Лабораторные работы по курсу

“Параллельное и распределённое программирование”

Гусев Евгений

1. Умножение матриц

Целью данной лабораторной работы является сравнение последовате-льной и параллельной реализации алгоритмов умножения матриц, времени их работы при разных способах обхода алгоритма.

Результатом программы является список, состоящий из строк, в которых через запятую указаны количество используемых потоков, размерность квадратной матрицы и время работы.

При запуске файлов home.sh и run.sbatch, формируется csv-файл, являющийся результатом выполнения программы, по данным которого, можно строить графики отражающие: ускорение, эффективность и время работы алгоритма (см. Рисунок 1, 2, 3, 4).

Программы собирались и запускались как на домашнем компьютере, так и на вычислительном кластере САФУ.

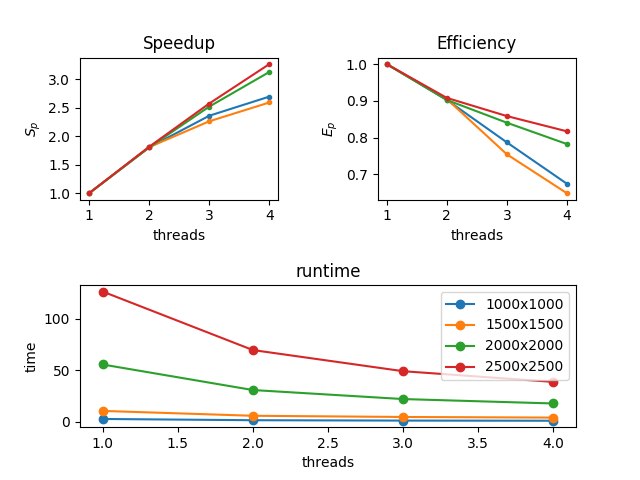


Рисунок 1 – графики до смены порядка обхода матриц.

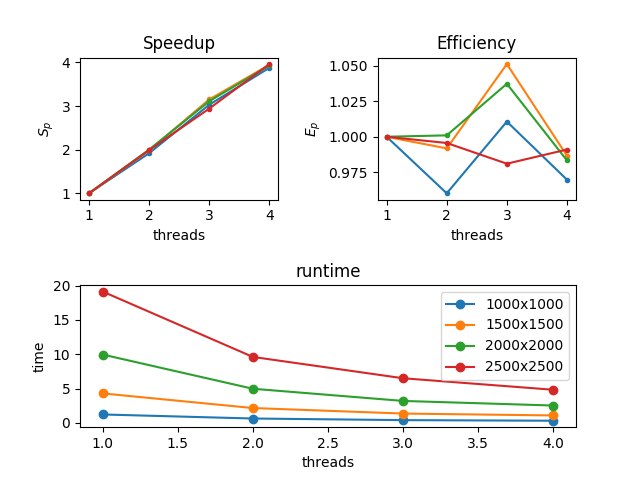


Рисунок 2 – графики после смены порядка обхода матриц.

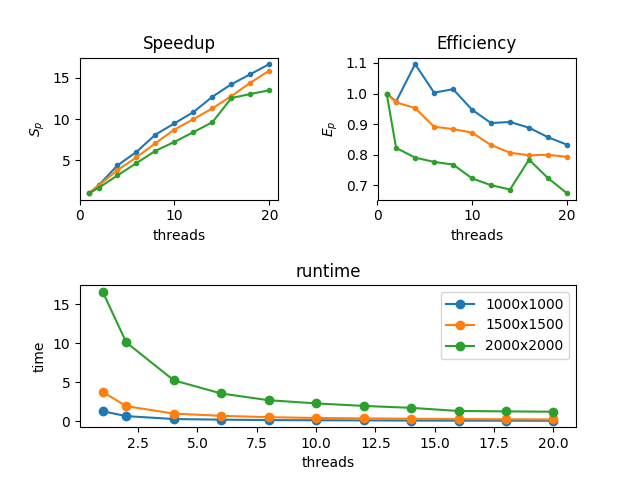


Рисунок 3 - графики до смены порядка обхода матриц (на кластере).

