Московский Государственный Университет им. М.В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Практикум по курсу «Суперкомпьютеры и параллельная обработка данных»

Разработка параллельной версии программы для релаксации трехмерной матрицы

Работу выполнил

Федоров В. В.

Студент 323 группы
факультета ВМК МГУ

Москва 2020

Постановка задачи и формат данных.

Задача: Реализовать параллельный алгоритм релаксации трехмерной матрицы

Формат ввода: На вход подается единственное число — размер трехмерной матрицы.

Описание алгоритма.

Математическая постановка: Трехмерная матрица размером NxNxN с нулями на границах обрабатывается следующим образом: её элементы обходятся по выбранной координате от 1 до N-2 (нумерация элементов матрицы начинается с нуля), и каждому присваивается среднее из двух его соседей по этой координате. Так выглядит подобный цикл по координате i:

```
for (size_t k = 1; k < n - 1; k++)
      for (size_t j = 1; j < n - 1; j++)
            for (size_t i = 1; i < n - 1; i++)
            {
                  A[i][j][k] = (A[i - 1][j][k] + A[i + 1][j][k]) / 2.;
            }
```

Аналогично по координатам j и k. В цикле по координате k считается максимальная разница между исходными и новыми значениями элементов. Данная процедура выполняется до тех пор, пока не выполнится заданное число итераций либо максимальная разность не окажется меньше заданного эпсилон.

Параллельная оптимизация: Для OpenMP оптимизация заключалась лишь в добавлении прагм. Так как в алгоритме присутствует зависимость по выбранной координате, распараллелить можно только два вложенных цикла из трёх.

MPI-версия программы устроена намного сложнее. Так как модель памяти MPI не предусматривает общей памяти как таковой, необходимо разбивать трехмерную матрицу на подблоки и передавать их каждому из процессов. Рассмотрим алгоритм разделения матрицы на примере матрицы $3 \times 3 \times 3$ и 2 процессов. Матрица хранится в памяти линейно, в одномерном массиве:

i	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
j	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
&	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

1. Разделение по i-столбцам: I-столбцом назовем множество элементов матрицы с одинаковыми координатами j и k, например $iline_{12}=\{A_{012,}A_{112,}A_{212}\}$. Каждому процессу с передается блок i-столбцов размером $n_k=\frac{n^2}{p}+(k<n^2 \bmod p)$, где n — размер матрицы, p - число процессов, k — номер процесса. В нашем случае процессу с номером 0 дается блок из 5 i-столбцов, а процессу с номером 1 — из 4. В таблице представлено такое разделение (одним цветом выделены элементы, принадлежащее одному i-столбцу, одним тоном выделены элементы, отравляемые одному процессу):

i	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
j	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
&	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Для отправки данных процессам от нулевого процесса используется функция `MPI_Scatterv`. Так как блоки элементов не расположены подряд, необходимо задать пользовательский тип при помощи `MPI_Type_vector`. После этого каждый процесс может выполнить цикл «усреднения» по i, после чего полученные данные собираются у нулевого процесса с помощью `MPI_Gatherv`.

2. **Разделение по jk-плоскостям**: Jk-столбцом назовем множество элементов матрицы с одинаковыми координатами i, например
$jkflat_1=\{A_{100,}A_{101,}A_{102,}A_{110,}A_{111,}A_{112,}A_{120,}A_{121,}A_{122}\}$. Каждому процессу с передается блок jk-плоскостей размером $m_k=\frac{n}{p}+(k<n \bmod p)$, где n — размер матрицы, p - число процессов, k — номер процесса. В нашем случае процессу с номером 0 дается блок из 2 jk-плоскостей, а процессу с номером 1 — из 1. В таблице представлено такое разделение (одним цветом выделены элементы, принадлежащее одной jk-плоскости, одним тоном выделены элементы, отравляемые одному процессу):

i	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
j	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
&	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Для отправки данных процессам от нулевого процесса используется функция `MPI_Scatterv`. После этого каждый процесс может выполнить циклы «усреднения» по j и по k, после чего полученные данные собираются у нулевого процесса с помощью

`MPI_Gatherv`. В цикле по k каждый процесс также считает максимальную разность между старыми и новыми значениями элементов матриц, после чего нулевой процесс получает максимум из собранных значений при помощи `MPI_Reduce`.

