

Лабораторные работы по курсу
"Параллельное и распределённое программирование"

Игорь Комолых, Сергей Луцик

24 мая 2018 г.

1 Умножение матриц

Эта лабораторная работа заключалась в сравнении последовательной и параллельной реализации алгоритмов умножения матриц, а так же в сравнении времени работы программы при разных способах обхода массива.

В результате выполнения работы были получены:

- описанный класс Matrix
- bash и sbatch файлы запуска программы на персональных компьютерах и кластере САФУ
- python-скрипт для построения графиков из полученных данных.

Результатом выполнения программы является строка, в которой через запятую указаны количество используемых потоков, размерность квадратной матрицы, время работы. При запуске bash или sbatch сценариев, происходит формирование CSV-файла, по данным которого в дальнейшем можно строить графики ускорения, эффективности и времени работы (см. Рис. 1 и 2). Программы собирались и запускались на вычислительном кластере САФУ.

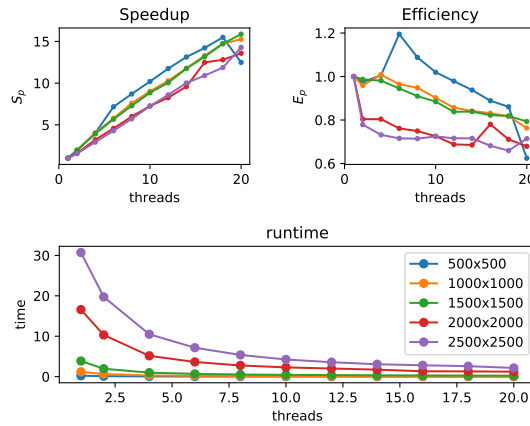


Рис. 1: графики до смены порядка обхода матриц.

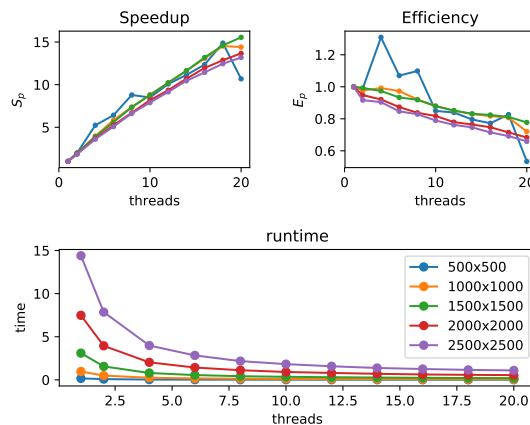


Рис. 2: графики после смены порядка обхода матриц.

По рис. 1 и 2 видно, что после изменения порядка обхода матрицы с привычного “строки-столбцы” на “столбцы-строки” (см. Листинг 1) время работы программы может сокращаться в 2 и более раза, в некоторых случаях удавалось достичь ускорения в 4-5 раз.

Данный пример демонстрирует особенности устройства кэша процессора и оперативной памяти. При обращении к какой-либо ячейке памяти, в кэш вместе с ней загружаются и несколько соседних ячеек. При обращении в порядке “строки-столбцы” два элемента, над которыми производятся операции в смежных итерациях алгоритма, в памяти будут находиться на расстоянии, равном размеру строки матрицы. Если же обходить массивы в порядке “столбцы-строки”, смежные итерации будут оперировать элементами одной строки матрицы, элементы которой располагаются в памяти друг за другом.

```

1  //
2  // строки-столбцы
3  //
4
5  #include <iostream>
6  Matrix mulParallel(const Matrix& first, const Matrix& second) {
7      Matrix result(first.rows(), second.cols());
8      if (first.cols() == second.rows())
9          #pragma omp parallel for shared(result, first, second)
10         for (size_t i = 0; i < result.rows(); ++i)
11             for (size_t j = 0; j < result.cols(); ++j) {
12                 result(i, j) = 0;
13                 for (size_t k = 0; k < result.rows(); ++k)
14                     result(i, j) += first(i, k) * second(k, j);
15             }
16     else
17         throw std::invalid_argument("Wrong dimensions");
18
19     return result;
20 }
21
22 //
23 // столбцы-строки
24 //
25
26 Matrix mulParallel2(const Matrix& first, const Matrix& second) {
27     Matrix result(first.rows(), second.cols());
28     if (first.cols() == second.rows()) {
29         #pragma omp parallel for shared(result, first, second)
30         for (size_t j = 0; j < result.cols(); ++j)
31             for (size_t i = 0; i < result.rows(); ++i) {
32                 result(i, j) = 0;
33                 for (size_t k = 0; k < result.rows(); ++k)
34                     result(i, j) += first(i, k) * second(j, k);
35             }
36     }
37     else
38         throw std::invalid_argument("Wrong dimensions");
39
40     return result;
41 }

