УДК 519.688:519.6:514.752

А.В. КАЛЮЖНЯК
Запорізький національний університет

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОБЧИСЛЕНЬ У РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ ЗА ДОПОМОГОЮ БІБЛІОТЕКИ MPI

Розвиток обчислювальної техніки, яка характеризується збільшенням кількості даних і масивним різноманітним паралелізмом, висуває перед розробниками нові виклики. Традиційно, розроблення масштабованих програм виконується на обчислювальному ядрі, ігноруючи продуктивність моделювання. Для додатків із меншим виходом, ніж зараз, учені можуть архівувати результати моделювання для подальшої інтерпретації. Однак для програм екстремального масштабу вихідні дані часто містять занадто багато даних для зберігання в основній пам'яті або обмежені шириною смуги вводу-виводу. Отже, зараз існує потреба в розробленні масштабованих додатків, які включають моделювання, імітацію, аналіз і візуалізацію.

Метою дослідження є розроблення паралельного розподіленого методу моделювання геометричних об'єктів із використанням функціонального підходу бібліотеки MPI. У статті подано опис роботи алгоритму «маршируючих кубів» у розподіленій системі, проаналізовано властивості і практичне застосування під час побудови об'єктів із використанням паралельного програмування бібліотеки MPI й OpenMP. Проаналізовано розроблення ефективного паралельного програмного компонента, який, окрім прямого рендерингу, дає змогу ефективно зберігати, будувати геометричні моделі й може водночас використовувати кілька пристроїв. Подано приклади побудови об'єктів у середовищі Qt Creator.

Ці результати будуть корисними для теоретичних і практичних досліджень візуального представлення моделей із розподіленою пам'яттю. Моделі, побудовані за допомогою вдосконаленого алгоритму «маршируючих кубів», дають можливість розв'язати деякі завдання моделювання без великих витрат часу і прийняти належні рішення стосовно побудови об'єктів.

Отже, побудова тривимірних об'єктів за допомогою функціонального підходу може бути більш ефективною завдяки використанню бібліотеки розподіленого підходу MPI.

Ключові слова: «маршируючі куби», розподілена пам'ять, MPI, R-функції.

A.V. KALIUZHNIAK
Zaporizhzhia National University

COMPUTATION EFFICIENCY ENHANCEMENT IN DISTRIBUTED COMPUTER SYSTEMS WITH THE MPI LIBRARY ASSISTANCE

The development of computing technology, characterized by an increase in the amount of data and massive and diverse parallelism, poses new challenges to developers. Traditionally, the development of scalable applications has been carried out on the computer core, ignoring simulation performance.

For applications with lower output than currently available, scientists can archive simulation results for later interpretation. However, for applications of extreme scale, the data output often contains too much data to be stored in the main memory or is limited by the I/O bandwidth. Hence, there is a current necessity to develop scalable applications that include modeling, simulation, analysis, and visualization.

The objective of the study is to develop a parallel-distributed method of modeling geometric objects using the functional approach of the MPI library. This article is focused on the description of the operation of the "marching cubes" algorithm in a distributed system, the analysis of its properties and practical application in the construction of objects using parallel programming of the MPI and Open MP libraries.

The development of an effective parallel software component is analyzed, which, in addition to direct rendering, enables efficient storage and construction of geometric models and is able to use several devices simultaneously. The examples of building objects in the Qt Creator environment are also presented.

The results will beneficial for theoretical and practical research on the visual representation of models with distributed memory.

Models built by means of the improved "marching cubes" algorithm enables to solve some modeling problems in a time-consuming way and make appropriate decisions regarding the object construction.

Therefore, 3D objects building based on a functional approach can be more efficient by using the MPI distributed approach library.

Key words: "marching cubes", distributed memory, MPI, R-function.

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

Постановка проблеми

Розбиття на ізоповерхні є фундаментальною операцією для багатьох наукових досліджень. Наприклад, ізоповерхні дають змогу перевіряти особливості тканин і форми органів у медичному аналізі, форму і взаємодію між молекулами в програмах біоінформатики й місцеві опади за результатами вимірювань погодних радіолокаторів.

Ізоповерхні скалярних полів, визначених над кубічними сітками, є важливими в широкому діапазоні застосувань, таких як медична візуалізація, геофізична зйомка, фізика й обчислювальна геометрія. Основна проблема полягає в тому, що кількість елементів зростає стосовно щільності вибірки, а величезні обсяги даних висувають жорсткі вимоги до потужності обробки та пропускної здатності пам'яті. Це особливо справедливо для програм, які потребують інтерактивної візуалізації скалярних полів.

