

Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Практическое задание 2.

Параллельная реализация операций с сеточными данными
на неструктурированной смешанной сетке

Мокров Кирилл Сергеевич
523 группа

Москва, 2020

Содержание

1	Описание задания и программной реализации	3
1.1	Краткое описание задания	3
1.2	Краткое описание программной реализации	3
2	Исследование производительности	6
2.1	Характеристики вычислительной системы	6
2.2	Результаты измерений производительности	7
2.2.1	Линейный рост времени	7
2.2.2	Параллельное ускорение	8
2.2.3	MPI и OpenMP	10

1 Описание задания и программной реализации

1.1 Краткое описание задания

Задание можно разбить на 4 этапа:

1. Generate - генерация графа/портрета по тестовой сетке. По двумерной неструктурированной смешанной сетке, состоящей из треугольников и четырёхугольников генерируется портрет разреженной матрицы смежности графа, дополненный главной диагональю в формате CSR.
2. Fill - заполнение матрицы и вектора правой части по заданному портрету.
3. Com - построение схемы обмена данными.
4. Solve - решение СЛАУ с полученной матрицей. Так как матрица, полученная на предыдущем этапе, симметричная, то используется метод сопряженных градиентов с предобуславливателем Якоби.
5. Report - проверка корректности и выдача измерений. Проверка, что невязка системы после работы решателя удовлетворяет заданной точности, выдача значения невязки и печать таблицы таймирования всех предыдущих этапов.

1.2 Краткое описание программной реализации

Сначала в функции *main* происходит считывание данных командной строки, проверка их корректности и выдача диагностики. Параметры программы: Nx , Ny , $K1$, $K2$, Px , Py , их значение хорошо поясняет рисунок 1, Px и Py - размеры процессорной решётки.

Пример для $Nx=10$, $Ny=9$, $K1=3$, $K2=4$:

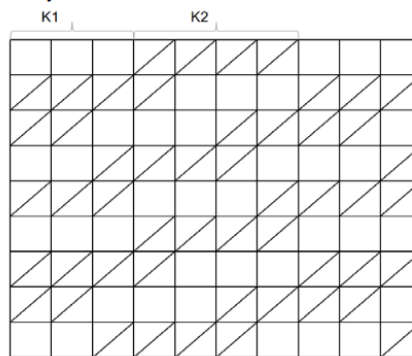


Рис. 1: Параметры командной строки

Программа компилировалась `mpic++`, с флагами `-Wall -Wextra -pedantic -std=c++11 -O3 -Wshadow`.

Листинг 1: Реализованные функции

```

// Inner product of two vectors vec_1 and vec_2, N_own - num of
own dots
double dotKernel(const int N_own,
                 const std::vector<double>& vec_1,
                 const std::vector<double>& vec_2);

// Linear combination of two vectors  $x = a * x + b * y$ 
void axpbyKernel(const double a, std::vector<double>& x,
                 const double b, const std::vector<double>& y);

// A function in which the data is transferred. vec - data
vector, com - exchange scheme
void update(std::vector<double>& vec, const Com_struct& com);

// Matrix vector product with sparse matrix. N_own - num of own
dots, row_ptr, col_ptr - CSR representation, res = Matrix *
vec, com - exchange scheme
void SpMVKernel(const int N_own,
                 const std::vector<double>& matrix, const std::
                 vector<int>& row_ptr,
                 const std::vector<int>& col_ptr, std::vector<
                 double>& vec,
                 const Com_struct& com, std::vector<double>& res)
;

// Determines whether a cell with the cell_idx index contains
oblique line, K1, K2 - parameters that determine the
distribution of oblique lines
bool hasOblique(const int K1, const int K2, const int cell_idx);

// Determines local boundaries, N - side len, p - num of
processes, idx - current process index, b, e - begin and end
void get_local_begin_end(const int N, const int p, const int idx
, int& b, int& e);

// Function corresponds to the first stage of the task. Nx, Ny,
K1, K2, Px, Py - determine the configuration of a two-
dimensional grid, rank - rank of the process, size - total

