DOI 10.31891/2307-5732-2021-295-2-89-96
УДК 004.75

П.Г. РЕГІДА, І.А. КОМІСАРОВ
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

ДОСЛІДЖЕННЯ СПОСОБУ ПЛАНУВАННЯ ОБЧИСЛЕНЬ НА ОСНОВІ АЛГОРИТМУ БУЛЬБАШКОВОГО РОЗПОДІЛУ В РІЗНИХ ТОПОЛОГІЯХ

У цій роботі досліджено алгоритм бульбашкового планування та розподілення для різних типів топологій: решітки, гіперкубу, топології де Бруйна, розширеної топології де Бруйна на основі надлишкового коду. Проаналізовано статичні алгоритми планування, результати сформовані у вигляді порівняльної таблиці за критеріями складності, необхідності знаходження критичного шляху, наявності таблиці маршрутизації та ефективності. Дослідження способу планування обчислень проведено на основі задачі знаходження коренів систем лінійних та нелінійних рівнянь за допомогою методів Крамера та Ньютона. Для даної задачі синтезовано відповідні графи ярусно-паралельної форми. Проведено експериментальні дослідження занурення отриманих графів в синтезовані топології за допомогою способу бульбашкового розподілу, представлено результати планування для вказаних топологій.

Ключові слова: планувальник BSA, система рівнянь, топологія де Бруйна.

P.H. REHIDA, I.A. KOMISAROV
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

COMPARISON OF PLANNING RESULTS USING BUBBLE SCHEDULING AND ALLOCATION (BSA) ALGORITHM FOR DIFFERENT TOPOLOGIES

In this article, the bubble scheduling and allocation algorithm is considered for different types of topologies: grid, hypercube, de Bruijn topology, extended de Bruijn topology based on ternary code. Static planning algorithms are analyzed; the results are presented in the form of a comparative table on the criteria of complexity, the need to find a critical path, the presence of a table of routing and efficiency. The study of the method of planning calculations is carried out based on the problem of finding the roots of systems of linear and nonlinear equations using Cramer's and Newton's methods. The corresponding graphs of tier-parallel form are synthesized for these methods.

The principles of synthesis for 4 types of topologies are shown. The synthesis of the grid, hypercube, and de Bruijn graph is considered in the classical form. The synthesis of the extended de Bruijn topology is a synthesis of de Bruijn topology [1, 2] using a ternary code. That is, with the same number of processors, the number of connections increases.

Experimental studies of the scheduling of the obtained graphs in the synthesized topologies using the method of bubble scheduling and allocation are conducted; the results of scheduling are presented for these topologies.

The best results were shown by extended de Bruijn topology based on ternary code due to the increased degree of units, which is especially noticeable for Newton's method where there are much more data transfers than in Cramer's method. The topology of a hypercube and de Bruijn topology demonstrated just about same results but hypercube topology did a little better. In addition to this, having a smaller diameter and cost, the hypercube is the most optimal topology and still used today. However, when constructing fail-safe topological organizations, it is better to use topologies based on ternary code, such as the topology based on the extended de Bruijn graph.

Keywords: bubble scheduling, systems of equations, de Bruijn topology.

Постановка задачі

Проблема мінімізації часу виконання задач в комп'ютерних системах (КС) є завжди актуальною. Для вирішення цієї проблеми створено багато різних КС для виконання певних класів задач. В роботі розглянуті КС, що складаються з автономних вузлів з локальною пам'яттю, які з'єднані між собою каналами зв'язку для передачі даних. Ці зв'язки формують певну топологію комп'ютерної системи. В даній статті розглянуто наступні топології: решітка, гіперкуб, топологію де Бруйна та розширена топологія де Бруйна на основі надлишкового кодування [1, 2]. Особливістю таких КС є гарна масштабованість, що дозволяє додавати вузли без необхідності додаткової синхронізації завдяки локальній пам'яті кожного вузла [3]. Проте, при зростанні кількості вузлів та, відповідно, кількості підзадач виникає проблема планування, а саме розподілення графу задач між вузлами КС. Саме ця проблема розглядається в даній роботі.

Аналіз досліджень та публікацій

Планування задач є важливою проблемою, так як неефективне планування задач може знизити обчислювальний потенціал КС, та, можливо, взагалі змарнувати переваги розпаралелювання. Тому для кожної задачі варто підбирати відповідні алгоритми планування та топологічні організації, що дозволить підвищити ефективніть КС.

Розглянемо основні статичні алгоритми планування, які відносяться до наступних класів [4]:

BNP (Bounded Number of Processors) – клас алгоритмів для систем з визначеною сталою кількістю процесорів та повною зв'язністю. В таких системах алгоритму не потрібно контролювати маршрутизацію (пересилки) між процесорами, так як вони всі зв'язані між собою. Відомі алгоритми цього класу: ETF (Earliest Time First), MCP (Modified Critical Path), ISH (Insertion Scheduling Heuristic) тощо.