Збільшення гетерогенних, масово паралельних архітектур ускладнило досягнення масштабованості й портативності. Наукові обчислювальні програми традиційно покладаються на бібліотеки OpenMP і MPI.

Але вимагати від розробників програмного забезпечення явно керувати ієрархіями пам'яті й обмежувати переміщення даних стає все більш обтяжливим і менш переносним. Поширення нових інструментів, доступних розробнику, бібліотек дали можливість ефективно писати продуктивні додатки. Навіть із новими інструментами деякі алгоритми можуть не масштабуватися в гетерогенних середовищах або різних паралельних парадигмах. Розуміння впливу вибору інструменту на продуктивність у реальних програмах до реалізації є необхідним для розроблення ефективних програм [1].

Аналіз останніх досліджень та публікацій

Сучасний розвиток дав змогу провести багато досліджень щодо обробки об'ємних даних на графічних процесорах (GPU), оскільки графічні процесори спеціально розроблені для виконання великих обчислювальних завдань із високими вимогами до пропускної здатності пам'яті, будуючись на простому й масовому паралелізмі замість CPU. Об'ємна трансляція – це одна з технік візуалізації скалярних полів, яка успішно реалізована на графічних процесорах.

Для заощадження часу та зменшення витрат реалізовано функціональний підхід геометричного моделювання в розподілених комп'ютерних системах із використанням MPI й алгоритму «маршируючих кубів». Задля кращого вивчення продуктивності візуалізації в численних робочих процесах і системах використано ОС Linux.

Існує багато альтернативних алгоритмів, таких як метод «Скелі», метод «Канейро» та «MT6», але під час вирішення прикладних завдань, пов'язаних із візуалізацією геометричних об'єктів, краще за все використовувати алгоритм «маршируючих кубів», так як важлива топологічна точність одержуваної поверхні.

Мета дослідження

Метою дослідження є розроблення паралельного розподіленого методу моделювання геометричних об'єктів із використанням функціонального підходу бібліотеки MPI.

Виклад основного матеріалу дослідження

1. «Маршируючі куби». Тривимірна побудова поверхонь зображень – важливе завдання, яке важко реалізувати на практиці. З усіх методів вилучення поверхонь зі скалярних полів алгоритм «маршируючих кубів» найбільш популярний завдяки його простоті. У статті розглядається застосування розподілених обчислень за допомогою бібліотеки MPI й алгоритму «маршируючих кубів». Розроблено надійний та ефективний алгоритм для побудови поверхонь зображень за допомогою функціонального підходу.

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

«Маршируючі куби» є найбільш широко використовуваною технологією для виділення ізоповерхні з тривимірних скалярних полів, які також називають об'ємними наборами даних. Об'ємний набір даних – це тривимірний масив кубічних елементів, комірок зі значеннями, пов'язаними з вісьма кутами. Ізоповерхня – це поверхня, на якій усі точки мають значення, що дорівнюють заданому користувачем пороговому значенню [2].

Комірка перетинається ізоповерхнею, якщо вона містить обидва значення вище та нижче за рівнозначну. Алгоритм «маршируючих кубів» витягує ізоповерхню по частинах, обробляючи комірки одну за одною.

Алгоритм припускає, що вихідні дані є дискретним тривимірним регулярним полем даних, визначеним на даних. Коли куб перетинає поверхню, алгоритм спочатку обчислює пересічні вершини поверхні й ребер куба за допомогою техніки лінійної інтерполяції, а потім з'єднує вершини, щоб утворити трикутники, які є апроксимацією поверхні в кубі. Після обходу всіх кубів виділяється повна поверхня.

Алгоритм обробляє кожну комірку незалежно, у результаті одна й та ж вершина може бути обчислена до чотирьох разів у сусідніх комірках. Обчислення повторюваних вершин є серйозним недоліком, оскільки дублювання даних збільшує розмір ізоповерхні без додавання будь-якої корисної інформації. Дублювання може спричинити значні проблеми з продуктивністю візуалізації ізоповерхні й наступних етапів обробки [3].

Тому запропоновано рішення для ефективного розв'язання цієї проблеми за допомогою використання кількох допоміжних структур даних. Координати вершини обчислюються лише під час першого розгляду відповідного активного ребра. Координати вставляються в таблицю вершин, а індекс, що відповідає положенню вершини в таблиці, зберігається в правильній позиції однієї з п'яти структур даних допоміжного масиву (рис. 1).

