УДК 519.688:519.6:514.752
DOI https://doi.org/10.26661/2786-6254-2022-2-09

ЗАСТОСУВАННЯ БІБЛІОТЕКИ MPI ДЛЯ ОБЧИСЛЕННЯ АЛГОРИТМУ «МАРШИРУЮЧИХ КУБІВ»

Калюжняк А. В.
аспірантка
Запорізький національний університет
вул. Жуковського, 66, Запоріжжя, Україна
orcid.org/0000-0002-4837-7566
anastasia.korgun@gmail.com

Мильцев О. М.
кандидат фізико-математичних наук,
доцент кафедри програмної інженерії
Запорізький національний університет
вул. Жуковського, 66, Запоріжжя, Україна
orcid.org/0009-0009-4382-7730
alexmyltsev@gmail.com

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

Візуалізація об'єктів геометричного моделювання стає невід'ємною складовою частиною багатьох підприємств машинобудування, будівництва та медицини. Використання комп'ютерних технологій допомагають проєктувати складні механізми. Уже існуючі методи геометричного моделювання вирішують проблеми візуалізації складних форм без великих матеріальних та часових затрат. Проте вирішуються не всі проблеми, та інколи застосування того чи іншого підходу неможливе, оскільки реалізація становиться досить складною. Задача побудови моделей є досить популярною. Візуалізація дозволяє автоматизувати процеси на підприємстві без великих матеріальних затрат.

Метою цього дослідження є розробка алгоритму методу моделювання геометричних об'єктів за допомогою функціонального підходу з використанням бібліотеки MPI та порівняння з використанням загальною пам'яттю OpenMP. Моделювання за допомогою функціонального підходу ґрунтується на використанні неявно заданих функцій, які дають можливість побудувати тривимірні моделі. У статті подано формальний опис алгоритму «маршируючих кубів», проаналізовано властивості і практичне застосування при побудові об'єктів з використанням паралельного програмування бібліотеки MPI та OpenMP. Подані приклади візуалізації об'єктів з використанням середовища Qt Creator.

Отримані результати будуть корисними для подальших теоретичних досліджень, а також для практичного використання візуального представлення моделей з розподіленою та загальною пам'яттю. Моделі, побудовані за допомогою алгоритму «маршируючих кубів», дають можливість розв'язання задач геометричного моделювання без можливих часових втрат і прийняття належних рішень стосовно представлення об'єктів.

Таким чином, задача побудови тривимірних об'єктів за допомогою функціонального підходу може бути успішно розв'язана завдяки використанню бібліотеки підходу MPI.

APPLICATION OF THE MPI LIBRARY FOR COMPUTATION OF THE "MARCHING CUBES" ALGORITHM

Kaliuzhniak A. V.
Postgraduate Student
Zaporizhzhia National University
Zhukovskoho str., 66, Zaporizhzhia, Ukraine
orcid.org/0000-0002-4837-7566
anastasia.korgun@gmail.com

Myltsev O. M.
Candidate of Physical and Mathematical Sciences,
Associate Professor at the Department of Software Engineering
Zaporizhzhia National University
Zhukovskoho str., 66, Zaporizhzhia, Ukraine
orcid.org/0009-0009-4382-7730
alexmyltsev@gmail.com

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

Visualization of geometric modeling objects is becoming an integral part of numerous engineering, construction and medical enterprises. The use of computer technologies enables to design complex mechanisms. Already existing methods of geometric modeling solve the problems of visualization of complex forms without involving large material and time costs.

However, not all problems can be solved and sometimes the application of a particular approach is impossible, as the implementation becomes relatively complicated. The task of building models is fairly common. Visualization enables to automate processes at the enterprise without large material costs.

The purpose of this study is to develop an algorithm for the method of modeling geometric objects using a functional approach with the MPI library implementation and its further comparison with Open MP shared memory.

Modeling by means of the functional approach is based on the use of implicitly specified functions that enables to build three-dimensional models.

The article presents a formal description of the "marching cubes" algorithm, analyzes its properties and practical application in the construction of objects using the MPI and OpenMP library parallel programming. Examples of visualization of objects with the Qt Creator environment usage are provided as well.

The obtained results will be advantageous for further theoretical research, as well as for the practical use of the visual representation of models with distributed memory.