UNC (Unbounded Number of Clusters) – клас алгоритмів для кластерних систем з довільною кількістю кластерів, але з повною зв'язністю між ними. Відомі алгоритми цього класу: EZ (Edge zeroing), LC (Linear Clustering), DSC (Dominant Sequence Clustering), DCP (Dynamic Critical Path), MD (Mobility Directed).

APN (Arbitrary Processor Network) – клас алгоритмів для комп'ютерних систем з довільною кількістю процесорів та довільною кількістю зв'язків між ними. Цей клас охоплює найбільшу кількість

комп'ютерних систем, тому алгоритми цього класу вважаються універсальними для різних топології. Відомі алгоритми цього класу: BU (Bottom-Up), MH (Mapping Heuristic), DLS (Dynamic Level Scheduling), BSA (Bubble Scheduling and Allocation).

Алгоритми планування класу APN

Детальніше розглянемо алгоритми класу APN:

MH (Mapping Heuristic). Цей алгоритм спочатку обчислює пріоритет кожної вершини графу та створює список вершин відсортованих в порядку спадання. Потім для першої задачі в списку шукає найкращий можливий час початку виконання, зважаючи на залежність від батьківських задач. Ці залежності зберігаються в таблиці маршрутизації. Після розподілення, вибрана вершина позначається як виконана та видаляється зі списку.

DLS (Dynamic Level Scheduling) схожий на MH, також базується на таблиці маршрутизації, але для обробки пересилок потребує додаткового методу маршрутизації. Результати, як правило, збігаються з MH.

BU (Bottom-Up). Цей алгоритм спочатку завантажує всі задачі критичного шляху (найдовший ланцюжок обчислень) на один процесор. Потім для кожної задачі розраховується довжина шляху від останньої задачі до цільової (в тактах). Ця довжина називається b-рівень. Розраховується максимальна кількість ребер від початкової задачі до цільової (p-рівень). У порядку зменшення значення p-рівня всі задачі з однаковим p-рівнем рівномірно завантажують на процесори за зменшенням b-рівня.

BSA (Bubble Scheduling and Allocation) був запропонований Yu-Kwong Kwok та Ishfaq Ahmad в 1995 році [5]. Спочатку всі задачі послідовно завантажуються на перший процесор. Потім для кожної задачі розраховується можливий час початку її виконання для інших процесорів на основі завантаженості процесора та часу завершення останньої пересилки даних від всіх батьківських задач. Якщо знаходиться процесор для якого «новий» час буде менше ніж «старий», то задача мігрує на цей процесор. У статті [5] наведені результати планування за допомогою цього алгоритму для різних топологій, а саме кільце на 8 та 16 вузлів та гіперкуб на 8 вузлів, при цьому BSA виявився найбільш ефективним.

Таблиця 1

Порівняння алгоритмів за різними параметрами.

Алгоритм / Характеристика	MH	DLS	BU	BSA
Складність, v – вершини, p – процесори, e – ребра	$O(v(p^3v+e))$	$O(v^3p\ f(p)))$, f(p) – складність обробки пересилок	$O(v^2\log(v))$	$O(p^2ev)$
Необхідність знаходження критичного шляху	+	-	+	-
Таблиця маршрутизації	+	+	-	-
Ефективність (згідно з [5])	2	3	4	1

Як видно з табл. 1 та результатів планування, наведених в статті [5], алгоритм BSA є найбільш привабливим з урахуванням зазначених характеристик, так як BU неефективно планує задачі, для DLS потрібно додатково реалізовувати метод обробки пересилань, а для MH потрібно розраховувати критичний шлях та використовувати таблицю маршрутизації. З наведених причин виберемо та реалізуємо BSA алгоритм для планування графа задач для комп'ютерної системи.

Синтез графів задач для експериментального порівняння планування в топологіях

Для проведення експериментів синтезуємо два графа задач для порівняння часу планування. В якості задачі пропонується використати метод Крамера для знаходження коренів системи лінійних алгебраїчних рівнянь (СЛАР) та ітераційний метод Ньютона для знаходження наближених коренів системи нелінійних рівнянь (СНР).

Метод Крамера рішення СЛАР

В табл. 2 представлено покрокове виконання методу Крамера для рішення системи з 4-х рівнянь [6]. Приймемо, що час виконання операцій додавання та віднімання – 1 такт, множення – 3, ділення – 4.

Таблиця 2

Обчислення методом Крамера системи з 4 рівнянь.

