Московский Государственный Университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра суперкомпьютеров и квантовой информатики

Отчёт

Солвер BICGSTAB для СЛАУ

(MPI реализация)

Работу выполнил:

Козлов Михаил Владимирович

523 группа

12.12.2018

Москва, 2018

1. Описание задания и программной реализации

1.1. Описание задания

Реализовать решатель СЛАУ BICGSTAB для разреженных матриц в формате CSR. Применить библиотеку MPI для распараллеливания решателя, а также применить другие методы оптимизации.

1.2. Описание программной реализации

В рамках поставленной задачи был реализован решатель СЛАУ BICGSTAB. Для этого были реализованы следующие модели данных:

- класс Matrix – класс для хранения разреженной матрицы в формате CSR и выполнения операций с матрицей. В приватной области хранится структура матрицы: *double *A* – ненулевые элементы матрицы, *int *JA* – номера столбцов ненулевых элементов, *int *IA* – позиция начала данных для строки i, *int sizeIA* – размер вектора IA, *int sizeA* – размер векторов A и JA. Были реализованы различные конструкторы (*Matrix(int Px, int Py, int Pz, int Nxp, int Nyp, int Nzp, int Nx, int Ny, int Nz, int *rows), Matrix(const Matrix &mat), Matrix(const Matrix &mat, int k)*). Последний из них служит генератором диагональной матрицы, элементами которой являются обратные элементы диагонали входной матрицы. Также были реализованы функции вывода.
- класс Vector – класс для хранения вектора и выполнения операций над ним. В приватной области также хранится структура вектора: *double *A* – элементы вектора, *int size* – размер вектора. Были реализованы различные конструкторы (*Vector(int selfSize, int haloSize, int *rows), Vector(const Vector &vec), Vector(int s, double c), Vector(int s, double *vec)*). Первый из них заполняет вектор синусами в цикле (sin(i)).

В данной реализации и матрица и вектор хранятся распределённо на узлах вычислительной системы. Разбиение размеров задаётся параметрами *int Px, int Py, int Pz.*

Были реализованы три базовые операции:

- friend *double dot(const Vector &vec1, const Vector &vec2)* – скалярное произведение двух векторов.
- friend *int axpby(Vector &vec1, const Vector &vec2, double a, double b)* – линейная комбинация двух векторов с коэффициентами a, b.
- friend *int SpMV(const Matrix &mat, const Vector &vec, Vector &res)* – умножение матрицы на вектор (Ax).
- friend *void sync(Vector &vec, int ** &halo, int ** &sHalo, int haloOptSize)* – «синхронизация» векторов между узлами: передача нужных частей векторов с одних узлов на другие для выполнения умножения матрицы на вектор.

Операции реализованы таким способом, чтобы работать с распределёнными матрицами и векторами: они обрабатывают только свой кусок данных, а если им нужны данные с других узлов – вызывается функция *sync*.

Функции решателей тестирования:

- *int solve(int N, Matrix &A, Vector &BB, double tol, int maxit, int debug, int ** &halo, int ** &sHalo, int haloOptSize, int rowsSize, int haloRedSize, int * &rows, int myRank)* – решатель СЛАУ Ax = BB методом BICGSTAB (с использованием MPI). N – размерность матрицы, tol – невязка системы, maxit – максимальное количество итераций решателя, debug – флаг отладки.
- *int testFunc(int N, int ** &halo, int ** &sHalo, int * &rows, int haloOptSize, int rowsSize, int haloRedSize, int myRank, int nProc, Matrix &A, Vector &BB)* – функция, проводящая комплексное тестирование алгоритма и функций: тестирование на разных размерах и разном количестве потоков.

Были применены различные методы оптимизации, например, более оптимальный алгоритм для умножения разреженной матрицы на вектор, развертка циклов, многопоточность.

2. Исследования производительности

2.1. Характеристики вычислительной системы

Тестирование проводилось на кластере ВМК IBM Polus. Данная система содержит 5 вычислительных узлов, один из которых выполняет функцию фронтэнда. Основные характеристики узла можно посмотреть на сайте http://hpc.cmc.msu.ru/polus.

Пиковая производительность кластера 55.84 Tflop/s.

lscpu даёт следующий вывод:

Architecture: ppc64le

Byte Order: Little Endian

CPU(s): 160

On-line CPU(s) list: 0-159

Thread(s) per core: 8

Core(s) per socket: 10

Socket(s): 2

NUMA node(s): 2

Model: 1.0 (pvr 004c 0100)

Model name: POWER8NVL (raw), altivec supported

CPU max MHz: 4023.0000

CPU min MHz: 2061.0000

Hypervisor vendor: (null)

Virtualization type: full

L1d cache: 64K

L1i cache: 32K

L2 cache: 512K

L3 cache: 8192K

NUMA node0 CPU(s): 0-79

NUMA node1 CPU(s): 80-159

Из информации из википедии (https://ru.wikipedia.org/wiki/POWER8) известно, что процессор POWER 8 имеет производительность 290 Gflop/s и 580 Gflop/s при обработке чисел двойной точности и одинарной точности соответственно. Максимальная пропускная способность памяти 230 GB/s.

