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 -
2 Описание алгоритма метода простой итерации Якоби ..... - 2 -
2.1 Последовательный алгоритм ..... - 2 -
2.2 Параллельный алгоритм ..... - 2 -
3 Результаты замеров времени выполнения ..... - 3 -
3.1 Таблицы ..... - 4 -
3.1.1 OpenMP ..... - 4 -
3.1.2 MPI ..... - 6 -
3.2 3D-графики ..... - 7 -
3.2.1 Bluegene OpenMP ..... - 7 -
3.2.2 Bluegene MPI ..... - 8 -
3.2.3 Regatta OpenMP ..... - 9 -
3.2.4 Regatta MPI ..... - 9 -
3.2.5 6700K OpenMP ..... - 10 -
3.2.6 6700K MPI ..... - 10 -
3.2.7 3630QM OpenMP ..... - 10 -
3.2.8 3630QM MPI ..... - 11 -
3.2.9 3217U OpenMP ..... - 11 -
3.2.10 3217U MPI ..... - 11 -
3.2.11 370M OpenMP ..... - 12 -
3.2.12 370M MPI ..... - 12 -
4 Комментарии к результатам ..... - 12 -
5 Выводы ..... - 13 -

1 Постановка задачи

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

$$x_{i,j}^{new} = \left(x_{i-1,j}^{old} + x_{i,j-1}^{old} + x_{i+1,j}^{old} + x_{i,j+1}^{old}\right)/4$$

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.

2 Описание алгоритма метода простой итерации Якоби

2.1 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 A. Предполагается выполнение $4n^2$ операций сложения и n^2операций деления элементов исходной матрицы. Количество выполненных операций имеет порядок O(n^2).

2.2 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 $N / (8 * T)$, 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 N / T, 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:

0	1	2	3	4
0	1	2	3	3
0	1	2	2	2
0	1	1	1	1
0	0	0	0	0

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

3 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

3.1.1 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.

Threads	Size	Bluegene	Regatta	i7 6700K	i7 3630QM	i3 3217U	i3 370M	Bluegene	Regatta	i7 6700K	i7 3630QM	i3 3217U	i3 370M
1	512	1,636	0,756	0,334	0,497	2,148	1,401						
	1024	5,915	2,948	1,310	1,883	5,956	3,590						
	1536	12,838	6,599	3,073	4,376	11,047	7,900						
	2048	22,326	15,375	5,579	7,895	26,594	20,643						
	2560	34,484	20,032	8,858	12,290	51,798	31,761						
	3072	49,219	30,163	12,603	17,052	75,498	40,510						
	4096	86,676	61,922	22,097	30,962	122,580	58,785						
	8192	344,373	249,681	86,044	120,888	445,078							
	16384		900,000	339,089	462,472								
	32768			1306,806	1807,370								
2	512	0,917	0,414	0,179	0,243	0,610	0,473	1,79	1,83	1,87	2,05	3,52	2,96
	1024	3,090	1,516	0,692	0,953	2,238	2,105	1,91	1,95	1,89	1,98	2,66	1,71
	1536	6,611	3,365	1,581	2,161	8,135	4,385	1,94	1,96	1,94	2,03	1,36	1,80
	2048	11,435	7,798	2,848	3,820	10,005	6,730	1,95	1,97	1,96	2,07	2,66	3,07
	2560	17,623	9,820	4,477	6,016	15,694	11,457	1,96	2,04	1,98	2,04	3,30	2,77
	3072	25,125	14,963	6,454	8,563	24,644	15,187	1,96	2,02	1,95	1,99	3,06	2,67
	4096	44,258	32,316	11,431	15,884	41,902	32,385	1,96	1,92	1,93	1,95	2,93	1,82
	8192	175,357	123,590	45,020	66,832	168,421		1,96	2,02	1,91	1,81	2,64	
	16384		480,940	176,095	291,367				1,87	1,93	1,59		
	32768			706,961	1137,351					1,85	1,59		
4	512	0,543	0,323	0,123	0,146	0,639	0,704	3,01	2,34	2,71	3,40	3,36	1,99
	1024	1,674	1,041	0,473	0,604	1,949	2,433	3,53	2,83	2,77	3,12	3,06	1,48
	1536	3,513	1,946	1,065	1,236	4,363	5,526	3,65	3,39	2,89	3,54	2,53	1,43
	2048	6,027	3,813	1,822	2,875	7,735	9,835	3,70	4,03	3,06	2,75	3,44	2,10
	2560	9,250	4,937	2,676	4,893	12,407	14,664	3,73	4,06	3,31	2,51	4,17	2,17
	3072	13,152	8,167	3,944	7,194	17,199	20,127	3,74	3,69	3,20	2,37	4,39	2,01
	4096	23,260	19,421	6,706	12,198	31,987	36,826	3,73	3,19	3,30	2,54	3,83	1,60
	8192	91,203	70,894	26,286	47,018	118,983		3,78	3,52	3,27	2,57	3,74	
	16384		281,464	91,882	186,267				3,20	3,69	2,48		
	32768			363,920	704,623					3,59	2,57		
8	512		0,529	0,111	0,212				1,43	3,00	2,34		
	1024		1,461	0,395	0,722				2,02	3,32	2,61		
	1536		3,967	0,858	1,620				1,66	3,58	2,70		
	2048		4,814	1,514	2,859				3,19	3,68	2,76		
	2560		5,973	2,298	4,439				3,35	3,85	2,77		
	3072		5,754	3,289	6,530				5,24	3,83	2,61		
	4096		12,832	5,893	11,364				4,83	3,75	2,72		
	8192		37,547	24,562	44,391				6,65	3,50	2,72		
	16384		141,854	85,756	173,321				6,34	3,95	2,67		
	32768			329,949	644,075					3,96	2,81		
16	512		3,981						0,19				
	1024		3,442						0,86				
	1536		2,822						2,34				
	2048		2,595						5,93				
	2560		3,773						5,31				
	3072		5,662						5,33				
	4096		6,234						9,93				
	8192		23,665						10,55				
	16384		92,232						9,76				