№	Дія	Час виконання
1	Розрахунок міноров для обчислення детермінантів 4 порядку $m11_1$ =a22·a33·a34+a42·a23·a34+a32·a43·a21-a42·a33·a24-a22·a43·a34-a32·a23·a44	41
2	$m12_1$ =a21·a33·a34+a41·a23·a34+a31·a43·a21-a41·a33·a24-a21·a43·a34-a31·a23·a44	41

Продовження табл. 2

№	Дія	Час виконання
3	$ml3_1$=a21·a32·a34+a41·a22·a34+a31·a42·a21-a41·a32·a24-a21·a42·a34-a31·a22·a44	41
4	$ml4_1$ =21·a32·a33+a41·a22·a33+a31·a42·a21-a41·a32·a23-a21·a42·a33-a31·a22·a43	41
5-16	Розрахунок мінорів для інших детермінантів. Мінори $ml1_1, ml2_2, ml3_3, ml4_4$ використовуємо для основного детермінанта, тому додатково не потрібно на них виділяти задачі.	41
17-21	Розрахунок всіх детермінантів $\Delta_i = a1_i \cdot ml1_i + a2_i \cdot ml2_i + a3_i \cdot ml3_i + a4_i \cdot ml4_i$, i=1..5, де Δ_5 – основний детермінант.	15
22-25	Обчислення коренів СЛАР: $x_i = \Delta_i / \Delta$, де i=1..4	4

На основі описаного методу створимо відповідний граф задач (рис. 1). Граф задач представляє собою ациклічний орієнтований граф. Вершини графа – це задачі, вага кожної вершини – це час виконання відповідної задачі, вага ребер – це час на пересилання результатів від однієї задачі до іншої. Приймемо, що час пересилання для всіх задач – 1 такт.

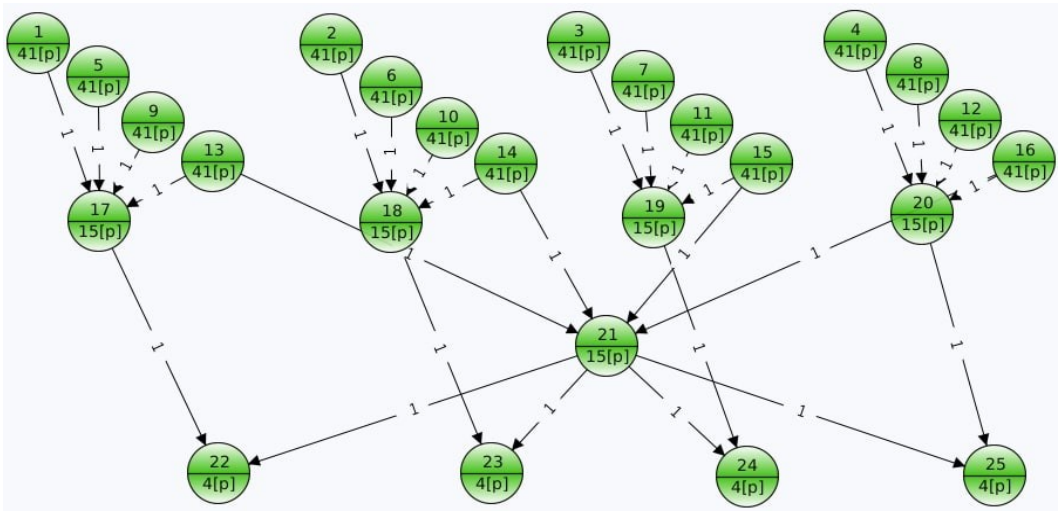

Рис. 1. Ярусно-паралельна форма алгоритму методу Крамера

Як видно з рис. 1 для завантаження даного графу в комп'ютерну систему достатньо 16 вузлів, так як ширина графу дорівнює 16. Тобто максимальна кількість задач, які можуть виконуватись незалежно дорівнює 16. Тому зі збільшенням кількісті вершин зростає значимість вибору найбільш ефективної топології.

Ітераційний метод Ньютона рішення СНР

В табл.3 представлено покрокове виконання ітераційного методу Ньютона для рішення системи з 4-х рівнянь [7]. Приймемо, що час виконання операцій додавання та віднімання – 1 такт, множення – 3, ділення – 4, знаходження похідної – 4. Отримані наближені значення коренів є результатами однієї ітерації методу Ньютона.

Таблиця 3

Обчислення методом Ньютона системи з 4 рівнянь.

№	Дія	Час виконання
1	Розрахунок часткової похідної $\frac{\partial f_1(x)}{\partial x_1}$	4
2-16	Розрахунок часткових похідних аналогічно до п.1	4
17	Розрахунок мінора $m_{11} = \begin{vmatrix} d_{22} & d_{23} & d_{24} \\ d_{32} & d_{33} & d_{34} \\ d_{42} & d_{43} & d_{44} \end{vmatrix} = d_{22}d_{33}d_{44} + d_{23}d_{34}d_{42} + d_{24}d_{32}d_{43} - d_{24}d_{33}d_{42} - d_{34}d_{43}d_{22} - d_{44}d_{23}d_{32}$	41
18-32	Розрахунок мінорів аналогічно п. 17	41

Продовження табл. 3.

