УДК 004.415.5
DOI: 10.18523/2617-3808.2023.6.76-80

Сідько А. А.

НОВА ВЕРСІЯ СЕРЕДОВИЩА ВИКОНАННЯ MATHPAR-DAP

У цій статті згадано основні особливості середовища виконання децентралізованого управління розподіленими обчисленнями DAP (Drop-Pine-Amine), які було опубліковано в [4]. Головною метою цієї статті є опис нових функціональних можливостей, які з'явилися в останньому випуску. Як приклад алгоритму з блоковою рекурсією описано факторизацію Холецького симетричної позитивно означеної матриці у вигляді блокового дихотомічного алгоритму. Результати експериментів демонструють гарну масштабованість запропонованого рішення. Запропоновано розвивати співпрацю у цій науковій сфері. Розроблений програмний пакет відкритий для спільного розроблення, його можна вільно використовувати для наукових і освітніх цілей.

Ключові слова: розподілені обчислення, паралельне програмування, середовище виконання, OpenMPI.

Вступ

Сучасні суперкомп'ютерні системи, що містять сотні тисяч ядер, стикаються з труднощами в організації паралельних обчислень (див. [1]). Основними проблемами є нерівномірне навантаження на апаратні засоби, накопичення помилок під час обчислень із великими матрицями та можливі відмови ядер під час процесу обчислень.

Нещодавно було розроблено універсальну схему виявлення динамічних задач (DTD) для середовища виконання PaRSEC [2; 3]. Це середовище може підтримувати системи з розділеною та спільною пам'яттю. Цей новий парадигм показав кращу продуктивність порівняно з параметризованим плануванням задач, яке використовувалося раніше.

У [4] описано нове середовище виконання для суперкомп'ютерів із розділеною пам'яттю. Воно призначене для вирішення матричних проблем за допомогою алгоритмів із блоковою рекурсією. Його головною перевагою є забезпечення ефективного обчислювального процесу та гарної масштабованості програм як для розріджених, так і для щільних матриць на кластері з розділеною пам'яттю. Ще однією перевагою є можливість реорганізації обчислювального процесу у випадку відмови окремих вузлів під час обчислень.

Приклад: факторизація Холецького

Дано симетричну позитивно означену матрицю A. Потрібно знайти нижню трикутну матрицю L, таку що $A = LL^T$.

Запропоновано дихотомічний блоковий рекурсивний алгоритм для факторизації Холецького. Обчислювальна складність всього алгоритму така сама, як у алгоритму множення матриць, який буде використовуватися для блокового множення.

Позначимо блоки матриці як α, β, β^T, γ а блоки матриці L, які потрібно знайти, позначимо як *a, 0, b, c*; і прирівняємо добуток LL^T до A:

$$L \times L^T = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \times \begin{pmatrix} a^T & b^T \\ 0 & c^T \end{pmatrix} =$$

$$= \begin{pmatrix} aa^T & ab^T \\ ba^T & bb^T + cc^T \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \beta^T & \gamma \end{pmatrix} = A.$$

Із цього випливає рівність для блоків:

$$aa^T = \alpha, b = \beta^T (a^{-1})^T, cc^T = \gamma - bb^T.$$

Для обчислення блоків *a*, *b* та *c* потрібно виконати блокове множення, транспонування, обернення та факторизацію Холецького для двох блоків.

Обчислення оберненої матриці можна уникнути, якщо обчислювати обернену матрицю L разом із матрицею L^{-1}.

Algorithm 1 Standard blocked Cholesky factorization

1: **for** each panel left to right **do**
2: Partition $A = \begin{bmatrix} A_{11} & \\ A_{21} & A_{22} \end{bmatrix}$, where A_{11} is NB×NB
3: Factorize $A_{11} = LL^T$ using unblocked algorithm
4: Update panel $A_{21} := A_{21} L^{-T}$ using triangular solver
5: Update trailing matrix $A_{22} := A_{22} - A_{21} A_{21}^T$ using symmetric rank-k update
6: Continue with $A = A_{22}$
7: **end for**

Процедуру факторизації Холецького розширено таким чином, щоб вона повертала не тільки матрицю L, а й її обернену матрицю L^{-1}. Потім,

разом з блоками a та b, обернені блоки a^{-1} та b^{-1}. У цьому випадку обернена матриця до L матиме вигляд:

$$L^{-1} = \begin{pmatrix} a^{-1} & 0 \\ -c^{-1}ba^{-1} & c^{-1} \end{pmatrix}.$$

Граф цього рекурсивного алгоритму показано на рис. 1.