На этом итерация программы завершается.

Анализ времени выполнения: Для оценки времени выполнения OpenMP-версии программы использовалась функция: `omp_get_wtime()`, для MPI-версии — MPI_Wtime();

Верификация: Для проверки корректности работы программы использовалась функция `verify()`, значение которой сравнивалось с её же значением для результата работы оригинального последовательного алгоритма.

Основные функции:

- `int main(...);`
  - В рамках функции осуществляется выделение памяти под матрицу, вызов остальных функций и замер времени.
- `int init(...);`
  - Инициализация элементов матрицы по формуле:
    - A[i][j][k] = 0, если одна из трех координат равна 0 или N — 1;
    - A[i][j][k] = 4 + i + j + k в противном случае.
- `int relax(...);`
  - Функция выполняет одну итерацию последовательного алгоритма релаксации.
- `int relax_parallel(...);`
  - Функция выполняет одну итерацию параллельного алгоритма релаксации.
- `int wrapper(...);`
  - Присутствует только в MPI-версии программы. Выделяет память для блоков матрицы для каждого процесса, выделяет память для массивов количеств и смещений для MPI_Scatterv и MPI_Gatherv и заполняет их, создает пользовательский тип для первого разделения матрицы, затем производит необходимое количество итераций релаксации и, наконец, освобождает выделенные им массивы и пользовательский тип.
- `int verify(...);`
  - Функция вычисляет хэш-сумму элементов матрицы по формуле:

$$S = \sum_{i,j,k=0}^{N-1} ((i+1)(j+1)(k+1)A_{ijk})/N^3$$

Полностью код можно посмотреть в репозитории на Гитхабе: https://github.com/P34K1N/skpod

Результаты выполнения.

1. OpenMP

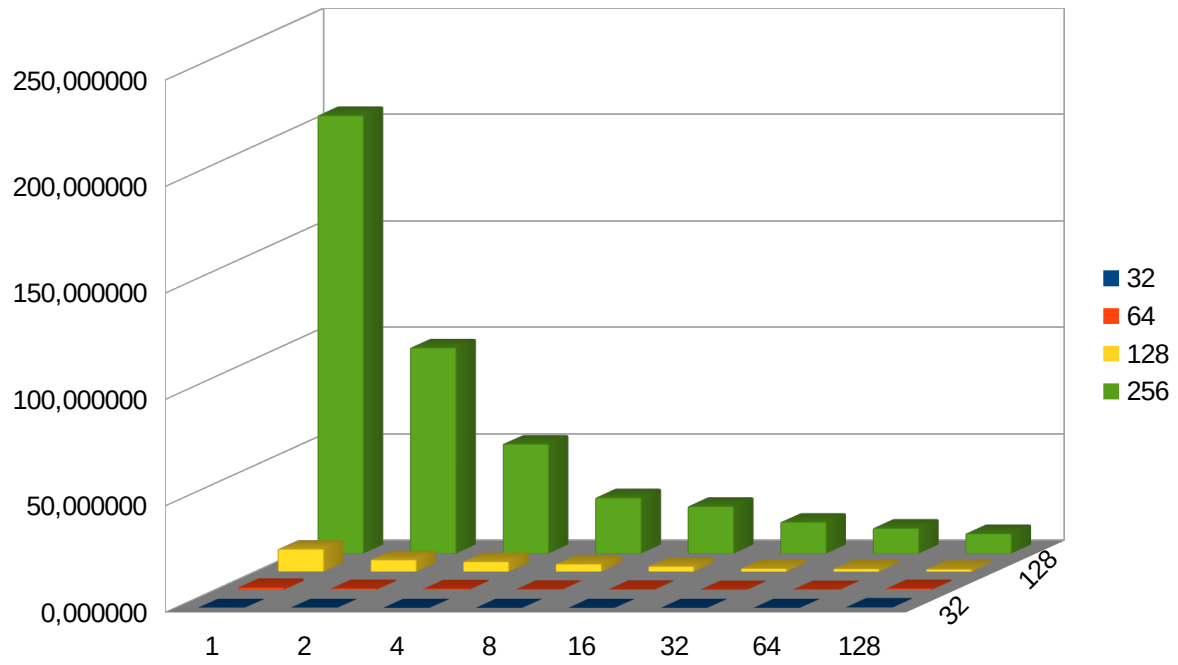
OpenMP-версия программы тестировалась на системе Polus