№	Дія	Час виконання
33	Розрахунок детермінанта $\Delta F = d_{11}m_{11} - d_{12}m_{12} + d_{13}m_{13} - d_{14}m_{14}$	15
34	Розрахунок елемента оберненої матриці $d_{11}^{-1} = \frac{m_{11}}{\Delta F}$	4
35-49	Розрахунок елементів оберненої матриці аналогічно п. 34	4
50	Розрахунок $x_1^{(1)} = x_1^{(0)} - \left(d_{11}^{-1} \cdot f_1(x^{(0)}) + d_{12}^{-1} \cdot f_2(x^{(0)}) + d_{13}^{-1} \cdot f_3(x^{(0)}) + d_{14}^{-1} \cdot f_4(x^{(0)})\right)$	16
51-53	Розрахунок інших частин $x^{(1)}$ аналогічно п.50	16

На основі описаного алгоритму створимо відповідний граф задач (рис. 2). Як і для методу Крамера, ширина цього графу також дорівнює 16, тому для його завантаження достатньо 16 процесорів.

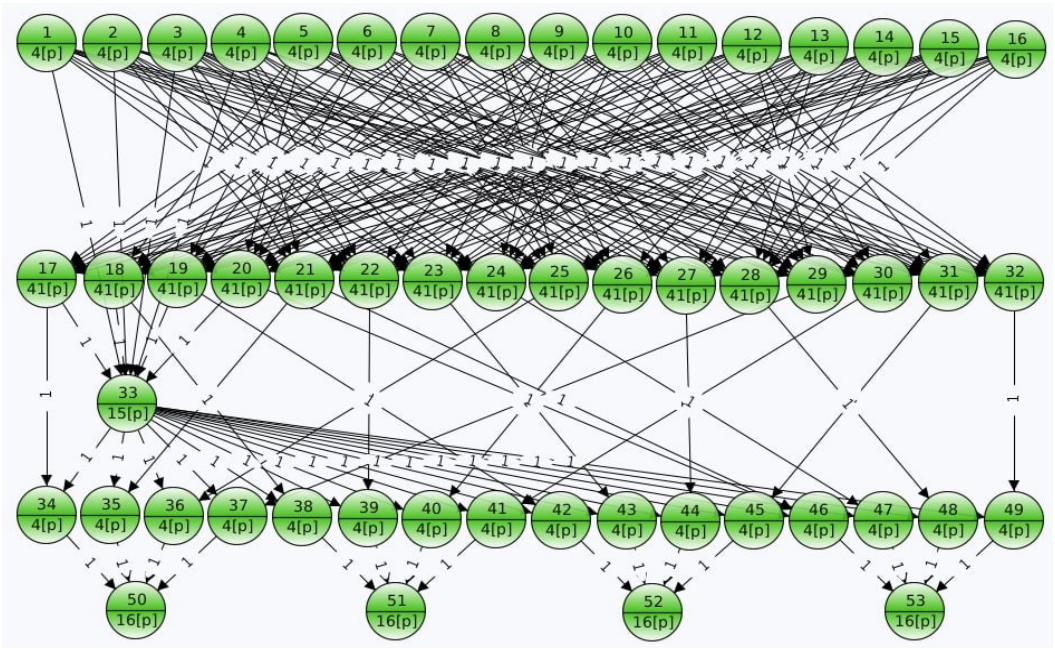

Рис. 2. Ярусно-паралельна форма алгоритму методу Ньютона

Синтезовані графи задач будуть використовуватись в якості вхідних даних при плануванні обчислень в різних топологіях комп'ютерних систем.

Синтез топологічних організацій для оцінки часу обчислень

Решітка

Синтез топології Решітка доволі простий – для кожної вершини сусідами будуть i+1 (окрім вершин, де i+1 mod N = 0), i-1 (окрім вершин, де i mod N = 0), i+N, i-N , де i – номер вершини, N – розрядність сітки (N^2). Дана топологія є простою, її діаметр дорівнює (N-1)*2, що відносно багато, тому частіше використовується модифікована решітка - топологія меш (mesh).

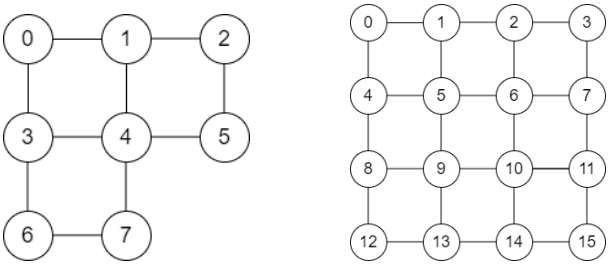

Рис. 3. Топологія Решітка для 8 та 16 процесорів

Рис.3 ілюструє топологію Решітка для N=3 та 4. Для N=3 кількість процесорів буде 9, тому видалимо останній процесор, щоб кількість процесорів для всіх топологій була рівною 8.

