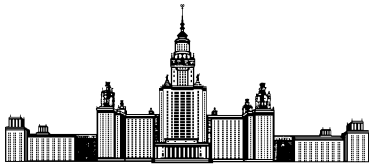

Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Практическое задание 1.
Параллельная реализация операций с сеточными данными на неструктурированной смешанной сетке

Мокров Кирилл Сергеевич
523 группа
Дата подачи: ???

Москва, 2020

Содержание

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. Solve - решение СЛАУ с полученной матрицей. Так как матрица, полученная на предыдущем этапе, симметричная, то используется метод сопряженных градиентов с предобуславливателем Якоби.
4. 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*. Их значение хорошо поясняет рисунок 1. В случае параллельной реализации после них ещё идёт количество используемых нитей. А в самом конце может быть необязательный аргумент включающий отладочную печать.

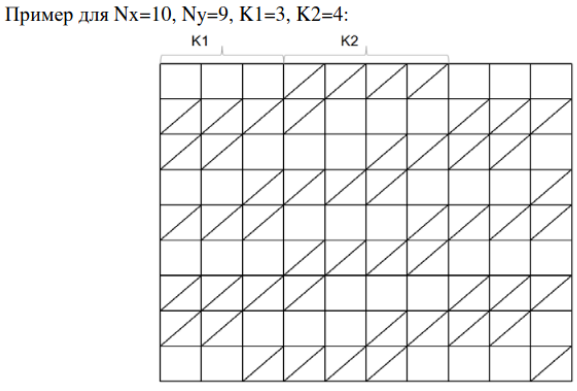

Рис. 1: Параметры командной строки

Листинг 1: **Реализованные функции**

```
// Inner product of two vectors vec_1 ans vec_2
```

```
double dotKernel(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);

// Matrix vector product with sparse matrix. row_ptr, col_ptr -
    CSR representation, res = Matrix * vec
void SpMVKernel(const std::vector<double>& matrix, const std::
    vector<int>& row_ptr, const std::vector<int>& col_ptr, const
    std::vector<double>& vec, std::vector<double>& res);

// Compution, how many cells with obliques line before cell with
     index cell_idx, K1, K2 - parameters that determine the
    distribution of oblique lines
int obliquesBefore(const int K1, const int K2, const int
    cell_idx);

// 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);

// Function corresponds to the first stage of the task. Nx, Ny,
    K1, K2 - determine the configuration of a two-dimensional
    grid, row_ptr, col_ptr - CSR representation
void generate(const int Nx, const int Ny, const int K1, const
    int K2, std::vector<int>& row_ptr, std::vector<int>& col_ptr)
    ;

// Function corresponds to the second stage of the task. Nx, Ny
    - determine grid size, row_ptr, col_ptr - CSR representation,
     A_arr, b_vec - fillable matrix and vector
void fill(const int Nx, const int Ny, const std::vector<int>&
    row_ptr, const std::vector<int>& col_ptr, std::vector<double
    >& A_arr, std::vector<double>& b_vec);
```

```
// Function corresponds to the third stage of the task. 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 std::vector<double>& A_arr, const std::vector<
   double>& b_vec, const std::vector<int>& row_ptr, const std::
   vector<int>& col_ptr, std::vector<double>& x_vec, const
   double TOL = 1e-4);
```

2 Исследование производительности

2.1 Характеристики вычислительной системы

Тестирования программы проводились 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 Последовательная производительность

N	Generation	Filling	Dot	Axpby	SpMV	Memory
			Solver			
10000	0.000106	0.002573	0.000472	0.000278	0.001478	1.37
			0.002410			
100000	0.000942	0.029858	0.004723	0.002842	0.014849	12.65
			0.023946			
1000000	0.009342	0.302991	0.051984	0.032798	0.163295	124.66
			0.263362			
10000000	0.086874	3.014130	0.517145	0.326723	1.640600	1224.79
			2.622970			

Таблица 1: Время (в секундах) работы 3 этапов программы, 3 основных вычислительных ядер и затраченная на работу всей программы память (в MB)

По результатам, представленным в таблице 1, видно, что с ростом размера задачи время этапов инициализации, основных ядер солвера, количества использованной памяти растёт линейно.