3.1.2 MPI

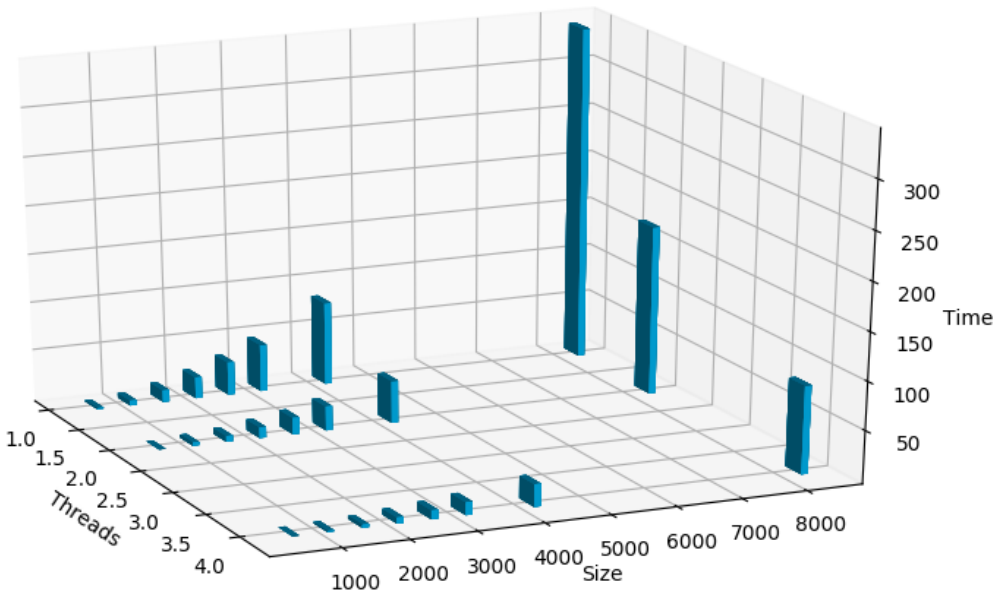
Threads	Size	Bluegene	Regatta	i7 6700K	i7 3630QM	i3 3217U	i3 370M	Bluegene	Regatta	i7 6700K	i7 3630QM	i3 3217U	i3 370M
1	512	4,238	3,037	0,113	0,151	0,683	1,773						
	1024	16,936	12,140	0,464	0,574	2,133	3,691						
	1536	39,120	27,291	1,040	1,264	5,564	11,421						
	2048	69,535	51,070	1,823	2,188	9,621	13,585						
	2560	108,639	76,038	2,821	3,386	16,127	25,901						
	3072	156,422	109,751	4,053	4,887	19,647	37,909						
	4096	278,085	205,703	7,210	8,676	22,804	83,354						
	8192	900,000	852,534	29,097	34,658	132,403							
	16384		900,000										
2	512	3,238	2,330	0,088	0,120	0,432	0,962	1,31	1,30	1,28	1,26	1,58	1,84
	1024	12,951	9,151	0,358	0,448	1,274	2,521	1,31	1,33	1,30	1,28	1,67	1,46
	1536	29,140	20,433	0,804	0,986	2,101	5,204	1,34	1,34	1,29	1,28	2,65	2,19
	2048	65,564	38,277	1,433	1,766	3,457	11,136	1,06	1,33	1,27	1,24	2,78	1,22
	2560	82,426	56,588	2,224	2,748	5,425	16,433	1,32	1,34	1,27	1,23	2,97	1,58
	3072	141,231	82,000	3,201	3,857	7,728	22,874	1,11	1,34	1,27	1,27	2,54	1,66
	4096	228,563	153,930	5,676	6,832	15,816	42,387	1,22	1,34	1,27	1,27	1,44	1,97
	8192	900,000	614,147	22,592	27,001	71,726		1,00	1,39	1,29	1,28	1,85	
	16384		900,000						1,00				
4	512	1,915	1,490	0,052	0,069	0,178	1,013	2,21	2,04	2,16	2,21	3,83	1,75
	1024	7,662	5,506	0,216	0,278	0,652	3,656	2,21	2,20	2,14	2,06	3,27	1,01
	1536	13,492	12,124	0,489	0,630	1,348	8,492	2,90	2,25	2,12	2,01	4,13	1,34
	2048	28,710	22,649	0,861	1,422	2,341	14,164	2,42	2,25	2,12	1,54	4,11	0,96
	2560	45,426	33,356	1,341	2,208	3,847	22,886	2,39	2,28	2,10	1,53	4,19	1,13
	3072	72,682	48,182	1,923	3,152	5,485	31,420	2,15	2,28	2,11	1,55	3,58	1,21
	4096	111,231	90,241	3,393	5,564	9,091	55,500	2,50	2,28	2,13	1,56	2,51	1,50
	8192	444,922	360,084	13,471	21,870	64,743		2,02	2,37	2,16	1,58	2,05	
	16384		900,000						1,00				
8	512	1,079	1,146	0,035	0,072			3,93	2,65	3,21	2,09		
	1024	4,103	3,461	0,141	0,269			4,13	3,51	3,28	2,13		
	1536	8,252	6,962	0,315	0,541			4,74	3,92	3,30	2,34		
	2048	19,140	12,686	0,547	0,921			3,63	4,03	3,33	2,37		
	2560	32,980	18,410	0,829	1,412			3,29	4,13	3,40	2,40		
	3072	56,067	26,317	1,180	2,081			2,79	4,17	3,44	2,35		
	4096	86,085	49,122	2,094	3,902			3,23	4,19	3,44	2,22		
	8192	363,408	194,421	7,969	13,510			2,48	4,38	3,65	2,57		
	16384		773,908						1,16				
16	512	0,645	1,625					6,57	1,87				
	1024	2,241	2,693					7,56	4,51				
	1536	4,864	4,544					8,04	6,01				
	2048	8,507	7,534					8,17	6,78				
	2560	15,980	10,482					6,80	7,25				
	3072	27,167	14,681					5,76	7,48				
	4096	42,308	26,543					6,57	7,75				
	8192	172,682	102,972					5,21	8,28				
	16384		408,611						2,20				