Гіперкуб

Для синтезу топології Гіперкуб [1] потрібно інвертувати кожен біт номера процесора (починаючи з 0) в двійковій системі, отримані числа і будуть «сусідами» процесора.

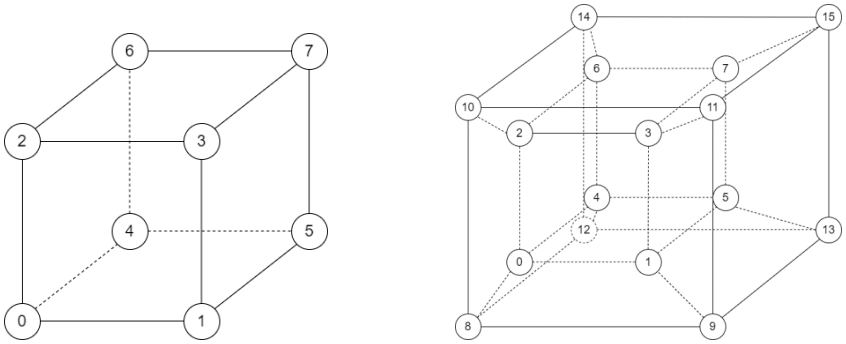

Рис. 4. Тривимірний і чотиривимірний гіперкуб

Представимо графічно топологію Гіперкуб для 8-и та 16-и процесорів (рис. 4).

Топологія де Бруйна [1]

Для топології де Бруйна, щоб отримати код сусідньої вершини потрібно виконати зсув вліво та вправо та вставити 0 і 1 у звільнений біт.

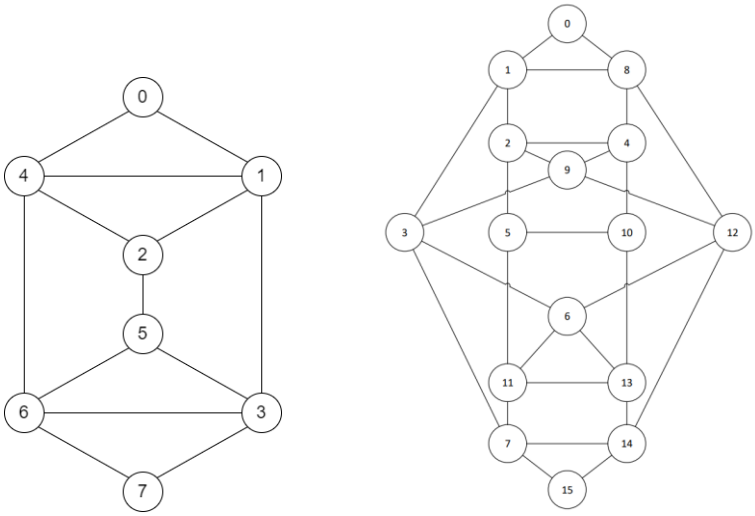

Рис. 5. Топологія де Бруйна для 8 та 16 процесорів

Представимо графічно топологію де Бруйна для 8-ми та 16-ти процесорів (рис. 5).

Розширена топологія де Бруйна

Синтез розширеної топології де Бруйна представляє собою синтез топології де Бруйна [1, 2] з використанням трійкового коду, тобто додається цифра -1 (позначається Т). При зсуві Т також вставляється у звільнений біт. Тобто при однаковій кількості процесорів кількість зв'зків збільшується. При конвертації трійкового коду в десяткову систему числення маємо, що 0Т та Т1 дорівнюють -1, а 01 і 1Т = 1, так як:

0Т = 0+-1=-1, Т1 = -10+01=-2+1=-1

01 = 1, 1Т = 10+0-1=2+-1=1

Для нумерації процесорів приймемо, що 0-3 відповідають номерам від 0 до 3, а -1 – -3 відповідають номерам від 4 до 6, для 3-розрядного коду відповідно – -1 – -7 – від 8 до 14. Таблиця 4 ілюструє синтез розширеної топології де Бруйна на основі 3-розрядного надлишкового коду.

Таблиця 4

Синтез розширеної топології де Бруйна на основі 2-розрядного надлишкового коду.

