****МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

**Практикум по курсу**

**"Суперкомпьютеры и параллельная обработка данных"**

**Разработка параллельной версии программы для метода релаксации Якоби**

**ОТЧЕТ**

**о выполненном задании**

студента 325 группы факультета ВМК МГУ

Бондаря Артёма Александровича

Москва, 2017 г.

Оглавление

[1 Постановка задачи - 2 -](#_Toc500099601)

[2 Описание алгоритма метода простой итерации Якоби - 2 -](#_Toc500099602)

[2.1 Последовательный алгоритм - 2 -](#_Toc500099603)

[2.2 Параллельный алгоритм - 2 -](#_Toc500099604)

[3 Результаты замеров времени выполнения - 3 -](#_Toc500099605)

[3.1 Таблицы - 4 -](#_Toc500099606)

[3.1.1 OpenMP - 4 -](#_Toc500099607)

[3.1.2 MPI - 6 -](#_Toc500099608)

[3.2 3D-графики - 7 -](#_Toc500099609)

[3.2.1 Bluegene OpenMP - 7 -](#_Toc500099610)

[3.2.2 Bluegene MPI - 8 -](#_Toc500099611)

[3.2.3 Regatta OpenMP - 9 -](#_Toc500099612)

[3.2.4 Regatta MPI - 9 -](#_Toc500099613)

[3.2.5 6700K OpenMP - 10 -](#_Toc500099614)

[3.2.6 6700K MPI - 10 -](#_Toc500099615)

[3.2.7 3630QM OpenMP - 10 -](#_Toc500099617)

[3.2.8 3630QM MPI - 11 -](#_Toc500099618)

[3.2.9 3217U OpenMP - 11 -](#_Toc500099619)

[3.2.10 3217U MPI - 11 -](#_Toc500099620)

[3.2.11 370M OpenMP - 12 -](#_Toc500099621)

[3.2.12 370M MPI - 12 -](#_Toc500099622)

[4 Комментарии к результатам - 12 -](#_Toc500099623)

[5 Выводы - 13 -](#_Toc500099624)

# Постановка задачи

Ставится задача реализации алгоритма решения системы линейных алгебраических уравнений методом простой итерации Якоби. Алгоритм выполняет следующие итерационные вычисления:

В предложенной версии программы исполняется только 100 итераций.

Требуется:

1. Реализовать параллельные алгоритмы предложенного алгоритма с помощью технологий параллельного программирования OpenMP и MPI.
2. Сравнить их эффективность.
3. Исследовать масштабируемость полученной параллельной программы.
4. Построить графики зависимости времени исполнения от числа процессоров для различного объёма входных данных.

# Описание алгоритма метода простой итерации Якоби

## Последовательный алгоритм

Предложенный алгоритм решения имеет следующий вид:

for (it = 1; it <= itmax; it++) {

eps = 0.;

for (i = 1; i <= N - 2; i++)

for (j = 1; j <= N - 2; j++) {

double e;

e = A[i][j];

A[i][j] = (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j + 1]) / 4.;

eps = Max(eps, fabs(e - A[i][j]));

}

if (eps < maxeps) break;

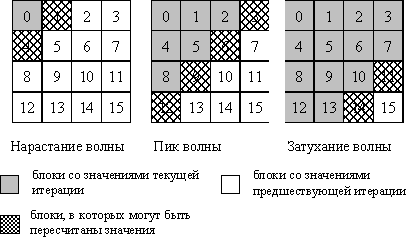
}

Алгоритм является итеративным и ориентирован на последовательное вычисление новых значений матрицы А. Предполагается выполнение операций сложения и операций деления элементов исходной матрицы. Количество выполненных операций имеет порядок O().

## Параллельный алгоритм

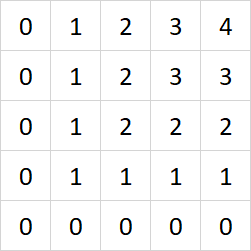
В виду наличия зависимости следующих значений от уже вычисленных для сохранения задачу нельзя распараллелить по строкам или столбцам (из-за состязания потоков), поскольку будут проводится другие вычисления. Поэтому вычисления необходимо проводить волнообразно.

