УДК 51:681.3.06

А.Ю. Дорошенко, І.С. Кононенко, О.Ю. Коротун

КООРДИНАЦІЙНІ АБСТРАКЦІЇ ПРОГРАМУВАННЯ ДЛЯ ЕФЕКТИВНОГО РОЗПАРАЛЕЛЮВАННЯ ЗА ДАНИМИ

Для найбільш поширеної в системах паралельного програмування парадигми розпаралелювання за даними характерна дуже проста організація керування обчисленнями за допомогою засобів бар'єрного типу. Однак навіть у випадках паралельних застосувань з регулярним характером обчислень ефективність паралельних програм, де використовуються ці засоби, є невисокою. Головною причиною такого недовикористання потенційної продуктивності паралелізму є надто обмежувальний характер зазначених бар'єрних засобів. У цій статті для класу паралельних за даними програм з розподіленою та спільною пам'яттю і статичною дисципліною доступу до спільної пам'яті з боку паралельних компонент нами запропоновано координаційні абстракції паралельного програмування у вигляді формальних регулярних виразів, що мають на меті підвищення ефективності синхронізації та обміну даними між цими компонентами. Показано застосування запропонованих засобів програмування для вирішення проблем автоматизації розробки ефективних паралельних програм.

1. Вступ

Розв'язання багатьох важливих проблем паралельної обробки в мультипроцесорних комп'ютерних системах так чи інакше стосується розробки засобів паралельного програмування достатньо загальних та абстрактних за своєю виразною силою і водночас простих і практичних за своїм використанням. Таке поєднання є суперечливим за своєю суттю, і тому часто розробники паралельного програмного забезпечення віддають перевагу якомусь одному з названих аспектів, найчастіше - простоті застосування. Наприклад, для найбільш поширеного в системах паралельного програмування стилю SPMD (одна програма для багатьох даних) характерна дуже проста організація керування обчисленнями за допомогою засобів бар'єрного типу (семафори або похідні від них засоби, а також операторні дужки на зразок PARDO...PAREND). Однак навіть у випадках паралельних застосувань з регулярним характером обчислень показники ефективності паралельних програм, де використовуються ці засоби, є невисокими [1]. Головною причиною такого недовикористання потенційної продуктивності паралелізму є надто обмежувальний характер зазначених бар'єрних засобів.

У останні роки з'явились ознаки нових підходів до поєднання властивостей виразної сили та простоти використання засобів паралельного програмування, зокрема, розробки спеціальних координаційних засобів і координаційних мов програмування [2-3], що є ортогональними до засобів та мов виразу власне обчислень. Хоча ідеї незалежного опису і взаємодії (координації) моделі обчислень та моделі керування цими обчисленнями не нові. Ще у 70-х роках були розроблені теорія дискретних перетворювачів [4], керуючих просторів [5] і засоби на зразок маршрутних виразів [6], в яких фактично вже використовувались елементи понять координації і взаємодії паралельних процесів. Проте лише тепер, коли набули такого широкого розвитку комп'ютерні мережі та Internet, приходить розуміння важливості концепцій координації та взаємодії як загальної парадигми програмування обчислень, про що свідчить спеціальні наукові конференції, присвячені цій тематиці [7].

У даній статті розвивається започаткований раніше [8] підхід до підвищення ефективності паралельних програм шляхом цілеспрямованого використання одного класу координаційних абстракцій програмування. Виявилося, що бар'єрні та семафорні засоби синхронізації загального призначення мають занадто обмежувальний характер для одержання ефективних паралельних програм, тому запропоновано аль-

