УДК 004.4

С. Д. Погорілий, д.т. н., проф.,
І. В. Білоконь, аспірант
О. В. Вареник, студент
Київський національний університет імені Тараса Шевченка
sdp77@i.ua, deimos@univ.net.ua, varenikav@mail.ru

Дослідження віртуальних кластерних архітектур на прикладі алгоритму Белмана-Форда

Запропоновано вдосконалення методу пошуку оптимальних конфігурацій віртуальних кластерних архітектур, що базується на використанні аналітичних оцінок поведінки алгоритму із використанням апарату модифікованих систем алгоритмічних алгебр В. М. Глушкова.

Ключові слова: віртуалізація, віртуальна машина, алгоритм Белмана-Форда, система алгоритмічних алгебр, розпаралелювання, HPC, MPI.

Вступ

Широке впровадження технологій апаратної підтримки віртуалізації у сучасних мікропроцесорах з архітектурою x64 дали можливість розгортання віртуальних машин (ВМ) на базі гіпервізорів не лише на серверних апаратних платформах, а і на доступних персональних комп'ютерах та ноутбуках. Параметри продуктивності таких віртуальних платформ дозволяють використовувати їх для побудови обчислювальних кластерів. Це дозволяє вийти на якісно новий рівень реалізації високопродуктивних обчислювальних систем. Концепцію побудови динамічно реконфігурованої кластерної обчислювальної архітектури на базі платформи віртуалізації Microsoft Hyper-V було запропоновано у роботі [1]. Перевагою подібних обчислювальних кластерів є можливість динамічної реконфігурації їх ресурсів для підвищення продуктивності або пропускної здатності кластера [2]. В роботі [3] запропоновано підхід до розв'язку задачі пошуку відповідних конфігурацій, що ґрунтується на застосуванні генетичних алгоритмів, а також метод побудови фітнес-функції. Метою цієї роботи є дослідження віртуальних кластерних архітектур на основі програмної реалізацією алгоритму Белмана-Форда (БФ) для вдосконалення запропонованого у [3] підходу, шляхом формування аналітичних оцінок поведінки програмних реалізацій паралельних алгоритмів.

Формалізація алгоритму БФ

Для проведення аналітичних оцінок треба привести паралельний алгоритм БФ до вигляду максимально абстрагованого від конкретної реалізації. Подібну задачу можна вирішити із використанням апарату модифікованих систем алгоритмічних алгебр В. М. Глушкова. Аналіз отриманої схеми алгоритму дозволить отримати результати, що будуть застосовні для довільної програмної реалізації. За такого підходу вибір певної платформи та парадигми програмування зведеться лише до зміни параметрів отриманої схеми.

Послідовна схема алгоритму

Алгоритм БФ вирішує задачу пошуку найкоротшого шляху з однієї вершини у загальному випадку, коли вага кожного з ребер може бути від'ємною. Також алгоритм БФ вказує на те, чи присутні у графі цикли з від'ємною вагою, досяжні з початкової вершини[4].

Введемо низку позначень (1)

N – розмірність графу

G – матриця суміжності графу

W – матриця розподілу ваги ребер

D – масив розподілу довжини шляхів від початкової вершини до усіх інших

P – масив для збереження маршруту

s – індекс початкової вершини

Тоді

g_{ij} – довжина ребра з вершини i в j

w_{ij} – вага ребра з вершини i в j

$d[v]$– довжина найкоротшого шляху з початкової вершини до вершини з індексом v

$p[v]$ – індекс передостанньої вершини найкоротшого шляху до вершини з індексом v

Операнди (2)

$$A_1 \equiv (d[i] = \infty)$$

$$A_2 \equiv (p[i] = \infty)$$

$$A_3 \equiv (d[i] = 0)$$

$$A_4 \equiv (p[i] = 0)$$

$$A_5 \equiv (d[i] = d[i] + w_{ij})$$

$$A_6 \equiv (p[i] = j)$$

$$A_7 \equiv (return\ ELSE)$$

$$A_8 \equiv (return\ TRUE)$$

Умови (3)

$\alpha_1(D,I,s) \equiv (d_i = s)$

$\alpha_2(D,W,I,j) \equiv (d_i > d_j + w_{ij})$

$\alpha_3(G,I,j) \equiv (g_{ij} = 1)$

$\alpha_4(i,N) \equiv (i \geq N)$

$\alpha_5(i,N) \equiv (j \geq N)$

$\alpha_6(i,N) \equiv (k > N-1)$

Всю роботу алгоритму можна розбити на N+1 кроки

1. Виконання ініціалізації.
2. На цьому етапі N-1 разів виконується обхід всіх існуючих ребер графа та застосування до них процедури ослаблення(Relax).
3. На останньому кроці, ще раз виконується обхід всіх ребер та їх ослаблення(Relax), метою якого є виявлення від'ємних циклів.