Для улучшения эффективности использования потоков (процессоров) так же была реализована блочная структура вычислений – это сделано для того, чтобы снизить общее количество синхронизаций и увеличить количество вычислений внутри одного потока (процессора). Организация волны вычислений при блочной схеме разделения данных представлена на следующем рисунке:



В случае OpenMP размер блока определялся как , где N – размерность матрицы, T – число потоков. Коэффициент 8 был выбран эмпирическим путем в ходе исследования результатов вычислений. При дальнейшем измельчении блоков большей производительности получить не удавалось, а при снижении размера сетки – длительность увеличивалась в виду более долгих ожиданий подсчета остальными потоками до перехода на подсчет следующей волны.

А для MPI размер блока определялся как , где N – размерность матрицы, T – число процессоров. Это обосновано тем, чтобы центральная самая большая волна (побочная диагональ матрицы) вычислялась одновременно. В этом случае будет максимальная скорость вычислений (2N – 1 подсчетов блока одним процессором) и минимальная количество синхронизаций между процессорами. Алгоритм распределения блоков (для T = 5) показан на рисунке:



Реализованные алгоритмы проверялись на совпадение по результатам с последовательным алгоритмом методом подсчета и сравнения суммы всей матрицы.

# Результаты замеров времени выполнения

Программа была запущена в конфигурациях:

* на i7-6700K @ 4.00GHz, 16GB RAM, Debian 8.9 x64 – 1, 2, 4, 8 ядер OpenMP & MPI (Q3’15)
* на i7-3630QM @ 2.4GHz, 16GB RAM, Debian 9.8 x64 – 1, 2, 4, 8 ядер OpenMP & MPI (Q3’12)
* на i3-3217U @ 1.8GHz, 4GB RAM, Debian 9.8 x64 – 1, 2, 4 ядер OpenMP & MPI (Q2’12)
* на i3-370M @ 2.4GHz, 3GB RAM, Lubuntu 17.10 x86 – 1, 2, 4 ядер OpenMP & MPI (Q3’10)
* на Regatta - 1, 2, 4, 8, 16 OpenMP & MPI (Power4 - 2001)
* на Bluegene - 1, 2, 4 для OpenMP; 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 для MPI (PowePC 450 - 2007)

Каждая конфигурация была запущена 3 раза и посчитаны усредненные результаты.

Такое большое разнообразие конфигураций получилось в связи с тем, что хотелось собрать свой собственный вычислительный кластер для MPI в локальной сети Wi-Fi роутера (1 стационарный компьютер + 3 ноутбука). Было создано подключение по ssh с использованием RSA-ключей + используя NFS server был подключен и успешно смонтирован общий каталог, в котором хранились запускаемые программы. Однако на стадии исполнения были обнаружены ошибки сегментирования при использовании MPI\_Barrier, решения найти не удалось, и идея была заброшена. Разве что получись запускать удаленно только на одной машине.

## Таблицы

### OpenMP

Коэффициенты в правых колонках – результаты деления времени текущей позиции на результаты одного потока при том же размере матрицы, т.е. своеобразный коэффициент ускорения. Подробная таблица приложена также отдельно к отчету.



