****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 перемножения матриц с использованием ленточного алгоритма.**

**ОТЧЕТ**

**о выполненном задании**

студента 321 учебной группы факультета ВМК МГУ

Аграновского Михаила Леонидовича

Москва, 2016 г.

Оглавление

[1 Постановка задачи - 2 -](#_Toc469570567)

[2 Описание алгоритма ленточного умножения матриц - 2 -](#_Toc469570568)

[2.1 Основа: последовательный алгоритм - 2 -](#_Toc469570569)

[2.2 Параллельный алгоритм - 3 -](#_Toc469570570)

[3 Результаты замеров времени выполнения - 3 -](#_Toc469570571)

[3.1 Таблицы - 3 -](#_Toc469570572)

[3.2 3D-графики - 4 -](#_Toc469570573)

[3.2.1 OpenMP на Regatta - 4 -](#_Toc469570574)

[3.2.2 MPI на Regatta - 5 -](#_Toc469570575)

[3.2.3 OpenMP на Bluegene - 5 -](#_Toc469570576)

[3.2.4 MPI на Bluegene - 6 -](#_Toc469570577)

[3.2.5 OpenMP на ноутбуке - 6 -](#_Toc469570578)

[4 Анализ результатов - 7 -](#_Toc469570579)

[5 Выводы - 7 -](#_Toc469570580)

# Постановка задачи

Ставится задача перемножения двух квадратных матриц при помощи т.н. ленточного алгоритма:

 $C= A x B, A, B, C \in Z^{n x n}$

Результатом перемножения матриц А и B является матрица С, каждый элемент которой есть скалярное произведение соответствующих строк матрицы A и столбцов матрицы B.

Требуется:

1. Реализовать параллельные алгоритмы ленточного перемножения матриц с помощью технологий параллельного программирования OpenMP и MPI.
2. Сравнить их эффективность.
3. Исследовать масштабируемость полученных программ и построить графики зависимости времени выполнения программ от числа используемых ядер и объёма входных данных.

# Описание алгоритма ленточного умножения матриц

## Основа: последовательный алгоритм

Простейшая форма алгоритма ленточного умножения матриц имеет следующий вид:

void ribbonMult(int \*a, int \*b, int \*c, int n)

{

 for (int i = 0; i < n; i++) {

 for (int j = 0; j < n; j++) {

 auto pElem = &(c[i \* n + j]);

 \*pElem = 0;

 for (int k = 0; k < n; k++) {

 \*pElem += a[i \* n + k] \* b[k \* n + j];

 }

 }

 }

}

Этот алгоритм является итеративным и ориентирован на последовательное вычисление строк матрицы С. Предполагается выполнение n\*n\*n операций умножения и столько же операций сложения элементов исходных матриц. Количество выполненных операций имеет порядок O(n\*\*3).

## Параллельный алгоритм

Разбиваем задачу вычисления конечной матрицы на подзадачи по вычислению строк и распределяем их по потокам (OpenMP) / процессам (MPI). Разбиение на вычисление отдельных полей не производим, т.к. размеры матриц при вычислениях и так будут на порядки превышать число потоков (процессов).

Ниже приведены краткие заметки по реализациям параллельных алгоритмах на OpenMP и MPI. Коды программ можно найти в github-репозитории: <https://github.com/agrml/ribbonMultiplicationSummary>

В OpenMP-версии вся модификация когда сводится к добавлению клаузы ***omp parallel for.***

В MPI-версии производится широковещательная рассылка заполненных матриц ***a, b***, а в конце работы – ***reduce*** результатов. Для синхронизации используются команды ***MPI\_Barrier.***

Реализованные алгоритмы проверялись на корректность и совпадение по результатам с последовательным алгоритмом (соответствующий код закомментирован).

# Результаты замеров времени выполнения

Ниже приведены результаты замеров времени программ на суперкомпьютерах Bluegene и Regatta: непосредственно в табличной форме и наглядно на 3D-графиках.

Программа была запущена в конфигурациях:

* на Regatta - 1,2,4,8,16 ядер для MPI и OpenMP-программы;
* на Bluegene - 1,2,4 для OpenMP; 1,2,4,8,16,32,64,128,256 для MPI.

Также для сравнения OpenMP-версия программы была запущена на ноутбуке (Core i5-6300HQ 2.30GHz × 4, 24GB RAM, Ubuntu 16.04)

Каждая конфигурация была запущена 3 раза. Ниже приведены усредненные результаты.

## Таблицы









--



## 3D-графики

### OpenMP на Regatta



### MPI на Regatta



### OpenMP на Bluegene



### MPI на Bluegene



Замечание к графику: может показаться, что на размере матрицы в 2560 строк скорость выполнения выросла. Это не так: на графике нету данных по вычислению на 1 процессоре по причине превышения ограничения по времени (см. таблицу).

### OpenMP на ноутбуке



# Анализ результатов

На одинаковых конфигурациях OpenMP показал результаты лучшие, чем MPI. Однако в абсолютном зачете побеждает MPI на 256 процессорах Bluegene. Так как система заточена под прогопроцессорные вычисления (множество относительно слабых процессоров), результаты запуска на Bluegene OpenMP-версии не впечатляют. В практических целях OpenMP (да и любые другие многопоточные (multithread) программы) следует выполнять на Regatta.

Заметим, что задача прекрасна поддалась распараллеливанию и зависимость скорости работы от числа вычислителей близка к линейной.

В сравнении с временем работы на ноутбуке принципиальный выигрыш дает выполнение на 256 вычислителях Bluegene. Заметим, что на ПК уже зависимость времени работы от числа потоков не линейная, а переход от 4 потоков (число ядер процессора, Intel Hiperthreading отсутствует) к 8 сопровождается спадом производительности. На суперкомпьютерах данную ситуацию поймать не удалось.

# Выводы

Выполнена работа по разработке параллельной версии алгоритма ленточного умножения матриц. Изучены технологии написания параллельных алгоритмов OpenMP и MPI. Проанализировано время выполнения алгоритмов на различных вычислительных системах.

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

MPI можно назвать более низкоуровневой технологией: разработка MPI-программы знакомит с основами взаимодействия вычислительных узлов суперкомпьютера. При этом MPI заточена именно на многопроцессорные системы и наибольшую скорость работы показала именно MPI-реализация, запущенная на наибольшем числе вычислителей суперкомпьютера Bluegene.