МЕТОДИ Й АЛГОРИТМИ СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

П. Кравець
Національний університет "Львівська політехніка"

УДК 004.272.2

СТОХАСТИЧНЕ РОЗПАРАЛЕЛЮВАННЯ ПРОГРАМ

Запропоновано новий принцип стохастичного розпаралелювання програм, який базується на концепціях валентності даних та валідності команд.

It is offered a new principle of a stochastic parallelizing of the programs which is based on concepts of data valency and commands validation.

Вступ

Сучасна тенденція ускладнення організації та зростання розміру програмних засобів ставить високі вимоги до продуктивності обчислювальних систем. Підвищення продуктивності може бути досягнуто за рахунок покращання технічних характеристик мікроелектронних компонентів, вдосконалення архітектури мікропроцесорів, використанням мультипроцесорних та мультимашинних мережних систем [1–6]. Ефективне використання таких систем вимагає нового способу планування обчислювальних процесів у напрямку їх розпаралелювання. Особливо важливою є адаптація існуючого програмного забезпечення для виконання на мультипроцесорних системах. Для цього є необхідним розроблення засобів розпаралелювання програм з мінімальними втручанням у цей процес з боку програмістів. У зв'язку з цим є актуальним розроблення концепції автоматичного розпаралелювання програм [7–11].

Аналіз методів розпаралелювання програм свідчить, що більшість з них розв'язують задачу розпаралелювання у детермінованому формулюванні. Однак траєкторії виконання команд розгалуженої програми залежать від наборів вхідних даних і можуть

розглядатися як реалізації випадкового процесу [12]. Тому актуальними є побудова теоретичних засад та методики стохастичного розпаралелювання програм.

У цій статті закладені нові принципи стохастичного розпаралелювання програм, які базуються на поняттях валентності даних та валідності команд. Практична реалізація цих принципів забезпечить ефективне проектування програмного забезпечення для мультипроцесорних обчислювальних систем.

Формулювання задачі

Виконання програми на однопроцесорній обчислювальній системі базується на послідовному виборі команд з оперативної пам'яті. Послідовний вибір є можливим тому, що команди програми є впорядкованими за значеннями своїх адрес. Послідовність вибору команд визначається алгоритмом роботи програми.

Для мультипроцесорної системи програма повинна складатися з окремих підпроцесів, які виконуються паралельно та локально взаємодіють між собою для обміну поточними значеннями даних. Виділення окремих підпроцесів з єдиного обчислювального процесу є досить складною задачею, яка може мати декілька розв'язків. Максимальний рівень розпаралелювання програми визначається кількістю процесорних елементів, а ефективність – способом планування обчислювальних робіт мультипроцесорної системи. Ефективним будемо вважати таке планування робіт в обчислювальній системі, яке при заданих обмеженнях на рівень розпаралелювання дозволяє виконати програму за мінімально можливий час. Очевидно, що ефективність розпаралелювання багато у чому визначається конкретним алгоритмом роботи програми.

Послідовність виконання команд програми часто визначається значеннями вхідних даних, тому в загальному можна вважати, що розгортання розгалужених обчислень є випадковим процесом. Одночасність виконання та локальна взаємодія мультипроцесорних програм дозволяє як їх модель використати сукупність керованих випадкових процесів з локальною взаємодією. Керування такими випадковими процесами полягатиме у поточному перерозподілі команд між процесорними елементами для мінімізації часу виконання програми. Співвідношення значень вхідних даних є основним джерелом невизначеності для ефективного розпаралелювання програм. Для керування випадковими процесами в умовах невизначеності використовуються адаптивні методи [13–17].

Для побудови методів розпаралелювання програм приймемо парадигму пріоритету даних над кодом, характерну для сучасних технологій програмування. На відміну від програми з послідовною вибіркою команд будемо розглядати мультипроцесорну програму як сукупність підмножин випадково вибраних команд. Кожна підмножина команд є поточною програмою одного із процесорних елементів. Для можливості правильного послідовного виконання таких програм необхідно, щоб кожна команда програми містила інформацію про поточну готовність її операндів. Якщо операнди пройшли усі потрібні попередні перетворення іншими командами, то дана команда виконується, інакше – ні. У такій програмі можна довільно переставляти команди, випадково передавати їх від одного процесорного елемента до іншого. Фізична послідовність розміщення команд тепер не матиме важливого значення для забезпечення правильності обчислень.