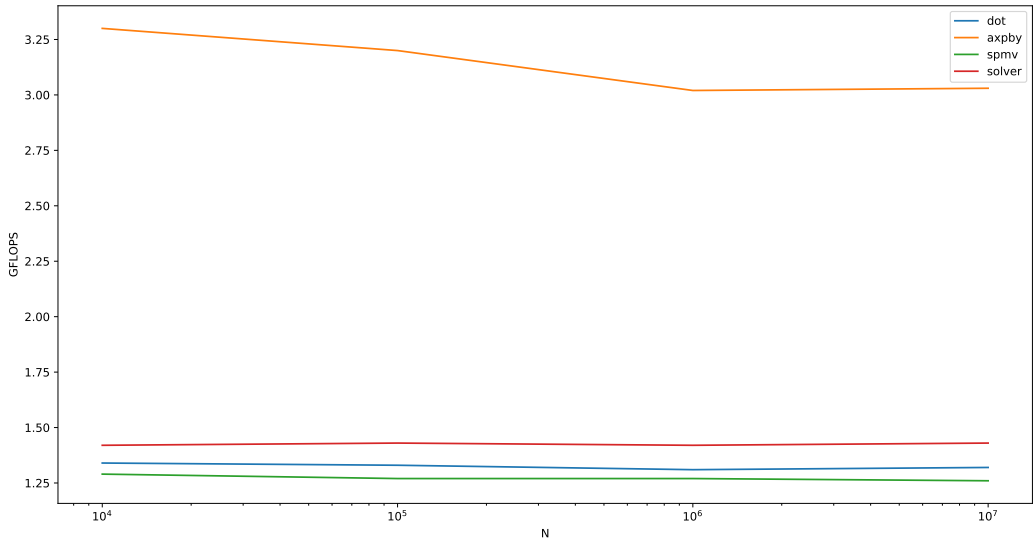

Рис. 2: Производительность 3 основных вычислительных ядер и всего солвера

2.2.2 Параллельное ускорение

T	Generation | S-up	Filling | S-up	Dot | S-up	Axpby | S-up	SpMV | S-up
			Solver | S-up		
1	1544 | 1,0	30508 | 1,0	4753 | 1,0	2085 | 1,0	14625 | 1,0
			22796 | 1,0		
2	847 | 1,8	15308 | 2,0	2397 | 1,9	2128 | 0,97	7356 | 1,9
			12836| 1,7		
4	471 | 3,3	7697 | 3,9	1262 | 3,7	1144 | 1,8	3768 | 3,9
			7005| 3,2		
8	436 | 3,6	3886 | 7,8	774 | 6,1	726 | 2,9	2067 | 7,0
			4350| 5,2		
16	518 | 3,0	2252 | 13,5	674 | 7,0	707 | 2,9	1572 | 9,3
			3858| 5,9		
32	799 | 1,9	1551 | 19,6	819 | 5,8	774 | 2,7	1472 | 9,9
			3962| 5,7		

Таблица 2: Время (в мкс) работы и ускорение 3 этапов программы и 3 основных вычислительных ядер, N = 100'000

T	Generation | S-up	Filling | S-up	Dot | S-up	Axpby | S-up	SpMV | S-up
			Solver | S-up		
1	1581610 | 1,0	3104950 | 1,0	473064 | 1,0	212899 | 1,0	1434150 | 1,0
			2259050 | 1,0		
2	791180 | 2,0	1566810| 1,9	239733 | 1,9	211287 | 1,0	716838 | 2,0
			1259280 | 1,8		
4	397837 | 3,9	782295 | 3,9	123937 | 3,8	114709 | 1,8	402956 | 3,5
			706907 | 3,2		
8	209928 | 7,5	401158 | 7,7	63847 | 7,4	70197 | 3,0	188113 | 7,6
			378095 | 6,0		
16	159830 | 9,9	238812 | 13,0	56495 | 8,3	67089 | 3,1	160765 | 8,9
			340134 | 6,6		
32	168950 | 9,3	167382 | 18,5	54456 | 8,6	65948 | 3,2	170258 | 8,4
			349474 | 6,4		