При реалізації цього алгоритму доводиться мати справу із моделюванням математичної нескінченності (зокрема це видно у операндах A_1, A_2). Для вирішення цієї проблеми було введено найбільше можливе значення ваги, що і відповідає математичній нескінченності.

Сформуємо послідовну схему алгоритму БФ, шляхом використання апарату систем алгоритмічних алгебр Глушкова(САА). Використовуючи позначення (1), (2) та (3) запишемо окремо всі кроки алгоритму у вигляді формул.

$$Init(D,P,V,s) = \{ A_1 * A_2 * (A_3 * A_4 \vee E) * (++i)\}^{\alpha_4}_{\alpha_1} \quad (4)$$

$$Rlax(W,G,D,P,i,j) = (A_5 * A_6 \vee E)^{\alpha_4} \quad (5)$$

$$BFs2(W,G,D,P) = \{ \{ (Rlax(W,G,D,P,i,j)^{\alpha_4\alpha_5\alpha_3} \vee E) * (++j)\}(++i)\} \quad (5.1)$$

$$BFs3(W,G,D) = \{ \{ ((A_7 \vee A_8) \vee E)^{\alpha_4\alpha_5\alpha_3\alpha_2} * (++j)\}(++i)\} \quad (6)$$

Для будь-якої α – ітерації будемо вважати, що на початку ітераційного процесу змінна i (або будь-яка інша змінна, що відповідає за ітерацію) ініціалізується нулем.

Перетворимо (6), скориставшись наступною властивістю САА-схем.

$$((A_7 \vee A_8) \vee E)^{\alpha_3\alpha_2} = (A_7 \vee A_8)^{\alpha_3 \wedge \alpha_2}$$

Тоді остаточно отримаємо

$$Bs3(W,G,D) = \{ \{ (A_7 \vee A_8)^{\alpha_4\alpha_5\alpha_3 \wedge \alpha_2} * (++j)\}(++i)\} \quad (7)$$

Використовуючи (4), (5), (5.1) та (6.1) сформуємо послідовну схему алгоритму БФ.

$$BF = Init(D,P,V,S) * \{ BF2(W,G,D,P)\}^{\alpha_6} * BF3(W,G,D) \quad (8)$$

Принцип обробки даних

У роботі використовується парадигма розпаралелювання алгоритму за даними, тобто кожна гілка обробляє певну частину інформаційної множини, не перетинаючись при цьому з іншими гілками.

До інформаційної множини належать матриці G, W – що містять інформацію про вхідний граф (довжина та вага ребер), а також індекс s, що вказує на початкову вершину. Також, додатково до інформаційної множини можуть бути включені ітератори – змінні, що відповідають за циклічні процеси у кожній з гілок. Виконаємо розбиття інформаційної множини, що також включає і поділ матриць G та V на підматриці. Будемо вважати, що швидкість виконання всіх потоків однакова, тому для оптимальної роботи доцільно розбити матриці на приблизно однакові частини. Якщо потрібно розділити матрицю $N*N$ на m приблизно рівних частин прямокутної форми, то можна скористатися наступними формулами

$$b_i^{N,m} = i * [\frac{N}{m}] \quad (9)$$

$$e_i^{N,m} = (i+1) * [\frac{N}{m}] \quad (10)$$

де b_i – перший рядок i–ї частини

e_i – останній рядок i–ї частини

В запропонованих формулах індекс i пробігає значення від 0 до $m-1$. Зрозуміло, що аналогічне розбиття можна зробити і для стовпчиків. Отже рівняння (9), (10) визначають об'єм даних для роботи кожної з гілок.

Підхід до формування паралельної схеми алгоритму

Перш ніж робити еквівалентні перетворення схеми (7), використовуючи апарат тотожних перетворень САА-М, розглянемо деякий узагальнений випадок.

Нехай існує α – ітерація

$$I = \{A\}^{\beta} \quad (11)$$

де A – оператор, що опрацьовує певний об'єм даних

$\beta = (i \notin \{1..N\})$

i – змінна, що ініціалізується на початку ітераційного процесу, а на кожному кроці неявно збільшується на одиницю.

Розглянемо можливе перетворення виразу (10). Для цього скористаємось загальним результатом доведеним у роботі [4].

Якщо ввести розбиття інтервалу $\{1...N\}$ на m приблизно однакових частин за допомогою формул (9), (10), тоді (11) можна представити у вигляді

$$I = \bigvee_{l=1}^{m} (\{ A\})_{\gamma_l} = \dot{\bigvee}_{l=1}^{m} (\{ A\})_{\gamma_l} \quad (12)$$

Параметр γ_l визначає межі інформаційної множини в яких працює і-та гілка.

Розглянемо тепер питання синхронізації гілок. Для синхронізації використаємо так звану бар'єрну синхронізацію, яка в певній точці блокує роботу тих потоків, що дійшли до неї, аж до тих пір, поки всі потоки не дійдуть до цієї точки. Таким чином, якщо крізь бар'єр пройшов хоча б один потік, то це гарантує, що поза бар'єром не залишилось жодного потоку.

