УДК 004.451.24

СТІРЕНКО С.Г.

МОДЕЛЬ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ ПАРАЛЕЛЬНИХ СИСТЕМ

Рассматриваются существующие модели организации вычислений в параллельных системах, применяемые в различных инструментальных средствах. Предложены подходы к оценке єффективности параллельных систем с локальной памятью. Приведен математический формализм и семантика операций, программная реализация которых позволит соответствовать математической модели.

Different models used in modern libraries and frameworks to organize computations in parallel systems are studied and reviewed. An improved approach, which allows consideration of data locality in distributed system. The mathematical formalism of this model and semantics of operations on the model are described.

Порівняльний аналіз технологій програмування паралельних систем

На сьогоднішній день існує велика кількість технологій та моделей програмування паралельних обчислювальних систем, що підтримуються більшістю сучасного апаратного забезпечення таких систем. Переважна кількість моделей програмування, що застосовуються в вищеперелічених технологіях заснована на особливостях організації паралельних обчислень в тому чи іншому апаратному забезпеченні. Зокрема виділяють два великих класи обчислювальних систем: обчислювальні системи зі спільною пам'яттю, та обчислювальні системи з локальною пам'яттю, — кожна з яких має свої особливості організації роботи з даними при паралельних обчисленнях. Так для систем зі спільною пам'яттю найбільш часто застосовується модель, заснована на спільних даних, які можуть бути розташовані в спільній пам'яті обчислювальної системи та доступні всім учасникам паралельних обчислень. При цьому виникає ряд задач, які необхідно розв'язати для забезпечення коректності доступу до даних в пам'яті, зокрема задачі взаємного виключення та синхронізації. Перша з них необхідна щоб запобігти накладення одночасних записів в одну комірку пам'яті, яке може призвести, наприклад, до некоректного результуючого значення, тобто такого, яке не було записано жодним з одночасно виконуваних потоків. В той час як друга дозволяє організовувати очікування певної події в паралельно виконуваному потоці. З іншого боку, для обчислювальних систем з локальною пам'яттю типовою є модель взаємодії, заснована на передачі повідомлень, при використанні якої необхідно розв'язувати проблему підтримання непротирічності копій даних, що знаходяться в локальній пам'яті різних обчислювальних вузлів, а також проблему синхронізації.

Особливості організації доступу до даних зведені до табл. 1. Можна відмітити, що більшість розглянутих широко застосовуваних у високопродуктивних обчисленнях технологій програмування орієнтовані лише на обчислювальні системи зі спільною пам'яттю або лише на обчислювальні системи з локальною пам'яттю. Також слід зазначити, що ряд технологій додатково підтримують використання так званих акселераторів для виконання довільних обчислень.

Розглянемо паралельне виконання обчислень в гетерогенній, тобто неоднорідній, обчислювальній системі з локальною пам'яттю, що містить K вузлів та N процесорів. Один вузол може містити від 1 до $(N - K + 1)$ процесорів. Гетерогенність системи може полягати в різній продуктивності обчислювальних вузлів або у різній пропускній здатності мережевих зв'язків між ними, при чому мережа організована таким чином, щоб забезпечити можливість зв'язку між будь-якою парою вузлів.

Виконання паралельних обчислень в системі з локальною пам'яттю передбачає пересилання даних, які необхідні для виконання певної частини обчислень, з локальної пам'яті одного вузла в локальну пам'ять іншого, на якому будуть виконуватися обчислення. Позначимо M_i обсяг пам'яті, що використано в i-тому вузлі, де $i \in$ – номер вузла. Позначимо m_k розмір k-го блоку даних. Позначимо $t^i_{rdy,k}$ час готовності k-го блоку даних в і-тому вузлі, необхідних для обчислення в певному іншому j-тому вузлі, такому що $i \neq j$ Причому $t^i_{rdy,k} = 0 \ \forall i \in \overline{(1,K)}$ для вхідних даних, які не обчислюються в процесі роботи (вважаємо, що вхідні дані перенесені засобами операційної системи на всі вузли, в яких вони необхідні для обчислень) Позначимо $t^j_{req,k}$ час запиту k-го блоку даних для виконання обчислень з його використанням в j-тому вузлі. Відмітимо, що для у загальному випадку, обсяг

