Практикум по курсу

“Суперкомпьютеры и параллельная обработка данных”

Разработка параллельной версии программы для вычисления обратной матрицы методом Гаусса.

Отчет студента 325 группы факультета ВМК МГУ

Шешукова Алексея

Москва, 2020

Постановка задачи

Требуется методом Гаусса найти обратную матрицу:

$$A^{-1}: A^{-1} * A = I,\ A,\ I \in R^{n \times n}$$

Требуется:

1. Реализовать параллельные алгоритмы поиска обратной матрице с помощью технологий параллельного программирования OpenMP и MPI.
2. Сравнить их эффективность.
3. Исследовать масштабируемость полученный программ.

Описание алгоритма для поиска обратной матрицы

Последовательный алгоритм

Построение обратной матрицы методом Гаусса заключается в следующем:

1. Присоединяем к исходной матрице единичную. Все элементарные преобразования, которые мы будем делать с исходной матрицей, мы будем делать и с единичной.
2. Приводим исходную матрицу к единичной с помощью элементарных преобразований. Для этого используем метод Гаусса: вычитаем строку с индексом i из всех остальных с таким коэффициентом, чтобы в i-ом столбце было только одно число, не равно нулю, и находилось в i-ой строке. Если элемент на позиции [i, i] равен нулю, меняем i-ую строку с местами со строкой, в которой i-ый элемент не нулевой. Если такой строки нет, то определитель матрицы равен нулю и обратной матрицы не существует.
3. Когда получили диагональную матрицу, каждую строку делим на единственный ненулевой элемент в ней, получая единичную матрицу. При этом присоединенная матрица в результате этих преобразований стала обратной к исходной.

Реализация этого алгоритма выглядит следующим образом:

```
double **inverseMatrix(double **m_orig, size_t n) {
   double **m = copyMatrix(m_orig, n);
   double **I = initMatrix(n , IDENTITY);
   int i, j, k;
   for (j = 0; j < n; j++) {
       if (m[j][j] == 0) {
           bool flag = true;
           for (k = j; k < n; k++) {
               if (m[k][j] != 0) {
                   swapRows(m, j, k, n);
                   flag = false;
                   break;
               }
           }
           if (flag) {
               throw ZERO_DET;
           }
       }
       double div = m[j][j];
       for (int i = 0; i < n; i++) {
           m[j][i] /= div;
           I[j][i] /= div;
       }
       for (int i = 0; i < n; i++) {
           if (i == j)
               continue;
               double c = m[i][j];
           for (int k = 0; k < n; k++) {
               m[i][k] -= c * m[j][k];
               I[i][k] -= c * I[j][k];
           }
       }
   }
   return I;
}
```

Сложность этого алгоритма для матрицы размера n * n составляет $O(n^3)$.

Параллельный алгоритм

Задачу вычитания строки, умноженной на коэффициент, из остальных строк можно распределить на потоки (OpenMP) / процессы (MPI). Для

этого у каждого процесса/потока должен быть доступ к своей части матрицы.

В OpenMP это решается добавлением директивы **omp parallel for shared(m, I) private(i, j, k)**.

В MPI создается общая матрица размера n * 2n, состоящая из исходной и присоединенной единичной.

1. Выполняется широковещательная рассылка исходной матрицы всем процесса
2. Для каждого процесса вычисляются границы, в которых он будет обрабатывать строки (вычитать ведущую строку)
3. Начинаем обрабатывать строки по порядку, процесс, которому принадлежит обрабатываемая строка, отправляет её другим с помощью **MPI_Send**, они получают её через **MPI_Recv** и вычитают с нужным коэффициентом из принадлежащих им строк.
4. В конце каждый процесс возвращает в качестве результата свою часть единичной матрицы.

Синхронизация реализована с помощью **MPI_Barrier**.

Исходный код программ можно найти в репозитории: github.com/alexeyshesh/InverseMatrixParallel.