Математично роботу бар'єру можна описати наступною функцією.[5]

$$B_m(i) = T(\lambda_i) * (\prod_{\substack{j=1 \\ j \neq i}}^{m} S(\lambda_j)) \quad (13)$$

де $T(\lambda_i)$ – контрольна точка бар'єру для i-ої гілки

$S(\lambda_i)$ – точка синхронізації (або синхронізатор) для і-ої гілки

m – кількість потоків

i – номер потоку який синхронізується

Вводячи об'єкт для синхронізації гілок (12.1), отримуємо наступні вирази

$$I = \dot{\bigvee}_{l=1}^{m} (\{ A * B_m(l)\}_{\gamma_l}) \quad (14)$$

Формування паралельної схеми алгоритму для систем із спільною пам'яттю

Застосуємо формулу (14) для кожного кроку алгоритму.

$$Init(D,P,V,s) = \dot{\bigvee}_{l=1}^{m} (\{ A_1 * A_2 * (A_3 * A_4 \vee E)_{\alpha_1} * (++i)\}_{\gamma_l}$$

$$* B_m(l)) = \dot{\bigvee}_{l=1}^{m} (P\_Init(D,P,V,s) * B_m(l)) \quad (15)$$

де $\gamma_l = i \notin \{b_l ... e_l\}$

$$BFs2(W,G,D,P) = \dot{\bigvee}_{l=1}^{m} (\{ \{ (Rlax(W,G,D,P,i,j) \vee E)_{\gamma_l \alpha_5 \alpha_3} * (++j)\} (++i)\} * B_m(l)) =$$

$$= \dot{\bigvee}_{l=1}^{m} (PBFs2(W,G,D,P) * B_m(l)) \quad (16)$$

$$BFs3(W,G,D) = \dot{\bigvee}_{l=1}^{m} (\{ \{ (A_7 \vee A_8)_{\gamma_l \alpha_5 \alpha_3 \wedge \alpha_2} * (++j)\} (++i)\} * B_m(l)) =$$

$$= \dot{\bigvee}_{l=1}^{m} (PBFs3(W,G,D) * B_m(l)) \quad (17)$$

Підставимо (15), (16), (17) до (7) та винесемо асинхронну диз'юнкцію за межі ітераційного процесу на другому етапі виконання алгоритму

$$BF = \dot{\bigvee}_{l=1}^{m} (P\_Init(D,P,V,s) * B_m(l)) * \dot{\bigvee}_{l=1}^{m} (\{ PBFs2(W,G,D,P) * B_m(l)\}_{\alpha_6}) * \dot{\bigvee}_{l=1}^{m} (PBFs3(W,G,D) * B_m(l)) \quad (18)$$

Отримана формула (18) – паралельна схема алгоритму БФ.

З точки зору узагальнення, схема (18) може бути трансформована і далі. Вже з першого погляду видно, що запропонований метод синхронізації є зручним для математичних перетворень, але не є універсальним і залежить від конкретної реалізації. Наприклад, для платформи .*NET* запропонована бар'єрна синхронізація з'явилась лише починаючи із версії *.NET Framework 4.0.* Тому доцільно розробити інші, більш універсальні методи синхронізації.

В найбільш загальному вигляді дію оператора синхронізації зручно описувати у термінах операторів контрольної точки T, циклу очікування S та оператора R(Reset) (дія цього оператора протилежна до дії оператора T). Введемо також умову для контрольних точок τ_x, що приймає одиничне значення при дії оператора $T(\tau_x)$ та нульове при дії оператора $R(\tau_x)$. Нижній індекс у виразі для умови контрольної точки визначає до якого потоку належить ця умова.

Запишемо нові оператори синхронізації у вигляді композиції операторів R, T, S.[6]

$$Sync_1(\tau_1, \tau_m) = T(\tau_1) * S(\tau_m) * R(\tau_m) * * T(\tau_1) * S(\tau_m) * R(\tau_m) \quad (19)$$

$$Sync_x(\tau_1, \tau_x) = S(\tau_{x-1}) * R(\tau_x) * T(\tau_x) * * S(\tau_{x-1}) * R(\tau_{x-1}) * T(\tau_x) \text{ при } x \neq 1 \quad (20)$$

З (19), (20) видно, що потрібно обрати одну із гілок, яка першою буде встановлювати контрольну точку, тим самим запускаючи і інші гілки. Зрозуміло, що у наслідок рівноправності всіх гілок, це може бути довільна гілка, тому нехай для зручності це буде гілка за номером 1.

Використовуючи (19), (20) запишемо схему (18) у наступному вигляді

$$BF = \dot{\bigvee_{l=1}^{m}}(P\_Init(D,P,V,s) * Sync_l(\tau_1,\tau_l))$$
$$* \dot{\bigvee_{l=1}^{m}}(\{ PBFs2(W, G, D, P) * Sync_l(\tau_1,\tau_l)\})^*_{\alpha_6}$$
$$* \dot{\bigvee_{l=1}^{m}}(PBF\_s3(W, G, D) * Sync_l(\tau_1,\tau_l)) \quad (21)$$

Схема (21) є аналогічною до (18), але тепер оператори синхронізації не прив'язані до конкретної реалізації.

