміні даними між двома вузлами по мережі SCI з використанням протоколів MPI може бути досягнута пропускна здатність на рівні 300 — 350 Мбайт/с. Пікова продуктивність кластера - 3,8 Gflops (при наявності мережі SCI), 1,3 Gflops (у відсутності мережі SCI).

В даний час в Інституті кібернетикі ім. В.М.Глушкова НАН України закінчується розробка більш потужного кластера із значно вищою продуктивністю.

Модель передачі повідомлень MPI та пакет MPICH: коротка характеристика, деякі угоди та позначення

MPI розшифровується як "message passing interface" ("взаємодія через передачу повідомлень") - це стандарт на програмний інструментарій для забезпечення зв'язку (інтерфейсу) між гілками паралельного додатку і являє собою бібліотеку процедур для передачі повідомлень [5,6]. При використанні бібліотеки MPI процеси розподіленої програми записуються на такій послідовній мові, як Сі або Фортран; їх взаємодія і синхронізація задається за допомогою визовів процедур бібліотеки MPI.

MPI надає програмісту єдиний механізм взаємодії гілок усередині паралельного додатку незалежно від машинної архітектури (однопроцесорні/мультипроцесорні з спільною/роздільною пам'яттю), взаємного розташування гілок (на одному процесорі / на різних) і API (розшифровується як "applications programmers interface" - "інтерфейс розроблювача додатків") операційної системи. Програма, що використовує MPI, легше налагоджується (звужується простір для здійснення стереотипних помилок паралельного програмування) і швидше переноситься на інші платформи (в ідеалі, простою перекомпіляцією).

Інтерфейс MPI був визначений в середині 1990-х років [7]. В даний час різними колективами розробників написано кілька програмних пакетів, що задовольняють специфікації MPI, зокрема: MPICH, LAM, HPVM і так далі. Вони виступають базовими при перенесенні MPI на нові архітектури комп’ютерів. У роботі використовується пакет MPICH, який написаний авторами специфікації, є безкоштовним і найбільш розповсюдженим. Таким чином, термін MPI використовується не тільки для позначення викладених у специфікації відомостей, але і, у визначеній мірі, для опису характеристик конкретної базової реалізації.

Мінімально до складу MPI входять: бібліотека програмування (заголовні і бібліотечні файли для мов Сі, Сі++ і Фортран) і завантажник додатків. Додатково включаються: профілюючий варіант бібліотеки (використовується на стадії тестування

паралельного додатку для визначення оптимальності розпаралелювання); завантажник із графічним і мережним інтерфейсом для X-Windows і інше. Структура каталогів MPICH виконана в повній відповідності з традиціями Юнікса: bin, include, lib, man, src, ... Мінімальний набір функцій простий в освоєнні і дозволяє швидко написати надійно працюючу програму.

Паралельний додаток складається з декількох гілок, чи процесів, чи задач, що виконуються одночасно. Різні процеси можуть виконуватися як на різних процесорах, так і на тому самому - для програми це ролі не грає, оскільки в обох випадках механізм обміну даними однаковий. Процеси обмінюються один з одним даними у вигляді повідомлень. Повідомлення проходять під ідентифікаторами, що дозволяють програмі і бібліотеці зв'язку відрізняти їх одне від одного. Для спільного проведення тих чи інших розрахунків процеси усередині додатку поєднуються в групи. Кожен процес може довідатися в бібліотеки зв'язку свій номер усередині групи, і, у залежності від номеру, приступає до виконання відповідної частини розрахунків.

У MPI гілка запускається і працює як звичайний процес Юнікса, пов'язаний через MPI з іншими процесами, що входять у додаток. В іншому процеси варто вважати ізольованими один від одного: у них різні області коду, стека і даних (як і в Юніксовських процесах). Говорять, що процеси мають роздільну пам'ять (separate memory).