Компиляция программы проводилась на фронтэнд узле командой *mpixlC -fopenmp matrix.cpp -o matrix*, а постановка в очередь проводилась командой *bsub -n * -R "span[hosts=1] affinity[core(1):distribute=pack]" -W 00:10 -o res.out mpirun matrix 1000 1000 100 0.00001 1000 1 * * * 1*. В данном случае, все параметры, кроме последнего, не имеют смысла, так как флаг тестирования (последний параметр) равен единице, и программа будет проводить тестирование. Вместо звёздочек надо поставить количество узлов и их разбиение Px, Py, Pz (важно, чтобы n = Px * Py * Pz).

2.2. Результаты измерений производительности.

Тестирование проводилось следующим образом: для матрицы размера (1000 1000 100) 100000000 (в формате (Nx, Ny, Nz) Nx*Ny*Nz) было произведено тестирование каждой базовой операции на одном узле и на различном количестве потоков и узлов (1, 2, 4, 8, 10, 16), масштабируемость проверялась путём увеличения размера и количества mpi-процессов в 2 раза (результаты и параметры можно увидеть в таблице).

Таблица масштабируемости солвера с использованием MPI:

N, P	10^8, 4	2 * 10^8, 8	4 * 10^8, 16
time	34.0633	32.3123	35.6444

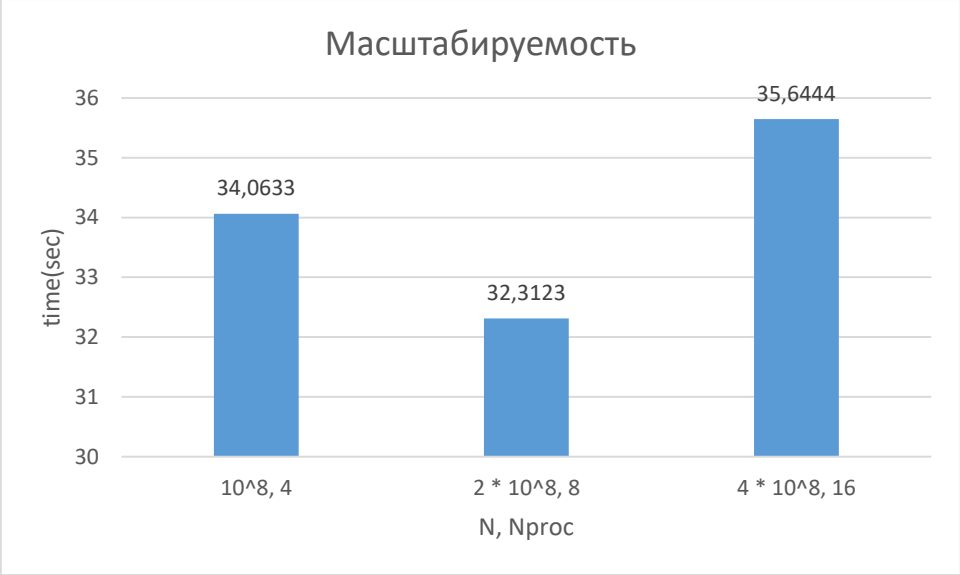
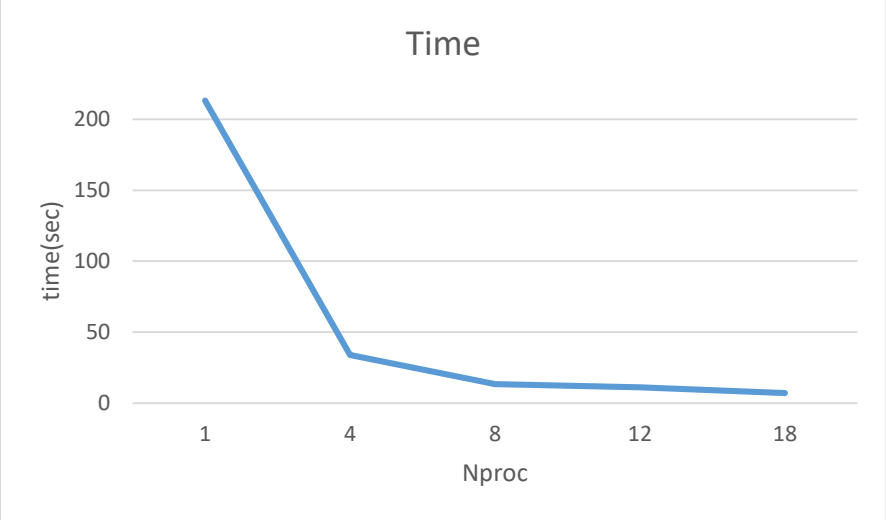