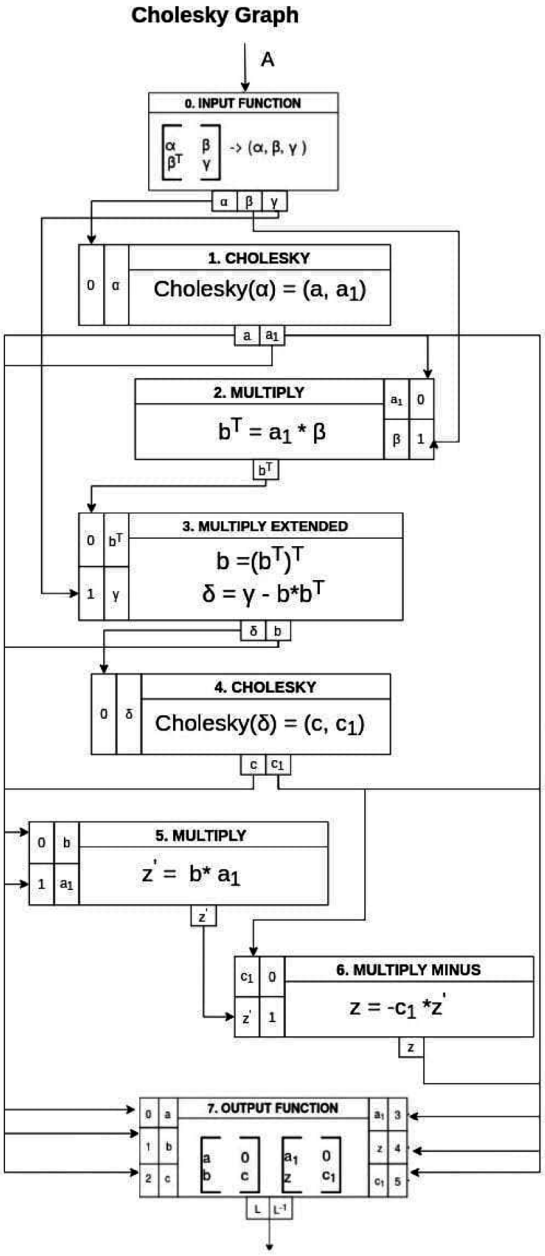

Рис. 1. Граф факторизації Холецького

На рисунку 2 наведено механізм управління обчислювальним процесом:

– Drop — це блоково-рекурсивна функція разом зі вхідними даними, яку можна передати на будь-який процесор;
– Amine — це набір усіх дропів першого рівня рекурсії деякої блоково-рекурсивної функції та набір їхніх зв'язків. Щоб обчислити Drop, потрібно створити для нього Amine;
– Pine — список усіх Amines, які було створено на цьому процесорі;
– Aerodrome — список усіх дропів, які було передано на цей процесор;
– Terminal — список дропів, які було відправлено на інші процесори.

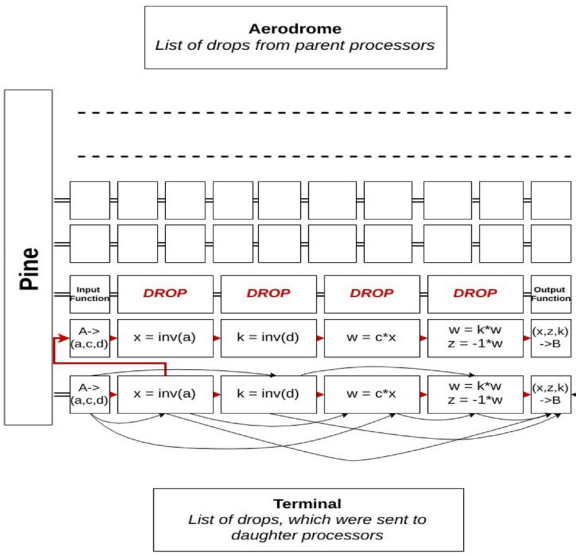

Рис. 2. Механізм управління обчислювальним процесом

Експерименти. Алгоритм факторизації Холецького

У таблиці 1 подано результати трьох серій експериментів. У кожній серії розмір листового блоку (LS), тобто розмір блоку, який обчислювався на одному процесорі, був постійним. У першій серії він становив 64, у другій — 128, а в третій серії — 256.

У кожній серії підтримувалося постійне обчислювальне навантаження на одне ядро: коли розмір матриці подвоювався, кількість ядер збільшувалась у вісім разів. Отже, кількість ядер була 1, 8, 64 і 512, а розмір матриці становив 256, 512, 1024 і 2048. Використовувалися числа BigDecimal зі 100 десятковими цифрами у дробовій частині.

Експерименти показали, що найкращим листовим блоком для кластеру був 128.

З другого рядка табл. 1 видно, що коли кількість ядер збільшується у вісім разів, час обчислень збільшується у 364/121=3.0, 896/364=2.4 та 1996/896=2.2 рази відповідно. Отже, відповідний TLR становить $[\sqrt[3]{3.0}, \sqrt[3]{2.4}, \sqrt[3]{2.2}] = [1.44, 1.34, 1.3]$.