### MPI

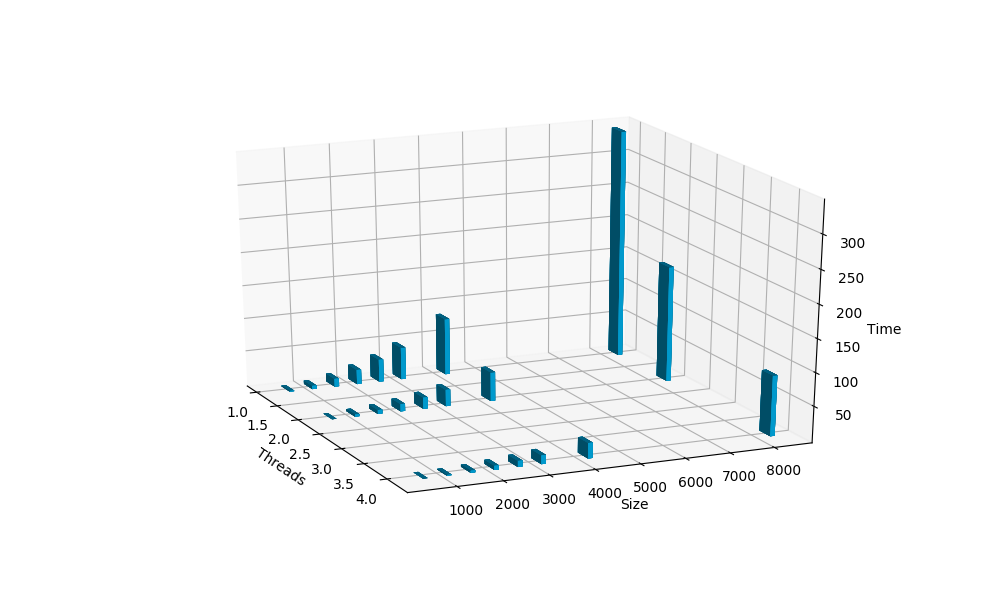




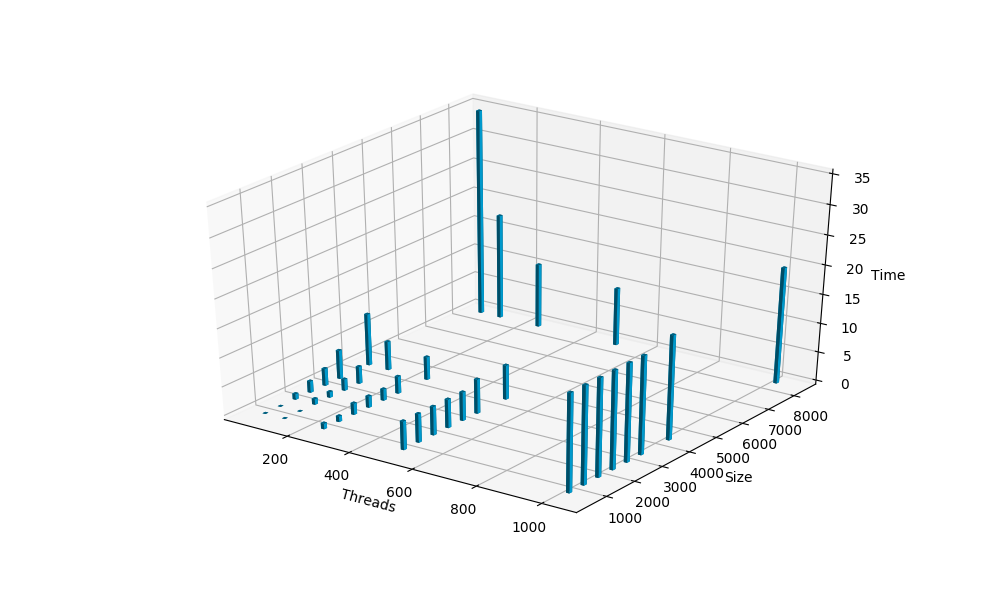
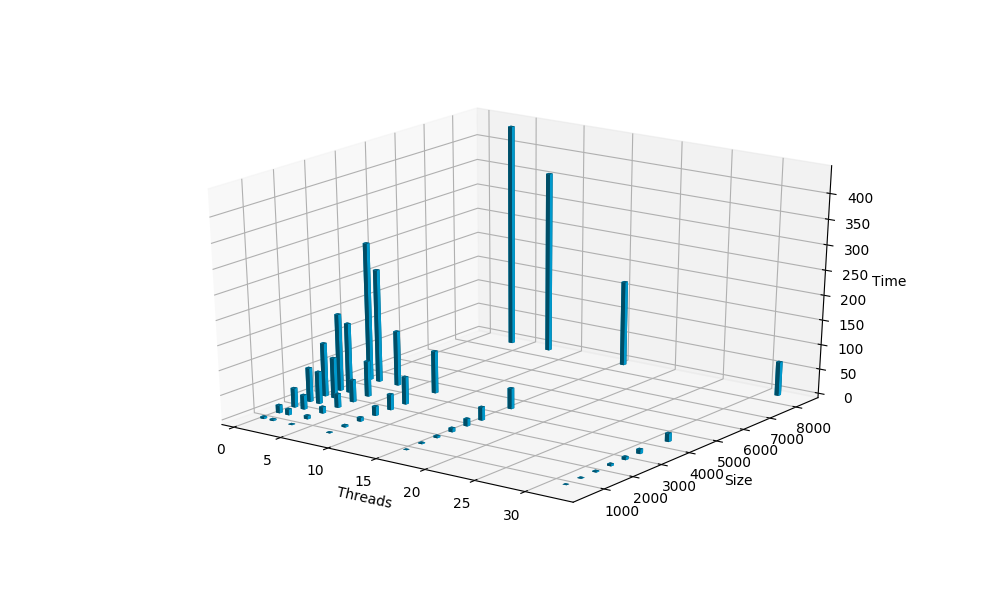
## 3D-графики