тернативні засоби синхронізації (форсуючі вирази) як більш ефективні у специфічних випадках, важливих для практичного застосування [8]. В процесі дослідження цього специфічного класу паралельних програм [9-12] була запропонована модель паралельних програм з локальною розподіленою і глобальною спільною пам'яттю і статичною структурою доступів до спільної пам'яті, де будь-яка пара залежних за даними операторів паралельної програми повинна завжди виконуватися в одному і тому ж порядку. Незважаючи, на перший погляд, на серйозне звуження через детермінізацію обмінів даними класу паралельних програм, що розглядаються, цей клас насправді досить широкий і включає багато важливих для практичного застосування методів обчислень, наприклад, всі прямі методи для розв'язування систем лінійних алгебраїчних рівнянь. Крім того виявилось, що детерміноване визначення порядку обмінів даними через форсуючі вирази може значно полегшити розробку схем ефективних обмінів даних для цього класу програм у середовищі мультипроцесорних систем з багаторівневою пам'яттю завдяки наданню їм статусу координаційних програмних засобів, коли вони можуть стати формальною основою для систематичного скорочення кількості операцій доступу до повільних рівнів пам'яті [13-14]. У даній статті викладено результати подальшого дослідження та реалізації форсуючих виразів як координаційних засобів високого рівня та їх використання для прискорення паралельних обчислень, організованих за парадигмою паралелізму за даними (data parallel paradigm [15]).

2. Модель паралельних програм

Наша модель паралельних програм складається з двох ортогональних частин: моделі обчислень і моделі координації.

Модель обчислень. Паралельні програми розглядаються як кортежі виду $p = (P,K,t,E)$, де $P = \{P_i\}$ – скінченна множина послідовних компонентів програм (модулів), K – множина логічних компонентів – імен паралельних процесів, $t : K \to P$ – (часткове) відображення ініціалізації, яке забезпечує початкову конфігурацію програми, E – множина імен зовнішніх масивів. Модулі будуються як регулярні програми алгебри алгоритмів [4].

Нехай V – множина змінних і D – область значень цих змінних. Множина часткових відображень $B = \{b : B \to D\}$ зветься множиною станів пам'яті. Вважаємо відомою множину базових операторів $Op = \{y : B \to B\}$ модулів з одиничним оператором ε і множиною базових умов $Co = \{u : B \to \{\mathbf{0},\mathbf{1}\}\}$, що також включає логічні константи **0** і **1**. Алгебра часткових перетворень Y, яка породжується на Op за допомогою трьох операцій: конкатенації $P;Q$, операції розгалуження $u \to (P\ else\ Q)$, і ітерації $while(u,P)$, де P, Q є оператори і u – умова, зветься алгеброю операторів. Алгебра U, яка породжується на Co за допомогою логічних операцій і операції множення Pu оператора P на умову u, яка означає "u після того, як P", зветься алгеброю умов. Двоосновна алгебра $A(Y,U)$, що розглядається як множина операторів і умов, замкнена відносно вищезгаданих операцій, зветься алгеброю алгоритмів. Кожний елемент алгебри $A(Y,U)$ може бути представлений як регулярний вираз, побудований з базових операторів і базових умов і операцій алгебри алгоритмів. Регулярні вирази алгебри операторів звуться регулярними програмами.

Отже, ми розглядаємо модулі як послідовні регулярні програми, де множина базових операторів включає паралельний виклик процедур у вигляді `Pcall F(x)`, і оператори обмінів даними. Оператори обмінів можуть бути двох видів: прямі парні обміни за допомогою операторів надсилання `x→k1` і отримання `y←k2`, що виконуються в компонентах *k2* і *k1* відповідно, і зовнішні обміни через спільну пам'ять за допомогою операторів читання і запису `x:←A` та `y:→A`, де x, y – імена внутрішніх і A – зовнішнього масивів. Виконання пари відповідних один одному операторів надсилання/отримання `x→k1` та `y←k2` семантично еквівалентне присвоєнню `y:=x`.

Обчислення програми починається з одночасного запуску ініційованих модулів і закінчується при завершенні їх усіх. Передбачається, що вхідні та вихідні дані програми знаходяться в зовнішній пам'яті, що являє собою сукупність зовнішніх масивів дворівневої структури: на першому рівні – багатовимірний масив блоків даних, кожен з яких є багатовимірним масивом елементів даних – на другому рівні. Для синхронізації зовнішніх обмінів передбачається, що існують стандартні можливості, які складаються з семафорних операторів закриття `l(x)` і відкриття `u(x)` блоку даних x зовнішнього масиву, і операторних дужок `PARDO...PAREND` для синхронізації операторів паралельного виклику всередині модуля. Останній змінює конфігурацію при додавання нового компонента для

нової конфігурації і вилученні його після завершення паралельного виклику.