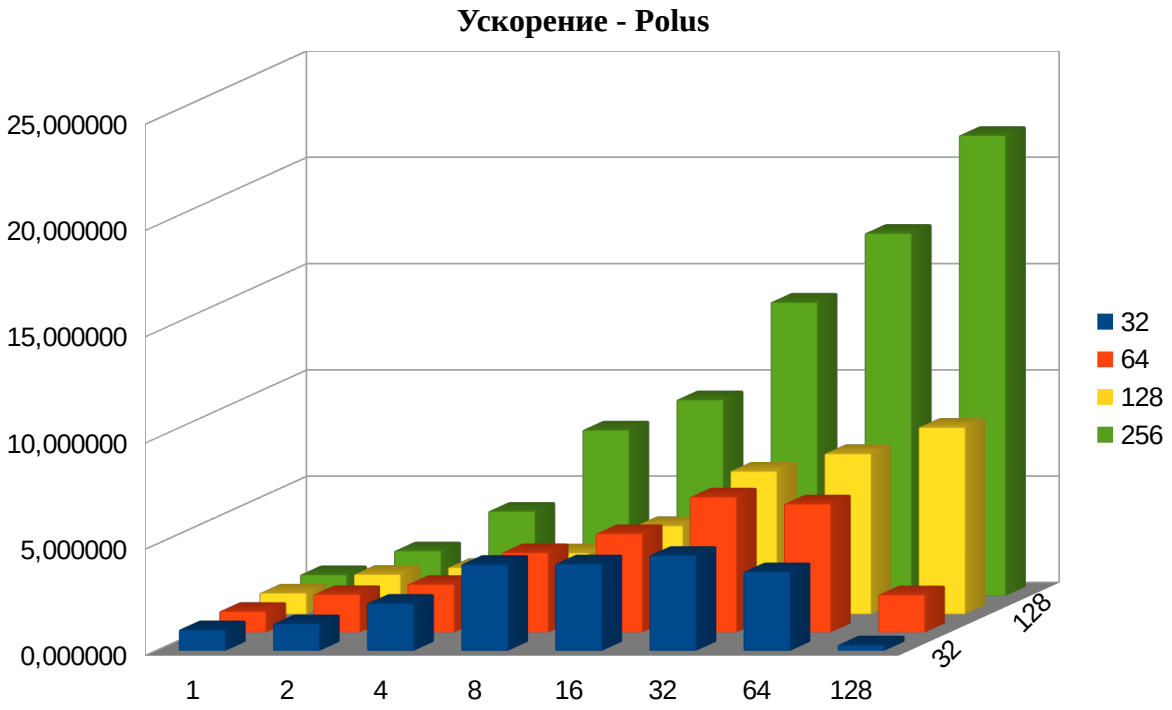
Время выполнения, с - Polus								
n \ t	1	2	4	8	16	32	64	128
32	0,131873	0,102631	0,058764	0,032335	0,032155	0,029281	0,035358	0,462693
64	1,173890	0,652107	0,516483	0,311478	0,251657	0,183679	0,193732	0,659674
128	10,659000	5,663030	4,834930	3,687920	2,556020	1,582460	1,409630	1,211480
256	205,923000	96,807600	51,585100	26,363200	22,296800	14,892600	12,059700	9,498050