У загальній комірці (тобто комірці, яка не знаходиться на межі об'єму) попередньо розглянуто дев'ять ребер. Таким чином, загальна активна комірка може створити не більше ніж три нові вершини. Значення в допоміжних структурах даних оновлюються під час подальшої обробки всіх комірок.

```
void MarchingCubesOriginal(int ncellsX, int ncellsY, int ncellsZ, float minValue, QVector<QVector4D>
 &points, Intersection intersection, QVector<Triangle> &triangles)
{
    printf("Original\n");
    int numTriangles = 0;
    int YtimeZ = (ncellsY + 1) * (ncellsZ + 1);

    triangles.resize(3 * ncellsX * ncellsY * ncellsZ);
    for(int i=0; i < ncellsX; i++)          //x axis
        for(int j=0; j < ncellsY; j++)      //y axis
            for(int k=0; k < ncellsZ; k++)  //z axis
            {
                //initialize vertices
                QVector4D verts[8];
                int ind = i*YtimeZ + j*(ncellsZ+1) + k;
                /*(step 3)*/ verts[0] = points[ind];
                verts[1] = points[ind + YtimeZ];
                verts[2] = points[ind + YtimeZ + 1];
                verts[3] = points[ind + 1];
                verts[4] = points[ind + (ncellsZ+1)];
                verts[5] = points[ind + YtimeZ + (ncellsZ+1)];
                verts[6] = points[ind + YtimeZ + (ncellsZ+1) + 1];
                verts[7] = points[ind + (ncellsZ+1) + 1];
```

Рис. 1. Реалізація алгоритму

Паралельна обробка є відповідним підходом для вирішення проблеми збільшення розміру даних, і пристрої, оснащені графічними процесорами (GPU), є відповідною платформою для виконання аналізу даних паралельно над структурованими даними як об'ємними наборами даних. Окрім продуктивності, бажані переваги.

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

Особливостями паралельної реалізації алгоритму «маршируючих кубів» для GPU є масштабованість у вхідних наборах даних, яка може оброблюватися, мати високу якість отриманих ізоповерхонь і можливість підключати програмне забезпечення до різних візуалізацій.

2. Реалізація розподіленого підходу. Як стандарт MPI широко прийнятий у високопродуктивних обчислювальних системах, оскільки він забезпечує незалежну від мови платформу. Він забезпечує зв'язки з мовами C, C++ і Fortran, що робить його можливим інструментом у практиці програмування. MPI має такі переваги, як хороша мобільність і висока ефективність, і може використовуватися в гетерогенних середовищах. Існує багато різних безкоштовних, ефективних і практичних версій MPI [3].

У паралельному програмуванні існує шість найбільш часто використовуваних базових інтерфейсів MPI. Розроблення програм у розподіленій системі ускладнюється через проблему ресурсів (кількість вузлів, їх архітектура, ефективність), визначаються вже в момент оброблення мережею виконання завдання. Також, незважаючи на високий рівень продуктивності бібліотеки MPI, сама технологія має недоліки (складність написання програм, необхідність надмірної специфікації типів даних у переданих повідомленнях) [4].