```

```

        processes, N - own + halo, N_own - own, row_ptr, col_ptr -
        CSR representation, Part - ranks of vertex owners, L2G, G2L -
        local to global and global to local mapping
void generate(const int Nx, const int Ny,
               const int K1, const int K2,
               const int Px, const int Py,
               const int rank, const int size,
               int& N, int& N_own,
               std::vector<int>& row_ptr,
               std::vector<int>& col_ptr,
               std::vector<int>& Part,
               std::vector<int>& L2G,
               std::vector<int>& G2L);

// Function corresponds to the second stage of the task. N - own
// + halo, N_own - own, row_ptr, col_ptr - CSR representation,
// L2G - local to global mapping, A_arr, b_vec - fillable matrix
// and vector
void fill(const int N, const int N_own,
           const std::vector<int>& row_ptr,
           const std::vector<int>& col_ptr,
           const std::vector<int>& L2G,
           std::vector<double>& A_arr,
           std::vector<double>& b_vec);

// Building a data exchange scheme. rank - rank of the process,
// size - total processes, N - own + halo, N_own - own, row_ptr,
// col_ptr - CSR representation, Part - ranks of vertex owners,
// L2G, G2L - local to global and global to local mapping, com
// - exchange scheme
void Com(const int rank, const int size,
          const int N, const int N_own,
          const std::vector<int>& row_ptr,
          const std::vector<int>& col_ptr,
          const std::vector<int>& Part,
          const std::vector<int>& L2G,
          const std::vector<int>& G2L,
          Com_struct& com);

```

```

// Function corresponds to the third stage of the task. N – own
+ halo, N_own – own, A_arr, b_vec – initial data for solving
systems of linear equations, row_ptr, col_ptr – CSR
representation, x_vec – equations solution, TOL – relative
error of the solution
void Solve(const int N, const int N_own, const int rank,
          const std::vector<double>& A_arr, const std::vector<
          double>& b_vec,
          const std::vector<int>& row_ptr, const std::vector<int
          >& col_ptr,
          const Com_struct& com, std::vector<double>& x_vec,
          const double TOL = 1e-7);

```

2 Исследование производительности

2.1 Характеристики вычислительной системы

Тестирования программы проводились на вычислительном комплексе *Ломоносов-2*. Ломоносов-2 - массивно-параллельная вычислительная система, состоящая из большого числа вычислительных узлов. Каждый узел оборудован 14 ядерным процессором *Intel Haswell-EP E5-2697v3*, 2.6 GHz, 64 GB оперативной памяти. Узлы связаны сетью Infiniband FDR. Некоторые из вычислений выполнены на вычислительном комплексе *IBM Polus*. Polus - параллельная вычислительная система, состоящая из 5 вычислительных узлов. Каждый узел оборудован двумя десятиядерными процессорами *IBM Power 8*, каждое ядро которого имеет 8 потоков, 256 GB оперативной памяти. Производительность кластера (TFlop/s): 55,84 (пиковая), 40,39 (Linpack).

2.2 Результаты измерений производительности

2.2.1 Линейный рост времени

P	Generation	Filling	Dot	Axpby	SpMV	Memory
			Solver			
10000	0.000111	0.000249	0.000757	0.000098	0.000686	
			0.001165			
100000	0.001051	0.002475	0.004964	0.000956	0.004021	
			0.007286			
1000000	0.010469	0.024775	0.044690	0.008970	0.038380	
			0.065353			
10000000	0.107075	0.122159	0.106816	0.040453	0.172713	
			0.318858			

Таблица 1: Время (в секундах) работы 3 этапов программы, 3 основных вычислительных ядер для 8 процессов. Polus

По результатам работы можно видеть, что время работы программы растёт линейно. Определить потребляемую память не получилось ни с помощью `valgrind`, ни по отчёту, выдаваемому планировщиком на Polus, так как он содержит некорректные данные.

2.2.2 Параллельное ускорение

P	Generation S-up	Filling S-up	Dot S-up	Axpby S-up	SpMV S-up
			Solver S-up		
1	0,0040965 1	0,0255193 1	0,0027106 1	0,0022277 1	0,0068099 1
			0,0141199 1		
2	0,0026110 1,57	0,0026110 1,87	0,0017039 1,59	0,0017808 1.25	0,0036808 1.85
			0,0078031 1.81		
4	0,0012251 3.34	0,0072235 3,53	0,0012177 2,23	0,0006634 3.36	0,0020253 3,36
			0,0044721 3.16		
8	0,0007207 5.68	0,0037086 6.88	0,0008872 3,06	0,0003113 7.16	0,0011562 5.89
			0,0010537 13.40		
16	0,0010537 3.89	0,0023110 11.04	0,0754179 0,04	0,0001532 14.54	0,0362415 0.19
			0,0767009 0.18		
32	0,0009311 4.40	0,0011646 21.91	0,1371020 0,02	0,0000810 27.51	0,0543343 0.13
			0,1381070 0.10		

Таблица 2: Время работы и ускорение 3 этапов программы и 3 основных вычислительных ядер, N = 100'000, Lom2