Валентність та валідність даних

Для визначення відповідності значень операндів конкретній команді введемо поняття валентності даних. Нехай програма оперує значеннями змінних $x=(x_j \mid j = 1, J)$. Кожному x_j має відповідати цілочислова змінна v_j, яку назвемо валентністю, тобто здатністю даних утворювати валідну команду. Команду назвемо валідною, якщо валентність її операндів дорівнює валентностям відповідних їм даних.

Припустимо, що дані та значення їх валентностей розміщені у глобальній області пам'яті, доступній для запису та читання усім командам одного процесу. На момент старту програми валентність усіх змінних є однаковою, наприклад, дорівнює нулю. При виконанні команди валентність її операндів змінюється. Спосіб зміни валентностей даних *k*-ї команди визначається функцією

$$v^k_{n+1} = f(v^k_n) \ , n = 0, 1, 2, \ldots. \qquad (1)$$

Команди мультипроцесорної програми складаються з ідентифікатора процесу *IdProcess*, коду операції *OpCode*, операндів $op = (op(j) \mid j = \overline{1,l};\ \ l \geq 1\ )$ та базових валентностей операндів $v^k = (v^k(j) \mid j = \overline{1,l})$. Для команд умовного, безумовного переходів, циклів додатково може вказуватися ознака зміщення *Shift*. Базові значення валентностей операндів визначаються в процесі компіляції програми за допомогою функції (1).

Можливість поточного виконання команди програми визначається функцією валідності

$$\varphi(v^k, \tilde{v}^k) \in \{\text{true}, \text{false}\},$$

де $\tilde{v}^k$ – поточне значення валентностей операндів команди.

Функція валідності команди перевіряє збіг поточних значень валентностей операндів $\tilde{v}^k$ *k*-ї команди із базовими значеннями v^k, заданими в оперативній пам'яті:

$$\varphi(v^k, \tilde{v}^k) = \bigwedge_{j=1}^{l} \left(v^k(j) = \tilde{v}^k(j) \right).$$

Якщо функція валідності набуває значення логічної істини, то команда виконується, інакше її виконання затримується.

Елементи розпаралелювання. У межах одного обчислювального процесу основними елементами розпаралелювання є: асоціативні блоки команд, вітки умовних операторів, оператори тіла циклу, підпрограми.

Розпаралелювання лінійного обчислювального процесу

Лінійний процес є найпростішим для мультипроцесорної компіляції та розпаралелювання. Зміна валентностей операндів *k*-ї команди може бути виконана за допомогою функції інкременту:

$$v_{n+1}^{k}(j) = v_{n}^{k}(j) + 1. \qquad (2)$$

Паралельне виконання лінійного процесу можливе за рахунок асоціативних блоків команд. Два блоки є асоціативними, якщо їх перестановка не порушує правильності виконання програми. Структура фрагмента лінійної програми з асоціативними блоками є такою: початковий блок; множина асоціативних блоків; кінцевий блок. У початковому блоці, як правило, вводяться або ініціалізуються дані та проводяться підготовчі обчислення, які не піддаються розпаралелюванню. У кінцевому блоці здійснюються заключні обчислення та виведення результатів роботи програми.

Для можливості правильного виконання асоціативних блоків команд приймемо такі правила:

1) асоціативний блок розмічається валентностями двох бінарних ознак – входу у блок та виходу з блоку;
2) для початкового блока, кінцевого блока та всередині асоціативних блоків зміна валентностей даних визначається функцією (2);
3) після виконання останньої команди початкового блока відбувається запам'ятовування даних та їх валентностей в додатковому сегменті оперативної пам'яті; пошук значень операндів валідних команд здійснюється в основному та додаткових сегментах даних;
4) валентність одноіменних змінних приймається однаковою на входах усіх асоціативних блоків, що дорівнює валентностям цих змінних на виході з початкового блоку;
5) валентності одноіменних змінних всередині асоціативних блоків команд відрізняються між блоками на фіксоване порогове значення;
6) вихідні валентності усіх асоціативних блоків мають дорівнювати їх вхідним валентностям (крім валентностей ознак початку та кінця асоціативного блока);
7) вхідна валентність одноіменних змінних кінцевого блока має дорівнювати вихідній валентності асоціативних блоків; можливість виконання кінцевого блоку визначається кон'юнкцією вихідних ознак асоціативних блоків.