Особливістю MPI є поняття області зв'язку (communication domains). При запуску додатка всі процеси розміщуються в створювану для додатка спільну область зв'язку. При необхідності вони можуть створювати нові області зв'язку на базі існуючих. Всі області зв'язку мають незалежну одна від одної нумерацію процесів. Програмі користувача в розпорядження надається комунікатор - дескриптор області зв'язку. Багато функцій MPI мають серед вхідних аргументів комунікатор, що обмежує сферу їхньої дії тією областю зв'язку, до якого він прикріплений. Для однієї області зв'язку може існувати декілька комунікаторів таким чином, що додаток буде працювати з нею як з декількома різними областями. У вхідних текстах для MPI часто використовується ідентифікатор MPI_COMM_WORLD. Це назва комунікатора, що створюється бібліотекою автоматично. Він описує стартову область зв'язку, що поєднує всі процеси додатку.

Номер процесу - ціле невід’ємне число, що є унікальним атрибутом кожного процесу. Атрибути повідомлення - номер процесу-відправника, номер процесу-отримувача й ідентифікатор повідомлення. Для них заведена структура MPI_Status, що містить три поля: MPI_Source (номер процесу відправника), MPI_Tag (ідентифікатор повідомлення), MPI_Error (код помилки); можуть бути і додаткові поля. Ідентифікатор повідомлення (msgtag) - атрибут повідомлення, який є цілим невід’ємним числом, що лежить у діапазоні від 0 до 32767. Процеси поєднуються в групи, можуть бути вкладені групи. Всередині групи всі процеси пронумеровані. З кожною групою асоційований свій комунікатор. Тому при здійсненні пересилання необхідно вказати ідентифікатор групи, всередині якої виробляється це пересилання. Усі процеси містяться в групі з визначеним ідентифікатором MPI_COMM_WORLD. При опису процедур MPI будемо користатися ідентифікатором OUT для позначення "вихідних" параметрів, тобто таких параметрів, через які процедура повертає результати.

В MPI існує три категорії функцій: блокуючі, локальні, колективні.

Блокуючі - зупиняють (блокують) виконання процесу доти, поки вироблена ними операція не буде виконана. Функції, що неблокують, повертають керування негайно, а виконання операції продовжується у фоновому режимі; за завершенням операції треба простежити особливо. Функції, що неблокують, повертають квитанції ("requests"), що погашаються при завершенні. До погашення квитанції із змінними і масивами, що були аргументами неблокуючої функції, нічого робити не можна.

Локальні - не ініціюють пересилань даних між гілками. Більшість інформаційних функцій є локальними, тому що копії системних даних уже зберігаються в кожній гілці. Функція передачі MPI_Send і функція синхронізації MPI_Barrier (див. далі) не є

 ISBN 7-776-8361-7

локальними, оскільки роблять пересилання. Відзначимо, що, приміром, функція прийому MPI_Recv (парна для MPI_Send) є локальною: вона усього лише пасивно чекає надходження даних, нічого не намагаючись повідомити іншим гілкам.

Колективні - повинні бути викликані всіма гілками-абонентами того комунікатора, що передається їм як аргумент. Недотримання для них цього правила приводить до помилок на стадії виконання програми (як правило, до повисання програми).

Розширення мови САА/Д засобами паралельного програмування і підключення цільової мови

З метою розробки САА/Д-схем для кластерів у мову САА/Д – вхідну для ДТМ – додатково введені оператори ПАРАЛЕЛЬНО і ЧЕКАТИ, які є реалізаціями операцій асинхронної диз'юнкції і фільтрації модифікованих алгебр Глушкова. У залежності від цільової мови програмування і використовуваної бібліотеки процедур, оператор ПАРАЛЕЛЬНО може приймати три форми:

1. ПАРАЛЕЛЬНО(B1, B2,...,BN);
2. ПАРАЛЕЛЬНО(N);
3. ПАРАЛЕЛЬНО.

Перша форма використовується для запуску паралельних гілок B1, B2,...,BN, друга – для створення і запуску на різних процесах N копій процесів, що описані САА/Д-схемою, третя форма використовується у випадку, коли число копій задається зовнішнім по відношенню до програми чином. Останнє має місце при використанні бібліотек MPI.