пам'яті, що буде використано, може залежати від значень вхідних даних та від результатів обчислень на попередніх кроках, тому його необхідно враховувати динамічно, тобто в процесі виконання обчислень. [1,2] Тоді необхідно визначити моменти часу для початку передачі даних $t_{comm,k}^{i\to j}$, такі що задовольняють нерівність $t_{comm,k}^{i\to j} \geq t_{rdy,k}^{i}$, максимізують ефективність використання обчислювальної системи $K_E \to \max$ та мінімізують обсяг пам'яті, що використано в усіх вузлах $\sum_{i=1}^{K} M_i \to \min$, з урахуванням можливості багаторазової зміни вузла j на j' у проміжок часу $\left[t_{rdy,k}^{i}, min\{t_{req,k}^{j}, t_{req,k}^{j'}\}\right]$. У якості критерію ефективності використовується коефіцієнт ефективності K_E, який визначається відношенням прискорення паралельної реалізації обчислень у порівнянні з послідовною реалізацією до кількості процесорів N. [3] При визначенні $t_{comm,k}^{i\to j}$ також врахувати час, необхідний на безпосередню передачу даних між вузлами $t_{T,k}^{i\to j}$, який залежить від структури системи та обсягу даних.

Необхідно розглянути три наступні випадки співвідношень між часом запиту даних $t_{req,k}^{j}$ і часом готовності даних $t_{rdy,k}^{i}$.

а) Дані обчислені раніше, ніж вони використовуються вперше: $t_{rdy,k}^{i} < t_{req,k}^{j}$. В цьому випадку моделі, що забезпечують підтримку негайної передачі даних виконають переміщення даних у момент готовності останніх $t_{comm,k}^{i\to j} = t_{rdy,k}^{i}$, а моделі, що забезпечують підтримку відкладеної передачі даних виконають переміщення у момент запиту $t_{comm,k}^{i\to j} = t_{req,k}^{j}$. При цьому, незалежно від підтримуваного способу передачі даних, обсяги пам'яті, що використовується в обох процесах, мають підпорядковуватися обмеженням

$$\begin{cases} M_i\left(t_{comm,k}^{i\to j} + t_{T,k}^{i\to j} + 1\right) = M_i\left(t_{comm,k}^{i\to j}\right) - m_k \\ M_j\left(t_{comm,k}^{i\to j} + t_{T,k}^{i\to j}\right) = M_j\left(t_{comm,k}^{i\to j} - 1\right) + m_k \\ M_i\left(t_{comm,k}^{i\to j}\right) \leq M_i(t) \leq M_i\left(t_{comm,k}^{i\to j} + t_{T,k}^{i\to j} + 1\right) \\ \forall t: 0 \leq \left(t - t_{comm,k}^{i\to j}\right) \leq t_{T,k}^{i\to j} \\ M_j\left(t_{comm,k}^{i\to j}\right) \leq M_j(t) \leq M_j\left(t_{comm,k}^{i\to j} + t_{T,k}^{i\to j} + 1\right) \\ \forall t: 0 \leq \left(t - t_{comm,k}^{i\to j}\right) \leq t_{T,k}^{i\to j} \end{cases} \quad (1)$$

