Практическое задание №1
Многопоточная реализация солвера CG для СЛАУ с разреженной матрицей

Введение

Солверы систем линейных алгебраических уравнений (СЛАУ) широко применяются в суперкомпьютерном моделировании. Большие разреженные системы возникают в расчетах сеточными методами.

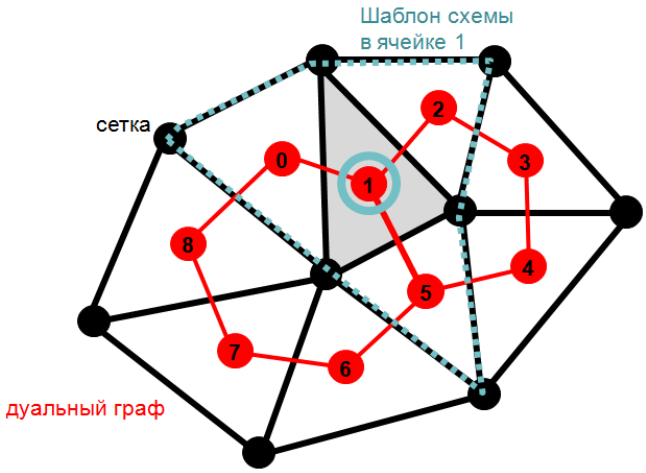
Метод CG хорошо подходит в качестве практического задания, поскольку он имеет достаточно простой алгоритм и использует всего три базовые операции линейной алгебры.

Дискретная аппроксимация разностных операторов на пространственных сетках определяет топологию матрицы СЛАУ, т.е. её портрет. Портретом матрицы *A* называется множество пар индексов строк и столбцов (i,j), для которых соответствующий коэффициент $a_{ij} \neq 0$.

Как правило, портрет матрицы совпадает с графом связей расчетных ячеек сетки. Вершинам графа сопоставлены расчетные ячейки – узлы сетки или сеточные элементы сетки (в зависимости от того, где заданы сеточные функции). Наличие ребра между двумя вершинами графа означает, что шаблон численной схемы пространственной дискретизации, центрированный в одной из ячеек, включает другую ячейку.

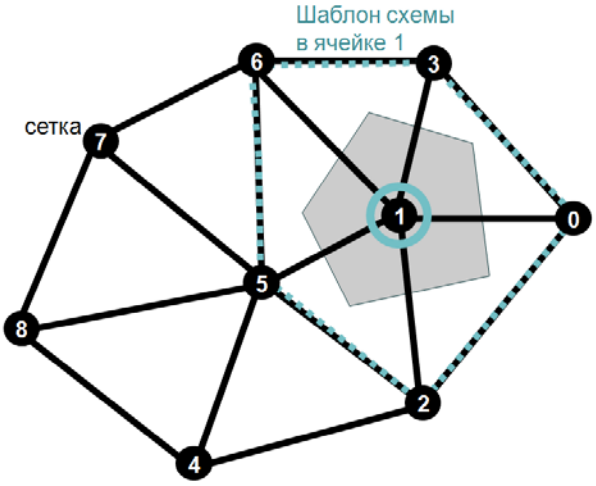
На рис. 1 показан пример двухмерной треугольной сетки и ее дуального графа, а также портрет матрицы, соответствующий этому примеру пространственной дискретизации с определением сеточных функций в элементах сетки. Аналогичный пример для случая, когда сеточные функции заданы в узлах, показан на рис. 2.

Значения коэффициентов матрицы определяются используемым численным методом. Способ их расчета в рамках данного задания нас не интересует. Портрет матрицы и значения ее коэффициентов являются входными данными для солвера.