Координаційна модель. Об'єктом визначення для координаційної моделі є зовнішні взаємодії паралельних компонент через спільну пам'ять, а традиційною метою її впровадження – підвищення рівня інтегрованості процесів, породжуваних паралельними програмами [3]. У цій статті координаційна модель застосовується для іншої мети – підвищення продуктивності паралельних програм. Як правило, в мовах паралельного програмування для синхронізації доступу паралельних процесів до спільних об'єктів використовуються семафори або інші механізми блокувань (критичні області, монітори та ін.), що базуються на семафорах. Ці засоби разом з структурними засобами опису паралелізму, такими як операторні дужки *fork...join*, є універсальними для програмування довільних схем взаємодій між процесами через спільну пам'ять. Однак їх застосування в загальному випадку може призвести до втрати ефективності. Важливим способом підвищення ефективності паралельних обчислень на мультипроцесорних ЕОМ є створення більш ефективних спеціалізованих засобів синхронізації, що дозволяють більш високий рівень асинхронності виконання обмінів з спільною пам'яттю. Зокрема, такими спеціалізованими засобами можуть бути координаційні абстракції програмування, альтернативні семафорним блокуванням загального призначення. Ідея полягає в тому, щоб використовуючи апріорну інформацію про порядок доступу компонент до спільної пам'яті при детермінованих обмінах, зафіксувати цей порядок у вигляді формальних виразів, названих *форсуючими*, і потім застосувати їх як новий декларативний засіб для синхронізації даного класу зовнішніх обмінів.

Поняття форсуючих виразів визначається для випадку детермінованих обмінів через спільну пам'ять в безгоночних паралельних програмах. Остання властивість означає, що будь-яка пара залежних за даними операторів, щонайменше один з яких змінює значення спільної змінної, завжди повинна бути виконана в одному і тому ж порядку. Формальне визначення форсуючих виразів подається нижче в загальних термінах алгебри алгоритмів [4].

Нехай $p = (P,K,t,E)$ – паралельна програма за даним вище означенням. Для кожного $k \in K$, позначимо через R_k символ читання і через W_k – символ запису. Припустимо також, що визначена бінарна операція паралельного читання (P,Q) є комутативною і асоціативною операцією, побудованою з регулярних виразів P і Q, що використовують тільки символи читання. Нехай $K(A)$ – множина компонентів безгоночної паралельної програми, які обмінюються через зовнішній масив A. Тоді *форсуючий вираз* (ФВ) є регулярний вираз $f(A)$, побудований з символів R_k і W_k, $k \in K(A)$, одиничного оператора та алгебри умов над пам'яттю за допомогою засобів регулярних операцій і операцій паралельного читання.

Уточнимо тепер поняття паралельної програми у такий спосіб. Будемо називати паралельну програму $p = (P,K,t,E,F)$, де F – множина форсуючих виразів, *безгоночною*, якщо її зовнішні обміни керуються форсуючими виразами з множини F. Виконання безгоночної програми полягає в сумісному виконанні модулів і форсуючих виразів. Виконання оператора читання (запису), яке ініціюється в модулі k, повинно відповідати інтерпретації символу $R_k(W_k)$ як поточного символу ФВ, інакше виникає синхронізаційна затримка в цьому модулі, доки не відбудеться потрібна операція читання (запису). Звичайно, не має значення, що ФВ визначається саме в термінах регулярних програм. Для цього можна було обрати іншу нотацію чи іншу базову мову програмування. Істотним є те, що ФВ – має бути узгоджений з модульними програмами. У цьому контексті ФВ як координаційна компонента є суттєвою і невід'ємною частиною паралельної програми, яка забезпечує очікувану поведінку обчислення програми. Формальна операційна семантика ФВ в загальних термінах транзитивних систем розроблена в [5,6], де показано, що форсуючі вирази мають більшу виразну силу і є більш ефективними, ніж бар'єроподібні і семафороподібні засоби.

3. Посилення паралелізму за даними: приклад з лінійної алгебри

Паралелізм за даними [15] є найбільш уживаним видом паралельних обчислень в мультипроцесорних системах. У своїй поширеній формі SPMD (Single Program Multiple Data) [1] він передбачає паралельне застосування однієї і тієї ж програми до незалежних даних. Популярність цих обчислень пояснюється перш за все простотою реалізації і засобів програмування, серед яких для синхронізації паралельних обчислень використовуються семафорні та бар'єрні засоби типу PARDO...PAREND. Проте простота програмування і використання часто призводить до

втрат продуктивності паралельних обчислень, навіть у випадках задач з регулярною структурою.