Наведені правила забезпечують можливість виконання асоціативних блоків команд у довільному порядку.

Розпаралелювання розгалуженого обчислювального процесу

Розгалуження обчислювального процесу виконується за допомогою операторів умовних переходів. Розпаралелювання розгалуженого процесу має ряд особливостей. Залежно від умови той або інший альтернативний блок команд програми не буде виконаний (усі команди блока будуть невалідними) протягом деякого або усього часу виконання програми. Передбачити це на етапі компіляції програми неможливо, оскільки введення та опрацювання даних здійснюється пізніше, на етапі виконання програми.

Команда переходу може передати керування командам, розміщеним нижче або вище по тексту програми. Напрямок переходу задається знаком зміщення валентностей: додатне значення визначає перехід донизу, а від'ємне -- догори по програмі.

Структура фрагмента розгалуженої програми є такою: початковий блок; команда умовного переходу; множина альтернативних блоків; кінцевий блок.

Розпаралелювання розгалуженого процесу може бути виконано за двома схемами:

1) звичайне, без випереджувального виконання альтернативних блоків команд;

2) з випереджувальним виконанням усіх альтернативних блоків команд.

Розмітка розгалуженої програми без випереджувального виконання альтернативних блоків команд з переходом донизу по тексту програми здійснюється так:

1) вхідні валентності команди умовного переходу дорівнюють вихідним валентностям початкового блока;
2) вихідні валентності команди умовного переходу визначаються фіксованим зміщенням для кожного альтернативного переходу;
3) вхідні валентності альтернативних блоків команд розрізняються на фіксоване порогове значення і дорівнюють валентностям відповідних виходів команди умовного переходу;
4) всередині альтернативних блоків зміна валентностей керується функцією (2);
5) вихідні валентності альтернативних блоків команд є однаковими і дорівнюють вхідним валентностям відповідних змінних кінцевого блока.

Якщо у черзі команд зустрічається валідна команда переходу, то у зв'язку із випадковим розміщенням команд в дійсності перехід не відбувається, а тільки змінюються значення валентностей даних, які зустрічаються у блоці, на який заплановано перехід. Це робить валідними команди цього блока, і вони можуть бути виконані у порядку, що регулюється зміною валентностей їх операндів.

Розпаралелювання циклічного обчислювального процесу

Переміщення догори по програмі є характерним для циклічних процесів. Оскільки при виконанні циклу валентність даних змінюється, то для забезпечення повторного виконання команд тіла циклу необхідно відновити значення валентностей операндів. Для цього при вході у цикл відбувається запам'ятовування значень та валентностей тих даних, які є операндами команд тіла циклу. Якщо у потоці команд зустрічається валідний перехід догори по програмі, що відповідає від'ємній ознаці зміщення, то запам'ятовані дані та їх валентності відновлюються і стають такими, як і при вході у цикл. При достроковому виході з циклу вниз по тексту програми валентності змінних циклу дорівнюють вихідним валентностям циклу плюс необхідне зміщення.

Структура програми з циклом з післяумовою є такою: початковий блок; тіло циклу; команда умовного переходу вгору програми; кінцевий блок. Правила зміни валентностей операндів команд циклу з післяумовою є такими:

1) після завершення виконання команд початкового блока, перед входом у цикл здійснюється запам'ятовування значень даних та їх валентностей у додатковому сегменті оперативної пам'яті; пошук значень змінних із заданими валентностями здійснюється в основному та додатковому сегментах даних;
2) вхідні валентності даних тіла циклу дорівнюють вихідним валентностям відповідних даних початкового блока;
3) зміна валентностей даних всередині тіла циклу здійснюється функцією (2);
4) команда умовного переходу на початок тіла циклу призводить до встановлення валентностей даних у значення, які вони мали перед початком виконання тіла

циклу; при виході із циклу значення валентностей даних мають дорівнювати вхідним валентностям кінцевого блока.

5) значення вхідних валентностей кінцевого блоку дорівнюють максимальним значенням валентностей, набутих у попередньому фрагменті програми.

При компіляції циклів, зокрема з фіксованою кількістю повторень, доцільно здійснювати реплікацію команд тіла циклу. Маркування валентностей операндів таких команд здійснюється аналогічно до лінійного процесу з асоціативними блоками. Хоча реплікація команд призведе до зростання розміру коду, але дозволить зменшити час виконання мультипроцесорної програми.