Формування паралельної схеми алгоритму для систем із розподіленою пам'яттю.

Отримана схема (21) або (21.1) може бути застосована лише для систем зі спільною пам'яттю, тому вона потребує наступних трансформацій, які дозволять застосовувати її і для систем із розподіленою пам'яттю. Для цього, на певному кроці виконання алгоритму, нам потрібно виконувати обмін даними таким чином, щоб кожна з гілок мала однакову інформаційну множину. Цю задачу можна вирішити шляхом введення нових операторів обміну даними, що не залежать від конкретної програмної реалізації.

Введемо деякий оператор $DS(a,b)$, який має наступні особливості:

1. Перший із параметрів вказує на масив даних, яким ми обмінюємося із іншими гілками.
2. Другий вказує номер гілки, яка відправляє ці дані, відповідно всі інші гілки приймають ці дані.
3. Виклик цього оператора у гілці із номером b, ініціює передачу даних від цієї гілки до всіх інших, при цьому гілка за номером b призупиняє свою роботу аж до тих пір, поки всі інші гілки не закінчать прийом даних.
4. Виклик цього оператора у гілці із номером відмінним від b, ініціює процес прийому даних.
5. Для повного вирівнювання інформаційної множини на довільному кроці виконання алгоритму, цей оператор треба викликати m разів в кожній гілці, де m – кількість гілок.
6. Із пунктів (3), (4), (5) випливає, що виконавши повний обмін даними, ми також додатково синхронізуємо роботу всіх гілок.

Аналіз алгоритму показує, що обмін даними необхідно здійснювати безпосередньо перед синхронізацією роботи гілок, але враховуючи пункт (6), оператор синхронізації може бути відкинутий.

$$BFd = \dot{\bigvee_{l=1}^{m}}(P\_Init(D,P,V,s) * B_m(l)) *$$
$$* \dot{\bigvee_{l=1}^{m}}(\{ PBFs2(W, G, D, P)\} * (root = 0) *$$
$$* \{ DS(a, root) * (root++)\})^*_{\alpha_6}$$
$$* \dot{\bigvee_{l=1}^{m}}(PBFs3(W, G, D)) \quad (22)$$

де a – це деякий масив в якому знаходяться всі дані, що очікують передачі.

Отримана формула (22) – паралельна схема алгоритму БФ для систем із розподіленою пам'яттю. Аналіз схеми (22) показує, що вона може бути застосована і для систем із спільною пам'яттю. Тоді виникає питання, а чи буде вона ефективніша за схему (21)? Для відповіді на питання, треба докладніше розглянути роботу оператора обміну даними, зокрема слід розглянути випадок передачі порожніх даних і подивитись які фактори будуть впливати на ефективність роботи.

Розглянемо наступну особливість: якщо здійснювати обмін порожніми даними, то процес обміну даними перетвориться у процес синхронізації(як це випливає із пункту (6)). Дійсно, запишемо процес обміну даними у вигляді формули із врахуванням особливостей роботи оператора $DS(a,b)$.

$$(root = 0)^*_{\alpha}\{ DS(a, root) * (root++)\} \quad (22.1)$$

$\alpha \equiv (root >= m)$, m – кількість гілок.

У свою чергу

$$DS(a, root) = (Send() * T(\lambda_i) \vee_{\beta}$$
$$\vee Accept() * T(\lambda_i)) * * (\prod_{\substack{j=1 \\ j \neq root}}^{m} S(\lambda_j)) \quad (23)$$

Введення операторів $Send()$ та $Accept()$ необхідне для врахування особливостей (3), (4) оператора $DS(a,b)$, звідки очевидно, що оператор $Send()$ виконує лише посилання даних, а $Accept()$ виконує прийом даних. Тоді відповідно умова β запишеться наступним чином

$$\beta \equiv (root = i)$$

(i-номер гілки, яка викликає оператор $DS(a,b)$

$T(\lambda_i)$ та $S(\lambda_j)$ раніше введені контрольна точка та точка синхронізації.

Для випадку передачі порожніх даних оператори $Send()$ та $Accept()$ перетворюються на тотожні оператори.

$$(E * T(\lambda_i) \vee E * T(\lambda_i)) * (\prod_{\substack{j=1 \\ j \neq root}}^{m} S(\lambda_j)) = (T(\lambda_i) \vee T(\lambda_i)) *$$
β β

$$*\Big(\prod_{\substack{j=1\\ j\neq root}}^{m} S(\lambda_j))=T(\lambda_i)\prod_{\substack{j=1\\ j\neq root}}^{m} S(\lambda_j)\Big) \quad (24)$$

Підставляючи (12.1) та (24) до (22) отримуємо

$$(root=0)*\ \{\ B_m(i)*(root++)\}=B_m(i) \quad (25)$$

Отриманий результат, а саме ліва частина формули (25), каже про те, що застосування схеми (21.1) для систем зі спільною пам'яттю дає набагато більший час виконання по зрівнянню із схемою (21), що пов'язано із викликом функції синхронізації m разів.