Models built by means of the "marching cubes" algorithm facilitates to solving geometric modeling problems without possible time losses and making appropriate decisions regarding the representation of objects.

Thus, the task of constructing three-dimensional objects using a functional approach can be successfully solved by applying the MPI approach library.

Вступ. Дослідження, пов'язане з використанням геометричних моделей, застосовується сьогодні майже у всіх областях життєдіяльності. Представлення моделей без великих фізичних та матеріальних затрат, а також мінімізація можливих втрат дає можливість автоматизувати процеси, проєктувати складні механізми, споруди.

Функціональні можливості кожного методу геометричного моделювання реалізовуються за допомогою програмного забезпечення, яке в процесі роботи взаємодіє з графічними пристроями вводу/виводу. При практичній реалізації виникає одна з основних проблем – це автоматизація процесів побудови складних об'єктів з найменшою похибкою. Для того щоб візуалізувати будь-яку

геометричну модель, необхідно враховувати її групову приналежність: мікрогеометрія чи мікрогеометрія.

Розробка додатків, які одночасно інтегрують весь конвеєр наукового моделювання та є масштабованими, залишається технічно складною задачею. Збільшення паралельних архітектур ускладнило досягнення масштабованості та портативності. Навіть з новими інструментами алгоритми можуть не масштабуватися в різних паралельних парадигмах. Перед розробкою необхідно зрозуміти вплив вибору інструменту на продуктивність у реальних програмах.

1 Постановка проблеми. Широке застосування паралельних архітектур обчислювальних систем викликає підвищену зацікавленість до засобів розробки програмного забезпечення, яке здатне повністю використовувати апаратні засоби даних типів.

Застосування багатопроцесорних комп'ютерів та відповідних технологій розпаралелювання алгоритмів дозволяє суттєво зменшувати час обчислень за рахунок паралельної обробки даних. Основні критерії якості паралельної реалізації алгоритму: прискорення розрахунків із зростанням числа процесорів, ефективність та адекватність відтворення явищ, що моделюються.

Прискорення визначають як відношення часу рахунку послідовного алгоритму до часу рахунку паралельного алгоритму, а ефективність – як відношення прискорення до кількості процесорів, в якому досягнуто у відсотках. Насправді, зазвичай відбувається зниження ефективності зі зростанням числа процесорів. Це пов'язано з такими чинниками: програми можуть мати послідовні фрагменти, що може послугувати розбалансуванню обчислень в паралельних процесах, тому деякий час може витрачатися на міжпроцесорні обміни. Для збалансування обчислень та мінімізації обмінів ключова роль відводиться вибору способу розподілу даних та обчислень за процесорами [1].

Проблема адаптації математичних моделей до багатопроцесорних обчислювальних комплексів – це проблема найбільш ефективної реалізації алгоритмів зі збереженням точності результатів моделювання. Найбільш універсальним та зручним методом побудови геометричного об'єкта є функціональний підхід. За його допомогою можна побудувати модель будь-якої складності. R-функції складаються з математичних відношень в неявному вигляді, які допоможуть візуалізувати складні моделі.

Найбільш ефективним для виконання багатовимірних завдань на багатопроцесорних обчислювальних системах з розподіленою пам'яттю вважається принцип геометричного паралелізму, який передбачає декомпозицію розрахункової області на підобласті відповідно до кількості процесорів. Технологія цього принципу заснована на розподілі області за процесорам, з вимогою рівномірного завантаження. При цьому розбиття області відбувається по блоках. Якщо розмірність сітки в блоці більша за середню розмірність у розрахунку на один процесор, то цей блок обслуговується кількома процесорами, і навпаки, один процесор обслуговує кілька сусідніх блоків, якщо їх сумарна розмірність не перевищує середньої.

2 Огляд бібліотек паралельного програмування. Для реалізації алгоритму геометричного паралелізму розроблено бібліотеки обміну повідомленнями MPI (Message Passing Interface). В ній кожен процесор кластера виконує одні й самі обчислення частини розрахункової області, розподіленого цьому процесору. Обчислення зводяться до взаємно узгодженої поетапної реалізації методу розщеплення за просторовими змінними. Винятки становлять процесори, які додатково виконують склеювання рішень на внутрішніх кордонах [2]. Умови склеювання виконуються на кожному кроці за часом. Процесори, що обслуговують сусідні блоки, передають необхідну інформацію одному з таких процесорів, який здійснює автономний розрахунок усієї межі в цілому та розсилає результати у зворотному напрямку. Це відбувається паралельно з усіх блоків, незначна затримка може виникнути тільки через необхідність передачі даних виконуючим процесорам для склеювання рішень на протилежній межі свого блоку. При цьому всі процесори, крім тих, що виконуються (тобто активних в даний момент часу), знаходяться в стані очікування (рис. 1).