За допомогою оператора ЧЕКАТИ виконується затримка функціонування гілок. Цей оператор продовжує виконання процесу лише після того, як усі процеси досягнуть точки програми, у якій знаходиться оператор ЧЕКАТИ.

Стандартна предметна область – база знань ДТМ - також містить поняття й операції над ними, за допомогою яких виконується обмін повідомленнями. Зокрема, для обміну інформацією між гілками в САА/Д вводиться поняття ПОВІДОМЛЕННЯ. Кожне повідомлення характеризується унікальним ідентифікатором. Для пересилання повідомлень визначені елементарні оператори відправлення і прийому повідомлень. В якості параметрів ці оператори приймають ідентифікатор повідомлення, номер процесу-отримувача або процесу-відправника, тип переданих даних і так далі. Наприклад, передача цілого числа i-ому процесу від поточного виконується в такий спосіб:

"ПОВІДОМЛЕННЯ 99 - ЧИСЛО intervals ВІДПРАВИТИ ПРОЦЕСУ З НОМЕРОМ ЧИСЛО i".

У розподілених програмах кожен процес має свій унікальний номер. Як правило, процес з номером 0 є головним і виконує керування іншими процесами. З цієї причини в предметній області існують елементарні оператори для визначення номера поточного процесу і загальної кількості процесів.

В даний час стандартна предметна область ДТМ дозволяє виконувати трансляцію паралельних САА/Д-схем у програми мовою С, що використовують пакет MPICH. Оскільки при використанні стандарту MPI число процесів задається не в самій програмі, а за допомогою спеціального файлу і параметра командного рядка, то в паралельних САА/Д-схемах використовується третя форма оператора ПАРАЛЕЛЬНО.

Хоча з теоретичної точки зору гілкам для організації обміну даними досить всього двох операцій (прийом і передача), на практиці усе обстоїть набагато складніше. Одними тільки комунікаціями "точка-точка", тобто такими, у яких рівно один передавальний процес і рівно один (приймаючий) займається порядку 40 функцій. Користаючись ними, програміст має можливість вибрати:

1) Спосіб зачеплення процесів - у випадку неодночасного виклику двома процесами парних функцій прийому і передачі може бути зроблений автоматичний вибір одного з трьох варіантів:

- буферизація на передавальній стороні - функція передачі заводить тимчасовий буфер, копіює в нього повідомлення і повертає керування процесу, що викликав (вміст буфера буде передано у фоновому режимі);

- чекання на прийомній стороні, завершення з кодом помилки - на передавальній;

- чекання на передавальній стороні, завершення з кодом помилки - на прийомній.

2) Спосіб взаємодії комунікаційного модуля MPI із викликаючим процесом:

- блокуючий - керування викликаючому процесу повертається тільки після того, як дані прийняті чи передані (чи скопійовані в тимчасовий буфер);

- неблокуючий - керування повертається негайно (тобто процес блокується до завершення операції), і фактична прийомо-передача відбувається в фоні. Функція неблокуючого прийому має додатковий параметр типу "квитанція". Процес не має права робити які-небудь дії з буфером повідомлення, поки квитанція не буде "погашена";

- персистентний - в окремі функції виділені: створення "каналу" для прийому/передачі повідомлення; ініціація прийому/передачі; закриття каналу. Такий спосіб ефективний, приміром, якщо прийомо-передача відбувається усередині циклу, а створення/закриття каналу винесені за його границі.