Формула (25), показує можливість граничного переходу схеми із роздільною пам'яттю до схеми зі спільною пам'яттю. Цей результат може бути покладений в основу побудови узагальненої параметричної схеми. Очевидно, що зворотній перехід без введення нових операторів неможливий.

Подальші узагальнення схеми (22) можна отримати, використовуючи вже описаний вище підхід для систем із спільною пам'яттю, а саме використовуючи формули (19), (20).

Формування паралельної параметричної схеми алгоритму.

Наступним кроком буде об'єднання всіх схем (27.1), (18) в паралельну параметричну схему, яка буде у граничному випадку переходити в одну із вже отриманих схем. Сформулюємо вимоги, яким повинна відповідати така схема.

1. Схема не повинна залежати від конкретної програмної реалізації.
2. Схема повинна бути справедливою для систем зі спільною пам'яттю і систем із розподіленою пам'яттю.

Для того, щоб задовольнити вимозі 2, згадаємо схему (21.1), а точніше її частину, що відповідає за передачу даних між гілками.

$$MemType()=(root=0)*\ \{\ DS(a,root)*(root++)\} \quad (26)$$

Як показує формула (25), для випадку спільної пам'яті (26) може інтерпретуватися, як синхронізація, а для випадку роздільної пам'яті (26) передає дані і синхронізує. Таким чином, позначивши (26) оператором $MemType()$, ми фактично отримуємо оператор, що параметрично визначає тип пам'яті з яким ми працюємо.

Задовольнивши, таким чином, всі вимоги, ми можемо синтезувати нашу схему

$$BF=\dot{\bigvee_{l=1}^{m}}(P\_Init(D,P,V,s)*Sync_l(\tau_1,\tau_l)*$$
$$*\ \{\ BF\_step2(W,G,D,P)\}*(root=0)*MemType()*$$
$$*PBF\_step3(W,G,D)) \quad (27)$$

При цьому оператор $Rlax(W,G,D,P,i,j)$ задається формулою (25.1).

Отримана схема є параметричною, причому в якості параметру виступає наявність чи відсутність даних, які ми пересилаємо оператором $DS(a,root)$. Тут важливо розуміти, що параметризація в нашому випадку є неявною, тому не слід її сприймати, як наявність змінної, що набуває певне значення і тим самим змінює схему. В отриманій схемі вибір параметру зводиться лише до того, яким чином ми інтерпретуємо оператор $DS(a,root)$. А вибравши потрібну інтерпретацію ми вже можемо перейти до потрібного частинного випадку.

Експериментальне дослідження.

Програмна реалізація виконувалася на мові програмування C# із використання технології MPI на основі бібліотеки MPI.NET [7].

Дослідження проводилися на базі комп'ютерного класу з 8 ПК наступної конфігурації: Intel Core 2 Quad CPU Q6600 2.4 GHz, 8 GB RAM., Gigabit Ethernet NIC, Windows Server 2008 R2 Enterprise.

Обчислювальний кластер побудовано з використанням пакету Microsoft HPC Server 2008 R2 [8].

Метою експериментальних досліджень є пошук оптимальних конфігурацій обчислювальної системи. Пошук таких конфігурацій відбувається на основі аналізу завантаженості кластеру при виконанні алгоритму БФ, синтезованого на основі отриманої схеми (27).

Наведемо залежність прискорення від кількості процесів на рис.1.

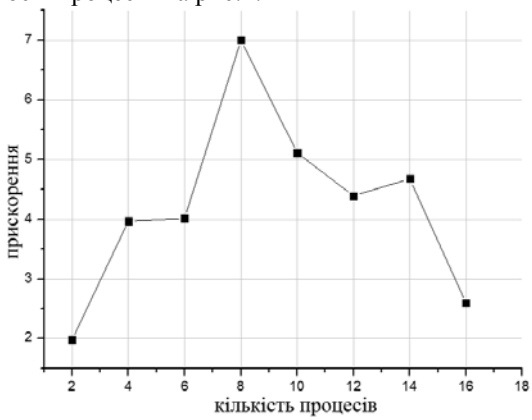

Рисунок 1 – Залежність прискорення від кількості процесів (N=5000).

Для початку дамо не великий, якісний аналіз отриманих залежностей. На графіку прискорення можна виділити наступні характерні ділянки.

1) Ділянка майже лінійного зростання прискорення(від 2 до 4 процесів).
2) Різке нелінійне зростання(від 4 до 8).