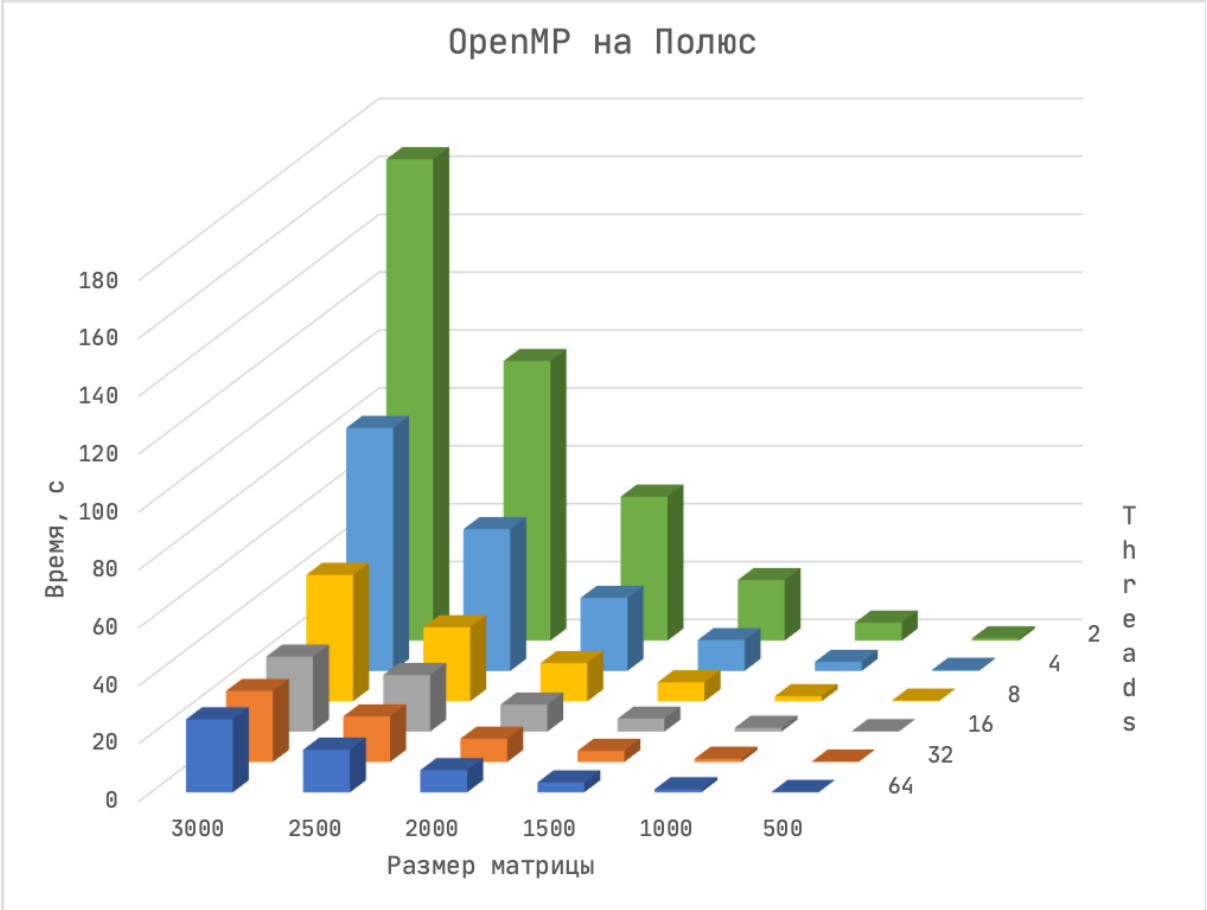
Результаты замеров времени выполнения

Замеры времени производились на суперкомпьютерах “Полюс” и Bluegene. Программы были запущены по три раза, в таблицах приведено усредненное время.

Полюс OpenMP

Размер матрицы	Количество потоков	Время, с
500	2	0,811974
500	4	0,397317
500	8	0,262807
500	16	0,179914
500	32	0,157514