Ці дані вказують на гарну масштабованість запропонованого програмного рішення.

Обчислювальні експерименти проводили на кластері MVS-10P CASCADE LAKE (2256 ядер, Intel Xeon Platinum 8268, 28 ядер / 48 потоків, 2.9 ГГц, 35.75 МБ кеш-пам'яті, 96 ГБ ОЗП на процесор).

Подолання проблеми синхронізації повідомлень із блокуванням

Під час відлагодження системи було виявлено проблему синхронізації повідомлень. Спочатку всі пересилання були неблокуючими. Однак за пересилання великих обсягів даних було виявлено переповнення буферів і руйнування масивів. Тому було прийнято рішення використовувати блокувальне пересилання для цих повідомлень.

Однак за таких умов постала проблема «мертвого замка». Почали виникати ситуації, коли декілька процесорів входили в команду відправлення повідомлень, але при цьому перший відправляв другому, другий — третьому і так далі, а останній відправляв першому. Очевидно, що програма зависала.

Для подолання цієї проблеми було введено певні правила. Усі повідомлення, крім тих, що відправляють великі об'єкти даних, можна відправляти в режимі iSend (неблокувальний режим). Для відправлення великих об'єктів даних необхідно використовувати блокувальні повідомлення. Однак для цього потрібно забезпечити, щоб повідомлення в блокувальному режимі не можна було відправити, якщо є ризик, що під час їхнього передавання є можливість отримати чиєсь повідомлення в блокувальному режимі. Тому необхідно заборонити відправлення блокувальних повідомлень, коли очікується прийом блокувального повідомлення. Для цього достатньо відокремити час відправлення та прийому блокувальних повідомлень.

Для розділення цих двох подій (відправлення та приймання блокувального повідомлення) введено булеву змінну recv, яка має такі стани:

– False (за замовчуванням) — дозволено відправлення блокувальних повідомлень, але заборонено їх приймання;
– True — очікування на приймання блокувальних повідомлень, заборонено відправлення блокувальних повідомлень.

Стан змінної recv має бути синхронізовано на всіх процесорах. Для цього введено нові структури даних. Виняток становить кореневий процесор, який може відправляти блокувальні повідомлення без запиту. Також, якщо номер процесора парний, він може відправляти повідомлення без запиту на вільний процесор із непарним номером. У всіх інших випадках потрібен запит.

Ці винятки введено для прискорення виконання програми, уникаючи зависання.

Усі операції пересилання поділяють на блокувальні та неблокувальні. Неблокувальне пересилання (iSend) передбачає відправлення стану процесора, відправлення списку вільних процесорів, відправлення запиту на відправку блокувального повідомлення, відправлення підтвердження прийому блокувального повідомлення та відправлення відмови на відправку блокувального запиту. Блокувальне пересилання (send) — відправлення завдання видалення на вільні процесори та відправлення результату на батьківські процесори.

Для синхронізації режиму приймання-відправлення було введено додаткові команди для запиту дозволу на відправлення блокувального повідомлення та повідомлення про те, що процесор готовий прийняти блокувальне повідомлення. Для керування цими повідомленнями створено три списки:

– waitingFromOthers — список процесорів, які запитали дозвіл на відправлення блокувального повідомлення;
– waitingOutput — список процесорів, яким уже було відправлено запити на дозвіл відправити блокувальне повідомлення;
– approvedOutput — список процесорів, які вже дали згоду на відправлення блокувального повідомлення.

Стани змінної recv змінюються в таких випадках:

– recv = false: після прийому одного блокувального повідомлення — не дозволено приймати;

Таблиця 1

Масштабованість алгоритму Холецького для чисел BigDecimal зі 100 десятковими цифрами (час у секундах)

#core / matrix size	1/256	TLR	8/512	TLR	64/1024	TLR	512/2048
Time for leaf size 64	1.22 min	1.14	1.8 min	2.4	24.84 min	1.05	28.92 min
Time for leaf size 128	1.21 min	1.44	3.64 min	1.35	8.96 min	1.31	19.96 min
Time for leaf size 256	1.15 min	1.96	8.72 min	1.21	15.5 min	1.16	23.96 min

– recv = true: після відправлення підтвердження на запит на відправку блокувального повідомлення — заборонено відправляти.

Висновки

У статті було описано середовище виконання DAP (drop-amine-pine). Основною особливістю DAP є його здатність послідовно розгортати функції в глибину, підтримуючи всі стани на будь-якому рівні вкладеності до завершення всіх обчислень у поточному обчислювальному піддереві. Такий підхід дає змогу будь-якому процесору вільно перемикатися між підзадачами без очікування завершення поточної підзадачі.