3) Повільне падіння величини прискорення (більше 8 процесів)

Поведінка залежності на першій ділянці може бути добре описана законом Амдала, оскільки розпаралелюванню підлягає приблизно 95% алгоритму.

Наявність другої ділянки пов'язана із тим, що кожен з вузлів кластера має лише чотири ядра, а додавання п'ятого процесу змушує задіяти ще один додатковий вузол, обмін даними з яким потребує вже більшого часу. Але подальше збільшення кількості процесів частково компенсує ці втрати і призводить до подальшого зростання величини прискорення.

Падіння величини прискорення на третій ділянці пояснюється постійно зростаючими витратами на обмін даними між процесами.

Перш ніж зробити певні математичні оцінки отриманих результатів, треба провести експеримент при різних комбінаціях розташування процесів на доступних ресурсах. Однак, велика кількість можливих конфігурацій робить цю задачу нерозв'язною у загальному випадку. Тому доцільно обрати декілька найбільш характерних конфігурацій та зробити порівняльний аналіз. В якості таких конфігурацій оберемо наступні.

- Всі вузли заповнюються доти, доки кількість процесів не стане рівною кількості ядер, подальші процеси розташовуються на інших вузлах. Позначимо її К1.
- На кожному вузлі запускається лише один процес. Позначимо її К2

Для таких конфігурацій знімемо залежність завантаженості кластеру. При цьому будемо користуватися наступними формулами.

$$P_i = \frac{1}{T}\sum_t \frac{P_i(t)+P_i(t+\Delta t)}{2} * \Delta t$$

$$P_A = \frac{1}{m}\sum_i P_i$$

$$P = \sum_i P_i$$

де T - час виконання алгоритму.

m - кількість процесів.

$P_i(t)$ - завантаженість i-го ядра у час t

P_A - середня завантаженість ядра

Δt - час між двома послідовними вимірами

P - завантаженість кластера.

Результат такого експерименту ми бачимо на рис. 2 та рис. 3.

Для аналізу отриманих результатів скористаємось наступною моделлю.

Загальний час виконання алгоритму складається з часу обчислення T_{calc} та часу втрат на синхронізацію, та обмін даними між задачами T_{wait}.

$$T_{total} = T_{calc} + T_{wait} \qquad (28.1)$$

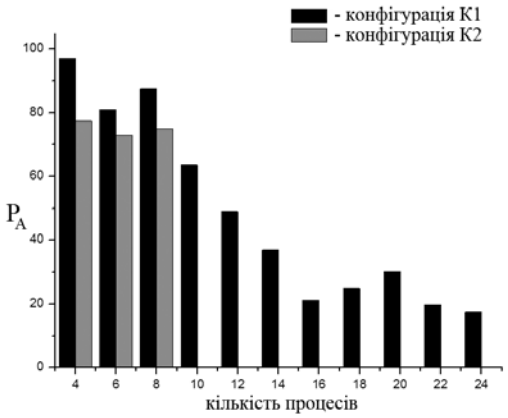

Рисунок 2 – Середня завантаженість ядра для різних конфігурацій від кількості процесів. (N=5000).

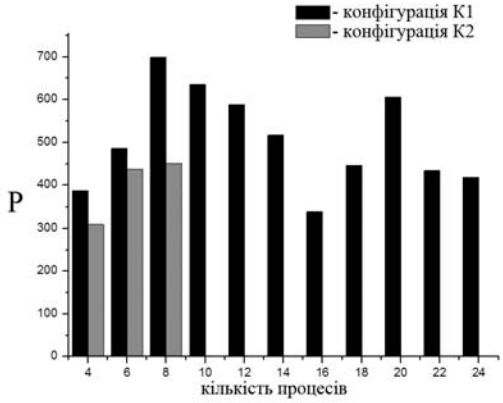

Рисунок 3 – Завантаженість кластеру для різних конфігурацій від кількості процесів(N=5000)

Для алгоритмів, що працюють з постійним об'ємом даних, величина T_{calc} може бути обрахована наступним чином.

$$T_{calc} = \frac{N_{total}}{mV_c} \qquad (28.2)$$

V_c - швидкість обробки даних біт/с

N_{total} - об'єм даних, що обробляється алгоритмом.

Величину T_{wait} можна представити наступним чином $T_{wait} = T_s + T_1 + T_2$ (28.3)

Тут

T_s - втрати безпосередньо на синхронізацію,

T_1 - постійна затримка при передачі по мережі,

T_2 - затримка безпосередньо на обмін по мережі.

Таким чином, ми повинні знайти таке значення m, при якому величина T_{calc} буде мати мінімальне значення, але все ж буде більшою за величину T_{wait}. Зважаючи на те, що з ростом m величина T_{calc} лише зменшується, а величина T_{wait} тільки збільшується, шукану величину m можна знайти із наступного рівняння.

$$T_{calc} \approx T_{wait} \qquad (28.4)$$

Усі попередні викладки були зроблені без втрати загальності, тобто отримані міркування справедливі для всіх алгоритмів, що працюють з постійним об'ємом даних протягом декількох ітерацій.

