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

Ухин Сергей Алексеевич

523 группа

Дата подачи: 18.10.2020

Москва, 2020

1. Описание задания и программной реализации

1.1 Краткое описание задания

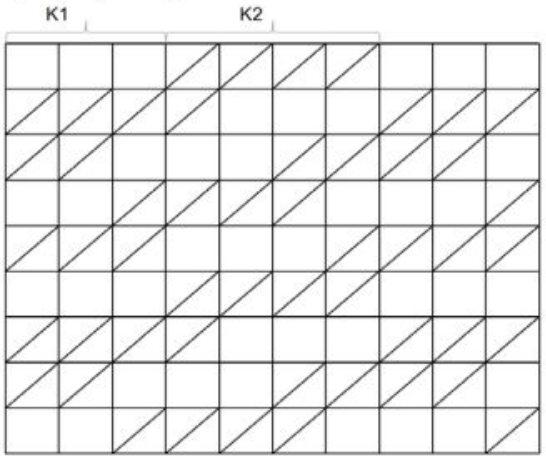
Задание можно разбить на 4 этапа:

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.

Пример для Nx=10, Ny=9, K1=3, K2=4:



Реализованные функции:

Функция для первого этапа задания. Векторы ia, ja для хранения CSR представления. N - итоговое количество узлов.

void Generate(int Nx, int Ny, int k1, int k2, int &N, std::vector<int> &ia, std::vector<int> &ja)

Функция для второго этапа задания. Заполняет матрицу A и b для решения слау. Также заполняет матрицу M^-1 , которая потом используется в солвере.

void Fill(int N, std::vector<int> &ia, std::vector<int> &ja, std::vector<double> &A, std::vector<double> &b, std::vector<double> &M)

Функция для нахождения скалярного произведения двух векторов.

double dot(const std::vector<double> &x, const std::vector<double> &y)

Функция для реализации линейной комбинации двух векторов. res = a * x + b * y

void axpby(const std::vector<double> &x, const std::vector<double> &y, double a, double b, std::vector<double> &res)

Функция для перемножении матрицы a, которая хранится в CSR формате на вектор b, результат сохраняется в res.

void SpMv(const std::vector<int> &ia, const std::vector<int> &ja, const std::vector<double> &a, const std::vector<double> &b, std::vector<double> &res)

Функция для третьего этапа задания. tol - необходимая относительная точность решения (отношение L2 нормы невязки к норме правой части. Невязка – это r = Ax–b, где x - полученное решение) . x - полученное решение. k - количество итераций алгоритма. res - – L2 норма невязки.

void Solve(int N, const std::vector<int> &ia, const std::vector<int> &ja, const std::vector<double> &A, const std::vector<double> &b, const std::vector<double> &M, double tol, std::vector<double> &x, int &k, double &res)

Функция для 4 этапа задания.

void Report(int N, const std::vector<int> &ia, const std::vector<int> &ja, const std::vector<double> &A, const std::vector<double> &b, const std::vector<double> &x, double tol, std::ofstream &fout)

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	0.001168	0.008615	0.0004077	0.0008681	0.0016579	1.52
			0.0055861			
100000	0.010783	0.044081	0.0040078	0.0081638	0.013074	14.31
			0.025571			
1000000	0.060024	0.25378	0.044341	0.075873	0.1362	142.45
			0.31101			
10000000	0.29022	2.6552	0.4574285	0.833832	1.3713102	1423.87
			3.1858829			

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.

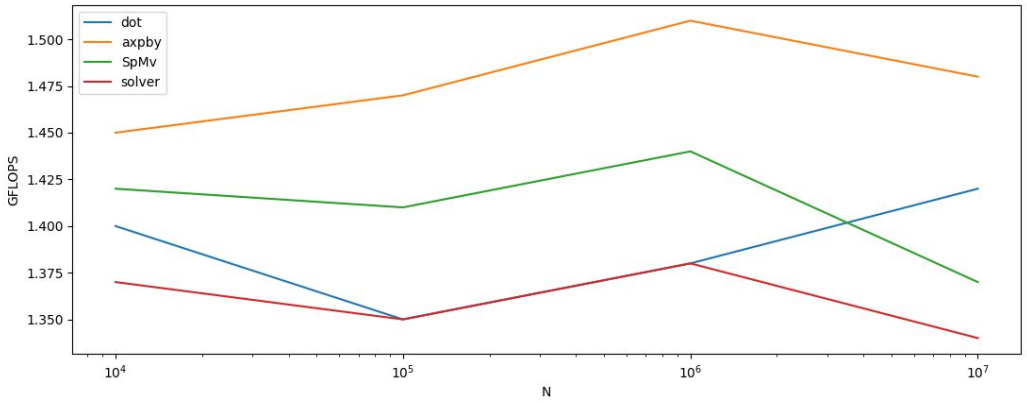


Рис. 1: Производительность 3 основных вычислительных ядер и всего солвера

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	10783 | 1	8615 | 1	407,7 | 1	868 | 1	1657,9 | 1
			5586,1 | 1		
2	7436.5 | 1.45	4582.4 | 1.88	242.6 | 1.68	647.7 | 1.34	1055.9 | 1.57
			3990 | 1.4		
4	5015.3 | 2.15	2843.2 | 3.03	129.8 | 3.14	466.6 | 1.86	632.7 | 2.62
			3019 | 1.85		
8	3267.5 | 3.3	1403.1 | 6.15	69.8 | 5.84	321.4 | 2.71	274.9 | 6.03
			1692.7 | 3.3		
16	5234.4 | 2.06	1323.3 | 6.51	125.8 | 3.24	394.5 | 2.2	301.4 | 5.5
			1995 | 2.8		

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	290220 | 1	2655200 | 1	457428,5 | 1	833832 | 1	1371310,2 | 1
			3185882,9 | 1		
2	183683.5 | 1.58	1390157 | 1.91	2265946|1.72	591370 | 1.41	8464877|1.62
			22435795 | 1.42		
4	123497 | 2.35	876303 | 3.03	145677 | 3.14	448296 | 1.86	523400 | 2.62
			1722098 | 1.85		
8	68448 |4.24	493531.5 | 5.38	142946 | 3.2	234222 | 3.56	315243 | 4.35
			977264 | 3.26		
16	140883 | 2.06	407864 | 6.51	141181 | 3.24	379014 | 2.2	249329 | 5.5
			1137815 | 2.8		

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

3 Анализ полученных результатов

Для начала рассчитаем $TBP = min(TPP, BWAI)$. 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: $TPP = 10C * 2.5GHz * 2 * 512/64 = 400\ GFlops$. За BW возьмём пропускную способность контроллера памяти - 230 GB/s.

Kernel	AI, Flop / byte	TBP, GFlops	%, TPP
dot	0.0625	14.375	3.5
axpby	0.075	17.25	4.3
spmv	0.058	13.34	3.3

Таблица 4: AI, TBP для трёх вычислительных ядер