Розробка паралельних програм ускладнюється також через такі проблеми: ресурси (кількість вузлів, їх архітектура, продуктивність) визначаються тільки в момент обробки мережею замовлення виконання завдання.

Незважаючи на те, що використання бібліотеки MPI показують високий рівень продуктивності, сама технологія має низку недоліків:

– низький рівень (програмування з використанням MPI часто порівнюють із програмуванням на асемблері), необхідність детального управління розподілом масивів та витків циклів між процесами, а також обміном повідомленнями між процесами – все це призводить до високої трудомісткості розробки програм;

– необхідність надмірної специфікації типів даних у переданих повідомленнях, а також наявність жорстких обмежень на типи даних, що передаються;

– складність написання програм, здатних виконуватися при довільних розмірах масивів та довільній кількості процесів, – унеможливлює повторне використання наявних MPI-програм;

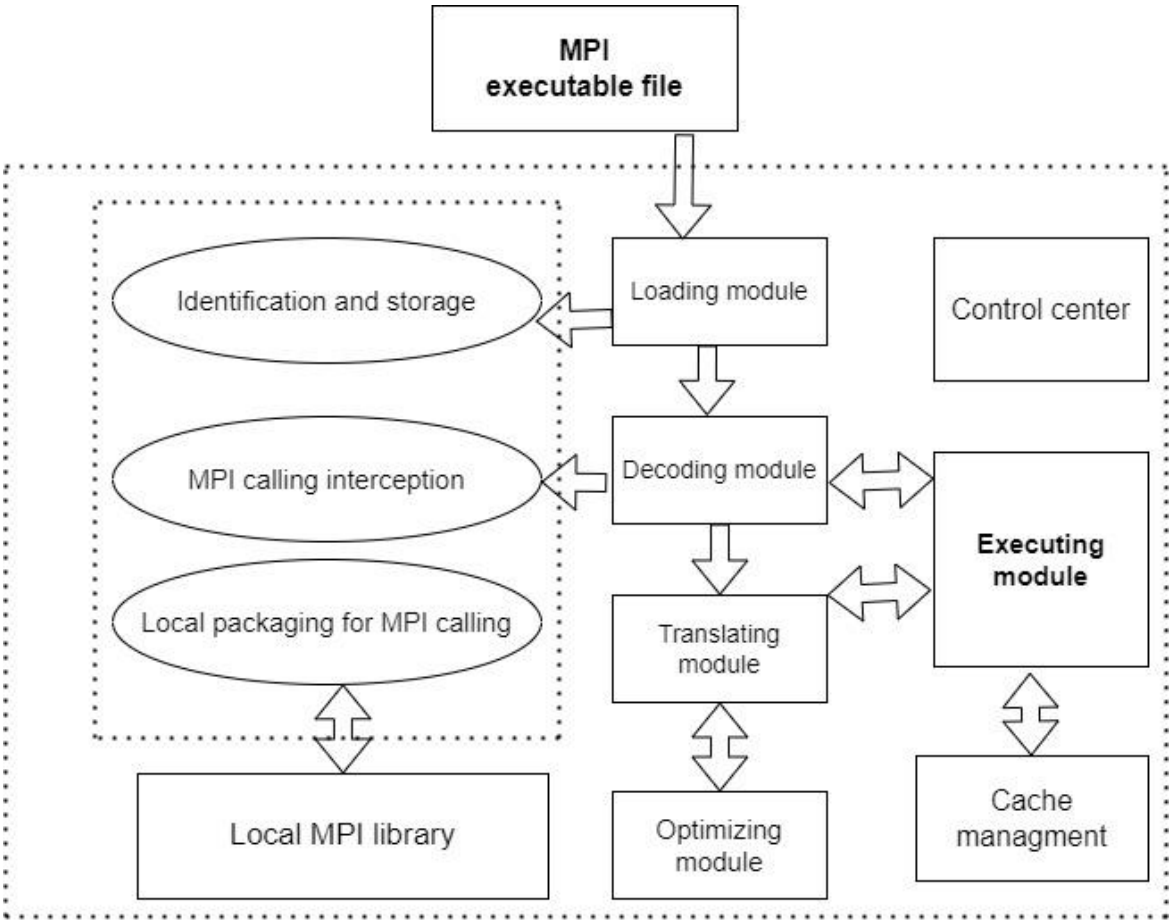

Рис. 1. Етапи роботи бібліотеки MPI

– відсутність підтримки об'єктно-орієнтованого підходу [3].

Але існує ще одна технологія, яка дозволяє візуалізувати модель за допомогою паралельного програмування багатопочних додатків із загальною памяттю – OpenMP.

Технологія OpenMP реалізує паралельні обчислення з допомогою багатопоточності, у якій «головний» (master) потік створює набір «підлеглих» (slave) потоків і з-поміж них розподіляються обчислення. Потім за допомогою директив препроцесора OpenMP породжується паралельна область із заданою кількістю потоків CPU рівним кількості GPU, і для кожного потоку призначається device-пристрій за допомогою функції cudaSetDevice(ndev), де ndev – номер пристрою. Після цього на кожному GPU розміщується відповідна частина даних, наприклад частина матриці необхідного розміру і паралельна область закривається. Далі, наприклад, при множенні матриці на вектор відкривається паралельна область, при цьому кожен потік використовує пристрій, який йому призначено. Це можливо завдяки тому, що OpenMP використовує безліч потоків, які не знищуються між паралельними і послідовними областями, а управління передається або заданій кількості потоків для паралельної області, або головному потоку для послідовної області. Постійна підтримка безлічі потоків дає виграш у продуктивності, оскільки створення потоків у кожній паралельній області є повільною операцією [4].