Паралельна реалізація «маршируючих кубів» розроблена шляхом поділу 3D-скалярного зображення на прямокутні секції. Вхідні параметри керують розміром кожного розділу вздовж *x*, *y* і *z*. Потім кожен розділ обробляється окремо. Кожен потік MPI заповнює локальні вихідні змінні точки, нормалі й трикутники. Точки й нормалі містять вектор сітки. Трикутники містять вектор із трійки індексів, де індекси стосуються значень у точках і нормалях. Кожен потік обробляє розділи, призначені йому MPI (рис. 2).

```
std::vector<TriangleForMPI> triangle_mpi;
int YtimeZ = (ncellsY + 1) * (ncellsZ + 1);
int total_size = ncellsX * ncellsZ * ncellsY;
QVector4D verts[8];
QVector3D intVerts[12];

if(id == MASTER_ID)
{ int local_size = total_size / nb_procs;
int start_l = local_size * id;
int numTriangles = 0;
for(int l=start_l; l < start_l + local_size; l++)
{ //initialize vertices
    //get the index
    int cubeIndex = int(0);
    for(int n=0; n < 8; n++)
       if(verts[n].w() <= minValue) cubeIndex |= (1 << n);
    //check if its completely inside or outside
    //get intersection vertices on edges and save into the array
    if(edgeTable[cubeIndex] & 1) intVerts[0] = intersection(verts[0], verts[1], minValue);
    //now build the triangles using triTable
    for (int n = 0; triTable[cubeIndex][n] != -1; n+=3) {
        Triangle tmp;
        tmp.p[0] = intVerts[triTable[cubeIndex][n+2]];
        tmp.p[1] = intVerts[triTable[cubeIndex][n+1]];
        tmp.p[2] = intVerts[triTable[cubeIndex][n]];
        tmp.norm = QVector3D::crossProduct(tmp.p[1] - tmp.p[0], tmp.p[2] - tmp.p[0]);
        tmp.norm.normalize();
        TriangleForMPI local_triangle_mpi;
        local_triangle_mpi.p1_x = tmp.p[0].x();
        local_triangle_mpi.p1_y = tmp.p[0].y();
        local_triangle_mpi.p1_z = tmp.p[0].z();
        numTriangles++; }}
MPI_Request request;
MPI_Isend(&numTriangles, 1, MPI_INT, MASTER_ID, 0, MPI_COMM_WORLD, &request);
MPI_Isend(triangle_mpi.data(), numTriangles, ptype, MASTER_ID, 1, MPI_COMM_WORLD, &request);
if(id == MASTER_ID)
```

Рис. 2. Реалізація розподіленого підходу алгоритму «маршируючих кубів»

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

Після того як усі розділи оброблено, локальний вихід потрібно об'єднати з глобальним. Для зручності обрано простий, але ефективний спосіб. Коли кожен потік завершує заповнення локального виводу, цей вивід додається до глобальних точок, нормалей і трикутників, де дозволено додавати до глобального виводу лише по одному потоку за раз (рис. 3).

```
    std::vector<TriangleForMPI> global_triangle_mpi;
int global_numtriangle = 0;
global_triangle_mpi.resize(100000);
for(int i = 0; i < nb_procs; i++)
{  int numtriangle = 0;
    MPI_Status status;
    MPI_Recv(&numtriangle, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status);
    global_numtriangle += numtriangle;
   TriangleForMPI* buffer;
      Triangle tmp;
      tmp.p[0].setX(buffer[s].p1_x);  }   }
global_triangle_mpi.clear();
global_triangle_mpi.shrink_to_fit();}
MPI_Type_free(&ptype);
MPI_Finalize();
```

Рис. 3. Тріангуляція розподіленого підходу за допомогою MPI

Цей алгоритм не розроблений для явної мети паралелізму, у ньому є кілька недоліків, які перешкоджають масштабованості.

В алгоритмі кінцевий розмір не визначено заздалегідь, тому вихідні дані, сформовані паралельно, мають бути об'єднані, що спричиняє паралельне вузьке місце та створює повторювані точки. Крім того, вихідні масиви не можуть бути правильно розподілені, тому знадобився динамічний розподіл пам'яті.

Проаналізувавши прискорення роботи алгоритму «маршируючих кубів» за допомогою MPI з різними потоками на кожному вузлі видно, що лінія показує прискорення за часом у секундах (рис. 4).

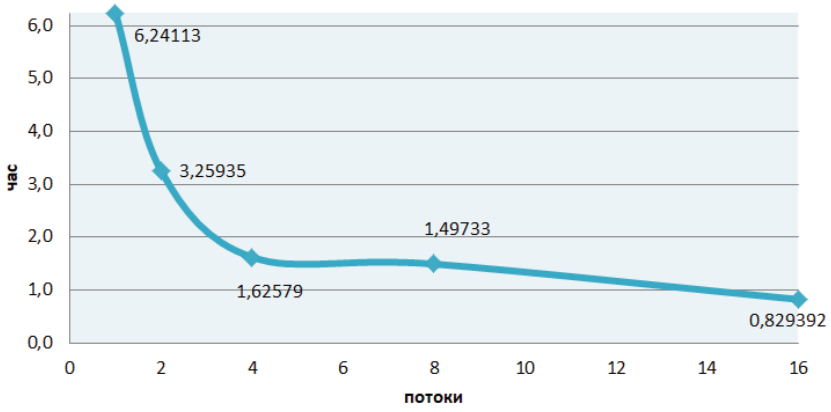

Рис. 4. Порівняння роботи MPI зі зміною в потоках

Очевидно, що код MPI масштабується краще, в обчислювальному кластері на ОС Linux не нижче за версії 20.0 і з використанням однакових параметрів серверу, ніж у кластері, де продуктивність обчислювальних машин різна. Тести MPI проводилися на декількох вузлах. Оскільки кожен код працює над окремими розділами загального зображення паралельно однаково, очікується, що результати продуктивності будуть подібними. Для проведення дослідження використовувався обчислювальний кластер, який складався з двох серверів Intel (R) Core(TM) i7–4770 CPU @ 3.40GHz 8 ядер і візуалізація геометричної моделі з розбиттям 240 (рис. 5).