Таблица 3: Время (в мкс) работы и ускорение 3 этапов программы и 3 основных вычислительных ядер, N = 10'000'000

В таблицах 2 и 3 представлены время работы параллельной программы на $1, 2, 4, 8, 16, 32$ нитях, при $N = 10^5$ и $N = 10^7$. Не самое хорошее ускорение некоторых этапов работы программы в первом случае можно объяснить небольшим размером задачи, иногда бывает выгодней использовать меньше нитей, но в то же время тратить меньше времени на манипуляции с ними. Для второго же случая, задача ощутимо лучше распараллеливается, ускорение для некоторых этапов доходит до 9 - 18 раз.

	Nx, Ny, K1, K2					
	100, 1000, 15, 4			1000, 10000, 15, 4		
T	DOT	AXPBY	SpMV	DOT	AXPBY	SpMV
1	1,302	4,205	1,48	1,312	4,090	1,512
2	2,604	4,118	2,96	2,589	4,112	3,026
4	4,960	7,685	5,78	5,008	7,592	5,384
8	8,091	12,09	10,5	9,721	12,40	11,53
16	9,300	12,44	13,8	10,98	12,98	13,49
32	7,626	11,34	*14,8*	*11,39*	*13,20*	12,74

Таблица 4: Производительность трёх основных вычислительных ядер, GFlops (лучшие результаты выделены курсивом)

3 Анализ полученных результатов

Для начала рассчитаем $TBP = min(TPP, BW\dot{A}I)$. На *Polus* используется процессор *Power 8*, обнаружить в интернете, какими векторными расширениями обладает данный процессор не удалось, поэтому предположим, что одной его ядро располагает двумя *AVX512*. Тогда так как это 10 ядерный процессор, частоты работы которого могут изменяться от 2.5GHz до 5GHz, TPP можно посчитать так: $TPP = 10C * 2.5GHz * 2 * 512/64 = 400GFlops$. За BW возьмём пропускную способность контроллера памяти - 230 GB/s.

Расчёт AI. Пусть X - длина вектора *row_ptr*, Y - *col_ptr*. DOT: цикл размера X, с телом sum = sum + a[i] * b[i], FLOP - 2 * X (сложение и умножение), BYTE - 2 * X * 8 (два вектора a, b). AXPBY: цикл размера X, с телом x[i] = a * x[i] + b * y[i], FLOP - 3 * X (два умножения и сложение), BYTE - 3 * X * 8 (два вектора прочитать, один записать). SpMV: вложенные циклы, если представить их одним, то его размер будет Y, тело sum = sum + a[i] * b[i], FLOP - 2 * Y (сложение и умножение), BYTE - 8 * (Y + 2X) + 4 * (Y + X) (матрица, *col_ptr* по Y, *row_ptr*, vec, res по X).

Kernel	AI, Flop / byte	TBP, GFlops	%,TBP	%, TPP
dot	0.125	28.75	40	2,8
axpby	0.130	29.90	44	3,3
spmv	0.098	22.54	65	3,7

Таблица 5: AI, TBP и количество процентов от достижимой и пиковой производительностей для трёх основных вычислительных ядер

Для вычислительных ядер проценты от достижимой производительности достаточно велики, от 40 до 65. Если же рассматривать пиковую производительность, то всё сильно хуже, и это несмотря на то, что при расчётах использовались минимальная тактовая частота и максимальная пропускная способность процессора.