При вивченні аспектів процесу розпаралелювання алгоритмів для наукових досліджень не можна залишити без уваги питання впливу на продуктивність подання даних з різною точністю, оскільки дуже часто реальні розрахунки необхідно вести з більш високою точністю (стандартною подвійною – real*8, максимально можливою – real*16). Відповідно до технічних характеристик продуктивність процесорів-вузлів кластера залежить від використовуваної точності обчислень. Найбільша продуктивність досягається при обчислення з одинарною точністю подання даних. При обчисленнях з подвійною точністю продуктивність знижується теоретично на порядок (порівняно з піковою продуктивністю, заявленою виробником) і приблизно вдвічі на практиці.

Число потоків у групі, що виконуються паралельно, можна контролювати кількома спо-

собами. Один із них – використання змінної оточення OMP_NUM_THREADS [5]. Інший спосіб – виклик процедури omp_set_num_threads (). Ще один спосіб – використання виразу num_threads у поєднанні з директивою parallel. (приклад з програми).

Розглянемо приклад використання даної бібліотеки для задачі підрахунку суми $\sum_{i=n} \frac{1}{1+i^2}$:

```
double sum_series(int begin, int end)
{
 auto sum = 0.0;

 for (auto i = begin; i < end; i++)
 sum += 1.0 / (1.0 + double(i * i));
 return sum;
}
int main()
{
 int num_thread = omp_get_max_threads() - 1, //
Максимальна кількість потоків
 n = 100000, // Кількість членів ряду
 step = n / num_thread;
 double sum = 0;

 cout << "Sum of series (without threads): " <<
sum_series(0, n) << '\n';

 cout << "Num of threads: " << num_thread <<
'\n';
 #pragma omp parallel num_threads(num_thread)
 {
 int i = omp_get_thread_num();
 double thread_sum = sum_series(i * step, (i ==
num_thread - 1) ? n : (i + 1) * step);
```

Скомпільована програма показує, що використання бібліотеки OpenMP пришвидшує обчислення даної суми. Тут ми використовували директиву #pragma omp parallel, яка розділяє цикл генерації геометрії між кількома потоками виконання. Кожен потік виконує ітерацію циклу окремо, що збільшує швидкість виконання програми (рис. 2):

```
Sum of series (without threads): 3.0767
Num of threads: 7
Sum of series (with threads): 2.0766
```

Рис. 2. Результат роботи обчислення суми

Алгоритм «маршируючих кубів» можна реалізувати паралельно за допомогою OpenMP шляхом поділу тривимірного скалярного зображення прямокутні секції. Вхідні параметри керують розміром кожної секції за координатами x, y та z. Потім кожна секція опрацьовується незалежно.