Найпростіші (але і самі повільні) функції - MPI_Recv і MPI_Send (див. далі) - виконують блокуючу прийомо-передачу з автоматичним вибором зачеплення (до речі, усі функції прийому сумісні з усіма функціями передачі). Для підключення до мови САА/Д засобів паралельного програмування і синтезу розподілених додатків в даній версії ДТМ використовуються наступні загальні процедури MPI:

int MPI_Init(int* argc, char* argv)**

MPI_Init - ініціалізація паралельної частини додатку. Реальна ініціалізація для кожного додатку виконується не більше одного разу, а якщо MPI уже був ініціалізований, то ніякі дії не виконуються і відбувається негайне повернення з підпрограми. Усі MPI-процедури, що залишилися, можуть бути викликані тільки після виклику MPI_Init. Повертає: у випадку успішного виконання - MPI_SUCCESS, інакше - код помилки. (Те ж саме повертають і всі інші функції, що розглядаються далі.)

int MPI_Finalize(void)

MPI_Finalize - завершення паралельної частини додатка. Усі наступні звертання до будь-яких MPI-процедур, у тому числі до MPI_Init, заборонені. До моменту виклику MPI_Finalize деяким процесом усі дії, що вимагають його участі в обміні повідомленнями, повинні бути завершені. Складний тип аргументів MPI_Init передбачений для того, щоб передавати всім процесам аргументи main:

```
int main(int argc, char** argv)
{ MPI_Init(&argc, &argv);  ...
 MPI_Finalize();}
```

int MPI_Comm_size(MPI_Comm comm, int* size)

Визначення загального числа паралельних процесів у групі comm. Тут comm - ідентифікатор групи, OUT size - розмір групи.

int MPI_Comm_rank(MPI_comm comm, int* rank)

Визначення номеру процеса в групі comm. Значення, що повертається за адресою &rank, лежить у діапазоні від 0 до size_of_group-1. Тут comm - ідентифікатор групи, OUT rank - номер викликаючого процесу в групі comm.

Прийом/передача повідомлень (із блокуванням) між окремими процесами здійснюється в ДТМ наступними функціями:

int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm)

Тут buf - адреса початку буфера посилки повідомлення, count - число переданих елементів у повідомленні, datatype - тип переданих елементів, dest - номер процесу-одержувача, msgtag - ідентифікатор повідомлення, comm - ідентифікатор групи. Посилка блокуючого повідомлення з ідентифікатором msgtag, що складається з count елементів

 ISBN 7-776-8361-7

типу datatype, процесу з номером dest. Всі елементи повідомлення розташовані підряд у буфері buf. Значення count може бути нулем. Тип переданих елементів datatype повинен указуватися за допомогою визначених констант типу. Дозволяється передавати повідомлення самому собі. Блокування гарантує коректність повторного використання всіх параметрів після повернення з підпрограми. Вибір способу здійснення цієї гарантії: копіювання в проміжний буфер чи безпосередня передача процесу dest, залишається за MPI. Слід спеціально зазначити, що повернення з підпрограми MPI_Send не означає ні того, що повідомлення вже передане процесу dest, ні того, що повідомлення залишило процесорний елемент, на якому виконується процес, що виконав MPI_Send.

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_comm comm, MPI_Status *status)

Тут OUT buf - адреса початку буфера прийому повідомлення, count - максимальне число елементів у прийнятому повідомленні, datatype - тип елементів прийнятого повідомлення, source - номер процесу-відправника, msgtag - ідентифікатор прийнятого повідомлення, comm - ідентифікатор групи, OUT status - параметри прийнятого повідомлення. Прийом повідомлення з ідентифікатором msgtag від процесу source із блокуванням. Число елементів у прийнятому повідомленні не повинне перевершувати значення count. Якщо число прийнятих елементів менше значення count, то гарантується, що в буфері buf зміняться тільки елементи, що відповідають елементам прийнятого повідомлення. Якщо потрібно довідатися про точне число елементів у повідомленні, то можна скористатися підпрограмою MPI_Probe. Блокування гарантує, що після повернення з підпрограми всі елементи повідомлення прийняті і розташовані в буфері buf. В якості номера процеса-відправника можна вказати визначену константу MPI_ANY_SOURCE - ознака того, що підходить повідомлення від будь-якого процесу. В якості ідентифікатора прийнятого повідомлення можна вказати константу MPI_ANY_TAG - ознака того, що підходить повідомлення з будь-яким ідентифікатором. Якщо процес посилає два повідомлення іншому процесу й обидва ці повідомлення відповідають тому самому виклику MPI_Recv, то першим буде прийняте те повідомлення, що було відправлено раніше.