Threads	Size	Bluegene		Threads	Size	Bluegene		Threads	Size	Bluegene	
32	512	0,442	9,586	256	512	1,770	2,394	64	512	0,441	9,605
	1024	1,320	12,827		1024	1,888	8,972		1024	0,908	18,658
	1536	2,686	14,563		1536	2,145	18,234		1536	1,633	23,960
	2048	4,582	15,177		2048	2,429	28,625		2048	2,699	25,764
	2560	6,980	15,564		2560	2,852	38,099		2560	3,939	27,584
	3072	9,902	15,796		3072	3,271	47,818		3072	5,439	28,757
	4096	17,308	16,067		4096	4,332	64,198		4096	9,264	30,019
	8192	69,231	13,000		8192	11,509	78,202		8192	35,001	25,713
128	512	0,722	5,868	512	512	5,209	0,814	1024	512	17,178	0,247
	1024	0,991	17,089		1024	5,320	3,183		1024	17,487	0,968
	1536	1,421	27,533		1536	5,425	7,212		1536	17,505	2,235
	2048	1,953	35,600		2048	5,563	12,501		2048	17,767	3,914
	2560	2,594	41,884		2560	5,772	18,822		2560	17,763	6,116
	3072	3,396	46,058		3072	6,016	25,999		3072	17,948	8,715
	4096	5,529	50,291		4096	6,593	42,181		4096	18,250	15,238
	8192	18,717	48,084		8192	10,382	86,685		8192	20,226	44,498