P	Generation S-up	Filling S-up	Dot S-up	Axpby S-up	SpMV S-up
			Solver S-up		
1	0,327389 1	2,99311 1	0,380903 1	0,374577 1	0,95245 1
			1,94247 1		
2	0,164463 1,99	1,47338 2.03	0,204223 1,87	0,197227 1,90	0,501543 1,90
			1,02163 1,90		
4	0,0931595 3,51	0,739782 4.05	0,131032 2,91	0,1366 2,74	0,290079 3,28
			0,606789 3,20		
8	0,0633809 5.17	0,389848 7,68	0,103776 3.67	0,118076 3,17	0,238827 3,99
			0,494773 3.93		
16	0,0369154 8,87	0,185666 16.12	0,109889 3,47	0,0550029 6,81	0,154094 6,18
			0,303414 6,40		
32	0,0278479 11.76	0,0931546 32.13	0,160808 2,37	0,0338331 11,07	0,119357 7,00
			0,277557 7,00		

Таблица 3: Время работы и ускорение 3 этапов программы и 3 основных вычислительных ядер, N = 10'000'000, Lom2

P	Generation S-up	Filling S-up	Dot S-up	Axpby S-up	SpMV S-up
			Solver S-up		
1	0,504931 1	0,917668 1	0,515702 1	0,231572 1	1,11597 1
			2,08155 1		
2	0,270138 1,87	0,464667 1,97	0,27418 1,88	0,169161 1,37	0,599347 1,86
			1,16755 1,78		
4	0,174666 2,89	0,497847 1,84	0,834341 0,62	0,146143 1,58	0,772503 1,44
			1,23706 1,68		
8	0,118437 4,26	0,346794 2,65	0,713116 0,72	0,106369 2,18	0,618075 1,81
			0,938596 2,22		
16	0,0915578 5,51	0,188397 4,87	0,434112 1,19	0,0581014 3,99	0,364974 3,06
			0,549978 3,78		
32	0,0822376 6,14	0,100647 9,12	0,26031 1,98	0,0223672 10,35	0,122083 9,14
			0,316899 6,57		

Таблица 4: Время работы и ускорение 3 этапов программы и 3 основных вычислительных ядер, $N = 10^6$ и $N = 10^7$, Polus

В таблицах представлены время работы параллельной программы и достигаемое ускорение на 1, 2, 4, 8, 16, 32 процессах, 6 и 3 - СК Ломоносов2 $N = 10^6$ и $N = 10^7$, 4 - СК Polus $N = 10^7$.

Коммуникации между процессами осуществляются в функциях DOT(allreduce) и SpMV(isend, irectv, waitall), которые испытывают из-за этого проблемы с масштабируемостью. Можно заметить, что на Ломоносове2 для малых размеров задач производительность этих функций начинает резко падать с 16 процессов и далее, так как в данном случае выгоднее выполнить больше локальных вычислений, чем гонять данные по сети, дополнительно усугубляет ситуацию необходимость задействовать межузловые связи (на узле максимум 14 процессов), а скорость коммуникаций внутри узла и между узлами отличается примерно на два порядка.

	Nx, Ny, K1, K2					
	1000, 1000, 10, 5			10000, 1000, 10, 5		
P	DOT	AXPBY	SpMV	DOT	AXPBY	SpMV
1	2.15	3.68	3.12	1,52	2,17	2,22
2	3,43	4.60	5.77	2,83	4,11	4,21
4	4.79	12.35	10.48	4,41	5.94	7,28
8	<i>6.58</i>	26.32	<i>18.36</i>	5.57	6,87	8,84
16	0,08	53.48	0.59	5.26	14,75	13,71
32	0,04	<i>101.18</i>	0.39	3.59	23,97	17,70

Таблица 5: Производительность трёх основных вычислительных ядер, GFlops (лучшие результаты выделены курсивом), Lom2

2.2.3 MPI и OpenMP

P / T	Generation	Filling	Dot	Axpby	SpMV
			Solver		
1	1.00 1,00	1.00 1,00	1.00 1,00	1.00 1,00	1.00 1,00
			1,00		
2	0.93 1,00	0.99 0,95	0.94 0,95	0.68 0,50	0.89 1,00
			0,9		
4	0.72 0,97	0.46 0,97	0.15 0,95	0.40 0,45	0.42 0,87
			0,80		
8	0.53 0,93	0.33 0,96	0.09 0,92	0.27 0,37	0.28 0,95
			0,75		
16	0.34 0,62	0.30 0,81	0.07 0,52	0.25 1,9	0.24 0,55
			0,41		
32	0.19 0,29	0.28 0,58	0.06 0,27	0.32 0,1	0.21 0,26
			0,20		

Таблица 6: Эффективность работы 3 этапов программы и 3 основных вычислительных ядер для MPI и OpenMP реализаций, $N = 10'000'000$, Polus