	0	1	2	3	4	5	6	7	8
0	#	#							#
1	#	#	#			#			
2		#	#	#					
3			#	#	#				
4				#	#	#			
5		#			#	#	#		
6						#	#	#	
7							#	#	#
8	#							#	#

Рис 1. Пример треугольной сетки с определением сеточных функций в элементах (треугольниках). В этом примере шаблон схемы в ячейке состоит из этой ячейки и ее соседних ячеек, т.е. ячеек, имеющих с данной ячейкой общую грань. Справа показан портрет матрицы. Номера в узлах обозначают номера неизвестных в векторе. Номер элемента в сетке (красные кружки) соответствует номеру строки матрицы. Номера столбцов в строке узла соответствуют номерам соседних с ним элементов.



	0	1	2	3	4	5	6	7	8
0	#	#	#	#					
1	#	#	#	#		#	#		
2	#	#	#		#	#			
3	#	#		#			#		
4			#		#	#			#
5		#	#		#	#	#	#	#
6		#		#		#	#	#	
7						#	#	#	#
8					#	#		#	#

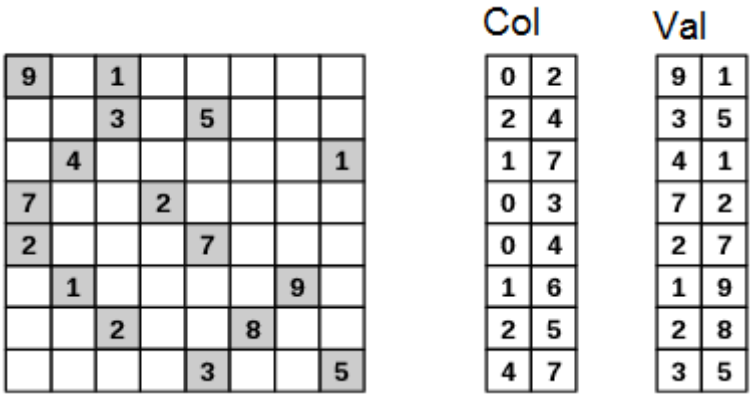
Рис 2. Пример треугольной сетки с определением сеточных функций в узлах сетки. В этом примере шаблон схемы в ячейке состоит из этой ячейки и ее соседних ячеек, т.е. ячеек, имеющих с данной ячейкой соединение ребром сетки. Номера в узлах обозначают номера неизвестных в векторе. Справа показан портрет матрицы. Номер узла в сетке соответствует номеру строки матрицы. Номера столбцов в строке узла соответствуют номерам соседних с ним узлов.

Формат разреженной матрицы

Предлагается использовать построчно-разреженный формат ELLPACK. Разреженная матрица A размера NxN, в которой имеется по M ненулевых коэффициентов в строке, представляется в виде двух массивов:

- **int Col[N*M];** хранит номера столбцов ненулевых коэф. подряд по всем строкам
- **double Val[N*M];** хранит значения ненулевых коэффициентов подряд по всем строкам

Пример:



Пример доступ к ненулевым коэффициентам i-й строки:

```
for(int _j= 0; _j<M; _j++){
     int j = JA[i*M + _j]; // номер столбца
     double a_ij = A[_j];  // значение коэффициента aij
     . . .
}
```

Генерация тестовой расчетной области

Для солвера СЛАУ портрет матрицы и значения ее коэффициентов являются входными данными.

Чтобы иметь возможность тестировать реализацию солвера, не прибегая к копированию объемных входных данных из реальных задач, предлагается сделать генератор расчетной области, которая будет определять портрет матрицы. Суть данного действия - получить протрет матрицы.

Предлагается в качестве тестовой расчетной области использовать трехмерную декартову решетку, размерность которой, Nx, Ny, Nz, можно варьировать при тестировании.

Тут все полностью аналогично рис. 2, есть набор узлов, соединенных ребрами, только связи между узлами имеют регулярную структуру, что всё

сильно упрощает. Узлы могут адресоваться по трем индексам, I=0,...,Nx-1; J=0,...,Ny-1; K=0,...,Nz-1. Размер СЛАУ при этом будет

N = Nx*Ny*Nz.

Позиция в векторе в СЛАУ *i*=0,...,N-1 соответствует набору (I,J,K) индексов узла в решетке:

i = K*(Nx*Ny)+ J*Nx + I.