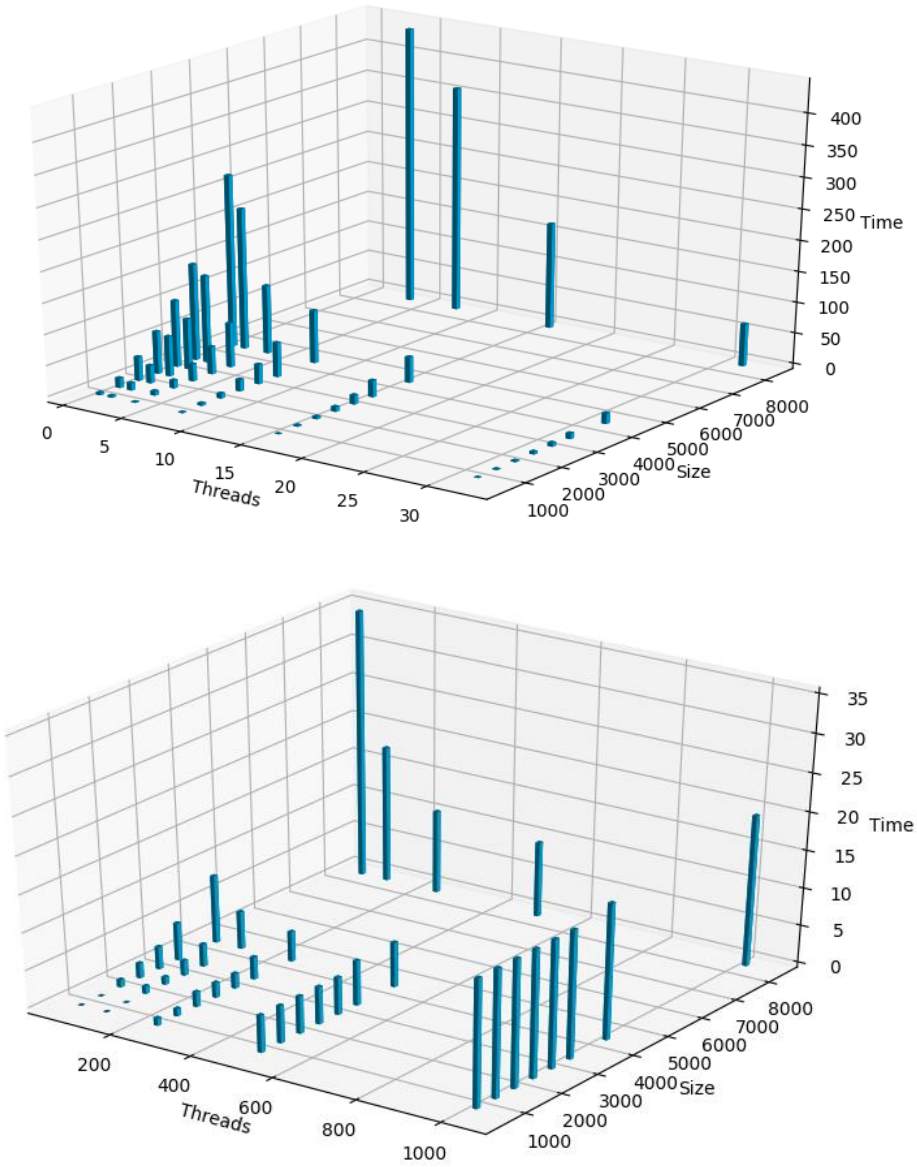
3.2 3D-графики

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

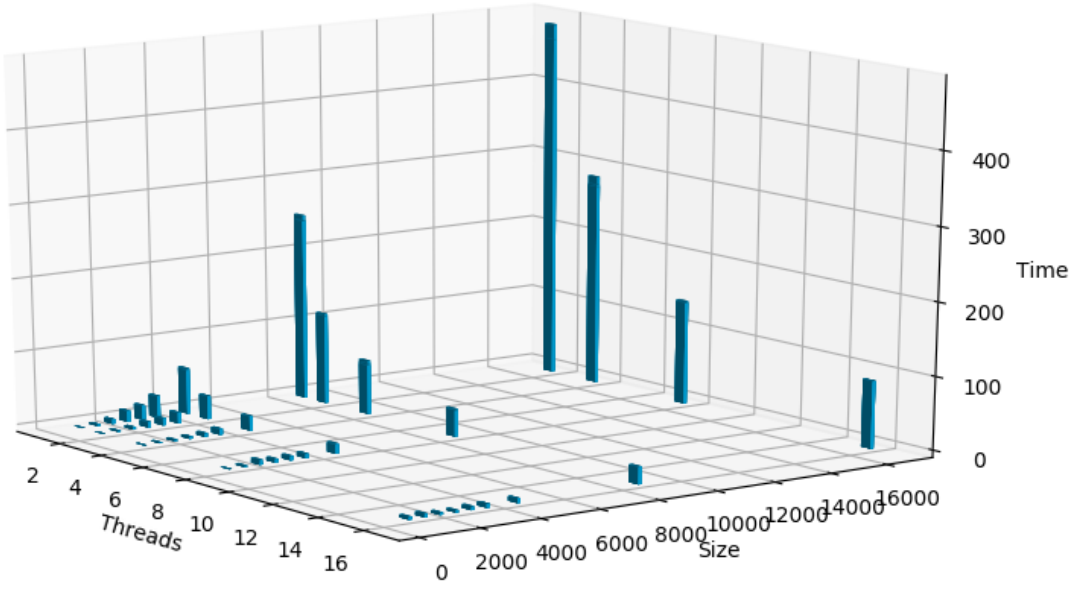
3.2.1 Bluegene OpenMP



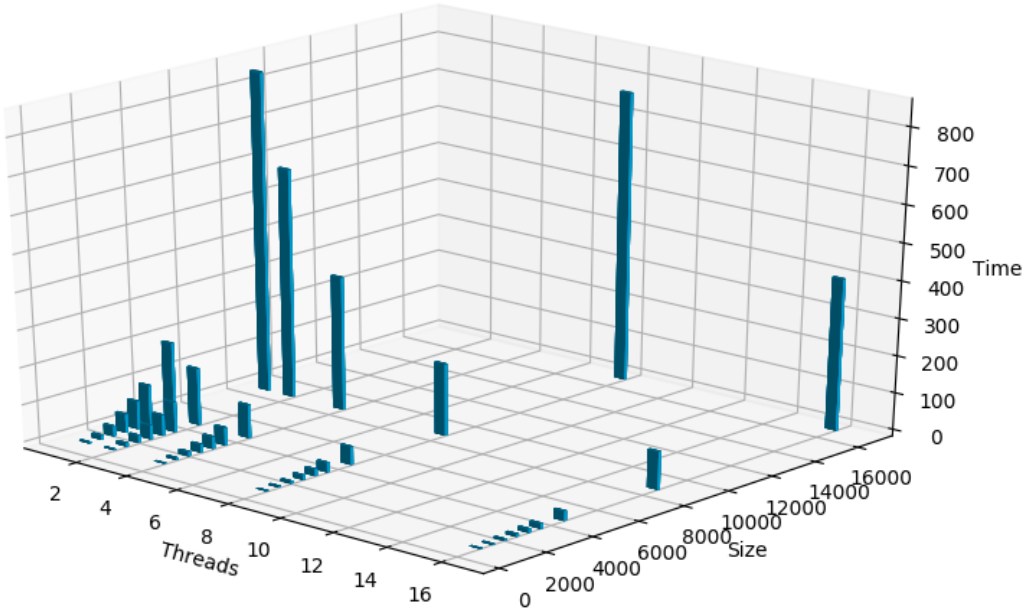
3.2.2 Bluegene MPI