Процесор			Зсув вправо			Зсув вліво		
№	Десятк. код	Трійковий код	1 >	0 >	Т >	< 1	< 0	< Т
0	0	00	10 (2)	00 (0)	Т0 (5)	01 (1)	00 (0)	0Т (4)
1	1	01	10 (2)	00 (0)	Т0 (5)	11 (3)	10 (2)	1Т (1)
4	-1	0Т	10 (2)	00 (0)	Т0 (5)	Т1 (4)	Т0 (5)	ТТ (6)
2	2	10	11 (3)	01 (1)	Т1 (4)	01 (1)	00 (0)	0Т (4)
3	3	11	11 (3)	01 (1)	Т1 (4)	11 (3)	10 (2)	1Т (1)
1	1	1Т	11 (3)	01 (1)	Т1 (4)	Т1 (4)	Т0 (5)	ТТ (6)
5	-2	Т0	1Т (1)	0Т (4)	ТТ (6)	01 (1)	00 (0)	0Т (4)
4	-1	Т1	1Т (1)	0Т (4)	ТТ (6)	11 (3)	10 (2)	1Т (1)
6	-3	ТТ	1Т (1)	0Т (4)	ТТ (6)	Т1 (4)	Т0 (5)	ТТ (6)

Так як після конвертації у нас виходить 7 процесорів, то додамо восьмий (за номером 7) з такими ж зв'язками, як для звичайної топології де Бруйна (3, 6), щоб кількість процесорів для всіх топологій була рівна 8.

На рис. 6. представимо розширену топологію де Бруйна для 8 процесорів на основі 2-розрядного надлишкового коду.

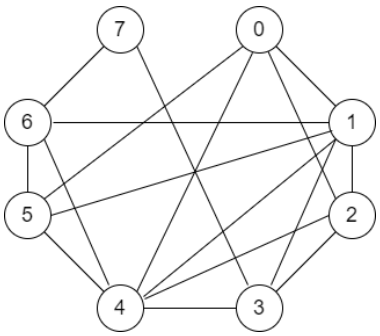

Рис. 6. Розширена топологія де Бруйна для 8 процесорів

Результати експериментів

Результати планування графів задач методів Крамера та Ньютона для вказаних топологій наведені в Таблиці 5.

$K_п$ – це коефіцієнт прискорення багатопроцесорної системи перед однопроцесорною і розраховується за формулою $K_п = t_1/t_n$, де t_1 – час виконання задачі використовуючи один процесор, t_n – час виконання задачі в багатопроцесорній системі.

K_e – це коефіцієнт ефективності багатопроцесорної системи перед однопроцесорною і розраховується за формулою $K_e = t_1/(n*t_n)$, де t_1 – час виконання задачі використовуючи один процесор, t_n – час виконання задачі в багатопроцесорній системі, n – кількість процесорів в багатопроцесорній системі.

Навантаження – це характеристика, яка показує відношення часу, який витрачає процесор на «корисну» роботу (виконання задачі) до всього часу виконання задачі. Розраховується за формулою $H = t_p/t_n$, де t_p – час, який який витрачає процесор на виконання «корисної» роботи (payload), t_n – час виконання всієї задачі в багатопроцесорній системі.

Таблиця 5

Результати планування.

Метод	**К-сть процесорів**	**Топологія**	**Час виконання**	**$K_п$**	**K_e**	**Середнє навантаження, %**	**Середня к-сть пересилок**	**Вартість системи**	**Durations * Cost**
Крамера	8	Гіперкуб	112	6.7	0.8	82	4	24	2688
		Де Бруйна	114	6.6	0.8	80	6	32	3648
		Р-на де Бруйна	111	6.7	0.8	83	4	48	5328
		Решітка	120	6.2	0.8	76	9	32	3840
	16	Гіперкуб	74	10	0.6	62	3	64	4736
		Де Бруйна	75	10	0.6	61	4	64	4800
		Р-на де Бруйна	70	11	0.7	65	3	160	11200
		Решітка	74	10	0.6	62	4	64	4736
Ньютона	8	Гіперкуб	192	4.5	0.6	44	65	24	4608
		Де Бруйна	193	4.8	0.6	44	64	32	6176
		Р-на де Бруйна	179	4.8	0.6	51	50	48	8592
		Решітка	205	4.2	0.5	40	71	32	6560
	16	Гіперкуб	141	6.1	0.4	34	31	64	9024
		Де Бруйна	150	5.8	0.4	31	37	64	9600
		Р-на де Бруйна	129	6.8	0.4	37	30	160	20640
		Решітка	158	5.5	0.3	28	45	64	10112

Середнє навантаження розраховується за формулою $H_c = \frac{\sum_0^{n-1} \frac{t_p}{t_n}}{n}$, де t_p – час, який витрачає процесор на виконання «корисної» роботи (payload), t_n – час час виконання всієї задачі в багатопроцесорній системі.

Вартість системи показує наскільки матеріально затратна в реалізації система і розраховується за формулою C=n*S, де S – максимальна кількість сусідів у одного процесора.

Durations * Cost – добуток часу виконання на вартість системи, показує наскільки вигідно використовувати певну топологію для досягнення відповідного часу виконання.

Представимо графічно залежність часу виконання завдання від типу топології.