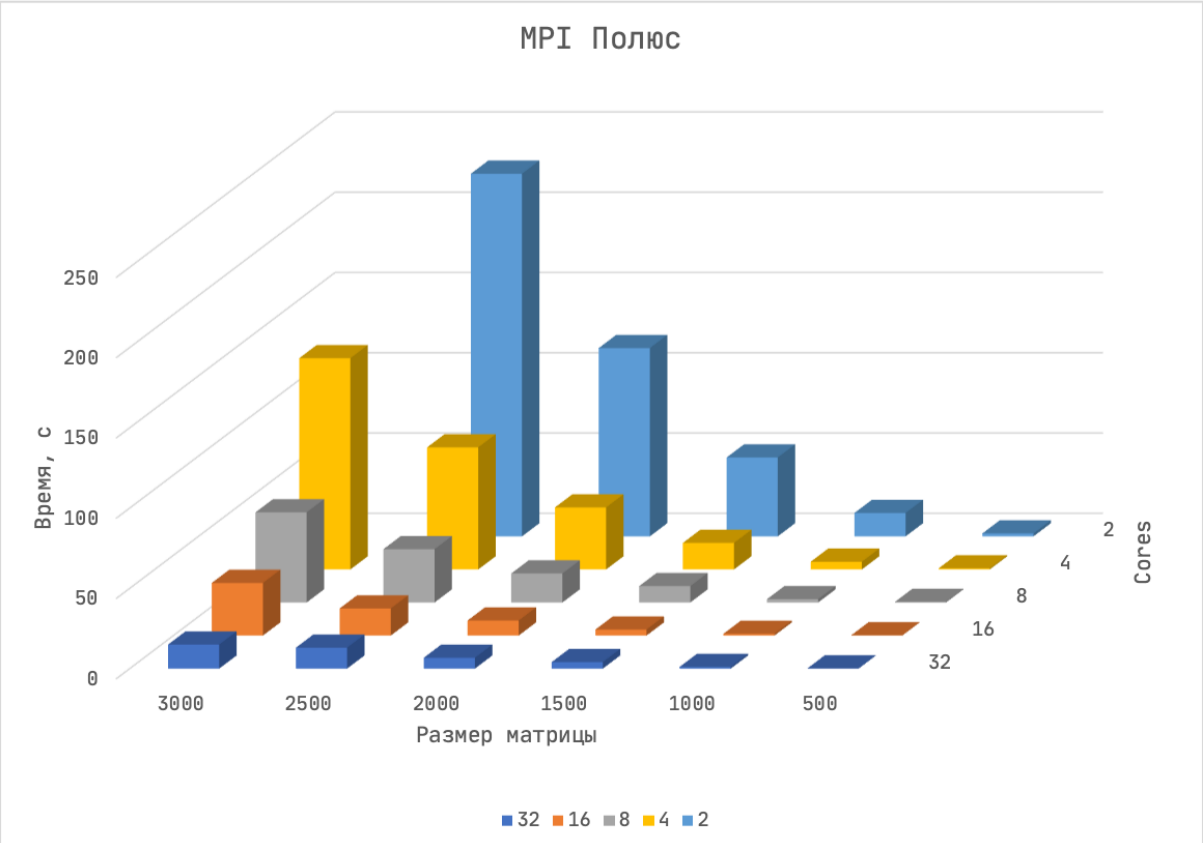
500	64	0,150024
1000	2	6,22197
1000	4	3,14688
1000	8	1,75769
1000	16	1,33029
1000	32	1,14029
1000	64	1,06858
1500	2	20,9363
1500	4	10,68
1500	8	6,58727
1500	16	4,49494
1500	32	3,85913
1500	64	3,39177
2000	2	49,7446
2000	4	25,2439
2000	8	13,1844
2000	16	9,3293
2000	32	8,01861
2000	64	7,64229
2500	2	96,5835
2500	4	48,9868
2500	8	25,5925
2500	16	19,5903
2500	32	15,7438
2500	64	14,7059
3000	2	166,332
3000	4	83,9897
3000	8	43,6901
3000	16	25,9238
3000	32	24,5511
3000	64	25,2194



Полюс MPI

Размер матрицы	Cores	Время, с
500	2	1,81872
500	4	0,669414
500	8	0,352541
500	16	0,150697
500	32	0,130442
1000	2	14,5885
1000	4	4,81829
1000	8	2,0879
1000	16	1,0381
1000	32	1,03756
1500	2	49,141
1500	4	16,5171
1500	8	10,2052