Головна ідея введення форсуючих виразів ФВ – забезпечити передбачуваний порядок доступів паралельних компонентів до спільної пам'яті (що може бути відомий наперед у випадку безгоночної програми) і тим самим усунути надмірний недетермінізм з метою підвищення продуктивності паралельної програми. У цьому відношенні ФВ є більш тонким засобом, ніж стандартні засоби синхронізації, прийняті для паралелізму за даними. Нижче показані переваги застосування ФВ для ефективної синхронізації обмінів зовнішніми даними в паралельних за даними обчисленнях на прикладі відомої обчислювальної задачі лінійної алгебри – приведення матриці системи лінійних рівнянь до трикутного виду методом Холецького [16]. Нагадаємо, що показниками продуктивності паралельної програми виступають: T_p – час виконання програми на p процесорах, прискорення обчислень програми на p процесорах $s(p) = T_1/T_p$ і ефективність паралельних обчислень на p процесорах $e(p) = T_1/pT_p$.

Для спрощення будемо вважати вхідну матрицю $\mathbf{A}(a_{ij})$ і матрицю Холецького $\mathbf{L}(l_{ij})$ – щільнозаповненими симетричними додатньовизначеними матрицями розмірності $N * N$, які розбиваються на квадратні блоки розмірності $B * B$, так що $n = N/B$ – є ціле число. Визначимо множину компонентів як $\{K(i,j):1\le j<i\le n\}$, і вважатимемо, що факторизація (i,j)-го блоку здійснюється компонентом $K(i,j)$. Позначимо через r_{ij}^{jk} момент зчитування компонентом блоку, що був створений компонентом $K(i,j)$, і w_{ij} – момент запису результуючого блоку в зовнішній масив компонентом $K(i,j)$. Тоді умови коректної синхронізації можуть бути представлені у вигляді $w_{ij}<r_{il}^{ij}$, $j < l \le i$, та $w_{jk} < r_{ij}^{jk}$, $l \le k \le j$. Ці нерівності (назвемо їх специфікаціями) можуть бути реалізовані за допомогою різних стратегій. Якщо фронти визначені як $Q_k = \{(i,j) : i + j = const, 1 < i + j \le 2n\}$ $1 \le k \le 2n - 1$ (Рис. 1a), то достатньо лише синхронізації засобами *PARDO...PAREND*. Для отримання оцінки складності доцільно прийняти за одиницю час, потрібний для приведення блоку з першого стовбчика. Тоді, $T_1 = n^3/6 + O(n^2)$ та $T_{n/2} = n^2 + O(n)$. Таким чином, в цьому випадку ефективність $e(p) = T_1/pT_p$ є порядку $1/3 + O(1/n)$.

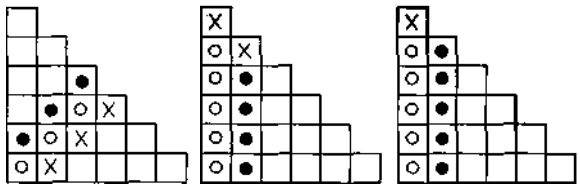

Рис. 1. Факторизація методом Холецького: вигляд фронтів

Якщо фронти організовані вздовж стовпчиків матриці компонентів і для синхронізації використовуються семафороподібні оператори, то фронт $Q_k, 1 \le k \le 2n - 1$ включає діагональний компонент, якщо k – непарне та складається з компонентів того ж стовпчика, але без діагонального компонента, якщо k – парне (Рис. 1b). Незважаючи на збільшення ширини фронту з $n/2$ до n, це не поліпшує порядку часу обчислень $T_n = n^2 + O(n)$ і навіть зменшує порядок ефективності до $1/6 + O(1/n)$ через дуже велику різницю в обсязі обчислень для різних точок одного й того ж фронту. Тут діагональні блоки грають роль бар'єра для синхронізації обчислень блоків одного стовпчика.