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

Рис. 5. Побудова геометричної моделі в середовищі QT Creator

Після читання даних зображення код MPI копіює дані розділу зображення в кожен процесор. Це швидка операція на одному обчислювальному вузлі, яка пізніше може мати додаткові переваги.

Висновки

Отже, представлено паралельний алгоритм побудови геометричних об'єктів за допомогою функціонального підходу в розподіленій системі, ефективність якого перевершує класичний алгоритм у часі. Скалярне поле розбивається на кілька частин, застосовується для кожної частини в різних процесах за допомогою MPI. Нитки виконання під час роботи MPI – довготривалі процеси (на відміну від динамічно породжених потоків), є симетричними за своїми можливостями й виконанням. Недостатнє усвідомлення симетрії процесів MPI може спричинити до неефективних і немасштабованих рішень. Експеримент показує, що на ефективність також впливає обрана архітектура та продуктивність комп'ютерної системи.

Список використаної літератури

1. Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation / A. Alexandrov et al. USA : Tech. Rep, 1995. P. 206.
2. Flexible collective communication tuning architecture applied to Open MPI, PVM/MPI / G.E. Fagg et al. USA : Manning Publications, 2006. P. 14.
3. Constructing a prior-dependent graph for data clustering and dimension reduction in the edge of AIoT, Future Gener / T. Guo et al. Comput. Syst. 128. Kobe, 2022. P. 381.
4. Predicting MPI collective communication performance using machine learning, in: 2020 IEEE International Conference on Cluster Computing (CLUSTER) / S. Hunold et al. Kobe, 2020. P. 259.
5. On construction of sensors, edge, and cloud (iSEC) framework for smart system integration and applications / E. Kristiani et al. *IEEE Int. Things J*. 2020. № 8(1). P. 309.
6. Rico-Gallego J., Lastovetsky A.L., Martín J.C.D. Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters. *IEEE Trans. Parallel Distrib. Syst*. 2017. № 28(11). P. 217.

References

1. Alexandrov, A., Ionescu, M.F., Schauser, K. E. & Scheiman, C. (1995) Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation, USA: Tech. Rep.

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7

2. Fagg, G.E., Pjesivac-Grbovic, J., Bosilca, G., Angskun, T., Dongarra, J. & Jeannot, E. (2006) Flexible collective communication tuning architecture applied to Open MPI, PVM/MPI. USA : Manning Publications.
3. Guo, T., Yu, K., Aloqaily, M. & Wan, S. (2022) Constructing a prior-dependent graph for data clustering and dimension reduction in the edge of AIoT, Future Gener. Comput. Syst. 128, Kobe.
4. Hunold, S., Bhatele, A., Bosilca, G. & Knees, P. (2020) Predicting MPI collective communication performance using machine learning, in: 2020 IEEE International Conference on Cluster Computing (CLUSTER), Kobe.
5. Kristiani, E., Yang, C.-T., Huang, C.-Y., Ko, P.-C. & Fathoni, H. (2020) On construction of sensors, edge, and cloud (iSEC) framework for smart system integration and applications, *IEEE Int. Things J*. Vol. 8(1). USA.
6. Rico-Gallego, J., Lastovetsky, A.L. & Martín, J.C.D. (2017) Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters. *IEEE Trans. Parallel Distrib. Syst*. Vol. 28(11). USA.

Калюжняк Анастасія Вікторівна – аспірантка кафедри програмної інженерії Запорізького національного університету, e-mail: anastasia.ko rgun@gmail.com, ORCID: 0000-0002-4837-7566.

Kaliuzhniak Anastasiia Victorivna – Postgraduate Student at the Department of Software Engineering of the Zaporizhzhia National University, e-mail: anastasia.korgun@gmail.com, ORCID: 0000-0002-4837-7566.

https://doi.org/10.32782/mathematical-modelling/2023-6-1-7