В *i*-й строке соответствующей матрицы будут следующие ненулевые позиции (т.е. набор номеров столбцов *Col(i)*):

i-Nx*Ny, если K>0;

i-Nx, если J>0;

i-1, если I>0;

i;

i+1, если I<Nx-1;

i+Nx, если J<Ny-1;

i+Nx*Ny, если K<Nk-1;

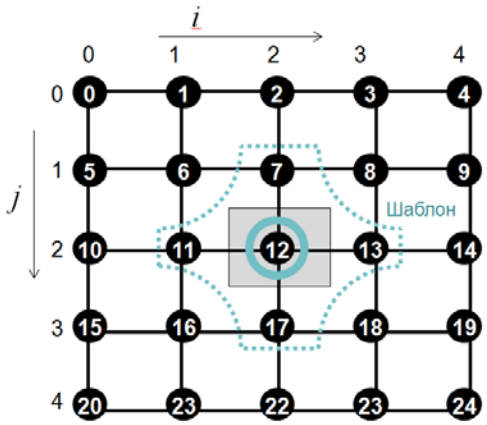
По решетке заданного размера необходимо сгенерировать описанный выше портрет матрицы в формате ELLPACK и заполнить коэффициенты матрицы произвольными значениями, но так, чтобы матрица имела диагональное преобладание (сумма модулей внедиагональных элементов строки должна быть меньше диагонального элемента).

Например, для внедиагональных элементов строки можно взять синус от индексов строки и столбца, $a_{ij} = cos(i * j + 3.14)$, $i \neq j$, $\boldsymbol{j} \in \boldsymbol{Col}(\boldsymbol{i})$ Набор индексов столбцов *Col(i)* для каждой строки *i* задан портретом матрицы!

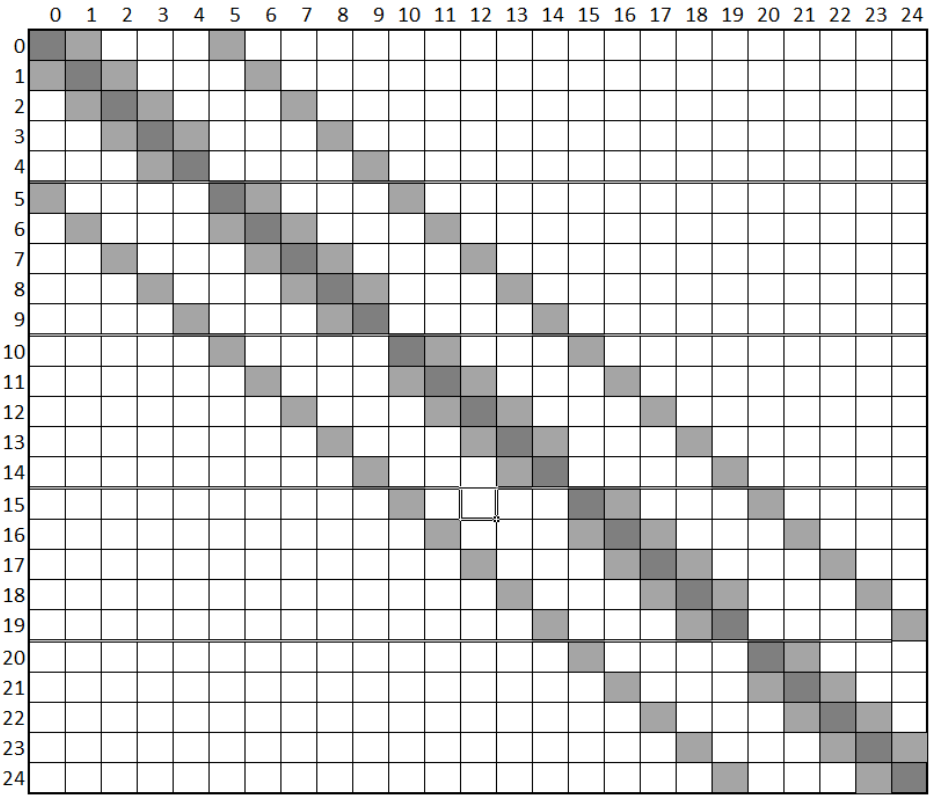
Диагональный элемент вычислить как сумму модулей внедиагональных элементов строки, умноженную на больший единицы коэффициент:

$$a_{ii} = 1.5 \sum_{j, j \neq i} |a_{ij}|$$

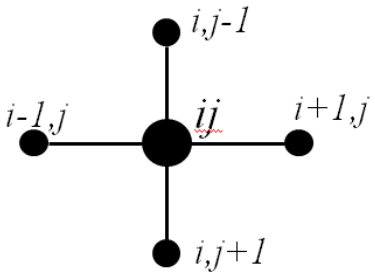
Для наглядности рассмотрим пример двухмерной решетки размера Nx*Ny, где Nx=Ny=5. Размер системы N=25.



Из топологии связей узлов в решетке следует портрет матрицы:



Шаблон схемы в данном случае выглядит так:



Солвер CG (с предобуславливателем Якоби)

Алгоритм предобусловленного метода CG имеет следующий вид:

Choose an initial guess x_0;
$r_0 = b - Ax_0$;
convergence = false;
k = 1;
repeat
  $z_k = M^{-1} r_{k-1}$;
  $\rho_k = (r_{k-1}, z_k)$;
  if $k = 1$ **then**
    $p_k = z_k$;
  else
    $\beta_k = \rho_k / \rho_{k-1}$;
    $p_k = z_k + \beta_k p_{k-1}$;
  end if
  $q_k = Ap_k$;
  $\alpha_k = \rho_k / (p_k, q_k)$;
  $x_k = x_{k-1} + \alpha_k p_k$;
  $r_k = r_{k-1} - \alpha_k q_k$;
  if $(\rho_k < \varepsilon)$ **or** $(k \geq maxiter)$ **then**
    convergence = true;
  else
    k = k + 1;
  end if
until convergence

В случае предобуславливателя Якоби матрица M – диагональная матрица, с диагональю из матрицы A. Начальное приближение – нулевое.

Реализация

Выполнять реализацию предлагается следующим образом.

1. Сделать генератор матрицы для расчетной области, представленной трехмерной декартовой решеткой заданного размера Nx, Ny, Nz, проверить корректность заполнения матрицы. Заполнять коэффициенты, например, можно так: для внедиагональных элементов строки можно взять синус от индексов строки и столбца, $a_{ij} = cos(i * j + 3.14)$, $i \neq j$, $\boldsymbol{j} \in \boldsymbol{Col}(\boldsymbol{i})$, а диагональный элемент вычислить как сумму модулей внедиагональных элементов строки, домноженную на больший единицы коэффициент: $a_{ii} = 1.5 \sum_{j,j \neq i} |a_{ij}|$

2. Сделать исходные последовательные реализации операций dot, axpby, SpMV, а также операций копирования вектора и заполнения вектора константой (соответствует оператору присваивания в алгоритме выше).

3. Сделать проверочный тест, выполняющий операции dot, axpby, SpMV с матрицей, заполненной как в п. 1, и векторами, заполненными произвольными различающимися значениями, например X[i]=cos(i*i), Y[i]=sin(i*i). Проверочный тест для каждой операции должен выдать контрольные значения: для dot – просто результат, для axpby и SpMV контрольные значения по выходному вектору, например: сумма всех элементов, L2 норма (sqrt(dot(X,X)), C норма. Последующие параллельные реализации можно будет сравнивать с исходными последовательными реализациями по этим контрольным значениям для подтверждения корректности.

4. Сделать солвер CG на основе исходных операций. Проверить работоспособность солвера на сгенерированной матрице и правой части, заполненной, например, аналогично: B[i]=cos(i). Начальное значение вектора решения нулевое.

