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

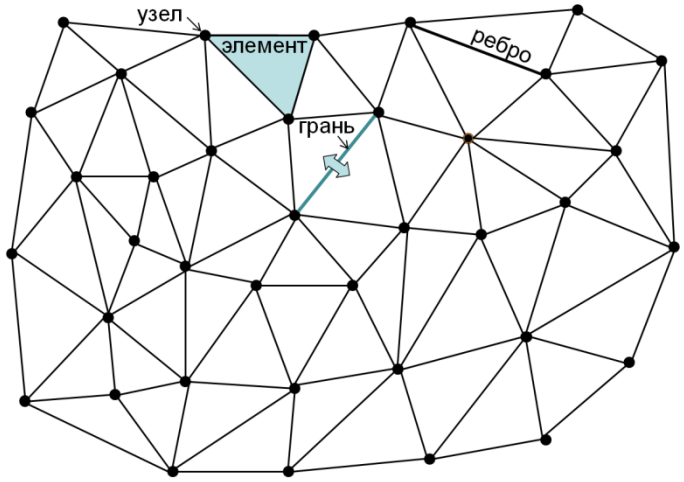
Цели задания:

- освоение базовых структур данных для представления неструктурированной сетки, графа связей расчетных ячеек, портрета разреженной матрицы;
- постижение взаимосвязи между сеткой, графом и разреженной матрицей;
- освоение многопоточного распараллеливания простейших операций.

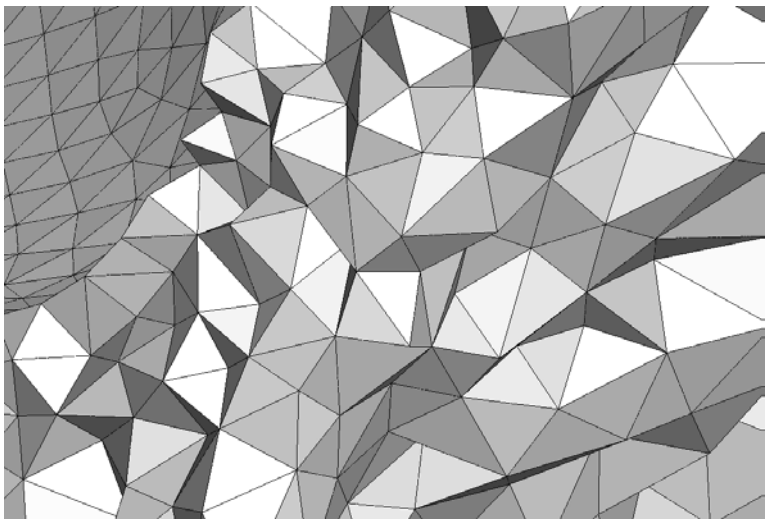
В качестве наиболее простой и репрезентативной операции выбрано матрично-векторное произведение с разреженной матрицей, портрет которой соответствует дискретизации на неструктурированной сетке. Задание будет разделено на этапы.

Введение

Для дискретизации по пространству используется неструктурированная сетка. Сетка – это разбиение некоторой пространственной расчетной области на сеточные элементы (в двумерном случае – многоугольники, в трехмерном – многогранники). Сетка задана в виде набора узлов и набора элементов. Объединение сеточных элементов полностью заполняет нашу расчетную область, а сами элементы не наползают друг на друга, а стыкуются через общие грани и узлы. Ниже на картинке показан пример двумерной треугольной сетки. Двумерной – для простоты рисования картинок. В случае трехмерной сетки всё, по сути, то же самое, для наших целей никакой разницы.

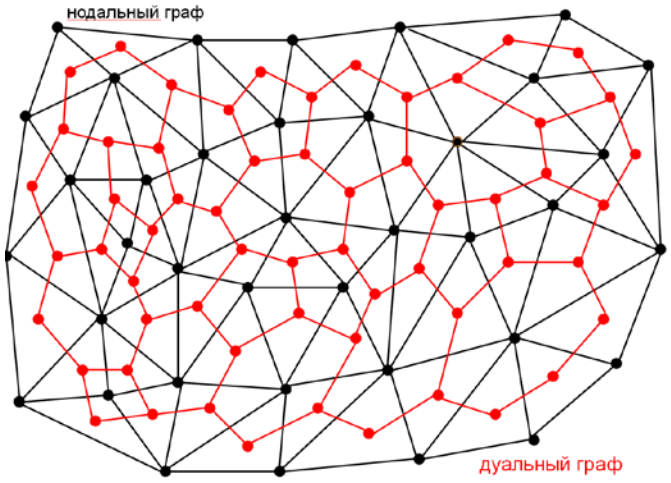


У нас есть несколько наборов сеточных объектов – узлов, элементов, ребер и граней этих элементов. На картинке элементы – это треугольнички. А ребра и грани в двумерном случае – это одно и то же. В трехмерном случае грани станут многоугольниками – гранями многогранных элементов, а ребра останутся ребрами. В 3D это выглядит как-то так:



Теперь надо понять, где на сетке заданы сеточные функции, то есть, какие-то переменные, хранящиеся в сеточных объектах. Можно задавать переменные 1) в узлах, а можно 2) в элементах. В первом случае контрольные объемы, то есть расчетные ячейки или просто ячейки, будут строиться некоторым образом вокруг узлов, во втором – эту роль будут выполнять сами сеточные элементы. Да не важно где, и то, и другое все равно сведется к графу.