```

Листинг 1: Два способа обхода матрицы

2 Задача Дирихле для уравнения Пуассона

С использованием класса матриц, полученного в ходе выполнения первой лабораторной были реализованы последовательный и параллельный алгоритмы решения задачи Дирихле для уравнения Пуассона, написаны python-скрипты для построения графиков поверхностей по полученным данным. Результат тестового запуска алгоритма для уравнения $f(x, y) = 4$ с краевыми условиями $g(x, y) = (x - 0.5)^2 + (y - 0.5)^2$ представлен на рис. 3.

Для того чтобы гарантировать получение точно таких же решений, как и в непараллельном алгоритме Гаусса-Зейделя, параллельный алгоритм был построен по волновой схеме. Для вычисления значения текущего элемента U_{ij} алгоритм Гаусса-Зейделя использует два ранее вычисленных элемента U_{i-1j} ,

threads: 8, iterations: 126, runtime: 1.24693s, dim: 102, eps: 0.001

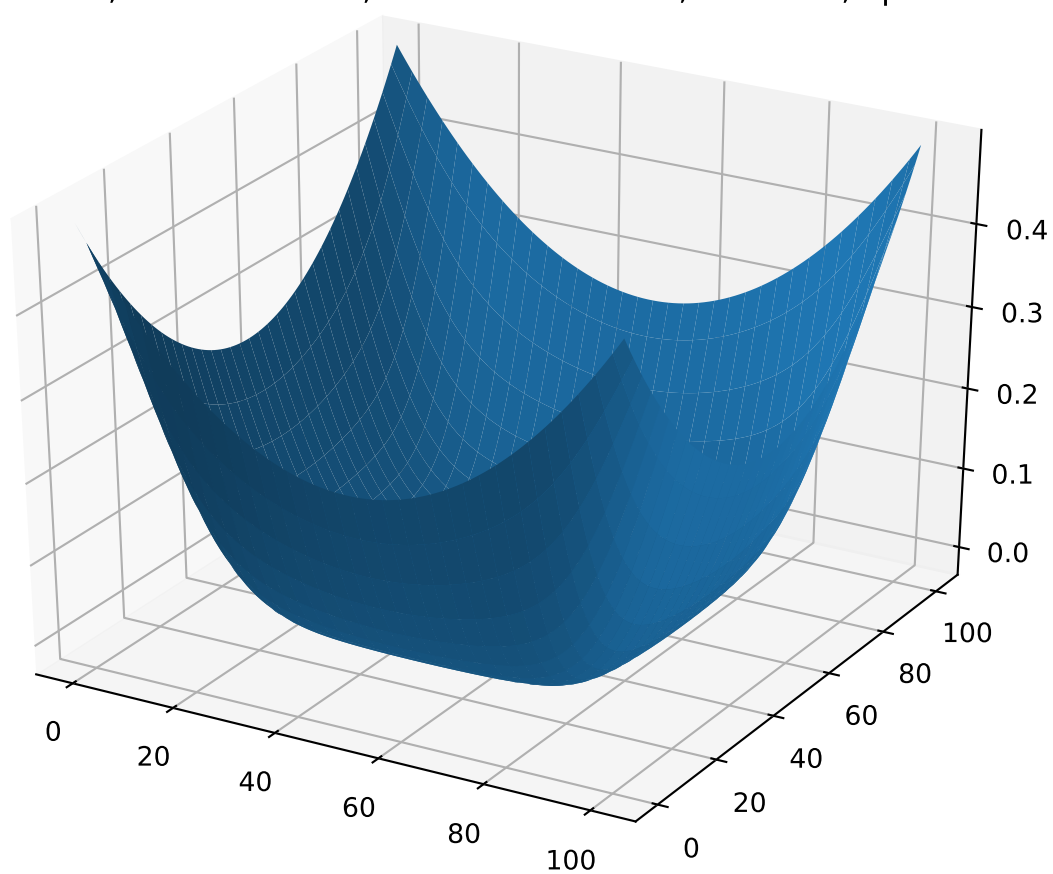


Рис. 3: График решения задачи Дирихле для уравнения Пуассона.