Для удобства отображения все результаты больше 900с не были отображены на графиках.

### Bluegene OpenMP

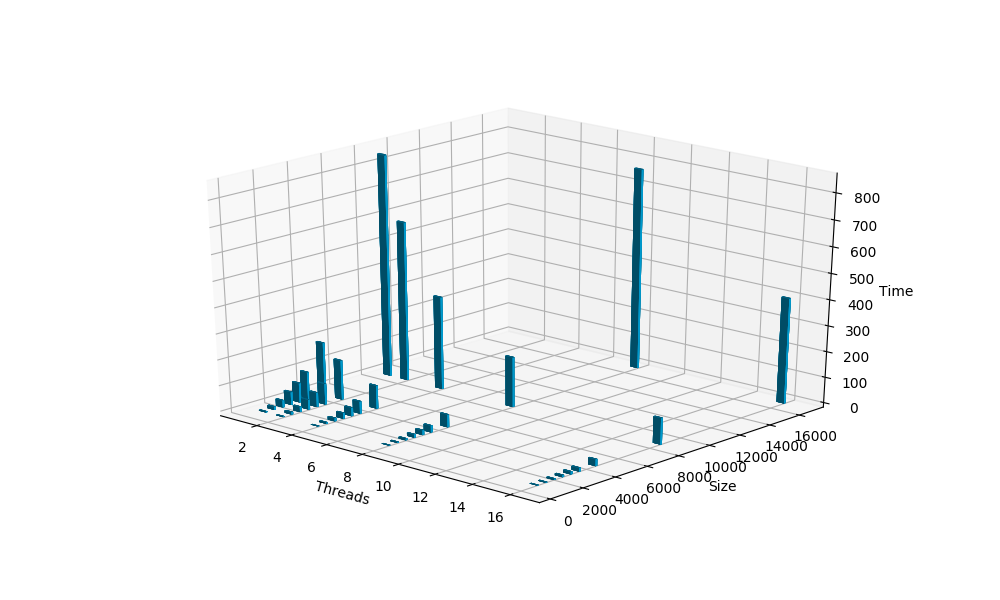


### Bluegene MPI

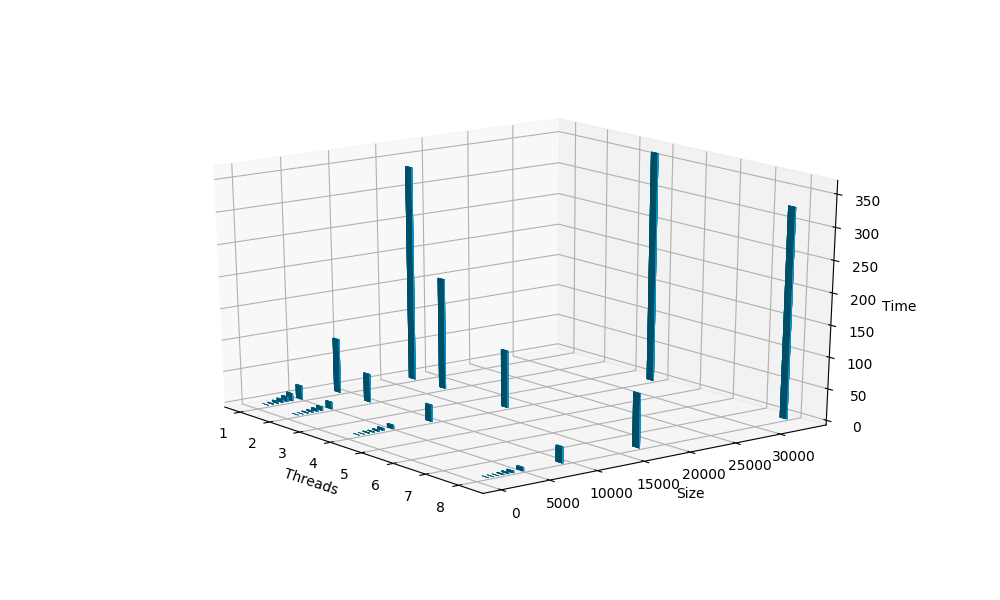


### Regatta OpenMPC:\Users\Артём Бондарь\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Regatta OpenMP.PNG

### Regatta MPI



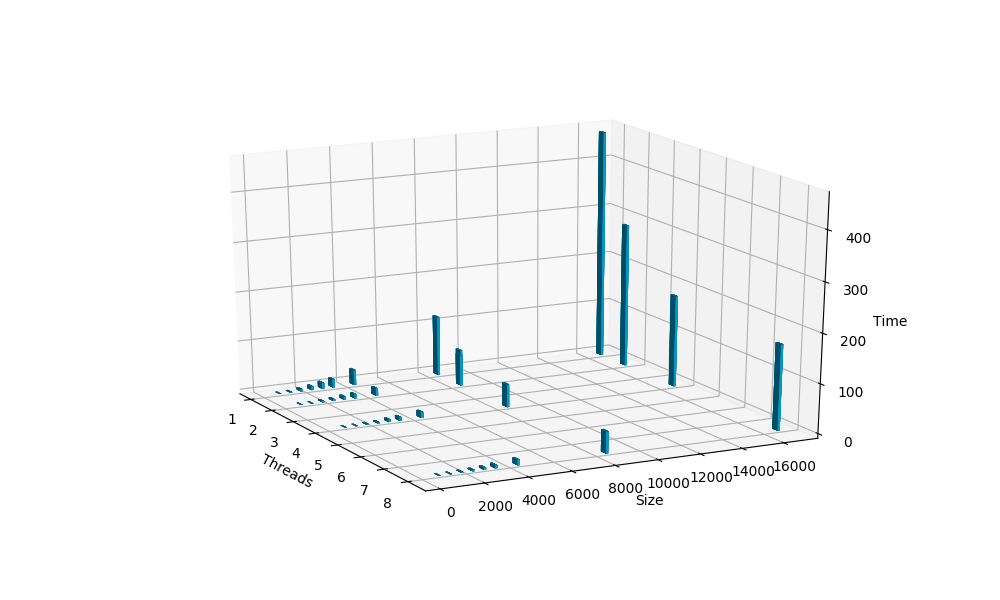
### 6700K OpenMP



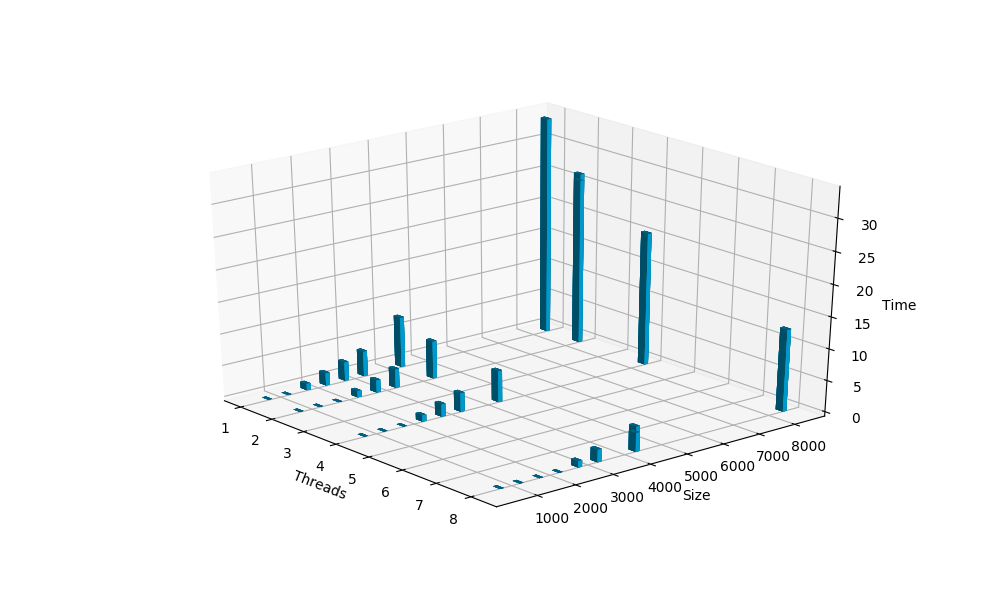
### 6700K MPI

### C:\Users\Артём Бондарь\AppData\Local\Microsoft\Windows\INetCache\Content.Word\6700K MPI.PNG

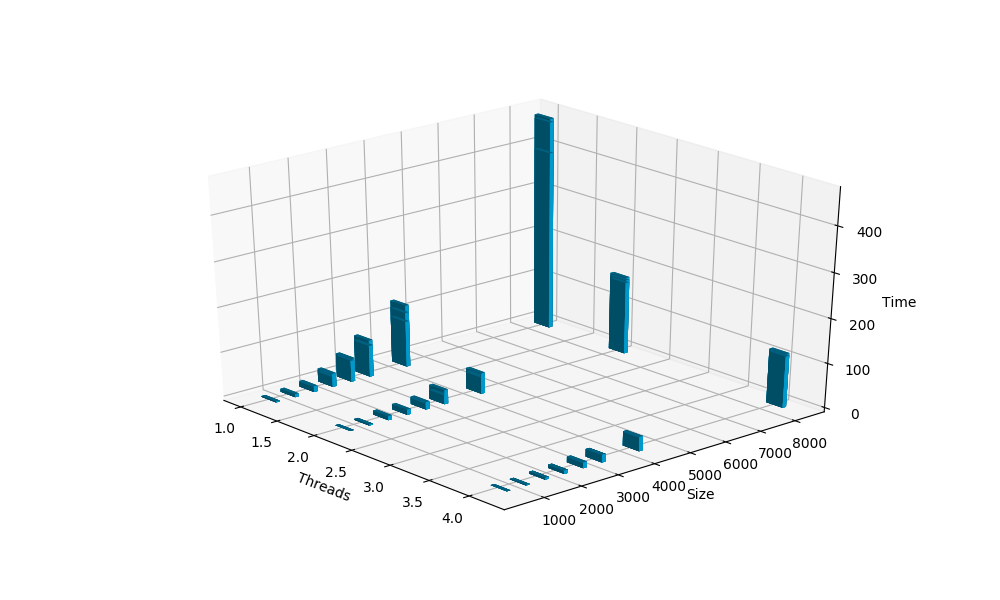
### 3630QM OpenMP



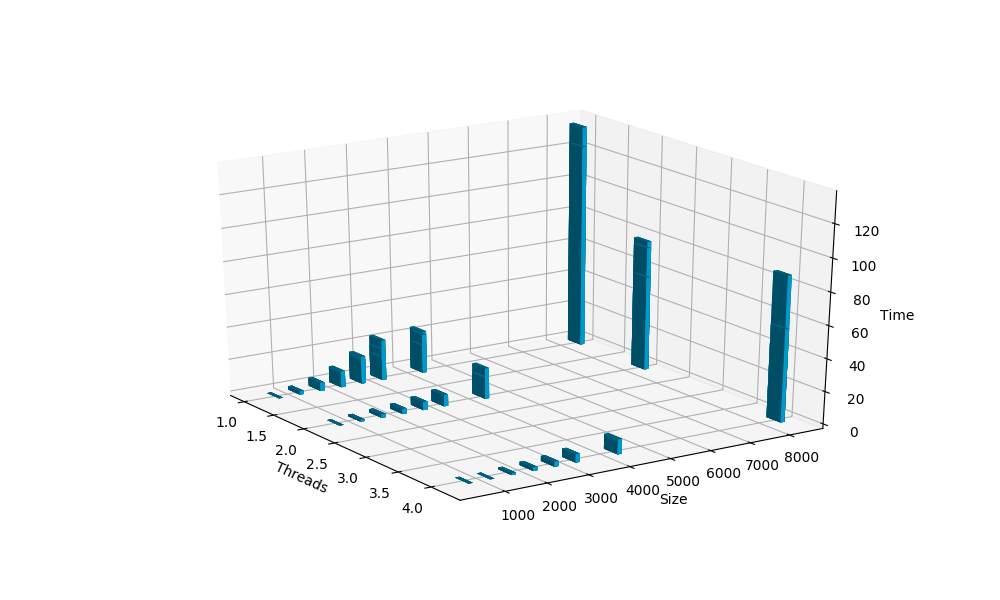
### 3630QM MPI



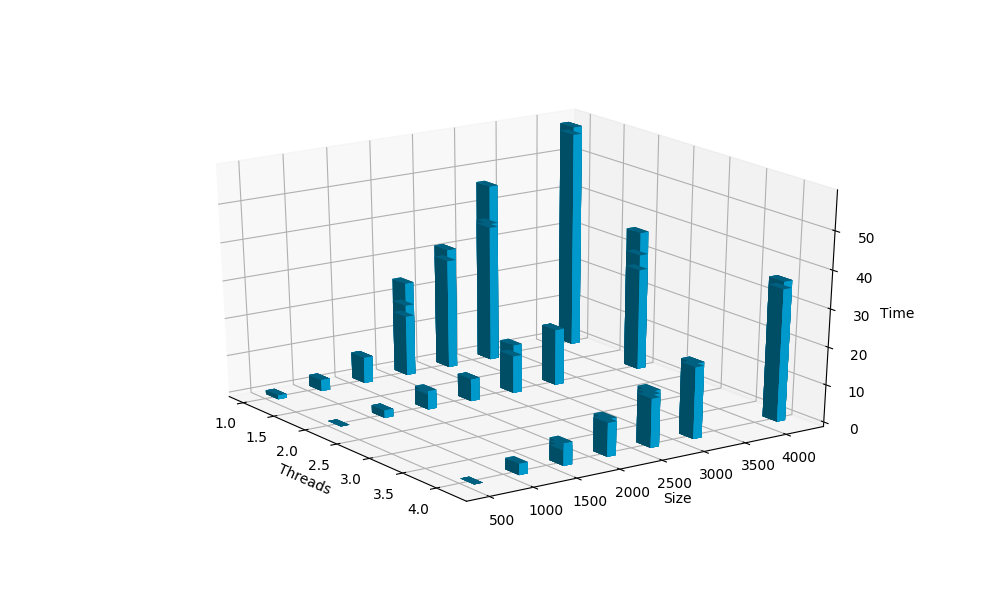