Розпаралелювання підпрограм

Команди підпрограми розміщуються у загальному потоці команд і можуть виконуватися паралельно з іншими валідними командами програми. Правильна послідовність виконання команд підпрограми забезпечується подібно до виконання циклів: зміщенням валентностей даних або використанням їх копій у додатковій області пам'яті.

Локальні дані підпрограми та їх валентності розмішуються в окремій області пам'яті. Це забезпечує однозначність їх інтерпретації та не впливає на глобальне керування валентностями даних.

Глобальні дані або ті, котрі передаються через вказівники, повинні мати глобально визначену послідовність зміни валентностей.

Розмітка базових валентностей операндів команд, розміщених за командою звертання до підпрограми, здійснюється із зміщеннями, значення яких визначаються після розмітки команд підпрограми.

Інший спосіб полягає у здійсненні локальної копії глобальних даних та їх валентностей. Такі дані будуть зберігатися разом із локальними даними підпрограми. Для даних з локальної пам'яті встановлюються свої початкові значення валентностей. Ці значення модифікуються у порядку роботи алгоритму підпрограми, що забезпечує відмінність команд програми та підпрограми.

Після завершення роботи підпрограми змінюються значення відповідних глобальних змінних та їх валентностей.

Приклад розпаралелювання програми

Виконаємо розпаралелюванням програми обчислення виразу $y = ax^2 + bx + c$ для цілочислових значень змінних a, b, c, x.

Будемо вважати, що початкова ініціалізація змінних виконана у сегменті даних. Змінні a, b, c, x займають у сегменті даних по одному байту, а змінна y – два байти. Відповідна програма обчислення виразу на мові *Assembler*, призначена для виконання мікропроцесором *Intel 80x86*, наведена у табл. 1.

У наведеному прикладі кількості машинних тактів виконання команд є умовними величинами, значення яких залежать від архітектури конкретного мікропроцесора.

Для зручності опису програми у ній виділені окремі блоки. Як видно із організації програми, блоки 1–3 є асоціативними і можуть виконуватися паралельно.

Таблиця 1

Початкова розмітка команд програми

№ команди	Валентність операндів до виконання команди	Команди програми	Кількість машинних тактів	Валентність операндів після виконання команди
1	v_{bx} 0	xor bx,bx	3	v_{bx} 1
		Блок 1		
2	v_{entry1} 0; v_{ax} 0 ; v_{al} 0; v_{ax} 0 v_{bx} 1; v_c 0	mov al,c	14	v_{entry1} 1; v_{ax} 1; v_{al} 1
3	v_{al} 1	cbw	2	v_{al} 2; v_{ax} 2
4	v_{bx} 1 ; v_{ax} 2 v_{bx} 1	add bx,ax	3	v_{ax} 0; v_{al} 0; v_{bx} 1; v_c 0; v_{exit1} 1
		Блок 2		
5	v_{entry2} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_b 0 ; v_x 0	mov al,b	14	v_{entry2} 1; v_{ax} 3; v_{al} 3; v_b 1
6	v_{al} 3; v_x 0	imul x	110	v_{ax} 4; v_{al} 4; v_x 1
7	v_{bx} 1; v_{ax} 4	add bx,ax	3	v_{ax} 0; v_{al} 0; v_{bx} 1; v_b 0; v_x 0; v_{exit2} 1
		Блок 3		
8	v_{entry3} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_a 0; v_x 0	mov al,x	14	v_{ax} 5; v_{al} 5; v_x 2; v_{entry3} 1
9	v_{al} 5; v_x 2	imul x	110	v_{ax} 6; v_{al} 6; v_x 3
10	v_{al} 6; v_a 0	imul a	110	v_{ax} 7; v_{al} 7; v_a 1
11	v_{bx} 1; v_{ax} 7	add bx,ax	3	v_{ax} 0; v_{al} 0; v_{bx} 1; v_a 0; v_x 0; v_{exit3} 1
12	v_{ax} 0; v_{al} 0; v_{bx} 1; v_y 0 ; v_a 0; v_b 0; v_y 0 v_c 0; v_y 0; v_{exit1} v_{exit2} v_{exit3} 1	mov y,bx	18	v_y 1; v_{bx} 2

Тепер довільно переставимо команди програми та визначимо послідовність їх виконання згідно із валентностями операндів.