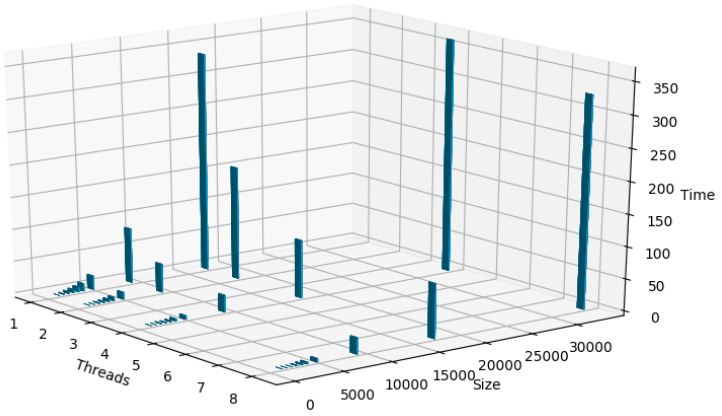
3.2.3 Regatta OpenMP



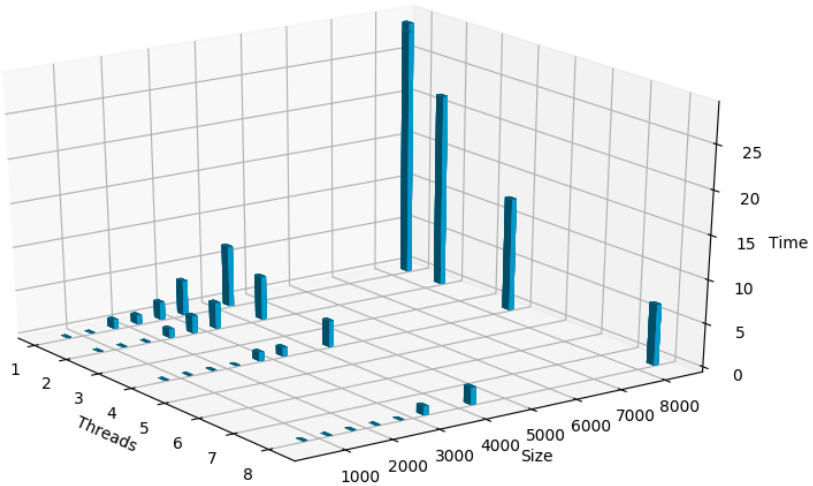
3.2.4 Regatta MPI



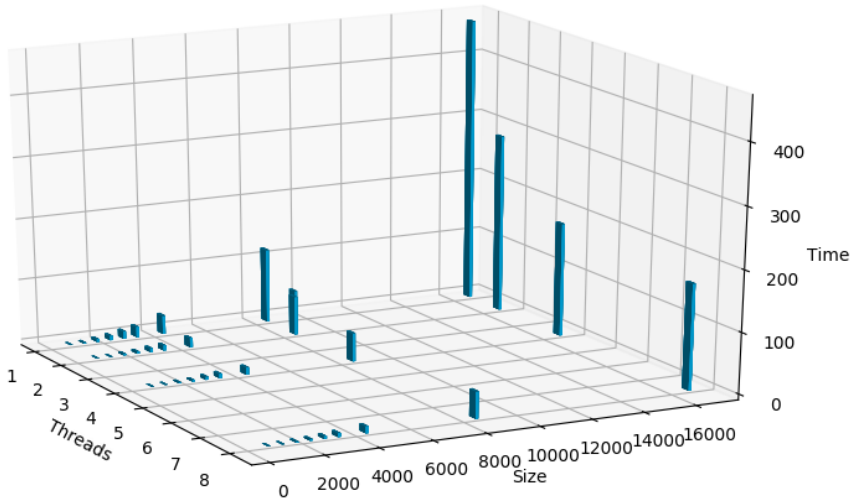
3.2.5 6700K OpenMP



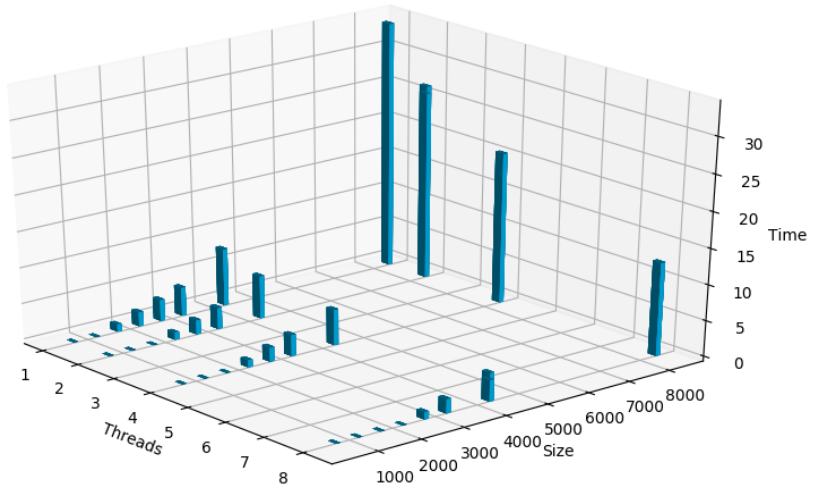
3.2.6 6700K MPI