Таблица времени выполнения и ускорения солвера с использованием MPI:

10^8	1	4	8	12	18
time	213.131	34.0633	13.4663	11.2961	7.0724
SpeedUp	1	6.2569	15.827	18.8677	30.1354



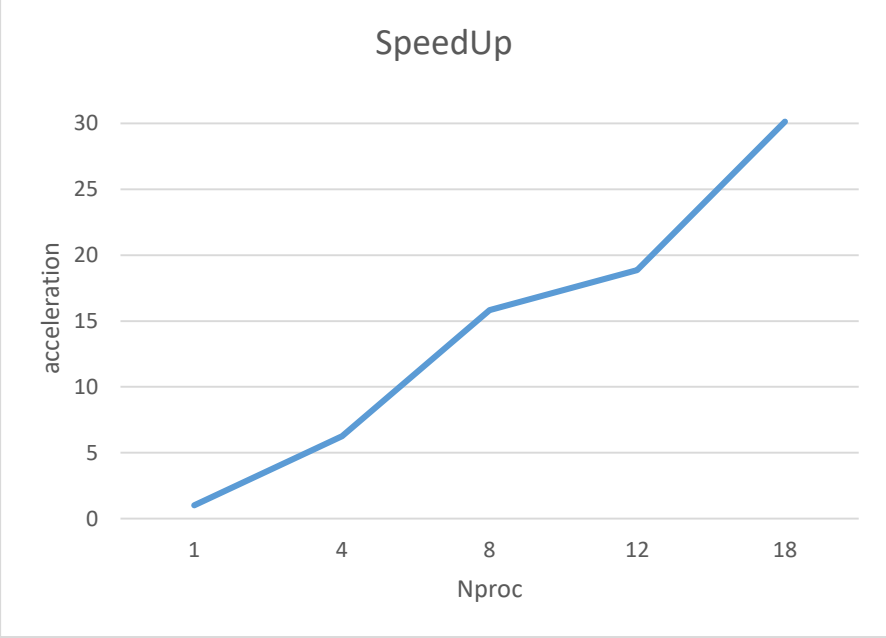


Таблица времени выполнения и ускорения базовых операций и солвера:

op	Nt/Np	timeOMP	accelerationOMP	timeMPI	accelerationMPI
Dot	1	0.176003	1	0.321443	1
	2	0.0906057	1.94251	0.0781535	4.11297
	4	0.074382	2.3662	0.0388826	8.26701
	8	0.024572	7.16273	0.0382613	8.40126
	10	0.0230405	7.63884	0.0261258	12.30366
	16	0.0244199	7.20734	0.0346484	9.27728
Axpby	1	0.0976478	1	0.194965	1
	2	0.0517899	1.88546	0.0416976	4.67569
	4	0.0399184	2.44619	0.0281741	6.92
	8	0.0263026	3.71247	0.0259514	7.5127
	10	0.0258585	3.77624	0.0249661	7.80919
	16	0.0277821	3.51478	0.016025	12.1663
SpMV + sync	1	3.75342	1	5.96461	1
	2	1.48512	2.52734	0.611438	9.75505
	4	0.956643	3.92353	0.31555	18.90227
	8	0.327823	11.4495	0.223497	26.68765
	10	0.271748	13.8121	0.169895	35.10763
	16	0.183652	20.4376	0.108993	54.72471
Solver	1	151.677	1	-	-
	2	78.3952	1.93477	-	-
	4	36.958	4.10403	-	-
	8	19.8494	7.64136	-	-
	10	16.8794	8.98592	-	-
	16	13.8382	10.9608	-	-

3. Анализ полученных результатов

Из полученных результатов можно сделать выводы:

- солвер масштабируемый
- ускоряется при увеличении узлов
- операции также ускоряются при увеличении узлов и потоков

Однако, хорошее MPI ускорение можно получить только при больших размерностях матрицы, иначе распараллеливание будет неэффективно и не целесообразно.

Также можно отметить, что сверхлинейное ускорение получено благодаря кэшу на одном узле (в данном случае справедливо только для ПОЛЮСа и при условии, что узел полностью отдан под данную задачу, то есть в это время на нём никто не считает). На «честной» параллельной системе, на которой каждый MPI процесс был бы на своём узле, такое сверхлинейное ускорение скорее всего не получилось бы.