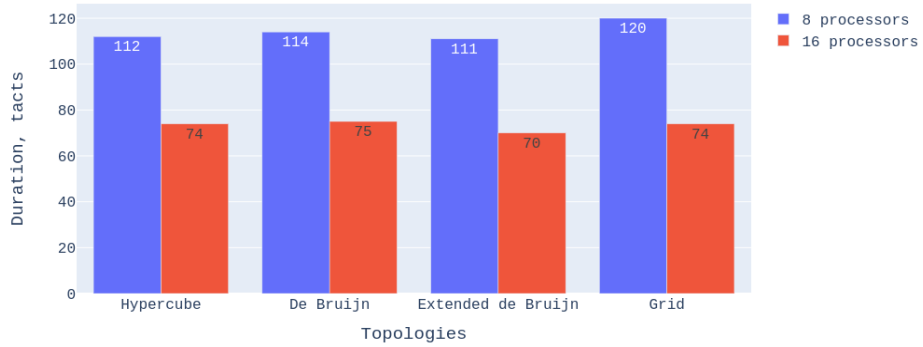

Рис. 7. Залежність часу виконання від топологій та кількості процесорів для методу Крамера

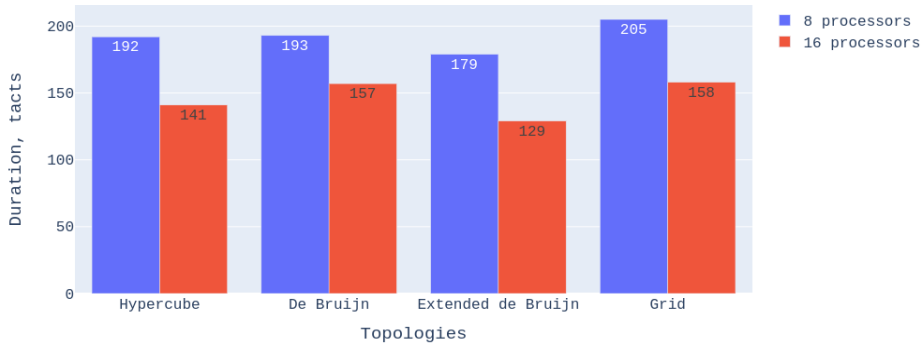

Рис. 8. Залежність часу виконання від топологій та кількості процесорів для методу Ньютона

Графічне представлення залежності добутку часу виконання на вартість системи (Durations * Cost) від топології:

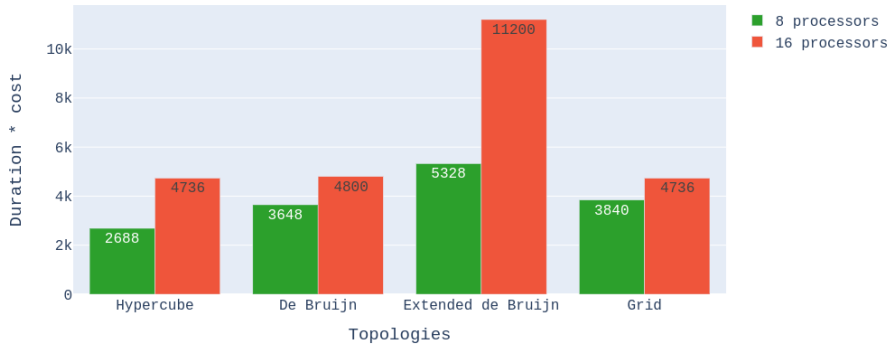

Рис. 9. Залежність добутку часу виконання на вартість системи від топологій та к-сті процесорів для методу Крамера

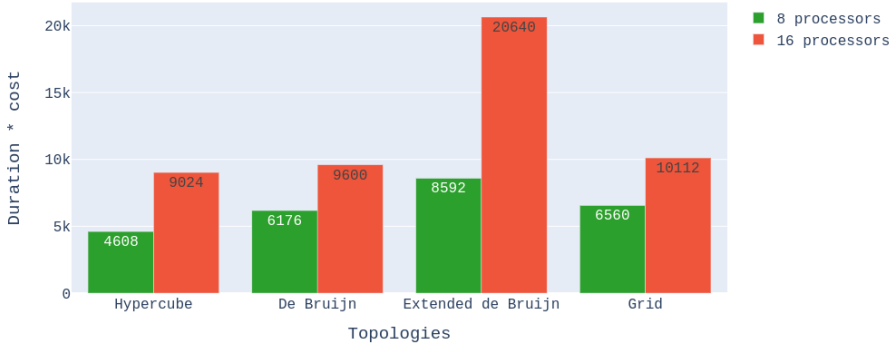

Рис. 10. Залежність добутку часу виконання на вартість системи від топологій та к-сті процесорів для методу Ньютона

Проаналізувавши результати планування можна зробити наступні висновки.