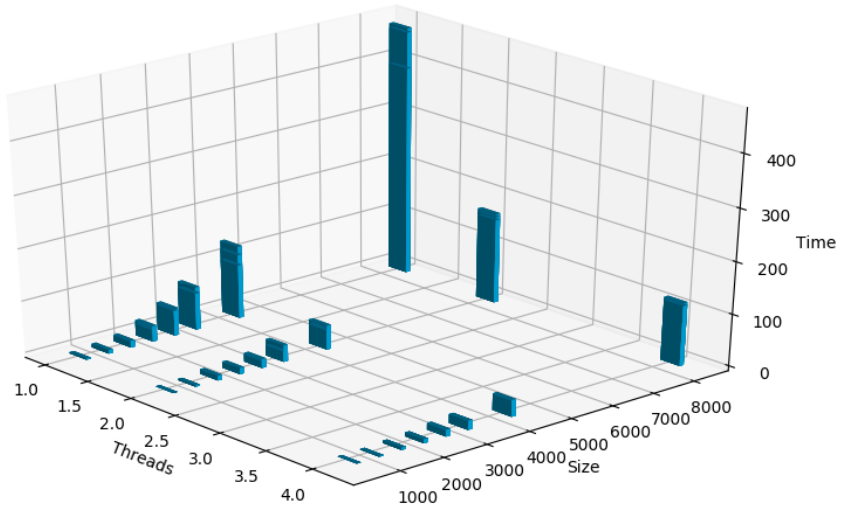
3.2.7 3630QM OpenMP



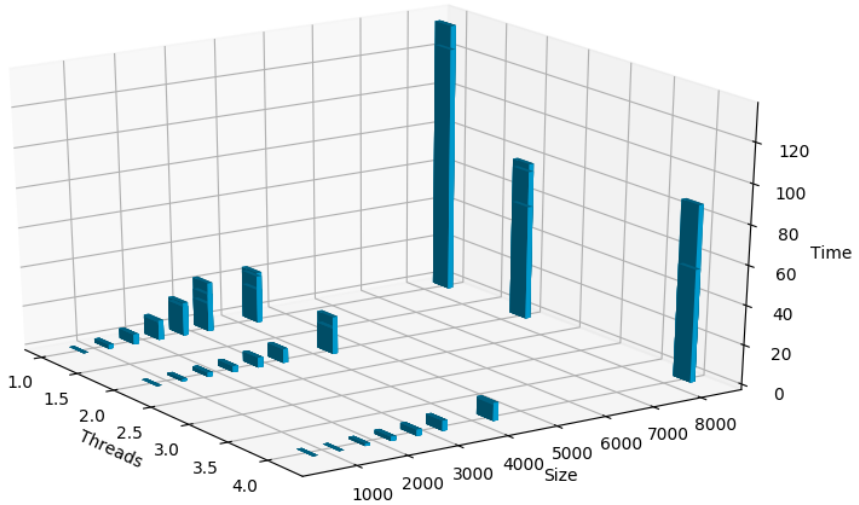
3.2.8 3630QM MPI



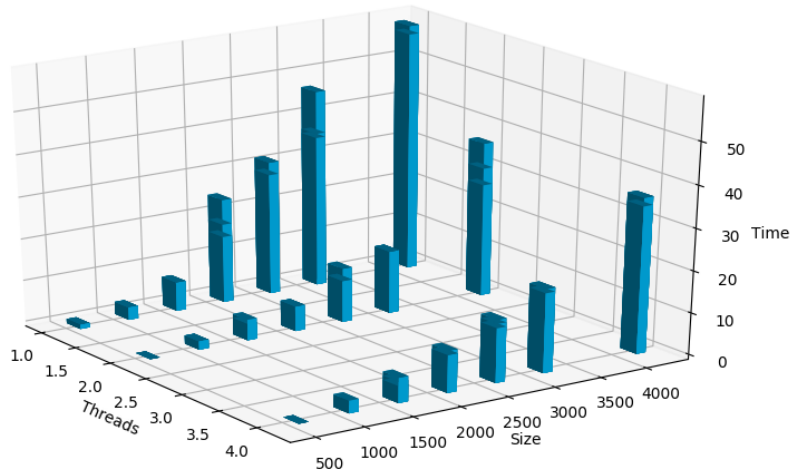
3.2.9 3217U OpenMP



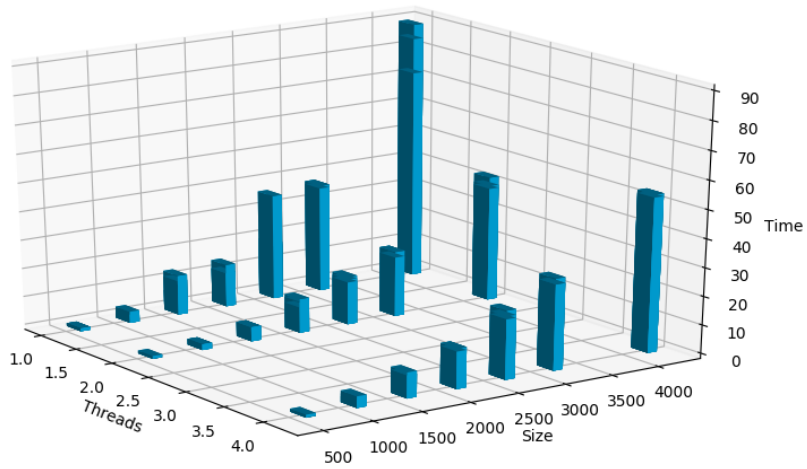
3.2.10 3217U MPI



3.2.11 370M OpenMP



3.2.12 370M MPI



4 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 370M, ситуация обратная – MPI-версии работают в 2-3 раза быстрее. Для самого “древнего” 370M 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 370M 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 $O((\frac{2*N}{T}) - 1) * C^2$), где C – время выполнения блока размером TxT одной нитью, N – размер матрицы.

5 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.