де перші два рівняння показують, що пам'ять, яка була використана для зберігання даних в i-тому вузлі може бути звільнена після завершення передачі, а в j-тому вузлі після передачі має бути виділено достатньо пам'яті для зберігання отриманих даних; а останні дві нерівності показують, що виділення та звільнення пам'яті може відбуватися будь-яким чином в процесі передачі, наприклад лінійно або стрибкогодібно на початку та у кінці передачі. В даному випадку необхідно визначити такий час початку передачі $t_{comm,k}^{i\to j}$, $t_{rdy,k}^{i}$, $\leq t_{comm,k}^{i\to j}$, $\leq t_{req,k}^{j}$, що максимізує ефективність та мінімізує максимальний обсяг використаної пам'яті на проміжку часу $\left[t_{rdy,k}^{i},\ t_{req,k}^{j} + t_{T,k}^{i\to j},\right]$:

$$\max_{t_{rdy,k}^{i} \leq t \leq t_{req,k}^{j} + t_{T,k}^{i\to j}} (M_i(t) + M_j(t)) \to min$$

Слід зазначити, що обсяг пам'яті, що використана в кожному з процесів, може залежати від значень вхідних даних або результатів попередніх обчислень, тому недоцільно виконувати оптимізацію статично.

В цьому випадку також необхідно врахувати можливість зміни вузла j, на якому фактично будуть виконані обчислення, у час $t_{migr,k}^{i\to j'}$, такий що $t_{rdy,k}^{i} < t_{migr,k}^{i\to j'} < \min\ \{t_{req,k}^{j}, t_{req,k}^{j'}\}$ внаслідок роботи методів балансування навантаження. При цьому, якщо k-тий блок даних вже було передано у локальну пам'ять j-того вузла, необхідно передати його із локальної пам'яті вузла j у локальну пам'ять вузла j'. Для мінімізації часу, необхідного на передачу даних, необхідно також мінімізувати різницю між часом передачі та часом запиту даних

$$t_{req,k}^{j} - t_{comm,k}^{i\to j} - t_{T,k}^{i\to j} \to min,$$

Оскільки рішення про балансування навантаження приймаються під час роботи програми, необхідно перерозрахувати $t_{comm,k}^{i\to j'}$ після нього.

б) Дані готові в той самий момент, у який вони мають бути використані вперше $t_{rdy,k}^{i} = t_{req,k}^{j}$. В цьому випадку, оскільки розглядаються лише різні вузли $i \neq j$, також необхідно виконати передачу даних. При цьому на безпосередню передачу буде витрачено час $t_{T,k}^{i\to j'}$. Розпочати передачу даних у час $t_{comm,k}^{i\to j'} < t_{rdy,k}^{i}$ неможливо, оскільки дані ще не були обчислені. Розпочати передачу даних із затримкою t_{ω} недоцільно, оскільки під час цієї обчислення виконуватись не будуть, звідки випливає, що відношення часу безпосередніх обчислень до загального часу виконання задачі зменшиться, тому загальна ефективність використання системи під час обчислень знизиться. Таким чином, в даному випадку необхідно ініціювати передачу даних у момент часу $t_{comm,k}^{i\to j'} = t_{rdy,k}^{i} = t_{req,k}^{j}$.

Оскільки запит на використання даних відбувається одночасно із закінченням їх підготовки, ані вузол *i*, який виконує підготовку, ані вузол *j*, якому необхідні ці дані для подальших обчислень, не можуть бути змінені через балансування навантаження: підготовка даних щойно закінчено, тому переміщення обчислень щодо підготовки не відбуватиметься; дані для подальшого використання вже запитані, тому будь-яке переміщення наступних неможливе до отримання цих даних, інакше дані можуть бути передані у невірний вузол.

в) Запит на використання даних відбувся раніше, аніж вони були підготовлені $t^i_{rdy,k} > t^j_{req,k}$. Цей випадок найбільш суттєво впливає на ефективність обчислень, оскільки система буде простоювати впродовж $(t^i_{rdy,k} - t^j_{req,k})$. А також на безпосередню передачу буде витрачено час $t^{i\to j}_{T,k}$. Розпочати передачу даних у час $t^{i\to j\prime}_{comm,k} < t^i_{rdy,k}$ неможливо, оскільки дані ще не були обчислені. Розпочати передачу даних із затримкою t_ω недоцільно за міркуваннями, аналогічними до пункту (б).