Колективні взаємодії процесів в ДТМ здійснюється наступними функціями.

В операціях колективної взаємодії процесів беруть участь усі процеси додатка. Відповідна процедура повинна бути викликана кожним процесом, можливо, зі своїм набором параметрів. Повернення з процедури колективної взаємодії може відбутися в той момент, коли участь процесу в даній операції уже закінчено. Як і для блокуючих процедур повернення означає те, що дозволено вільний доступ до буфера чи прийому посилки, але не означає ні того, що операція завершена іншими процесами, ні навіть того, що вона ними розпочата (якщо це можливо за змістом операції).

int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int source, MPI_Comm comm)

Тут OUT buf - адреса початку буфера посилки повідомлення, count - число переданих елементів у повідомленні, datatype - тип переданих елементів, source - номер процесу, що розсилає, comm - ідентифікатор групи. Розсилання повідомлення від процесу source усім процесам, включаючи процес, що розсилає. При поверненні з процедури вміст буфера buf процесу source буде скопійовано в локальний буфер процесу. Значення параметрів count, datatype і source повинні бути однаковими у всіх процесів.

int MPI_Reduce(void *sbuf, void *rbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Тут sbuf - адреса початку буфера для аргументів, OUT rbuf - адреса початку буфера для результату, count - число аргументів у кожного процесу, datatype - тип аргументів, op - ідентифікатор глобальної операції, root - процес-одержувач результату, comm - ідентифікатор групи. Функція аналогічна попередній, але результат буде записаний у буфер rbuf тільки процесу root.

Синхронізація процесів в ДТМ здійснюється наступною процедурою.

int MPI_Barrier(MPI_Comm comm)

Тут comm - ідентифікатор групи. Блокує роботу процесів, що викликали дану процедуру, доти, доки всі процеси групи comm, що залишилися, також не виконають цю процедуру.

Приклад розподіленого додатку, що синтезований в ДТМ

В якості прикладу розглядається задача обчислення значення визначеного інтегралу на відрізку [0;1]:

$$I = \int_0^1 f(x)dx \qquad (1)$$

На відрізку [0, 1] введемо рівномірну сітку $\{x_j \mid x_j = j*h;\ j=0,N;\ h*N = 1\}$. Наближене значення інтеграла будемо обчислювати згідно з наступною формулою:

$$I_N = \sum_{j=0}^{N-1} f(x_j + 0.5h)h \qquad (2)$$

Задача полягає в розподілі обчислення I_N на необхідній кількості процесорів. Нехай ми маємо необмежене число процесорів, тоді розіб'ємо вхідну задачу на елементарні

$$I_j = f(x_j + 0.5h)h,\ \ j = \overline{1,N} \qquad (3)$$

і призначимо j-у задачу на один процесор, усього буде задіяно N процесорів. Послідовність обчислень можна визначити в такий спосіб:

1. Процесор P_0 - запитує в користувача число інтервалів N;
2. Процесор P_0 - передає N всім іншим процесорам P_1 ..P_{N-1};
3. Процесори P_1 ..P_{N-1} приймають від P_0 значення N;
4. Процесори P_0 ..P_{N-1} обчислюють h і I_j по формулі (3);
5. Процесори P_1 ..P_{N-1} передають P_0 значення I_j, $j = \overline{1, N-1}$;
6. Процесор P_0 - приймає значення I_j від процесорів P_1 ..P_{N-1} і обчислює суму I_N.