5. Попробовать выполнить оптимизацию последовательных версий базовых операций, сохранив также исходные версии для сравнения. Можно применить развертку циклов, SIMD инструкции и т.д. Сделать таймирование по всем базовым операциям, оценить улучшения.

6. Сделать многопоточные параллельные реализации базовых операций. Получить многопоточную корректно работающую версию солвера.

Реализовать все вышеописанное можно в виде программы, которая выполняет генерацию тестовой системы, решает ее солвером, выдает результат в виде таймирования по всем базовым операциям и по общему времени работы солвера, выдает затраченное число итераций и отношение нормы невязки к норме правой части.

Исходный код на C или C++, осторожно, новые стандарты (C++11 и выше) могут не поддерживаться на кластерах. Сборка make.

Аргументы с командной строки

nx=<int> ny=<int> nz=<int> размер решетки топологии для генерации матрицы размера NxN, N=nx·ny·nz

tol=<double> невязка относительно нормы правой части

maxit=<int> максимальное число итераций

nt=<int> число нитей

qa флаг тестирования базовых операций

Отчет

По результатам нужно будет подготовить отчет, в котором привести данные по исследованию производительности солвера. Для каждой из базовых операций и для всего алгоритма исследовать зависимость достигаемой производительности от размера системы N, построить графики GFLOPS от N. Измерить OpenMP ускорение для различных N. Оценить максимально достижимую производительность (TBP) с учетом пропускной способности памяти и сравнить с фактической производительностью. Оценить достигаемую скорость передачи данных.

Сдавать код нужно вместе с инструкцией по компиляции и запуску.
Желательно сделать make-файл.
Желательно, чтобы исполнимый файл печатал хэлп при запуске без аргументов.
PDF отчета нужно загружать в систему отдельным файлом, не в общем архиве с кодом.

Требования к отчету:

Титульный лист, содержащий
1.1 Название курса
1.2 Название задания
1.3 Фамилию, Имя, Отчество(при наличии)
1.4 Номер группы
1.5 Дата подачи

Содержание отчета:

2 Описание задания и программной реализации
2.1 Краткое описание задания
2.2 Краткое описание программной реализации - как организованы данные, какие функции реализованы (название, аргументы, назначение)
Просьба указывать, как программа запускается – с какими параметрами, с описанием этих параметров.
2.3 Описание опробованных способов оптимизации последовательных вычислений (по желанию)

3 Исследование производительности
3.1 Характеристики вычислительной системы:
описание одной или нескольких систем, на которых выполнено исследование (подойдет любой многоядерный процессор).
тип процессора, количество ядер, пиковая производительность, пиковая пропускная способность памяти.
по желанию - промерять и на своем декстопе/ноуте, и на кластере
3.1.11 Описание компиляции под конкретную систему
3.2 Результаты измерений производительности
3.2.1 Последовательная производительность
Для каждой из трех базовых операций и для всего алгоритма солвера исследовать зависимость достигаемой производительности от размера системы N, построить графики GFLOPS от N. *Нескольких N достаточно: N=1000, 10000, 100000, 1000000*
Для повышения точности измерений, замеры времени лучше производить, выполняя набор операций многократно в цикле, чтобы осреднить время измерений. Суммарное время измерений чтобы получалось порядка нескольких секунд.
Оценить выигрыш от примененной оптимизации (по желанию)
3.2.2. Параллельное ускорение
Измерить OpenMP ускорение для различных N для каждой из 3-х базовых операций и для всего алгоритма солвера.

4 Анализ полученных результатов

4.1 Процент от пика

оценить для каждой из трех базовых операций, какой процент от пиковой производительности устройства составляет максимальная достигаемая в тесте производительность

4.2 Процент от достижимой производительности

аналогично оценить для каждой операции процент от максимально достижимой производительности с учетом пропускной способности памяти.

Приложение1: исходный текст программы в отдельном c/c++ файле

Требования к программе:

1 Программа должна использовать OpenMP или posix threads для многопоточного распараллеливания

2 Солвер должен корректно работать, т.е. показывать быструю сходимость.