В цьому випадку необхідно врахувати можливість зміни вузла і, на якому виконується підготовка даних. Якщо на момент часу $t^j_{req,k}$ підготовка даних ще не розпочалась, у час t < $t^i_{rdy,k}$внаслідок роботи алгоритмів балансування навантаження може бути змінено вузол, на якому фактично будуть виконані обчислення щодо підготовки *k*-го блоку даних. Завдяки подібній заміні може бути змінений час отримання даних, запит на використання яких вже видано, $t^{i\prime}_{rdy,k} \leq t^i_{rdy,k}$, що в свою чергу може змінити час ініціювання передачі даних.

Вироджений випадок *i* = *j* не розглядається, оскільки виконувати передачу даних між вузлами для нього не потрібно, та вони стають доступні відразу після їх обчислення. Для узгодженості приймемо у цьому випадку $t^{i\to j}_{comm,k} = t^i_{rdy,k}$.

Питання визначення оптимального маршруту для передачі даних між вузлами обчислювальної системи з точки зору швидкості такої передачі відноситься до розв'язання задачі маршрутизації, для якої запропоновано ряд ефективних алгоритмів. [4]

Питання керування ресурсами обчислювальної системи та виділення ресурсів для виконання обчислень, а також розподілу виділених ресурсів системи для виконання певних частин обчислень відносяться до функцій операційної системи. Підходи до розв'язання цієї задачі глибоко вивчені та існує велика кількість способів її розв'язання, [5] тому в даному досліджені ці питання не розглядаються, а використовуються відомі та реалізовані підходи.

Оцінка ефективності

Будемо вважати, що обчислення виконуються на паралельній обчислювальній системі з локальною пам'яттю, вузли якої можуть містити спільну для певної невеликої кількості процесорів пам'ять. Обчислювальна система з локальною пам'яттю також може містити акселератори певного виду, які розглядаються як окремі вузли з локальною пам'яттю, оскільки вимагають явного перенесення необхідних даних між пам'яттю акселератора та основною пам'яттю вузла. Нехай система містить K вузлів, при чому *i*-тий вузол в свою чергу містить ni процесорів. В цьому разі загальна кількість процесорів, що доступні в обчислювальній системі $N = \sum_{i=1}^{K} n_i$. Для спрощення вважаємо, що в системі виконуються лише обчислення що розглядаються, а також складові частини операційної системи та система керування паралельними обчисленнями. При цьому обсяги обчислень, що виконуються двома різними процесорами за одиницю часу, можуть відрізнятися, а також певна частина обчислень може виконуватись лише на певних заздалегідь визначених процесорах, що має бути коректно забезпечено операційною системою.

Позначимо час виконання деякої паралельної реалізації певного алгоритму на обчислювальній системі, що розглядається, T. Час виконання розраховується від моменту початку обчислень до моменту отримання кінцевого результату. При цьому, час активного використання *j*-го процесора даними обчисленнями складає t_j, таке що

$$t_j \leq T \qquad (2)$$

через необхідність очікування введення та виведення даних, комунікації з іншими процесами, отримання доступу до спільного ресурсу тощо. Таким чином, час послідовних обчислень тієї ж самої реалізації алгоритму на даній обчислювальній системі складався б з суми часу обчислень кожним процесором окремо $\sum_{j=1}^{N} t_j$. Тоді коефіцієнт прискорення визначається відношенням оціненого часу обчислень послідовно до часу обчислень паралельно

$$K_{\text{п}} = \frac{\sum_{j=1}^{N} t_j}{T} \quad (3)$$

при чому, з урахуванням (2), можна відмітити, що коефіцієнт прискорення не має перевищувати кількості наявних процесорів

$$K_{\text{п}} = \frac{\sum_{j=1}^{N} t_j}{T} \leq \frac{\sum_{j=1}^{N} T}{T} = \frac{NT}{T} = N \quad (4)$$