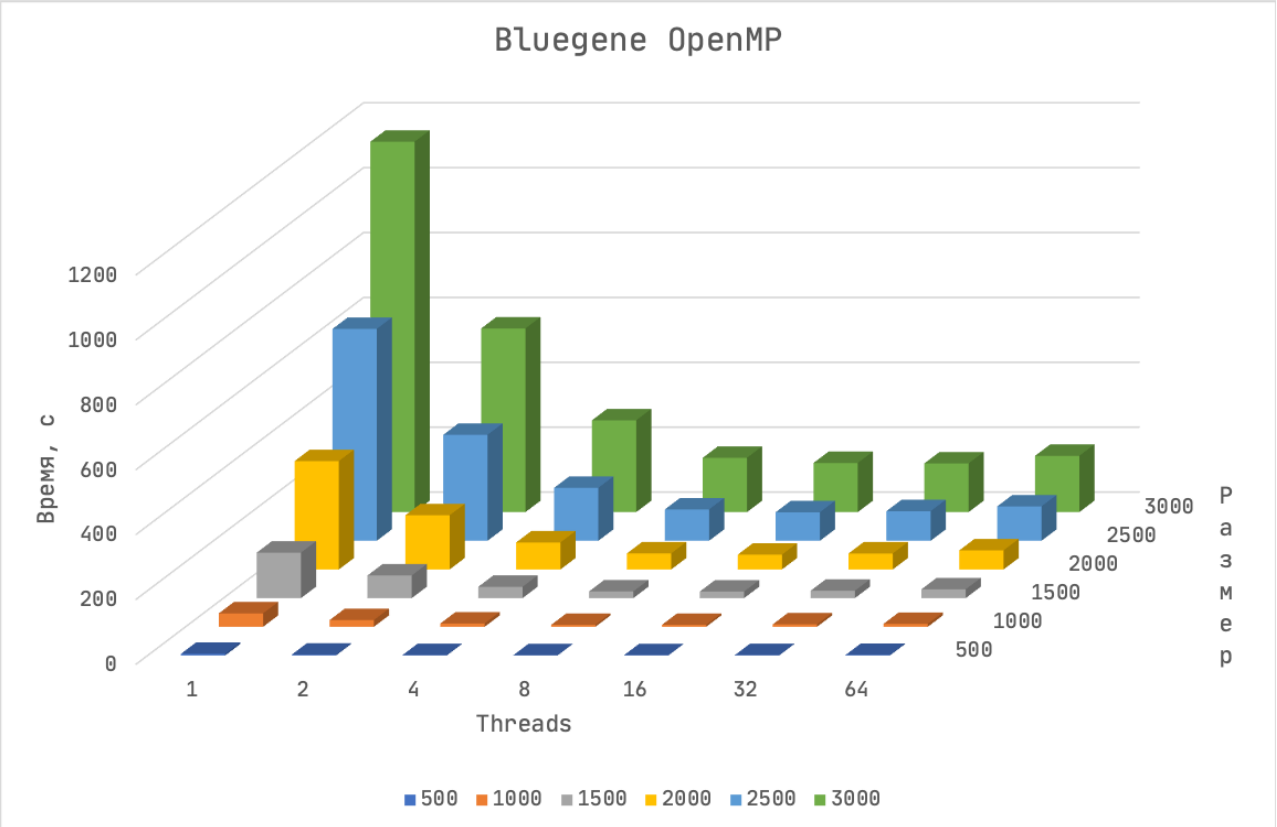
1500	16	3,75393
1500	32	4,06463
2000	2	117,276
2000	4	38,6415
2000	8	18,0877
2000	16	9,36737
2000	32	6,77579
2500	2	225,87
2500	4	76,0485
2500	8	33,1967
2500	16	16,8918
2500	32	13,0554
3000	4	131,585
3000	8	56,1771
3000	16	32,7888
3000	32	14,9901
3000	64	10,331



Результата измерения времени для вычисления матрицы размера 3000 * 3000 на двух ядрах нет, так как оно превосходит ограничение по времени.