Таким чином, для застосування отриманих формул для конкретного алгоритму потрібно знати наступні параметри

1. Кількість синхронізацій за одну ітерацію
2. Кількість операцій обміну даними між процесами за одну ітерацію
3. Об'єм даних, що передається протягом однієї операції обміну даними.

Для знаходження цих параметрів для нашого алгоритму, слід звернутися до отриманих САА схем. Зокрема із формул (27) і (26) та (23) видно, що загальна кількість операцій обміну даними між процесами за одну ітерацію дорівнює $m(m-1)$. Але тут також треба врахувати, що обмін даними може відбуватися, як між процесами на одній фізичній машині так і між процесами на різних фізичних машинах. Для врахування цього факту введемо деякий коефіцієнт C, що дорівнює кількості всіх обмінів всередині фізичних машин. Тому формула для часу T_1 може бути наближено записана наступним чином:

$$T_1 = (m(m-1) - C)\Delta t_2 + C\Delta t_3 \quad (28.5)$$

У цій формулі

Δt_2 - константна затримка при передачі між фізичними машинами,

Δt_3 - константна затримка при передачі в середині фізичної машини.

Величину C можна знайти із наступного рівняння

$$C = 12*[\frac{m}{b}] + (m - b*[\frac{m}{b}])(m - b*[\frac{m}{b}] - 1) \quad (28.6)$$

де b - кількість ядер на вузлі

В рівнянні (28.6) величина $12[\frac{m}{b}]$ показує кількість обмінів всередині процесорів на яких знаходиться b процесів. Тоді відповідно кількість обмінів на процесорі із меншою кількістю процесів можна виразити наступним чином $(m - b*[\frac{m}{b}])(m - b*[\frac{m}{b}] - 1)$.

Далі з формули (26) та (22.1) можна визначити кількість операцій протягом однієї синхронізації, вона дорівнює величині m. Тоді величину T_S можна представити у наступному вигляді

$$T_S = m\Delta t_1 \quad (28.7)$$

Δt_1 - середня затримка на синхронізацію

З формул (8) та (9) можна оцінити об'єм даних, що передається за одну ітерацію. Тоді запишемо величину T_2 у наступній формі.

$$T_2 = \frac{m(m-1)16N}{mV_S} = \frac{(m-1)16N}{V_S} \quad (28.8)$$

V_S - швидкість передачі по мережі

З формул (8), (9) також видно, що об'єм даних, що обробляється алгоритмом дорівнює величині $16N^2$.

Підставляючи тепер отримані формули (28.5), (28.6), (28,7), (28.8) та (28.2) у (28.4) отримаємо наступне рівняння.

$$\frac{16N^2}{mV_c} = m\Delta t_1 + (m(m-1) - C)\Delta t_2 + C\Delta t_3 + \frac{(m-1)16N}{V_S} \quad (29)$$

Підставляючи відповідні параметри у (29) та вирішуючи його відносно величини m знайдемо оптимальну кількість процесів для потрібної конфігурації.

Тут ми бачимо переваги використання САА схем. Отримана формула (29) є дійсною лише для обраною нами інтерпретації, але користуючись параметричною схемою (27), ми можемо обрати будь-яку іншу потрібну нам інтерпретацію та побудувати аналог рівняння (29) для нової схеми.

Зрозуміло, що аналогічну до (29) формулу можна побудувати і для інших алгоритмів, потрібно лише знати три вище наведені параметри. В нашому випадку ми лише перевіряємо дійсність запропонованої моделі на прикладі обраного алгоритму.

Для чисельної оцінки скористаємося наступними значеннями(конфігурація К1)

$N = 5000$

$V_S = 10^9 \, біт/с$

$\Delta t_1 = 2.5*10^{-3} c$

$\Delta t_2 = 160*10^{-6} c$

$\Delta t_3 = 80*10^{-6} c$

$V_c \approx 1.1*10^9 \, біт/с$

Чисельне вирішення рівняння (29) для конфігурації (К1) дає величину $m \approx 9.8$. Аналізуючи залежності на рис. 4, бачимо наявність локального максимуму при m=8. Глобальний максимум і є точка, де кількість процесів найбільш оптимальна. Бачимо, що теоретичний результат близький, але не дорівнює практичному, що пов'язано із тим, що отримана формула лише наближено описує роботу алгоритму. Сукупність експериментальних даних вказує на те, що найбільша завантаженість з'являється при кількості процесів кратній кількості ядер на вузлі(в нашому випадку чотирьом), тому отримане значення слід округляти до найближчого кратного кількості ядер.

Для конфігурації (К2) формула (29) залишається вірною, але тепер $\Delta t_2 = \Delta t_3$ і є константною затримкою при передачі між фізичними машинами. Чисельне вирішення рівняння (29) для цього випадку дає величину $m \approx 9.6$. Експериментальні дані вказують на те, що максимум знаходиться при кількості процесів не меншій восьми.