4. Дополнение к отчёту: замеры на BlueGene/P

Компиляция программы проводилась на фронтэнд узле командой *mpixlcxx_r –qsmp=omp matrix.cpp -o matrix*, а постановка в очередь проводилась командой *mpisubmit.bg -n 1 -w 00:15:00 matrix -- 100 100 100 0.00001 1000 1 1 1 1 1*. В данном случае, все параметры, кроме последнего, не имеют смысла, так как флаг тестирования (последний параметр) равен единице, и программа будет проводить тестирование.

Тестирование проводилось следующим образом: для матрицы размера (100 100 100) 1000000 (в формате (Nx, Ny, Nz) Nx*Ny*Nz) было произведено тестирование каждой базовой операции на одном узле и на различном количестве потоков (1, 2, 4), на разном количестве узлов (1, 2, 4, 8, 16, 32, 64).

Таблица времени выполнения и ускорения солвера с использованием MPI:

10^6	1	2	4	8	16	32	64
time	25.621	12.643	6.317	3.172	1.611	0.818	0.415
SpeedUp	1.000	2.027	4.056	8.078	15.900	31.316	61.717
efficiency	1.000	1.013	1.014	1.010	0.994	0.979	0.964

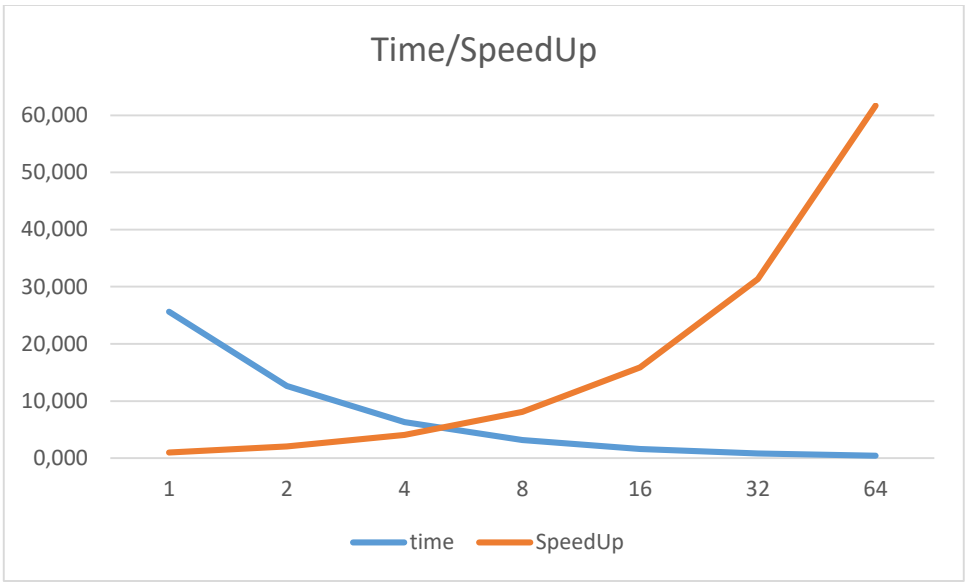

Time/SpeedUp
60,000
50,000
40,000
30,000
20,000
10,000
0,000
1
2
4
8
16
32
64
time
SpeedUp

Таблица времени выполнения и ускорения базовых операций и солвера:

op	Nt/Np	timeOMP	accelerationOMP	timeMPI	accelerationMPI
Dot	1	0.011200	1.000000	0.011201	1.000000
	2	0.005654	1.980830	0.005753	1.947033
	4	0.003353	3.339790	0.003075	3.642498
	8	-	-	0.001887	5.936933
	16	-	-	0.001006	11.136133
	32	-	-	0.000543	20.618870
	64	-	-	0.000316	35.467629
Axpby	1	0.023540	1.000000	0.023539	1.000000
	2	0.011806	1.993930	0.011773	1.999482
	4	0.005953	3.954550	0.005889	3.997415
	8	-	-	0.002947	7.987221
	16	-	-	0.001476	15.944955
	32	-	-	0.000952	24.717790
	64	-	-	0.000479	49.153863
SpMV + sync	1	0.229486	1.000000	0.229557	1.000000
	2	0.114892	1.997400	0.105365	2.178684
	4	0.057843	3.967370	0.052257	4.392864
	8	-	-	0.026415	8.690535
	16	-	-	0.013552	16.938601
	32	-	-	0.007359	31.193879
	64	-	-	0.003640	63.069615
Solver	1	10.313300	1.000000	-	-
	2	5.318230	1.939240	-	-
	4	2.826260	3.649110	-	-
	8	-	-	-	-
	16	-	-	-	-
	32	-	-	-	-
	64	-	-	-	-