Виконання програми зі стохастичною структурою на однопроцесорній системі. Нехай команди базової програми розміщені так, як показано у табл. 2.

Виконання програми розпочинається з нульових значень валентностей даних. Для цього відбувається перегляд черги команд і виконується перша валідна команда. Валідність команди визначається відповідністю поточних значень валентностей даних валентностям операндів та правилами 1–7, визначеними у розділі "Розпаралелювання лінійного обчислювального процесу". Валентність даних наслідується в порядку виконання команд. Значення валентностей даних, які є операндами команд, змінюється згідно з табл. 1.

Порядок виконання команд програми задається номерами, розміщеними у четвертій колонці табл. 2. Для даної випадкової послідовності команд спочатку буде виконаний блок 3, потім – блок 2 і на закінчення – блок 1. Команди програми будуть виконані на однопроцесорній обчислювальній системі за 404 машинних тактів.

Таблиця 2

Виконання програми на однопроцесорній системі

№	Валентність операндів	Команди програми	Порядок виконання команд	Зміна валентностей даних													
				entry1	entry2	entry3	exit1	exit2	exit3	y	a	b	c	x	ax	al	bx
				0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
4	v_{bx} 1; v_{ax} 2	add bx,ax	11	1	1	1	1	1	1	0	0	0	0	0	0	0	1
8	v_{entry3} 0; v_{ax} 0; v_{al} 0, v_{bx} 1; v_a 0; v_x 0	mov al,x	2	0	0	1	0	0	0	0	0	0	0	2	5	5	1
10	v_{al} 6; v_a 0	imul a	4	0	0	1	0	0	0	0	1	0	0	2	7	7	1
2	v_{entry1} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_c 0	mov al,c	9	1	1	1	0	1	1	0	0	1	1	0	1	1	1
9	v_{al} 5; v_x 2	imul x	3	0	0	1	0	0	0	0	0	0	0	3	6	6	1
11	v_{bx} 1; v_{ax} 7	add bx,ax	5	0	0	1	0	0	1	0	0	0	0	0	0	0	1
7	v_{bx} 1; v_{ax} 4	add bx,ax	8	0	1	1	0	1	1	0	0	0	0	0	0	0	1
5	v_{entry2} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_b 0; v_x 0	mov al,b	6	0	1	1	0	0	1	0	0	1	0	0	3	3	1
1	v_{bx} 0	xor bx,bx	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12	v_{ax} 0; v_{al} 0; v_{bx} 1; v_y 0; v_a 0; v_b 0; v_c 0; v_x 0; v_{exit1} v_{exit2} v_{exit3} 1	mov y,bx	12	1	1	1	1	1	1	1	0	0	0	0	0	0	2
6	v_{al} 3; v_x 0	imul x	7	0	1	1	0	0	1	0	0	1	0	0	4	4	1
3	v_{al} 1	cbw	10	1	1	1	0	1	1	0	0	1	1	0	2	2	1

Випадкове розміщення команд програми, призначеної для виконання на однопроцесорній системі, має ілюстративний характер і демонструє працездатність механізму валентності даних. Практичний інтерес має виконання такої програми на мультипроцесорній системі. При паралельному виконанні команд можна очікувати зменшення загального часу роботи програми.

Виконання програми зі стохастичною структурою на двопроцесорній системі. Випадкова структура програми дозволяє виконати її довільний перерозподіл між процесорними елементами. Наприклад, нехай перша половина команд програми, наведених у табл. 2, виконується першим, а друга половина – другим мікропроцесором. Послідовність виконання команд та зміна валентностей даних при виконанні програми двопроцесорною системою наведені у табл. 3.

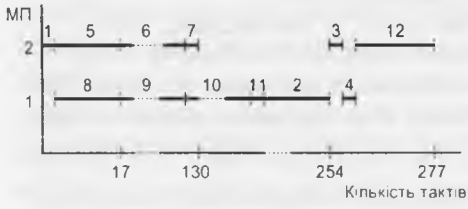
Перед початком роботи у локальну пам'ять мікропроцесорів завантажуються значення даних та їх валентностей. Зміна валентностей даних відбувається у локальній пам'яті мікропроцесора. Якщо процесор не знаходить валідної команди і черга команд не виконана, то відбувається міжпроцесорний обмін векторами даних та їх валентностей, який керується процесором-арбітром.