Подальші дослідження цієї конфігурації обмежені апаратними ресурсами.

Слід відмітити, що похибку можуть давати величини V_S та Δt_1, значення яких можна визначити лише експериментально для конкретного алгоритму. Тому отримана формула лише вказує точку в околі якої слід експериментально шукати оптимальну конфігурацію.

Подібний підхід можна застосовувати лише для алгоритмів, що не потребують значних процесорних ресурсів на обмін даними, оскільки в протилежному випадку завантаженість буде лише збільшуватися із збільшенням кількості процесів, але задача буде вирішуватися повільніше.

Висновок

Виконано формалізацію алгоритму Белмана-Форда з використанням математичного апарата модифікованих систем алгоритмічних алгебр В.М. Глушкова. На основі отриманих схем побудовано формалізовану оцінку максимально доцільної кількості процесів задачі. Проведено низьку експериментів із використанням віртуальних кластерних архітектур, результати яких підтверджують актуальність запропонованої моделі.

Список використаної літератури

1. Білоконь І. Побудова динамічно реконфігурованої обчислювальної архітектури з використанням технологій віртуалізації / І. Білоконь, Д. Грязнов, С. Погорілий // Вісник Київського національного університету імені Тараса Шевченка. Серія: радіофізика та електроніка. – 2010. – Вип. 14. – С. 4 – 6.

2. Білоконь І. Дослідження впливу конфігурації обчислювальної системи на продуктивність програмних реалізацій паралельних алгоритмів / І. Білоконь, Д. Грязнов, С. Погорілий // Праці VII Міжнар. конф. «Електроніка та прикладна фізика», 19-22 жовтня 2011 р. Україна, Київ. – К., 2011.

3. Погорілий С.Д. До задачі оптимізації завантаженості ресурсів обчислювального кластера з вузлами у вигляді віртуальних машин / С.Д. Погорілий, І.В. Білоконь // Матеріали 8 Міжнар. науково-практичної конф. з програмування «УкрПРОГ–2012», 22-24 травня 2012 р, Україна, Київ. – К., 2012. – Проблеми програмування – № 2-3 (у друці).

4. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест и др. – 2-е изд. – М., 2005. – С. 672 - 674.

5. Дослідження паралельних схем алгоритму Данцига для обчислювальних систем зі спільною пам'яттю / С.Д. Погорілий, В.А. Мар'яновський, Ю.В. Бойко и др. // Математичні машини і системи. – 2009. – №4. – С. 27-37.

6. Формування узагальнених паралельних схем алгоритму Флойда-Уоршала / С.Д. Погорілий, В.А. Мар'яновський, Ю.В. Бойко и др. // Системні дослідження та інформаційні технології. – 2010. – №1. – С. 52 – 68.

7. MPI.NET [Електронний ресурс]. – Режим доступу: http://osl.iu.edu/research/mpi.net/

8. Microsoft HPC Server 2008 R2. Доступ до документів (22.01.2012): http://www.microsoft.com/hpc/en/us/white-papers.aspx

9. MicrosoftHyper-V [Електронний ресурс]. – Режим доступу: http://technet.microsoft.com/en-us/library/cc753637%28WS.10%29.aspx

10. Ющенко Е.Л. Многоуровневое структурное проектирование программ / Е.Л. Ющенко, Г.Е. Цейтлин, В.П. Грицай. – М.: «Финансы и статистика», 1989. – 192 с.

Надійшла до редколегії 30.04.2012

С. Д. ПОГОРЕЛЫЙ И. В. БЕЛОКОНЬ
А. В. ВАРЕНИК
Киевский национальный университет имени Тараса Шевченка

ИССЛЕДОВАНИЯ ВИРТУАЛЬНЫХ КЛАСТЕРНЫХ АРХИТЕКТУР НА ПРИМЕРЕ АЛГОРИТМА БЕЛМАНА-ФОРДА

Предложено усовершенствование метода поиска оптимальных конфигураций виртуальных кластерных архитектур, которое базируется на использовании аналитических оценок с использованием аппарата модифицированных систем алгоритмических алгебр В. М. Глушкова.

Ключевые слова: виртуализация, виртуальная машина, алгоритм Белмана-Форда, система алгоритмических алгебр, распараллеливание, HPC, MPI.

S. D. POGORILYY I. V. BILOKON O. V. VARENIK
Taras Shevchenko National University of Kyiv

THE RESEARCH OF VIRTUAL CLUSTER ARCHITECTURES FOR BELMAN-FORD ALGORITHM EXAMPLE

An improvement method of finding the optimal configuration of cluster architecture based on using analytical evaluations of algorithm behavior using mathematical means of V.M. Glushkov modified systems of algorithmic algebras was suggested.

Keywords: virtualization, virtual machine, Belman-Ford algorithm, systems of algorithmic algebras, paralleling, HPC, MPI.