Форсуючі вирази для цієї безгоночної програми такі:

$$f(\mathbf{A}) = R_{ij};$$

$$f(\mathbf{L}_{ij}) = W_{ij};(while(u,R_{ij+k}),while(v,R_{ij})),$$

де $0 < j < i \le n$, $u = (1 \le k \le i - j)$, $v = (i \le l \le n)$. Може бути показано, що ці вирази забезпечують коректний доступ компонентів до блоків зовнішніх масивів і дозволяють часткове перекриття обчислень діагонального блоку і стовпчиків (як зображено на Рис. 1c фронти перекриваються). Форсуючі вирази залишають ефективність на рівні $1/3 + O(1/n)$, але дозволяють одержати варіант програми, який приблизно вдвічі швидший за попередні: $T_n = n^2 + O(n)$.

Ці результати підтверджуються експериментальною реалізацією на мові Java (JDK 1.1.5 від Sun, на процесорі Pentium 60 MHz), де три наведені вище стратегії моделюються з використанням потоків. В наступній Табл.1 наведена тривалість виконання програми для цих стратегій, яка реалізує задачу факторизації Холецького при кількості рівнянь $N = 200$ та різних розмірах блоку T (10, 20 і 50), за умови, що паралелізм обчислення є необмеженим.

Табл. 1. Тривалість виконання Java-програми для різних стратегій синхронізації (в msec)

Стратегія	T=10	T=20	T=50
PARDO-PAREND	1970	4170	7850
Семафори	2143	3907	7638
Форсуючі вирази	1610	2580	5440

4. Прискорення обмінів даними методом керованої буферизації

Сучасні мультипроцесорні системи, що створюються для розв'язання крупномасштабних задач, як правило, мають ієрархічну багаторівневу структуру підсистеми пам'яті, рівні якої відрізняються за показниками обсягу і швидкодії. Підвищення швидкості обмінів в середовищі з багаторівневою пам'яттю за допомогою приховування затримки і покращення просторової/часової локальності доступів до пам'яті є важливим джерелом збільшення ефективності паралельної програми. Серед програмних методів вирішення цих питань можна відзначити методи рівня операційної системи і системи програмування, а також метод формальних програмних перетворень на рівні вхідних текстів програм. Перевага останнього полягає у тому, що він недорогий і найбільш ефективний завдяки цільовій спрямованості на конкретні класи паралельних програм. В даній статті розвивається метод формальних програмних перетворень програм, що має на меті систематичне видалення обмінів з повільними рівнями пам'яті шляхом застосування різних способів буферизації цих обмінів у розподіленій пам'яті багатопроцесорної ЕОМ.

Для керування буферизацією зовнішніх обмінів в паралельних програмах ми пропонуємо метод прогнозування, зумовлений використанням форсуючих виразів для здійснення такого прогнозування в класі безгоночних програм. Цей метод, названий керованою буферизацією, дає можливість не тільки мінімізувати розмір буфера і скоротити до мінімуму кількість звернень до зовнішньої пам'яті, але й автоматизувати перетворення програм для реалізації керованої буферизації. Формальне визначення поняття буфера однозначних зовнішніх обмінів безгоночної програми за допомогою ФВ можна подати у наступному вигляді. Нехай $p = (P,K,T,t,E,F)$ – макроконвеєрна програма, де обміни із зовнішніми змінними з множини E синхронізуються форсуючими виразами з F. Тоді буфером називається часткове відображення $B{:}E \to D$ таке, що $B \subseteq M$, де M – стан зовнішньої пам'яті програми. Таким чином, буфер містить копії значень деяких зовнішніх змінних і зберігає їх у розподіленій пам'яті процесорів.

Ми використовуємо метод, який замість алгоритмів загальноцільової буферизації бере до уваги спеціальне знання, яке отримується з форсуючих виразів. У багатьох важливих випадках перетворення дуже прості і буферизація коштує небагато. Проілюструємо застосування нашої координаційної моделі на прикладі множення матриць, коли використовується частковий випадок техніки буферизації, а саме програмна конвеєризація [17].