В такому разі коефіцієнт ефективності визначається як відношення коефіцієнту прискорення до кількості наявних процесорів

$$K_E = \frac{K_{\text{п}}}{T} \quad (5)$$

при чому, з урахуванням (4), коефіцієнт ефективності може приймати значення в проміжку [0, 1], оскільки

$$0 \leq K_E = \frac{K_{\text{п}}}{T} \leq \frac{N}{N} = 1$$

Для подальшої оптимізації необхідно більш детально розглянути фактори, що впливають на сумарний час виконання обчислень. Зокрема слід проаналізувати складові сумарного часу обчислень, що складається з

$$T = t_c + t_{i/o} + t_w + t_s$$

де $t_c = \sum_j t_j$ – час безпосереднього виконання обчислень (англ. computations); $t_{i/o}$ – час введення та виведення даних, в тому числі з використанням мережі для обміну даних з іншими процесами (англ. input and output); t_w – час очікування (англ. wait); t_s – час неактивності через роботу системних потоків (англ. sleep). Можна відмітити, що коефіцієнт ефективності обмежуються відношенням часу безпосереднього виконання обчислень до суми всіх інших компонентів сумарного часу виконання

$$K_E = \frac{K_{\text{п}}}{N} = \frac{\sum_{j=1}^{N} t_j}{TN} = \frac{t_c}{(t_{i/o} + t_w + t_s)N}$$

що відповідає обмеженням на коефіцієнт прискорення, встановленими законом Амдала [6]. Також необхідно більш детально розглянути складові часу очікування, які можуть бути зменшені в рамках запропонованого підходу

$$t_w = t_{i/o} + t_{w,swap} + t_{w,e}$$

де $t_{w,i/o}$ — час очікування введення або виведення даних, в тому числі з використанням мережі для зв'язку з іншими процесами, $t_{w,swap}$ – час очікування підкачки даних в основну пам'ять в рамках одного вузла, $t_{w,e}$ – час очікування інших подій, зокрема взаємного виключення. Оцінка часу очікування підкачки даних в основну пам'ять є вкрай важливою характеристикою, оскільки за умови передачі даних у вузол значно раніше, аніж вони будуть оброблені, виникає необхідність їх зберігати. При цьому збільшується загальний обсяг використовуваної пам'яті а також на певний час після завершення передачі змінюється статистика звернень до сторінок пам'яті, що може негативно вплинути на ефективність роботи системи підкачки даних у віртуальній організації пам'яті. Аналогічний вплив може мати і зберігання даних на певному вузлі, після їх обчислення, тому необхідно розглядати роботу з основною пам'яттю більш детально. Для цього позначимо загальний обсяг використаної основної пам'яті

$$M(t) = \sum_{i=1}^{k} M_i(t) = \sum_{i=1}^{K} \sum_{l=1}^{n_i} M_{i,l}(t) + M_{i,s}(t) \quad (6)$$

де M_i – обсяг пам'яті, використаної в і-тому вузлі зі спільною пам'яттю, $M_{i,l}$ – обсяг пам'яті, використаної в l-тому процесі, що виконується на i-тому вузлі, а $M_{i,s}$ – спільна пам'ять, доступна одночасно всім процесам, що виконуються на i-му вузлі. Такий поділ є доцільним, оскільки в обчислювальній системі, що розглядається, вузли мають спільну пам'ять, і вплив від зберігання даних в одному з процесів, що виконується на даному вузлі, розповсюджується також на інші процеси, що виконуються на даному вузлі. Тому недостатньо розглядати використання пам'яті лише в одному процесі. Обсяг пам'яті, що використовується, є функцією часу. Для спрощення аналізу, можна ввести статистичні показники M_Σ — сумарний обсяг пам'яті, який було виділено під час виконання обчислень; $\widehat{M}$ – середній обсяг пам'яті, який було виділено під час виконання обчислень; M^* – піковий обсяг пам'яті, який одночасно було виділено під час виконання обчислень. Для характеристики роботи системи підкачки даних в основну пам'ять також необхідно аналізувати кількість сторінок пам'яті, які були збережені поза основною пам'яттю та пізніше були завантажені за запитом. Це типово характеризується кількістю так званих *кеш-промахів* (англ. cache miss), тобто кількістю звернень до сторінок пам'яті, які не були розміщені в основній пам'яті на момент звернення. Під час такого промаху виконується очікування підкачки даних в основну пам'ять, яке впливає на час очікування tw,swap. Позначимо загальну кількість кеш-промахів C, та кількість кеш-промахів за одиницю часу $C' = \frac{C}{T}$.