Таблиця 3

Виконання програми на двопроцесорній системі

№	Валентність операндів	Команди програми	Порядок виконання команд	Зміна валентностей даних													
				entry1	entry2	entry3	exit1	exit2	exit3	y	a	b	c	x	ax	al	bx
Мікропроцесор 1																	
			*	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
4	v_{bx} 1; v_{ax} 2	add bx,ax	6*	1	0	1	1	0	1	0	0	0	0	0	0	0	1
8	v_{entry3} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_a 0; v_x 0	mov al,x	1	0	0	1	0	0	0	0	0	0	0	2	5	5	1
10	v_{al} 6; v_a 0	imul a	3	0	0	1	0	0	0	0	1	0	0	3	7	7	1
2	v_{entry1} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_c 0	mov al,c	5**	1	0	1	0	0	1	0	0	0	1	0	1	1	1
9	v_{al} 5; v_x 2	imul x	2	0	0	1	0	0	0	0	0	0	0	3	6	6	1
11	v_{bx} 1; v_{ax} 7	add bx,ax	4	0	0	1	0	0	1	0	0	0	0	0	0	0	1
Мікропроцесор 2																	
				0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
7	v_{bx} 1; v_{ax} 4	add bx,ax	4*	0	1	0	0	1	0	0	0	0	0	0	0	0	1
5	v_{entry2} 0; v_{ax} 0; v_{al} 0; v_{bx} 1; v_b 0; v_x 0	mov al,b	2	0	1	0	0	0	0	0	0	1	0	0	3	3	1
1	v_{bx} 0	xor bx,bx	1*	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12	v_{ax} 0; v_{al} 0; v_{bx} 1; v_y 0; v_a 0; v_b 0; v_c 0; v_x 0; v_{exit1} v_{exit2} v_{exit3} 1	mov y,bx	6	0	1	0	0	1	0	1	0	0	0	0	2	2	2
6	v_{al} 3; v_x 0	imul x	3	0	1	0	0	0	0	0	0	1	0	1	4	4	1
3	v_{al} 1	cbw	5**	0	1	0	0	1	0	0	0	0	0	0	2	2	1

У табл. 3 символом '*' позначено крок, після якого відбувається міжпроцесорний обмін значеннями даних та їх валентностей. Після обміну кожен мікропроцесор формує матрицю даних та їх валентностей, за якою перевіряється валідність команд.



Діаграма паралельного виконання програми на двопроцесорній системі

Послідовність виконання програми зі стохастичним розміщенням команд на двопроцесорній обчислювальній системі зображена на рисунку.

Для заданого розподілу команд між двома процесорами обчислення будуть виконані за 277 машинних тактів.

Порівняно з результатами виконання програми на однопроцесорній системі зменшення кількості машинних тактів відбулося завдяки паралельному виконанню блоків програми двома процесорами.

Ефективність розпаралелювання може бути покращена (до певної межі) введенням нових процесорних елементів та пошуком іншого розподілу команд між ними. Визначення оптимального розподілу, який забезпечує мінімізацію часу виконання усієї програми, можна виконати за допомогою стохастичних методів. Методи стохастичного розпаралелювання базуються на основі випадкового пошуку. У класі методів випадкового пошуку найбільшу ефективність мають селективні та адаптивні методи [13–17].

Висновки

Поняття валентності даних забезпечує організацію програми із стохастичним розміщенням команд. Властивість валентності даних задавати порядок обчислення стохастичної програми робить можливим її мультипроцесорне виконання з довільним початковим розподілом команд. Блоки програми, які допускають паралельне виконання, визначають пошуковими методами шляхом цілеспрямованого перерозподілу черг команд між процесорними елементами. Глобальним критерієм випадкового пошуку є мінімізація часу виконання програми при обмеженнях на кількість процесорних елементів та топологію обчислювальної системи. Недоліком стохастичного розпаралелювання порівняно з детермінованими методами є велика кількість ітерацій, необхідних для збіжності до оптимального розв'язку. Цей недолік долається застосуванням сучасних засобів обчислювальної техніки.