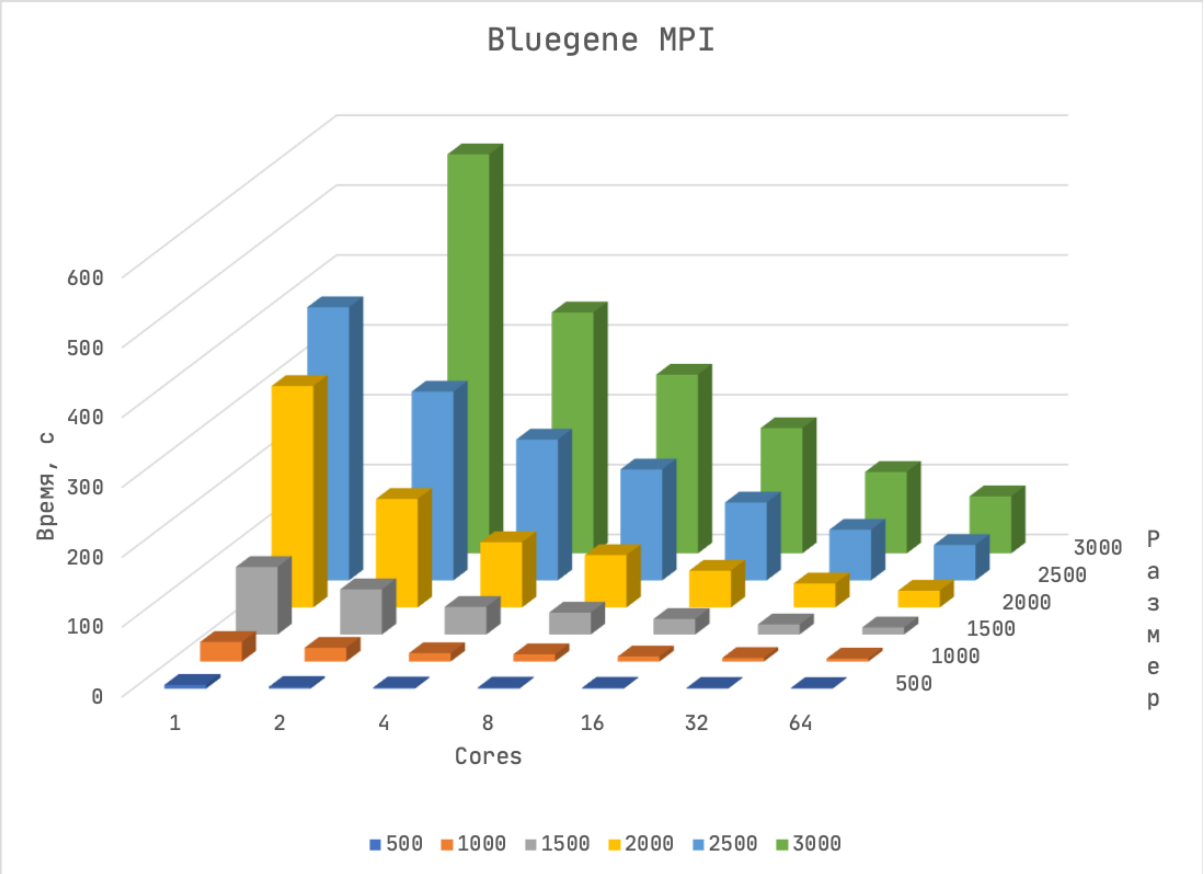
Bluegene OpenMP

		Размер матрицы					
		500	1000	1500	2000	2500	3000
Ядра	1	4,969943	41,29697	140,179	334,0277	652,938	1141,65
	2	2,405697	20,58657	69,7923	166,9587	326,4567	565,9907
	4	1,1306	10,21883	34,9141	83,1685	163,0107	282,865
	8	0,608896	5,976497	20,66897	49,46783	96,57247	167,3293
	16	0,918648	5,978103	20,24513	45,63447	87,83873	151,3937
	32	1,043303	7,43392	22,738	49,2503	91,2172	149,964
	64	1,650917	8,9185	27,13803	58,77997	105,9483	173,4917



Bluegene MPI

		Размер матрицы					
		500	1000	1500	2000	2500	3000
Ядра	1	5,31935	28,0028	96,6259	316,941	350,9871	TL
	2	2,49691	19,6554	64,6369	155,325	270,2389	571,138
	4	1,59875	11,9871	39,601	93,4508	201,5634	344,9
	8	1,42164	10,4162	31,3989	74,8963	158,877	255,895
	16	1,20082	7,20521	22,2532	52,7014	111,488	179,239
	32	1,02134	5,17115	14,5041	34,2955	72,5067	116,545
	64	0,979639	4,24981	10,1868	24,0407	50,7886	81,6154



Анализ результатов

Лучший результат на матрице размера 3000 показала программа, написанная на MPI на “Полюс” на 64 ядрах.

На Bluegene лучшие результаты показала программа на MPI, так как суперкомпьютер создан для многопроцессный вычислений. На графике OpenMP виден прогиб, который может быть связан с тем, что Bluegene заточен под многопроцессные вычисления.

Программу достаточно сложно эффективно распараллелить, так как постоянно нужно взаимодействие всех частей матрицы со всеми.

Выводы

Был реализован алгоритм вычисления обратной матрицы методом Гаусса. параллельные версии программы были реализованы с использованием библиотек OpenMP и MPI. Их работа была протестирована на суперкомпьютерах Полюс и Bluegene.

Сделать параллельную программу с помощью OpenMP значительно проще, и при этом появляется заметный прирост в скорости вычислений.

Распараллелить программу с помощью MPI значительно сложнее, так как это более низкоуровневая технология и важно понимать, как должны взаимодействовать процессы. Но результат в итоге получается лучше, чем на OpenMP.