Ряд характеристик варто також розглядати не лише для обчислювальної системи в цілому, а також для кожного процесу окремо. До таких

характеристик відносяться час очікування введення та виведення $t^j_{w,i/o}$ в j-тому процесі, час очікування підкачки даних $t^j_{w,swap}$ в j-тому процесі, а також час виконання введення та виведення $t^j_{i/o}$ та час безпосередньо обчислення t^j_c в кожному процесі. Це також дозволяє розглядати ряд статистичних значень, які загалом характеризують процес виконання обчислень, таких як середні за процесами значення часу очікування введення та виведення $\hat{t}^j_{w,i/o}$часу очікування підкачки даних $\hat{t}^j_{w,swap}$ або максимальні значення цих характеристик $t^*_{w,i/o}$ $t^*_{w,swap}$. Перші впливають на загальний час виконання обчислень в цілому, в той час як останні відіграють роль обмежуючого фактора.

Розглянуті вище характеристики мають опосередкований вплив на результуючий коефіцієнт ефективності, шляхи цього впливу показані на рис. 1. Тобто обсяг використаної в кожному вузлі пам'яті Мі впливає на кількість кеш-промахів С, однак не є єдиним фактором, що її визначає. Кількість кеш-промахів впливає на час очікування підкачки даних $t_{w,swap}$, який в свою чергу впливає на загальний час виконання обчислень. З іншого боку, час очікування введення та виведення $t_{w,i/o}$ безпосередньо впливає на загальний час виконання обчислень. На рис. 1 рамкою виділені дві характеристики, M_i та $t_{w,i/o}$, які можна змінювати при різних підходах до організації пересилання даних.