САА/Д-схема, що відповідає даному алгоритмові (процес проектування в ДТМ САА/Д-схем детально описаний в [1,2]), має вигляд:

СХЕМА ІНТЕГРАЛ; "ІНТЕГРАЛ" === (**ПАРАЛЕЛЬНО** * "ЧИСЛО me ЦЕ НОМЕР ПОТОЧНОГО ПРОЦЕСУ" * "ЧИСЛО size ЦЕ КІЛЬКІСТЬ ЗАДАЧ" * **ЯКЩО** ('ЧИСЛО me=ЧИСЛО 0') **ТО** ("РЯДОК "ВВЕДІТЬ ЧИСЛО ІНТЕРВАЛІВ" ВИВЕСТИ НА ЕКРАН"*"ЧИСЛО intervals ВВЕСТИ З КЛАВІАТУРИ") **КІНЕЦЬ_ТО*** **ЧЕКАТИ** * **ЯКЩО** ('ЧИСЛО me = ЧИСЛО 0') **ТО** ("ЧИСЛО i ЦЕ ЧИСЛО 1" * **ПОКИ НЕ** ('ЧИСЛО i >= ЧИСЛО size') **ЦИКЛ** ("**ПОВІДОМЛЕННЯ 99** - ЧИСЛО intervals ВІДПРАВИТИ ПРОЦЕСУ З НОМЕРОМ ЧИСЛО i" * "ЧИСЛО i ЗБІЛЬШИТИ НА ЧИСЛО 1") **КІНЕЦЬ_ЦИКЛ**) **КІНЕЦЬ_ТО**) **ІНАКШЕ** ("**ПОВІДОМЛЕННЯ 99** ВІД ПРОЦЕСУ З НОМЕРОМ ЧИСЛО 0 ПРИЙНЯТИ В ЧИСЛО intervals І ЗАПАМ'ЯТАТИ СТАТУС У СТАТУС st") **КІНЕЦЬ_ІНАКШЕ *** "ДІЙСНЕЧИСЛО h це ДІЙСНЕЧИСЛО (1/intrevals)" * "ДІЙСНЕЧИСЛО ik це ДІЙСНЕЧИСЛО 0" * "ЧИСЛО j ЦЕ ЧИСЛО me" * **ПОКИ НЕ** ('ЧИСЛО j >= ЧИСЛО intervals') **ЦИКЛ** ("ДІЙСНЕЧИСЛО x ЦЕ ДІЙСНЕЧИСЛО ((j+0.5)*h)" * "ДІЙСНЕЧИСЛО ik ЦЕ ДІЙСНЕЧИСЛО (ik+4/(1+x*x))" * "ЧИСЛО j ЗБІЛЬШИТИ НА ЧИСЛО size") **КІНЕЦЬ_ЦИКЛ** "ДІЙСНЕЧИСЛО ik ЦЕ ДІЙСНЕЧИСЛО (ik*h)" * **ЧЕКАТИ** * **ЯКЩО** ('ЧИСЛО me = ЧИСЛО 0') **ТО** ("ЧИСЛО i ЦЕ ЧИСЛО 1" * **ПОКИ НЕ** ('Число i >= Число size') **ЦИКЛ** ("**ПОВІДОМЛЕННЯ 98** ВІД ПРОЦЕСУ З НОМЕРОМ ЧИСЛО І ПРИЙНЯТИ У ДІЙСНЕЧИСЛО x І ЗАПАМ'ЯТАТИ СТАТУС У СТАТУС st" * "ДІЙСНЕЧИСЛО ik ЦЕ ДІЙСНЕЧИСЛО (ik+x)" * "ЧИСЛО i ЗБІЛЬШИТИ НА ЧИСЛО 1") **КІНЕЦЬ_ЦИКЛ** * "ДІЙСНЕЧИСЛО ik ВИВЕСТИ НА ЕКРАН") **КІНЕЦЬ_ТО ІНАКШЕ** ("ПОВІДОМЛЕННЯ 98 - ДІЙСНЕЧИСЛО ik ВІДПРАВИТИ ПРОЦЕСУ З НОМЕРОМ ЧИСЛО 0") **КІНЕЦЬ_ІНАКШЕ**) **КІНЕЦЬ СХЕМИ**;