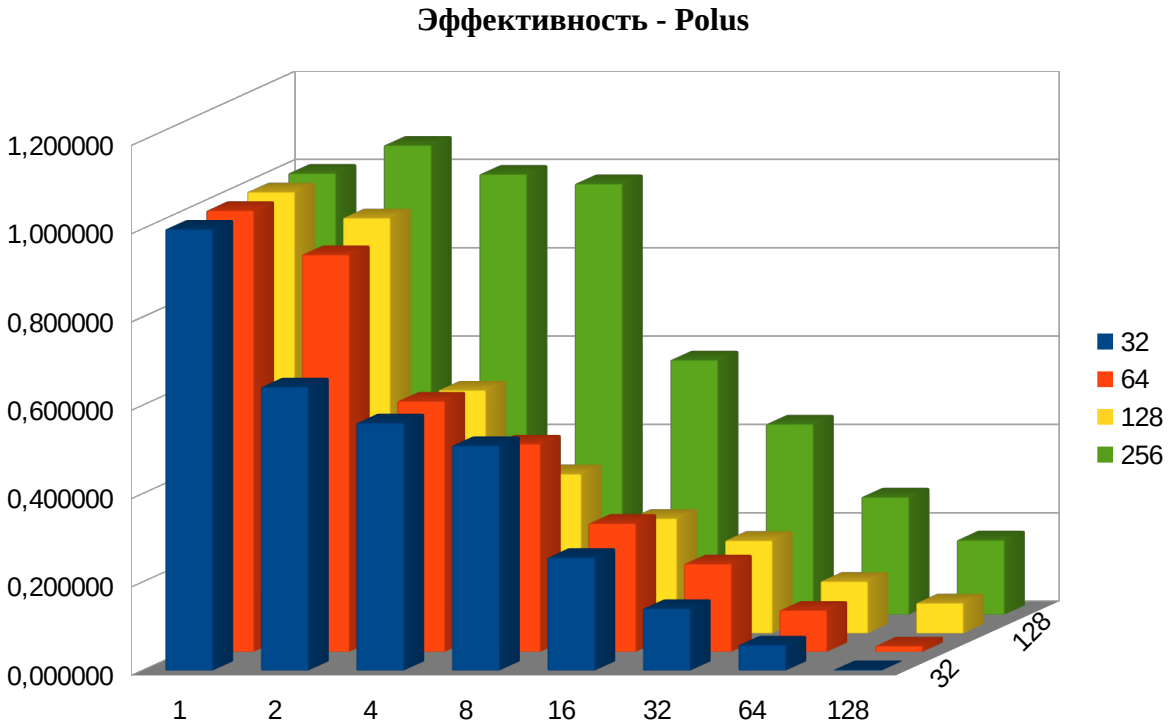
Ускорение - Polus								
n \ t	1	2	4	8	16	32	64	128
32	1,000000	1,284924	2,244112	4,078336	4,101166	4,503705	3,729651	0,285012
64	1,000000	1,800149	2,272853	3,768773	4,664643	6,390986	6,059350	1,779500
128	1,000000	1,882208	2,204582	2,890247	4,170155	6,735715	7,561559	8,798329
256	1,000000	2,127137	3,991909	7,811002	9,235541	13,827203	17,075300	21,680555

Эффективность - Polus								
n \ t	1	2	4	8	16	32	64	128
32	1,000000	0,642462	0,561028	0,509792	0,256323	0,140741	0,058276	0,002227
64	1,000000	0,900075	0,568213	0,471097	0,291540	0,199718	0,094677	0,013902
128	1,000000	0,941104	0,551146	0,361281	0,260635	0,210491	0,118149	0,068737
256	1,000000	1,063568	0,997977	0,976375	0,577221	0,432100	0,266802	0,169379

Время выполнения, с - Polus







2. MPI

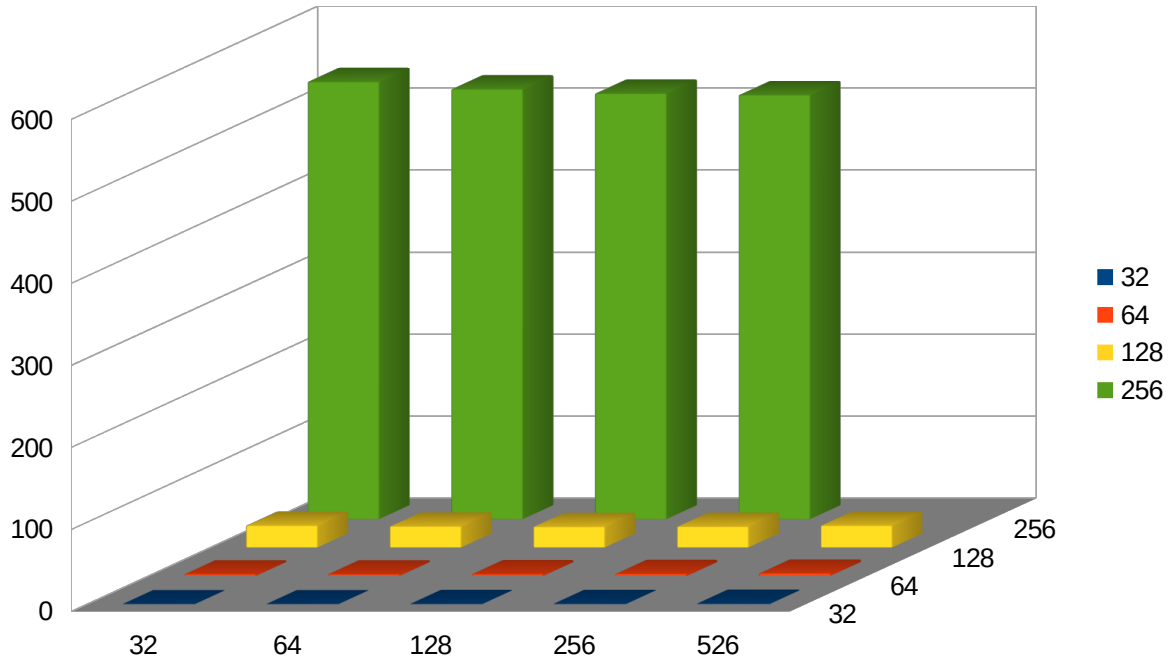
MPI-версия программы тестировалась на системе BlueGene.