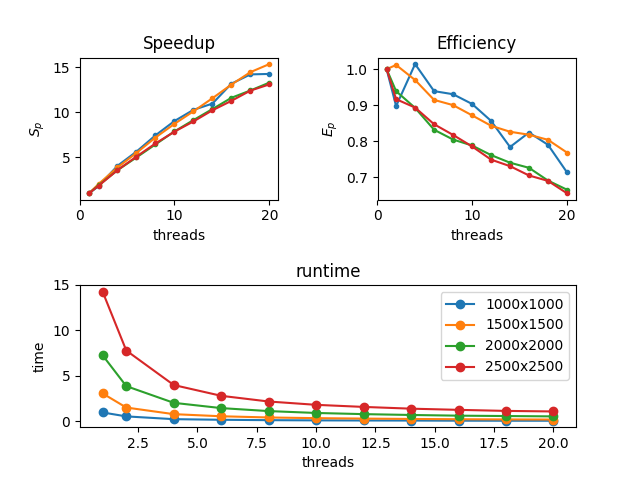


Рисунок 4 - графики после смены порядка обхода матриц (на кластере).

По рисункам 1-4 видно, что после изменения порядка обхода матриц

(см. Листинг 1) время работы программы может сокращаться в несколько

(2-5) раз.

Листинг 5 – Два способа обхода матрицы.

// строки-столбцы

Matrix mulParallel(const Matrix& first, const Matrix& second) {

Matrix result(first.rows(), second.cols());

if (first.cols() == second.rows()) {

#pragma omp parallel for shared(result, first, second)

for (size\_t i = 0; i < result.rows(); ++i)

for (size\_t j = 0; j < result.cols(); ++j) {

result(i, j) = 0;

for (size\_t k = 0; k < result.rows(); ++k)

result(i, j) += first(i, k) \* second(k, j);

}

}

else

throw std::invalid\_argument("Wrong dimensions");

return result;

}

// столбцы-строки

Matrix mulParallel2(const Matrix& first, const Matrix& second) {

Matrix result(first.rows(), second.cols());

if (first.cols() == second.rows()) {

#pragma omp parallel for shared(result, first, second)

for (size\_t j = 0; j < result.cols(); ++j)

for (size\_t i = 0; i < result.rows(); ++i) {

result(i, j) = 0;

for (size\_t k = 0; k < result.rows(); ++k)

result(i, j) += first(i, k) \* second(j, k);

}

}

else

throw std::invalid\_argument("Wrong dimensions");

return result;

}

Данный пример демонстрирует особенности устройства кэша процессора и оперативной памяти. При обращении к какой-то ячейке памяти в кэш записывается не только она, но и несколько соседних ячеек. В первом случае два элемента, над которыми производятся операции в смежных итерациях алгоритма, в оперативной памяти расположены на расстоянии равному размеру одной строки массива. Во втором случае смежные операции будут оперировать элементами одной строки матрицы, элементы которой располагаются в памяти подряд.

2. Задача Дирихле для уравнения Пуассона

Для решения задачи Дирихле для уравнения Пуассона, были реализованы последовательный и параллельный алгоритмы с использованием класса матриц из первой лабораторной работы. Так же были написаны python-скрипты для построения графика поверхностей. По уравнению f(x,y)=1 с краевыми условиями вида:

Был построен график решения задачи, который представлен на рисунке 6.

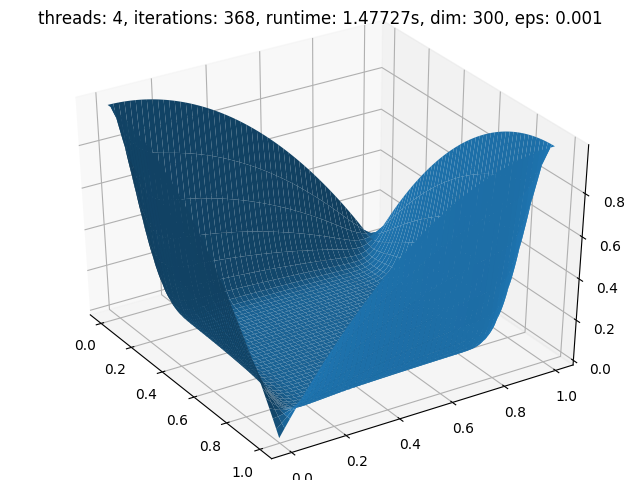


Рисунок 6 – График решения задачи Дирихле для уравнения Пуассона.

Чтобы решение было таким же, как и в непараллельном алгоритме Гаусса-Зейделя, параллельный алгоритм был построен по волновой схеме. Алгоритм Гаусса-Зейделя использует два ранее вычисленных элемента: , они необходимы для вычисления текущего элемента . Вначале, единственным, удовлетворяющим этим условиям, элементом является , но, в результате вычисления становится доступна следующая диагональ . Следовательно, выполнение одной итерации можно разбить на несколько шагов, на которых вычисляются узлы, которые расположены на одной из диагоналей исходной сетки.