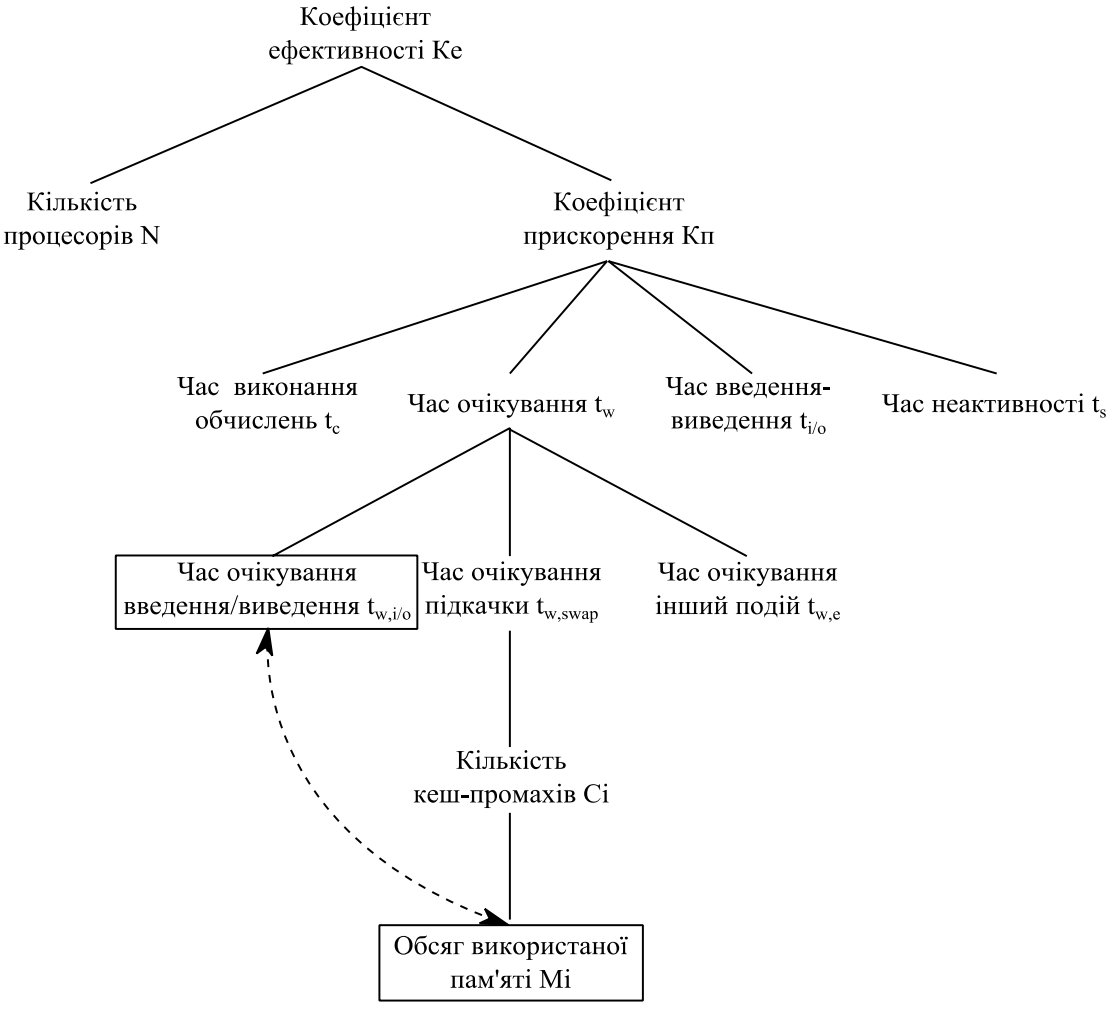

Рисунок 1. Вплив динамічних характеристик на коефіцієнт ефективності

Однак, дані характеристики також впливають одна на одну, що відмічено пунктирною лінією на рис. 1. Так, зменшення часу очікування передачі даних, тобто введення та виведення, може бути досягнуте завдяки буферизації k-го блоку даних на вузлі j, в якому будуть виконуватись обчислення, перед тим, як вони будуть запитані

$$\left(t^{i\to j}_{comm,k} + t^{i\to j}_{T,k}\right) < t^i_{req,k} \Rightarrow \left(t^j_{w,\frac{i}{o}} - t^{j'}_{w,\frac{i}{o}}\right) \geq t^{i\to j}_{T,k}$$

однак це призведе до збільшення обсягів використаної пам'яті

$0 \le M_i(t)' - M_j(t) \le m_k \forall t > t_{comm,k}^{i \to j}$.

Таким чином, характеристики вступають у протиріччя, покращення однієї з них може мати погіршити іншу та навпаки, тому необхідно досліджувати їх сумарний вплив на коефіцієнт ефективності

$$\begin{cases} \sum_{j=1}^{N} t_{w,i/o}^{j} \to min \\ \max_j t_{w,i/o}^{j} \to min \\ \sum_{i=1}^{K} \widehat{M_i} \to min \\ \max_i \widehat{M_i} \to min \\ K_E \to max \end{cases} \quad (7)$$

Відповідно до вищесказаного, найбільш важливим критерієм є коефіцієнт ефективності, який характеризує виконання обчислень в цілому. Однак для його підвищення з використанням автоматизованих засобів, необхідно проаналізувати ряд характеристик, що впливають на його значення. До них відносяться час очікування введення та виведення даних, час очікування підкачки даних в основу пам'ять, обсяг використаної пам'яті та кількість кеш-промахів. Застосування певних засобів мінімізації використаної пам'яті безпосередньо впливає на час введення та виведення даних та навпаки, тому для збільшення коефіцієнту ефективності необхідно розглядати та покращувати перелічені характеристики одночасно.