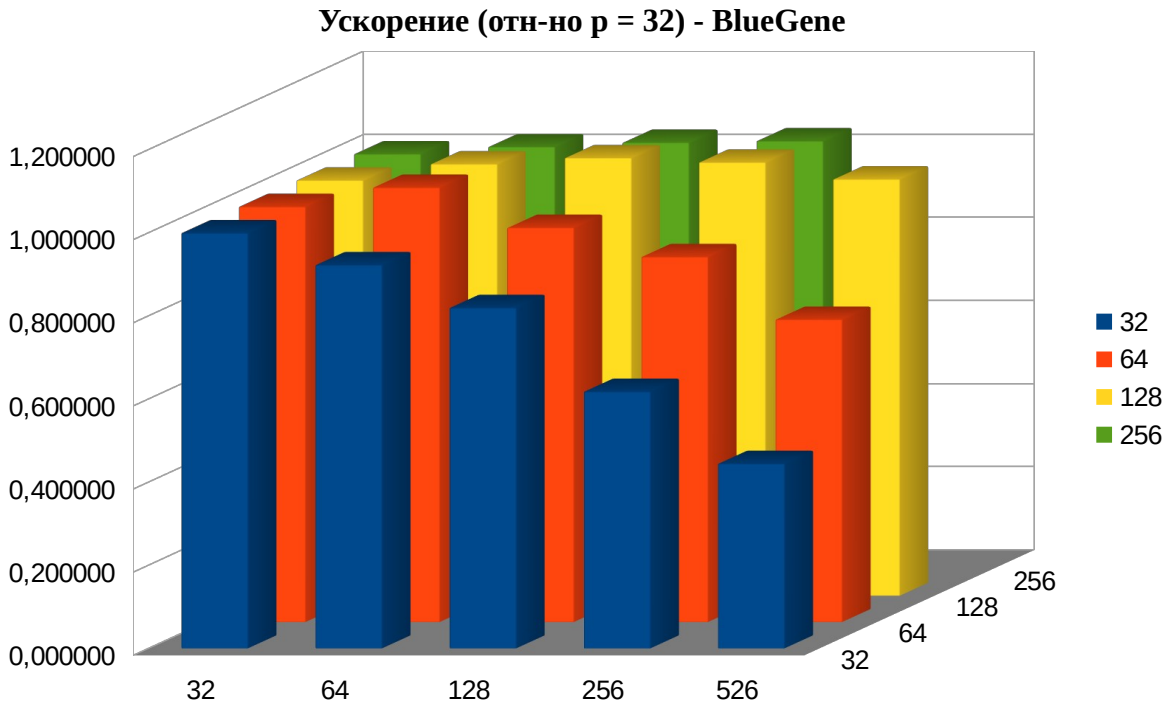
Время выполнения, с - BlueGene					
n \ p	32	64	128	256	512
32	0,406351	0,440108	0,495347	0,657134	0,912564
64	2,404193	2,298232	2,532088	2,734338	3,299374
128	27,086957	26,065618	25,702247	25,961079	27,015422
256	533,639595	524,678305	519,430124	517,500907	N/A*