1. Хокни Р., Джасхоуп К. Параллельные ЭВМ. – М.: Радио и связь, 1986.
2. Смирнов А. Д. Архитектура вычислительных систем. – М. Наука, 1990.
3. Кун С. Матричные процессоры на СБИС: – М.: Мир, 1991.
4. Вальковский В.А., Вирбицкайте И.Б. Потоковые вычислительные системы // Системная информатика. Вып. 2. – Новосибирск: ВО "Наука", 1993.
5. Уоссерман Ф. Нейрокомпьютерная техника. – М.: Мир, 1992.
6. Головкин Б.А. Вычислительные системы с большим числом процессоров.– М.: Радио и связь, 1995.
7. Грицык В.В. Распараллеливание алгоритмов обработки информации в системах реального времени. – К.: Наук. думка, 1981.
8. Трахтенгерц Э. А. Введение в теорию анализа и распараллеливания программ ЭВМ в процессе трансляции. – М.: Наука, 1981.
9. Воеводин В.В. Математические модели и методы в параллельных процессах. – М.: Наука,1986.
10. Программирование на параллельных вычислительных системах: Пер. с англ. / Р Бэбб, Дж. Мак-Гроу, Т. Акселрод и др.: Под ред. Р. Бэбба II. – М.: Мир. 1991.
11. Абрамов С.М., Адамович А.И., Позлевич Р.В. Т-система - среда программирования с поддержкой автоматического динамического распараллеливания программ // Программные системы: Теоретические основы и приложения. Под ред: А.К. Айламазян. – М., Наука, Физматлит, 1999.
12. Основы теории вычислительных систем / Под ред. С.А. Майорова. – М.:, Высш. школа, 1978.
13. Растригин Л.А., Рипа К.К., Тарасенко Г.С. Адаптация случайного поиска. – Рига: Зинатне, 1973.
14. Назин А.В., Позняк А.С. Адаптивный выбор вариантов: Рекуррентные алгоритмы. - М.: Наука, 1986.
15. Кравець П.О. Регуляризований ігровий метод керування випадковими процесами в умовах невизначеності // Комп'ютерна інженерія та інформаційні технології: Вісник НУ "Львівська політехніка". – 2002. – № 468. – С. 101 – 109.
16. Кравець П.О. Селективні методи випадкового пошуку // Матеріали Міжнародної науково-практичної конференції "Динаміка наукових досліджень". – Т. 1 Сучасні комп'ютерні інформаційні технології. – Дніпропетровськ: Наука і освіта, 2002. – С. 18 – 21.

17. Кравець П.О. Оптимізація випадкового пошуку генетичним методом з розпаралелюванням // Інформаційні системи та мережі: Вісник НУ "Львівська політехніка". – 2002. – № 464. – С. – 158–171.

Р.Базилевич, І. Подольський
Національний університет "Львівська політехніка"

УДК 621.382

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОПТИМІЗАЦІЇ ДЛЯ ЗАДАЧ ДЕКОМПОЗИЦІЇ

Для оптимізаційних задач розбиття запропоновано декілька алгоритмів, що використовують ієрархічну кластеризацію, сформовану методом оптимального згортання схеми. Досліджено ефективність алгоритмів з точки зору якості отриманих результатів та обчислювальних затрат.

Several algorithms for partitioning optimization are suggested. Hierarchical clustering by the Optimal Circuit Reduction method is used as a basic approach. Efficiency and effectiveness of proposed algorithms are investigated.

Вступ

Значна частина прикладних задач математичного моделювання мають високу та надвисоку розмірність – тисячі та мільйони невідомих. Ефективним підходом для їх якісного розв'язування є декомпозиція – розбиття схеми на підмножину окремих підсхем з подальшою їх апроксимацією макромоделями та розв'язанням основної задачі з використанням утворених макромоделей як основних елементів. Проте з математичної точки зору ця допоміжна задача також є трудомісткою, має експоненційну обчислювальну складність, що у випадку великої розмірності робить проблематичним отримання її оптимального розв'язку. Як конструктивні, так і ітераційні алгоритми декомпозиції зазвичай дають розв'язок, що відповідає деякому локальному екстремуму, часто суттєво віддаленому від оптимуму. В зв'язку з цим методологія розробки алгоритмів декомпозиції потребує подальшого вдосконалення. На відміну від традиційних запропоновані алгоритми використовують як базові елементи кластери довільної розмірності, що дає можливість покращити якість розв'язання з поміркованими обчислювальними затратами. Для формування кластерів використовується метод оптимального згортання