Вихідні дані кожного розділу потім об'єднуються в один масив. Реалізація MPI аналогічна формою. Кожен потік OpenMP заповнює локальні вихідні змінні точками, нормалями та трикутниками. Точки та нормалі містять вектор точок сітки. Трикутник містить вектор трійки індексів, де індекси відносяться до значень у точках та нормалях. Кожен потік обробляє розділи, призначені OpenMP. Після обробки всіх розділів локальний висновок необхідно з'єднати з глобальним висновком. Для зручності читання був обраний простий, але ефективний спосіб зробити це. Коли кожен потік завершує заповнення локального висновку, цей висновок додається до глобальних точок, нормалів і трикутників, де тільки один потік може додавати до глобального висновку.

На діаграмі показано зміну часу побудови геометричного об'єкта зі зміною потоків. Очевидно, що програма реалізована за допомогою MPI працює краще, ніж код OpenMP. Оскільки кожна реалізація однаково працює з окремими частинами алгоритму паралельно, очікується, що результати продуктивності не дуже відрізнятимуться.

Однак після зчитування даних код MPI копіює дані розділу на кожен процесор. Це швидка операція, яка може мати додаткові переваги. Використання алгоритму «маршируючих кубів» з технологією MPI дозволяє значно пришвидшити побудову складних геометричних моделей за рахунок розбиття на вокселі. Дане дослідження проводилося на AMD EPYC 7502P 32-Core Processor 32 ядра, 64 потоків, 256 GB RAM.

Алгоритм визначає, як поверхня перетинає цей куб, потім переміщається до наступного кубу. Щоб знайти поверхневий перетин в кубі, необхідно присвоїти 1 до вершини куба, якщо значення даних в цій вершині перевищує (або дорівнює) значенням поверхні, яку будуємо [6]. Ці вершини знаходяться всередині (або на) поверхні (рис. 4).

```
// Scalar field data
// Function to classify a cube as either inside
or outside
int classify_cube(int i, int j, int k) {
 int cube_index = 0;
 if (scalar_field[i][j][k] < ISO_VALUE) cube_
index |= 1;
 if (scalar_field[i+1][j][k] < ISO_VALUE) cube_
index |= 2;
 if (scalar_field[i+1][j+1][k] < ISO_VALUE)
cube_index |= 4;
 if (scalar_field[i][j+1][k] < ISO_VALUE) cube_
index |= 8;
 if (scalar_field[i][j][k+1] < ISO_VALUE) cube_
index |= 16;
 if (scalar_field[i+1][j][k+1] < ISO_VALUE)
cube_index |= 32;
 if (scalar_field[i+1][j+1][k+1] < ISO_VALUE)
cube_index |= 64;
```