Нехай ми маємо великоблокове розпаралелювання з двома вхідними і однією результуючою матрицями розмірності $nm * nm$, які поділені на квадратні блоки елементів, що будуть розміщені у двовимірних зовнішніх масивах розмірності $n * n$ з двовимірних блоків даних $m * m$, відповідно, **A** , **B** і **C** . Алгоритм обчислює результуючу суму добутків $\mathbf{C}(i,j) = \mathbf{A}(i,k)*\mathbf{B}(k,j)$, $1 \le i \le n$, на рівні блоків елементів матриці, де + і * є звичайні операції суми і множення матриць. Компонент цієї програми, який обчислює (i,j)-блок матриці результату наведено нижче в двох видах: початковий і перетворений фрагмент програми.

```
Z:= 0;
  for k:=1 to n do{
     U:<- A(i,k);
     V:<- B(k,j);
     Z:=Z+U*V;
  }
Z:->C(i,j);
```

```
Z:= 0;
for k:=1 to n do{
   if q2=1 then {U:<-A(i,k); U->K(q1,q2+1)};
   else {U<-K(q1,q2-1);
        if q2<n then U->K(q1,q2+1)};
   if q1=1 then {V:<-B(k,j);V->K(q1+1,q2)};
   else {V<-K(q1-1,q2);
        if q1<n then V->K(q1+1,q2)};
  Z:=Z + U * V
}
Z:->C(i,j);
```

Рис. 2. Види компонент для блочного множення матриць

Зауважимо, що це перетворення може бути виконане цілком формально в термінах алгебри алгоритмів. Позначивши три оператори тіла циклу на Рис. 2 (зліва) як a, b, c відповідно, і допустивши, що алгебра містить тотожність $(abc)^n = ab(cab)^{n-1}c$, ми можемо одержати перетворення з Рис. 2 (справа) автоматично.

Наведена на Рис. 2 (справа) реалізація являє собою керований форсуючими виразами алгоритм для цієї задачі за тієї умови, що координати компонента зберігаються в парі змінних $(q1,q2)$ для кожного компонента. Якщо (i,j)-елемент матриці-результату обчислюється компонентом, який називається (i, j), форсуючі вирази для зовнішніх масивів будуть такими:

$$f(\mathbf{A}_{ij}) = while(u,R_{ik}),\ f(\mathbf{B}_{ij}) = while(u,R_{kj}),$$
$$f(\mathbf{C}_{ij}) = W_{ij},\ u = (1 \le k \le n),\ 1 \le i,j \le n.$$

Отже, стає зрозуміло, що буферизація обмінів масиву **C** непотрібна, тому що одного разу записаний блок матриці **C** потім ніколи не буде прочитаний. Але обміни масивів **A** і **B** – багатократні, і тому потрібно щоб вони були буферизовані засобами одноразового читання елемента до одного компонента і розповсюдження його потім до інших за допомогою швидких обмінів. У результаті, майже всі нелокальні обміни, а саме $2n^2(n-1)$, будуть замінені швидкими прямими обмінами між компонентами. Зазначимо, що внаслідок того, що буфер мігрує, зовнішня пам'ять не потрібна, за винятком тієї, яка забезпечується компілятором.

5. *Висновки*

Координаційні моделі, що є ортогональний до моделей обчислень, дають можливість по-новому підійти до розуміння двох найголовніших проблем паралельних обчисленнь - підвищення рівня інтелектуалізації програмування і збільшення продуктивності паралельних систем.

Головна координаційна мета форсуючих виразів - нав'зувати заздалегідь відомий для безгоночних програм порядок доступів компонентів до спільної пам'яті, що має призвести до ліквідації недетермінізму обмінів і тим самим збільшити продуктивність паралельної програми. Крім того, завдяки тій же властивості форсуючих виразів можна полегшити розробку схем ефективних обмінів даними в середовищі з багаторівневою пам'яттю і досягти систематичного зменшення доступів до повільної пам'яті. У поєднанні з методами розпаралелювання, такими як програмна реалізація конвеєрного режиму, вони можуть забезпечити нетривіальні методики розробки ефективних паралельних програм. Формальні перетворення програм, які базуються на методах форсуючих виразів, були реалізовані в програмному забезпеченні проектування макроконвеєрної багатопроцесорної системи [9], і тепер використовуються для розробки методів паралельного програмування на платформі Java [18].

1. *Giloi G.K.* Parallel supercomputer architecture and their programming models // Parallel Computing. - 1994. - Vol. 20. -N 10/11. - P. 1443-1470.