Топологія решітка показала непогані результати (на рівні з топологіями гіперкуб та де Бруйна) для методу Крамера для 16 процесорів, так як «крок» до наступного сусіда дорівнює 4, а так як для метода Крамера для задач 17-20, батьківські задачі також мають «крок» 4, то зв'язки ідеально підходять для пересилок. Для методу Ньютона результати вже не настільки «вражаючі», так як позначається великий діаметр, через який потрібно більше часу витрачати на пересилки.

Топології гіперкуб та де Бруйна показують практично ідентичні результати, порівнюючи час

виконання, хоча гіперкуб показує кращий результат (рис. 7, 8). Причому добуток часу виконання на вартість системи для топології гіперкуб найменший серед всіх топологій, як для 8, так і для 16 процесорів, як видно з рисунків 9 та 10, з чого можна зробити висновок про вигоду використання даної топології.

Проте найкращі результати показала розширена топологія де Бруйна на основі надлишкового коду, це особливо помітно для методу Ньютона, де пересилок значно більше ніж в методі Крамера. Це доволі очікуваний результат, адже кількість зв'язків в цій топології найбільша, при чому зі збільшенням розрядності кількість зв'зків збільшується більше ніж для інших, це показує великий відрив у часі виконання для 16 процесорів для методу Крамера.

Висновки

В роботі розглянуто планування графів задач методів Крамера та Ньютона для різних топологій. Топологія гіперкуб є найбільш оптимальною та найбільш універсальною, а топологія на основі розширеного графу Де Бруйна, має кращі характеристики по швидкості за рахунок збільшеного ступеня вузлів. Маючи невелику вартість та діаметр, гіперкуб показує, в середньому, гарні результати (2 місце за часом виконання після розширеної топології де Бруйна), тому ця топологія часто використовується і в сьогоденні. Проте, при побудові відмовостійких топологічних організацій краще використовувати топології з резервуванням, такі як топологія на основі розширеного графу Де Бруйна.

Література

1. H. Loutskii, A. Volokyta, P. Rehida, O. Honcharenko, B. Ivanishchev and A. Kaplunov, "Increasing the fault tolerance of distributed systems for the Hyper de Bruijn topology with excess code," 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, 2019, pp. 1–6.

2. Loutskii, H., Volokyta, A., Rehida, P., Goncharenko, O. (2019). Using excess code to design fault-tolerant topologies. Technical sciences and technologies, 1 (15), 134–144.

3. Богданов А. Архитектуры и топологии многопроцессорных вычислительных систем / А. Богданов, В. Корхов, В. Мареев, Е. Станкова // Многопроцессорные компьютерные системы. – 2004. – С. 25-26.

4. Kwok Y., Ahmad I. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. Static scheduling algorithms. 1998. P. 11–14, 55–65.

5. Kwok Y., Ahmad I. Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures. Computer Systems. 1995. P. 36–42.

6. Метод Крамера [Електронний ресурс]. – 2016 – Режим доступу : http://cyclowiki.org/wiki/Метод_Крамера

7. Шохин К. Метод Ньютона для систем нелинейных уравнений [Електронний ресурс] / Шохин К., Лебедев А. – 2018. – Режим доступу : https://algowiki-project.org/ru/Метод_Ньютона_для_систем_нелинейных_уравнений

References

1. H. Loutskii, A. Volokyta, P. Rehida, O. Honcharenko, B. Ivanishchev and A. Kaplunov, "Increasing the fault tolerance of distributed systems for the Hyper de Bruijn topology with excess code," 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, 2019, pp. 1–6.

2. Loutskii, H., Volokyta, A., Rehida, P., Goncharenko, O. (2019). Using excess code to design fault-tolerant topologies. Technical sciences and technologies, 1 (15), 134–144.

3. Bogdanov A. Arhitektury i topologii mnogoprocessornykh vychislitel'nykh sistem / A. Bogdanov, V. Korkhov, V. Mareev, E. Stankova // Mnogoprocessornye komp'yuternye sistemy. – 2004. – S. 25-26.

4. Kwok Y., Ahmad I. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. Static scheduling algorithms. 1998. P. 11–14, 55–65.

5. Kwok Y., Ahmad I. Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures. Computer Systems. 1995. P. 36–42.

6. Metod Kramera [Elektronnij resurs]. – 2016 – Rezhim dostupu : http://cyclowiki.org/wiki/Metod_Kramera

7. Shokhin K. Metod N'yutona dlya sistem nelinejnykh uravnenij [Elektronnij resurs] / Shokhin K., Lebedev A. – 2018. – Rezhim dostupu : https://algowiki-project.org/ru/Metod_N'yutona_dlya_sistem_nelinejnykh_uravnenij

Надійшла/Paper received : 06.04.2021 р. Надрукована/Printed : 02.06.2021 р.