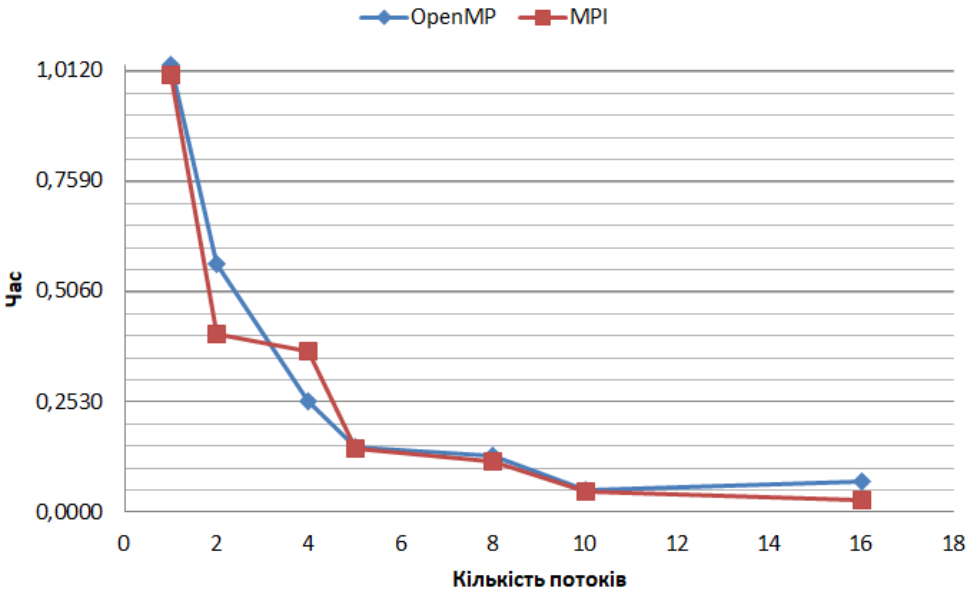

Рис. 3. Порівняння часу роботи алгоритмів

Рис. 4. Візуалізація геометричного об'єкта за допомогою «маршируючих кубів»

```
 if (scalar_field[i][j+1][k+1] < ISO_VALUE)
cube_index |= 128;
 return cube_index;}
// Function to construct the final polygonal mesh
void construct_mesh(){
 // the Marching Cubes algorithm to construct
the mesh}
int main(int argc, char** argv){
 MPI_Init(&argc, &argv);
 int num_procs, rank;
 MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 // Load scalar field data, the grid into
subgrids
 int start_i = rank * N / num_procs;
 int end_i = (rank + 1) * N / num_procs;
 int subgrid_size = end_i - start_i;
 float subgrid[subgrid_size][N][N];
MPI_Scatter((void*)scalar_field[start_i],
subgrid_size*N*N, MPI_FLOAT,
 (void*)subgrid, subgrid_size*N*N, MPI_FLOAT,
0, MPI_COMM_WORLD);
 // Classify each cube in the subgrid as either
inside or outside } } }
 // Exchange information about cube faces on
boundaries
```

При такому допущенні ми визначаємо топологію поверхні всередині куба, знаходження перетину визначається пізніше.

Застосування алгоритму «маршируючих кубів» в такому обсязі дозволяє повторно створити сітку оригіналу з рівномірним розподілом вершин.

Отже, використання бібліотеки з розподіленою пам'яттю дозволяє візуалізувати складні структури, для яких, за необхідності, можна створити свій тип, на основі вже існуючих, надає можливість паралельного вводу-виводу в різні частини файлу, створення та зупинку нових процесів, синхронізації та доступу до пам'яті на віддаленій машині. MPI використовує обмін повідомленнями для обміну даними між процесами. Вона зазвичай використовується для розподілених задач, які не можуть бути розбиті на незалежні частини, наприклад, коли потрібно вирішити задачу на великій кількості незалежних вузлів. А використовувати OpenMP можна, коли завдання може бути розбито на незалежні частини та оброблено паралельно на спільній пам'яті.

Використання директив MPI дозволяє розробнику на будь-якому етапі створення паралельної частини повернутись до послідовного варіанту програмного продукту. Це забезпечує гнучкість при роботі з розпаралеленням програми і надає можливість розробнику почати створювати програму послідовно і лише в потрібний момент здійснити процедуру розбиття виконання задачі на окремі потоки.

ЛІТЕРАТУРА

1. Alexandrov A, Ionescu M.F., Schauser K,E, Scheiman C. Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation, USA: Tech. Rep, 1995, p. 206.
2. Fagg G.E., Pjesivac-Grbovic J., Bosilca G., Angskun T., Dongarra J., Jeannot E. Flexible collective communication tuning architecture applied to Open MPI, PVM/MPI, USA : Manning Publications, 2006, p. 14.
3. Guo T., Yu K., Aloqaily M., Wan S. Constructing a prior-dependent graph for data clustering and dimension reduction in the edge of AIoT, Future Gener. Comput. Syst. 128, Kobe, 2022, p. 381.
4. Hunold S., Bhatele A., Bosilca G., Knees P. Predicting MPI collective communication performance using machine learning, in: 2020 IEEE International Conference on Cluster Computing (CLUSTER), Kobe, 2020, p. 259.
5. Kristiani E., Yang C.-T., Huang C.-Y., Ko P.-C., Fathoni H. On construction of sensors, edge, and cloud (iSEC) framework for smart system integration and applications, IEEE Int. Things J. 8(1), USA, 2020 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. 28(11), USA, 2017 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.
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. 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. 28 (11)*, USA.