### 3217U OpenMP



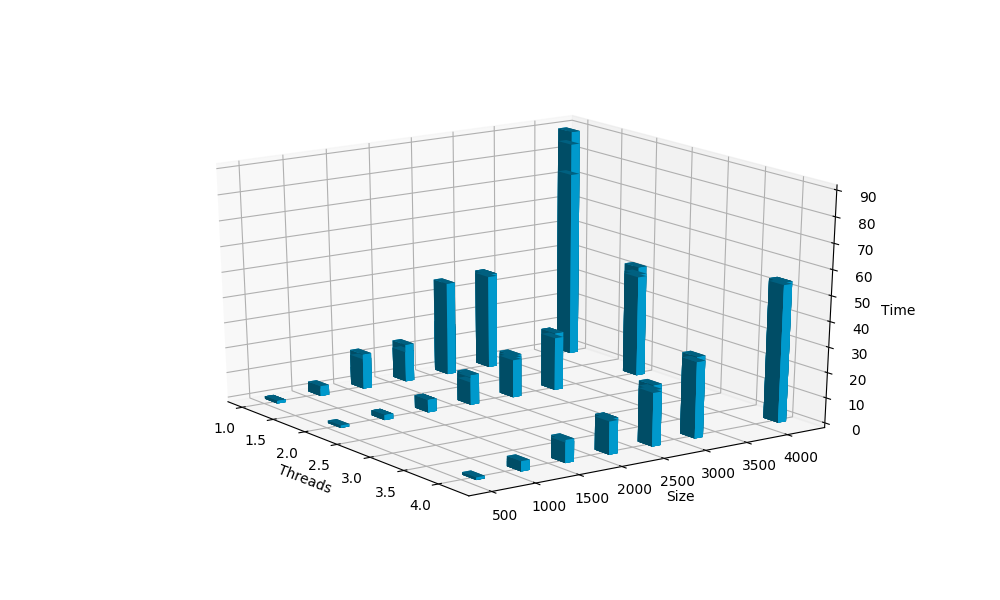
### 3217U MPI



### 370M OpenMP



### 370M MPI



# Комментарии к результатам

Сравнивая OpenMP и MPI получаем очень противоречивые результаты: на суперкомпьютерах на одинаковом количестве нитей и размерах матриц MPI-версии работают в 3-5 раз дольше, чем OpenMP, когда для обычных процессор, кроме 370М, ситуация обратная – MPI-версии работают в 2-3 раза быстрее. Для самого “древнего” 370М процессора MPI-версии уже в 1.5 раза дольше, чем OpenMP. В итоге можно все списать на оптимизации выполнения чего-то там в более свежих Intel-процессорах. Однако самое интересное в том, что MPI реализация на свежих Intel-процессорах получается еще и быстрее