Таблиця 1

Порівняння обраних моделей та технологій програмування паралельних обчислювальних систем

Технологія	Цільова система	Особливості доступу до даних
Вказівки компілятору (OpenMP)	Обчислювальна система зі спільною пам'яттю	Копіювання автоматичне відповідно до вказівок; додаткові вказівки для взаємного виключення; лише спільна пам'ять.
Низькорівневий інтерфейс передачі повідомлень (MPI)	Обчислювальна система з локальною пам'яттю	Обов'язкове явне копіювання даних між вузлами; можливість програмованої буферизації під час передачі.
Низькорівневий інтерфейс передачі повідомлень з оптимізацією (MPI-2/3)	Обчислювальна система з локальною пам'яттю, вузли якої містять спільну пам'ять	Те ж саме, що MPI, та додатково використання спільної пам'яті в рамках вузла за можливості, технології прямого дос-тупу у віддалену пам'ять.
Інтерфейс програмування графічних акселераторів CUDA	Обчислювальна система зі спільною пам'яттю та застосуванням графічних акселераторів nVidia	Обов'язкове явне копію-вання даних в пам'ять акселератора. Багато-рівнева ієрархія пам'яті.
Низькорівневий інтерфейс передачі повідомлень з урахуванням графічних акселераторів (CUDA-MPI)	Обчислювальна система з локальною пам'яттю та застосу-ванням графічних акселераторів nVidia	Те ж саме, що CUDA, та додатково можливість явного копіювання у пам'ять акселератора в іншому вузлі.
Технологія Array Building Blocks	Обчислювальна система зі спільною пам'яттю	Автоматичне взаємне виклю-чення в екземплярах спе-ціально структурованих типів; лише спільна пам'ять.
Технологія Threading Building Blocks	Обчислювальна система зі спільною пам'яттю та застосу-ванням акселераторів типу Xeon Phi	Автоматичне копіювання на акселератор та взаємне вик-лючення в екземплярах спе-ціально структурованих типів; лише спільна пам'ять.
Стандартна бібліотека Futures мови C++	Обчислювальна система зі спільною пам'яттю	Можливість відкладеного обчислення та доступу (за без-посереднім запитом), зокрема паралельного. Лише спільна пам'ять.
Стандартна техно-логія відкладених обчислень мови Haskell	Обчислювальна система зі спільною пам'яттю	Можливість відкладеного обчислення та доступу. Лише спільна пам'ять.

Список литературы

1. Стиренко, С.Г. Модель организации вычислений в распределjнной системе / С.Г. Стиренко, А.И. Зиненко, Д.В. Грибенко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. – 2012. –Т. 57. – С. 101–109.
2. Stirenko, S.G. Dynamic analysis of MPI applications / S.G. Stirenko, A.I. Zinenko, D.V. Gribenko // Proceedings of the 1st international conference on High-performance computing (HPC-UA), Kiev, 2011. – K. : ВЕК+, 2011. – P. 152–153.
3. Hennessy, John L. Computer architecture: a quantitative approach/ John L Hennessy, David A Patterson. – Amsterdam, Noord-Hooland, Netherlands : Elsevier, 2011. – 824 p.
4. Tanenbaum, A.S. Computer Networks/ A.S. Tanenbaum, D.J. Wetherall. – London, UK : Pearson Education, Limited, 2010. – 1240 p.
5. Симоненко, В. П. Организация вычислительных процессов в ЭВМ, комплексах, сетях и системах / В. П. Симоненко. – К. : ВЕК+, 1997. –304 с.
6. Amdahl, Gene M. Validity of the single processor approach to achieving large scale computing capabilities / Gene M. Amdahl // Proceedings of the April 18-20, 1967, spring joint computer conference. – AFIPS '67 (Spring). – New York, NY, USA : ACM, 1967. – P. 483–485.