 ISBN 7-776-8361-7

Ця схема за допомогою ДТМ автоматично транслюється в наступну програму:

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include «mpi.h»
int main(int argc, char *argv[]); float h; float ik; float x; MPI_Status st; int me; int size; int i; int intervals; int j; MPI_Init(&argc, &argv); MPI_COMM_WORLDrank (MPI_COMM_ WORLD, &me); MPI_COMM_WORLDsize(MPI_COMM_WORLD,&size); if(me == 0) {puts("Уведіть число інтервалів"); scanf("%d", &intervals); }/* if */ MPI_Barrier(MPI_ COMM_WORLD); if(me == 0) { i = 1; while(!(i > size)) {MPI_Send(&intervals, 1, MPI_INT, i, 99, MPI_COMM_WORLD); i += 1;} /* while */ }/* if */ else { MPI_Recv(&intervals, 1, MPI_INT, 0, 99, MPI_COMM_WORLD, &st);}/* else */ h = (1.0/intrevals); ik = 0; j = me; while(!(j > intervals)) { x = ((j+0.5)*h); ik = (ik+4/(1+(x*x))); j += size; } /* while */ ik = (ik*h); MPI_Barrier(MPI_COMM_WORLD); if(me == 0) { i = 1; while(!(i > size)) {MPI_Recv(&x, 1, MPI_DOUBLE, i, 98, MPI_COMM_WORLD, &st); ik = (ik+x); i += 1; } /* while */ printf("%f\n", ik); }/* if */ else {MPI_Send(&ik, 1, MPI_DOUBLE, 0, 98, MPI_COMM_WORLD); }/* else */ MPI_Finalize(); return 0; } /* main */
```

Висновок. Стаття присвячена актуальній тематиці створення інструментальних засобів автоматизації проектування розподілених додатків для кластерів (багатомашинних систем) – галузі знань, що інтенсивно розвивається як в Україні, так і за кордоном. В статті проаналізовані деякі апаратні і мовні особливості кластерів в порівнянні з іншими розподіленими обчислювальними системами – мультипроцесорами і комп'ютерними мережами; мова САА/Д розширена деяким засобами паралельних обчислень не лише модифікофаних алгебр Глушкова, але і обміну повідомлень; до мови САА/Д підключені відповідні засоби паралельного програмування пакету MPICH; нові можливості ДТМ експеременталько продемостровані на прикладі синтезу в ДТМ розроділеного додатку.

Створення інструментарію синтезу розподілених додатків для кластерів відкриває можливість рішення важливого класу задач (економічних, охорони навколишнього середовища і інших), які, в силу своїх властивостей – величезні об'єм і кількість виконуваних операцій, не можуть бути вирішені в прийнятний час на розповсюджених зараз "персональних" комп'ютерах.

In this work a language SAA/D – the input dialog transformation machine's language (DTM) – is broaden some means parallel computation modified Glushkov's algebra, to whom (means) installed the answer to them means of parallel programming of product MPICH. DTM's opportunity on synthesis apportionate tasks for clusters is demonstrating at example.

1. Петрушенко А.Н., Хохлов В.А., Ткачев В.А., Шепетухин Е.С. Диалоговая трансформационная машина: некоторые функциональные возможности // Проблемы программирования. – 2000. – № 1–2 (Спец. выпуск) – С. 323–334.

2. Петрушенко А.М., Хохлов В.А. Концепція діалогових обчислень та деякі проблеми автоматизації програмування // Там же. – 2004. – № 2–3 (Спец. выпуск) – С. 37–47.

3. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования: Пер. с англ. – М.: Издательский дом "Вильямс", 2003. – 512 с.

4. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – СПб: Питер, 2005. – 864 с.

5. M. Snir, S. Otto, S. Huss-Ledermann, D. Walker and J. Dongarra. 1996. MPI: The Complete Reference. Cambridge, MA: MIT Press.

6. http://parallel.ru/tech/tech.dev/mpi.html

7. www-unics.mcs.anl.gov/mpi/mpich