обычной последовательной (приблизительно в 2 раза), хотя алгоритмически там по сути должен выполнятся тот же код. А для 3630QM получили вообще аномальный результат – одна MPI нить посчитала 8К матрицу за 34с, когда 8 (!) OpenMP нитей посчитали её за 44с.

6700К быстрее 3630QM на 30-45% для 1-2 потоков и в 2 раза для 4-8, что обусловлено более новой архитектурой и большей максимальной тактовой частотой. 3217U проигрывает 370М в режимах 1 и 2 ядра, однако выигрывает при 4. Проигрыш обоснован разницей тактовой частоты, а выигрыш скорее всего из-за более новой технологии Hyperthreading.

В среднем, hyperthreading дает прирост производительности около 10% для OpenMP и до 50% для MPI для i7, около 30% для 3217U и вообще отрицательный для 370M.

В итоге при любых конфигурациях Regatta проигрывает десктопному 6700K (даже для 16 ядер), однако все еще лучше любых ноутбучных процессоров.

Самая большая производительность очевидная получена при большом количестве ядер на Bluegene (128-512). На 1024 уже получаем просадку, поскольку размер блоков уже слишком маленький и получаем большие накладные расходы на синхронизацию.

В общем случае из-за большой зависимости по данным задача плохо поддается распараллеливанию и максимальная теоретическая продолжительность выполнения приближается к ), где С – время выполнения блока размером TxT одной нитью, N – размер матрицы.

# Выводы

Выполнена работа по разработке параллельной версии метода релаксации Якоби. Изучены технологии написания параллельных алгоритмов OpenMP и MPI. Проанализировано время выполнения алгоритмов на различных вычислительных системах.

Технология OpenMP крайне удобна в использовании, причем дает ощутимый прирост производительности на рассчитанных на многопоточные вычисления системах, в том числе и на персональных компьютерах.

MPI же в свою очередь заточена именно на многопроцессорные системы и наибольшую скорость работы показала именно MPI-реализация запущенная на большом числе вычислителей. Однако, в конкретной текущей задаче, MPI-реализации показали результаты лучше, чем OpenMP и при малом количестве процессоров, однако только на современных процессорах.