Важливою особливістю цього середовища виконання є захист від відмов деяких вузлів під час обчислень. Батьківський вузол, який надіслав дроп (drop) своєму дочірньому вузлу, повинен отримати результат. Однак, замість результату, він може отримати повідомлення про статус дочірнього вузла. У таких випадках завдання цього дропа перенаправляється на альтернативний вузол. Додаткові зміни в інших вузлах не потрібні. В результаті буде втрачено та в подальшому перераховано лише піддерево, що відповідає цьому дропу.

Це середовище виконання було реалізоване на мові програмування Java і використовує інтерфейс для комунікації між процесорами OpenMPI. Вихідний код програми доступний за URL-адресою: https://bitbucket.org/mathpar/dap01/src/master/src/main/java/com/mathpar/parallel/dap/.

Список літератури

1. Dongarra J. With Extreme Scale Computing the Rules Have Changed. Mathematical Software / J. Dongarra // ICMS 2016, 5th International Congress, Procdistributed memoryeedings (G.-M. Greuel, T. Koch, P. Paule, A. Sommese, eds.), Springer. — 2016. – No. 9725. — Pp. 3–6.
2. Dynamic Task Discovery in PaRSEC- A data-flow task-based Runtime / R. Hoque, T. Herault, G. Bosilca, J. Dongarra // Proc. ScalA17, Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, November 12–17, 2017, Denver, CO, USA. — 2017.
3. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability [Electronic resource] / [G. Bosilca, A. Bouteiller, A. Danalis et al.] // Computing in Science and Engineering 99 (2013). — 2013. — https://doi.org/10.1109/MCSE.2013.98.
4. Supercomputer Environment for Recursive Matrix Algorithms [Electronic resource] / G. I. Malaschonok, A. A. Sidko // Program Comput Soft. — 2022. — Vol. 48. — Pp. 90–101. – 2022. — https://doi.org/10.1134/S0361768822020086.

References

Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Hérault, T., & Dongarra, J. (2013). PARSEC: Exploiting Heterogeneity to enhance scalability. *Computing in Science and Engineering*, *15* (6), 36–45. https://doi.org/10.1109/mcse.2013.98.

Dongarra, Jack. (2016). With Extreme Scale Computing the Rules Have Changed. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '16)*. Association for Computing Machinery, New York, NY, USA, 123. https://doi.org/10.1145/2907294.2926972.

Hoque, Reazul, Herault Thomas, Bosilca George, & Dongarra, Jack. (2017). Dynamic task discovery in PaRSEC: a data-flow task-based runtime. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA '17)* (Article 6, pp. 1–8). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3148226.3148233.

Malaschonok, G., & Sidko, A. (2022). Supercomputer environment for recursive matrix algorithms. *Programming and Computer Software*, *48* (2), 90–101. https://doi.org/10.1134/s0361768822020086.

A. Sidko

WHAT IS NEW IN THE LATEST RELEASE OF MATHPAR-DAP RUNTIME

In this paper, we recall the main features of the DAP runtime, that was published in [4]. But the main purpose of this paper is to describe the new functionality that appeared in our latest release. As an example of a block recursive algorithm, the Cholesky factorization of a symmetric positive definite matrix in the form of a block dichotomous algorithm is described. The results of experiments demonstrate good scalability of the proposed solution. Modern supercomputer systems containing hundreds of thousands of cores face difficulties in the organization of parallel computations (e.g., see [1]). The three main difficulties are the nonuniform hardware workload, accumulation of errors in the process of computations with large matrices, and possible failures of cores during the computation process.

Recently, a universal Dynamic Task Discovery (DTD) scheme for the PaRSEC runtime environment [2], [3] has been developed. This environment can support systems with shared and distributed memory. This new paradigm demonstrated better performance compared with the parameterized task scheduling that was used earlier.

In [1] we described a new runtime environment for supercomputers with distributed memory. It is designed for solving matrix problems using block recursive algorithms.

Its main advantage is to provide an efficient computational process and good scalability of programs both for sparse and dense matrices on a cluster with distributed memory. Another advantage is the ability to reorganize the computational process in the event of failure of individual nodes during computations.

A key feature of DAP is its ability to sequentially unroll functions in depth, maintaining all states at any nesting level until all computations in the current computational subtree are complete. This design allows any processor to switch freely between subtasks without waiting for the completion of the current subtask.

An important feature of this runtime environment is protection against failures of some nodes during computations. The parent node that sent a drop to its child node must receive a result. However, instead of a result, it may receive a message regarding the status of the child node. In such cases, the drop task is redirected to an alternate node. No additional changes to the other nodes are required. As a result, only the subtree corresponding to this drop will be lost and subsequently recalculated.

We would like to develop cooperation in this scientific area. The software package developed by us is open for joint development, and can be freely used for scientific and educational purposes.

Keywords: distributed computing, parallel programming, OpenMPI, runtime.

Матеріал надійшов 26.11.2023