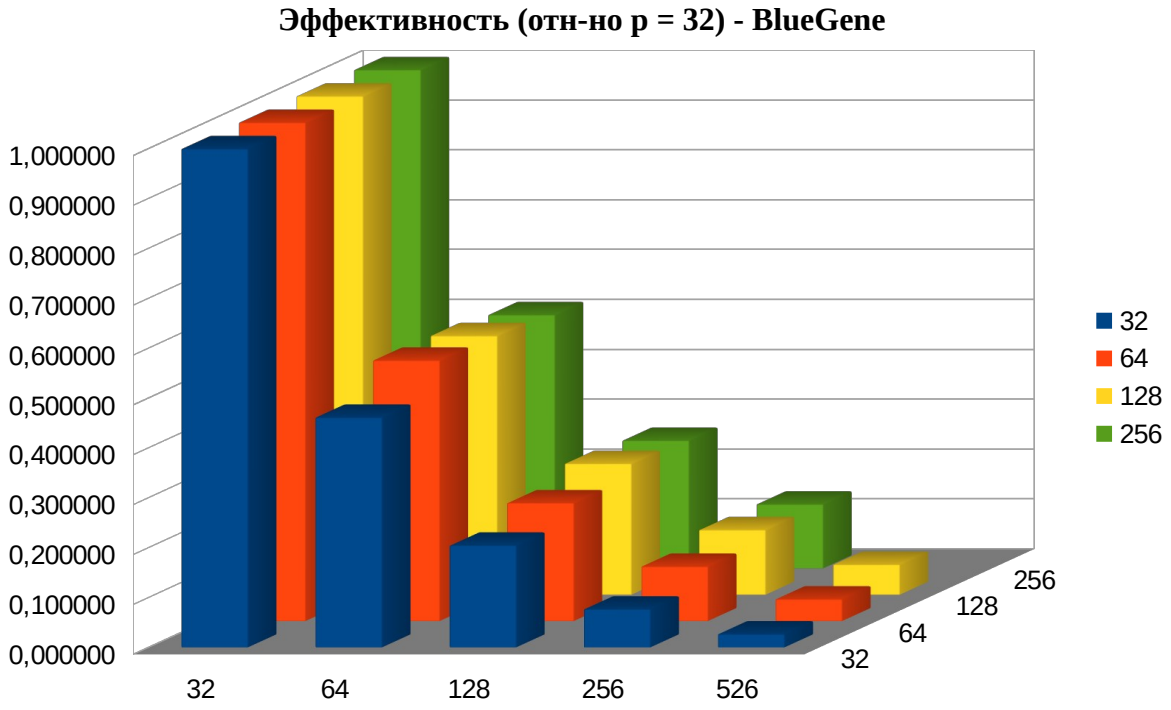
Ускорение (отн-но p = 32) - BlueGene					
n \ p	32	64	128	256	512
32	1,000000	0,923298	0,820336	0,618369	0,445285
64	1,000000	1,046105	0,949490	0,879260	0,728682
128	1,000000	1,039183	1,053875	1,043368	1,002648
256	1,000000	1,017080	1,027356	1,031186	N/A*

Эффективность (отн-но p = 32) - BlueGene					
n \ p	32	64	128	256	512
32	1,000000	0,461649	0,205084	0,077296	0,027830
64	1,000000	0,523053	0,237373	0,109907	0,045543
128	1,000000	0,519592	0,263469	0,130421	0,062665
256	1,000000	0,508540	0,256839	0,128898	N/A*

Время выполнения, с - BlueGene







Основные выводы.

OpenMP-версия программы при малых значениях N не ускоряется значительно при увеличении числа нитей, а порой даже и замедляется. Только при увеличении N можно заметить значительные улучшения времени работы программы, особенно при малом числе нитей, где время выполнения уменьшается почти пропорционально росту их числа. В целом зависимость эффективности от числа процессов следующая:

- 1-8 нитей — программа обрабатывается одним ядром, эффективность близка к 1;
- 16-64 нитей — программа обрабатывается несколькими ядрами в пределах одного процессора, эффективность средняя;
- 128 нитей — программа обрабатывается двумя процессорами, эффективность низкая.

Эффективность же ядер на BlueGene при тестировании MPI-программы оказалась низкой — увеличение числа процессов практически не давало ускорения.