2. *Malone T.W., Crowston K.* The Interdisciplinary Study of Coordination // ACM Computing Surveys. - 1995. -Vol. 26. -N 1. - P. 87-120.

3. *Gelernter D., Carriero N.* Coordination Languages and Their Significance // Commun. ACM - 1992. - Vol. 35. - N 2 . - P. 97-107.

4. *Г лутков В.М., Капитонова Ю.В., Летичевский Л.А.* Автоматизация проектирования вычислительных машин. - К.: Наукова думка, 1975. - 232 с.

5. *Глушков В.М., Анисымов Л.В.* Управляющие пространства в асинхронных параллельных вычислениях / / Кибернетика. - 1980. - N 5. - С. 1-9.

6. *Campbell R.H., Habermann N.A.* The specification of process synchronization by path expressions // Lcct. Notes Comput. Sci. - 1974. - Vol. 16. - P. 89-102.

7. *Coordination Languages and Models / D. Garlan, D. Le Metayer (eds.) //* Proc. 2-nd Int. Conf. COORDINATIONS. - Berlin, Germany, Sept. 1997, Springer Verlag, LNCS-V.1282.

8. *Дорошенко Л.Е.* Метод синхронизации внешних обменов в макроконвейерных программах // Кибернетика и системный анализ. - 1991. - №5. - С. 68-76.

9. *Дорошенко Л.Е.* Методы повышения производительности параллельных программ с многоуровневой памятью // Кибернетика и системный анализ. - 1994. - № 4. - С. 117-128.

10. *Дорошенко А.Е.* Динамические модели параллельных вычислений для макроконвейерных программ // Кибернетика и системный анализ. - 1995. ~ № 6. - С. 45-65.

11. *Doroshenko A.* Programming Abstracts for Synchronization and Communication in Parallel Programs / Parallel Computing Technologies, Proc. Third Int. Conf. PaCT'95 // Lect. Notes Comput. Sci. - 1995 - Vol. 964. - P.157-162.

12. *Doroshenko A.* Modeling synchronization and communication abstractions for dynamical parallelization / High-Performance Computing and Networking, Vienna, Austria, Apr.1997, Proc.Int. Conf. // Lect. Notes in Computer Sci. - 1997 - Vol. 1225. - P. 752-761.

13. *Doroshenko A.* Coordination Programming Abstractions for Efficient Parallel Programs // Праці І

міжнародн. науково-практ. конф. з програмування УкрПРОГ98. - К.: Ін-т кібернетики ім. В.М.Глушкова, 1998. - С 235-242.

14. *Doroshenko A.* Coordination Facilities to Enhance Concurrency of Race Free Parallel Algorithms / High-Performance Computing and Networking, Amsterdam, Netherland, Apr.1998, Proc.Int. Conf. // Lect. Notes in Computer Sci. - 1998 - Vol. 1401. - P. 854-856.

15. *Hillis W.D., Steele G.L.* Data parallel algorithms // Commun. ACM. - 1986.- Vol. 29. - N 12.

16. *Уилкинсон Дж., Райниш К.* Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. - М.: Машиностроение, 1976. - 389 с.

17. *Allan V.H., Jones R.B., Lee R.M., Allan S.J.* Software Pipelining, ACM Computing Surveys, 27. - 1995. - No. 3 - P. 367-432.

18. *Lea D.* Concurrent Programming in Java. - Reading: Addison Wesley, 1997. - 339 p.

A.E. Doroshenko, I.S. Kononenko, A.Y. Korotun

COORDINATION PROGRAMMING ABSTRACTIONS FOR EFFICIENT DATA PARALLEL COMPUTATION

The most popular data parallel style of programming is mainly characterized by simple control organization and barrier-like synchronization facilities used. However even in fairly regular cases of application programs these facilities can not expose sufficient performance. In major part this is because of restrictive nature of barriesr-like facilities used for synchronization of parallel computation. In this paper, a class of distributed/shared memory parallel programs with static, race free structure of accesses to shared memory is considered and formal regular expressions, called forcing expressions (FE), are defined as synchronization facilities for these programs. It is shown that alongside with more concurrency these facilities can facilitate formal development and design of efficient parallel programs.