Листинг 7 – Реализация задачи Дирихле для уравнения Пуассона

Results\_of\_Dirichlet Solution\_of\_Dirichlet\_OMP(size\_t N, double eps) {

auto startTime = std::chrono::steady\_clock::now();

Matrix u\_mat(N+2, N+2);

Matrix f\_mat(N, N);

double h = 1.0 / (N + 1);

for (size\_t i = 0; i < N; i++) {

for (size\_t j = 0; j < N; j++)

f\_mat(i, j) = f((i + 1) \* h, (j + 1) \* h);

}

for (size\_t i = 1; i < N + 1; i++) {

u\_mat(i, 0) = g(i \* h, 0);

u\_mat(i, N + 1) = g(i \* h, (N + 1) \* h);

}

for (size\_t j = 0; j < N + 2; j++) {

u\_mat(0, j) = g(0, j \* h);

u\_mat(N + 1, j) = g((N + 1) \* h, j \* h);

}

double max, u0, d;

size\_t i = 0, j = 0, iterations = 0;

std::vector<double> mx(N+2);

do{

iterations++;

// нарастание волны (k - длина фронта волны)

for (size\_t k = 1; k < N+1; k++) {

mx[k] = 0;

#pragma omp parallel for shared(u\_mat, k, mx) private(i, j, u0, d) schedule(static, 1)

for (i = 1; i < k+1; i++) {

j = k + 1 - i;

u0 = u\_mat(i, j);

u\_mat(i, j) = 0.25 \* (u\_mat(i-1, j) + u\_mat(i+1, j) + u\_mat(i, j-1) + u\_mat(i, j+1) - h\*h\*f\_mat(i-1, j-1));

d = std::fabs(u\_mat(i, j) - u0);

if (d > mx[i]) mx[i] = d;

}

}

for (size\_t k = N-1; k > 0; k--) {

#pragma omp parallel for shared(u\_mat, k, mx) private(i, j, u0, d) schedule(static, 1)

for (i = N-k+1; i < N+1; i++){

j = 2\*N - k - i + 1;

u0 = u\_mat(i, j);

u\_mat(i, j) = 0.25 \* (u\_mat(i-1, j) + u\_mat(i+1, j) + u\_mat(i, j-1) + u\_mat(i, j+1) - h\*h\*f\_mat(i-1, j-1));

d = std::fabs(u\_mat(i, j) - u0);

if (d > mx[i]) mx[i] = d;

}

}

max = 0;

for (i = 1; i < N+1; i++) {

if (mx[i] > max) max = mx[i];

}

} while (max > eps);

auto runtime = std::chrono::steady\_clock::now();

auto runtimeDuration = std::chrono::duration\_cast<std::chrono::duration<double>>(runtime - startTime);

Results\_of\_Dirichlet result(omp\_get\_max\_threads(), u\_mat, iterations, runtimeDuration.count(), eps);

return result;

}

Скорость работы программы на различном количестве ядер была зафиксирована с помощью графика, который представлен на рисунке 8.

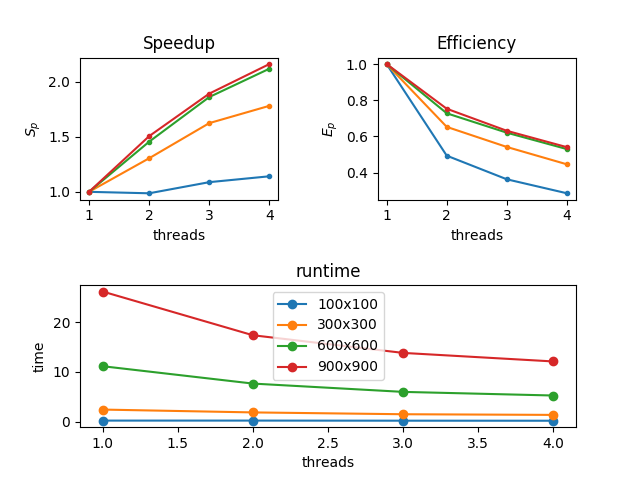


Рисунок 8 – графики ускорения, эффективности и времени работы волнового алгоритма.

Сравнение результатов двух лабораторных работ показывает, что все алгоритмы работают с разной эффективностью и по-разному распараллеливаются.