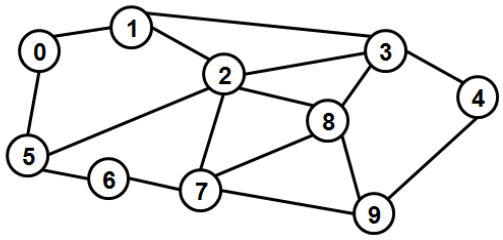
Граф описывает смежность ячеек, то есть элементов или узлов. В таком графе связей (или смежности) наших ячеек вершины графа соответствуют ячейкам – элементам или узлам сетки. Ребрам графа соответствуют общие грани между ячейками. Если между ячейками i и j есть общая грань, то в графе вершины i и j соединены ребром. Наша сеточка на картинке выше, это, по сути, и есть граф. Но это узловой граф. Его вершины соответствуют узлам. Для варианта с определением переменных в элементах нужен двойственный (dual) граф. Вот тут он красным нарисован.



Теперь у нас есть вершины графа, с которыми ассоциированы ячейки и заданные в них наборы переменных, и его ребра, которым соответствуют связи между ячейками. Если ячейки построены вокруг узлов, то это черный граф на картинке выше, если ячейками являются элементы, то это красный граф.

У графа есть матрица смежности. Это такая матрица NxN, где N – число вершин в графе, в которой ненулевые позиции соответствуют ребрам между вершинами графа. Коэффициент матрицы a_{ij} в j-м столбце i-й строки равен 1, если между вершинами i и j графа есть ребро, иначе там ноль. Если такое ребро есть, то и a_{ji} – неноль. Значит, число ненулей в матрице равно удвоенному числу ребер в графе.

Возьмем для примера какой-нибудь граф. Построим для него матрицу смежности и запишем ее портрет. Вот такой граф:



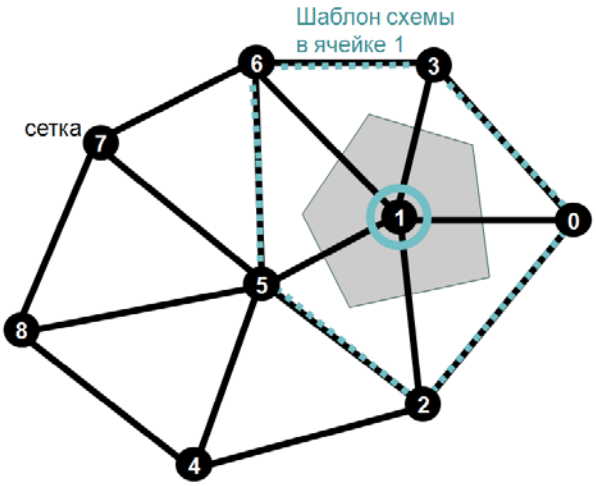
Матрица смежности для этого графа:

	0	1	2	3	4	5	6	7	8	9
0	0	**1**	0	0	0	**1**	0	0	0	0
1	**1**	0	**1**	**1**	0	0	0	0	0	0
2	0	**1**	0	**1**	0	**1**	0	**1**	**1**	0
3	0	**1**	**1**	0	**1**	0	0	0	**1**	0
4	0	0	0	**1**	0	0	0	0	0	**1**
5	**1**	0	**1**	0	0	0	**1**	0	0	0
6	0	0	0	0	0	**1**	0	**1**	0	0
7	0	0	**1**	0	0	0	**1**	0	**1**	**1**
8	0	0	**1**	**1**	0	0	0	**1**	0	**1**
9	0	0	0	0	**1**	0	0	**1**	**1**	0

Часто в вычислительных алгоритмах приходится иметь дело с разреженными матрицами, портрет которых имеет ненули на главной диагонали, а во внедиагональных позициях совпадает с портретом матрицы смежности. С такими матрицами мы и будем работать: **матрица смежности + главная диагональ.**

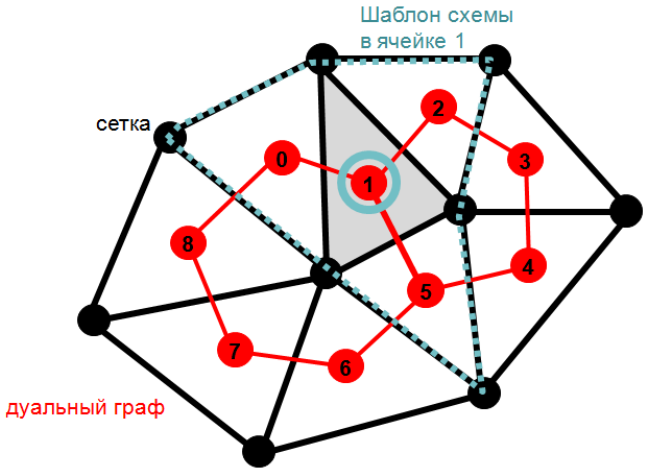
Пример для случая, когда сеточные функции заданы в узлах, показан на рис. 1.

На рис. 2 показан пример для случая определения сеточных функций в элементах сетки. Значения коэффициентов матрицы определяются используемым численным методом. Способ их расчета в рамках данного задания нас не интересует, мы их нагенерим потом как попало.



	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 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. Также присутствует диагональный элемент – каждый узел сам себе сосед.



	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 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. Также присутствует диагональный элемент – каждый элемент сам себе сосед.

Формат хранения графа и разреженной матрицы

Граф можно описать в виде портрета его матрицы смежности. Запишем портрет матрицы, то есть описание, где в матрице ненулевые позиции, в построчено-разреженном формате CSR (Compressed Sparse Row). Пусть в графе N вершин и E ребер. Для хранения этого добра нам нужны 2 интовых массива:

IA – размера N+1, также называют rowptr, xadj, ... В нем для каждой строки храним позицию начала списка столбцов данной строки. Последний элемент в массиве, IA[N], хранит общее число ненулей в матрице (равное удвоенному числу ребер графа – 2E).

JA – размера 2E, также называют colptr, adjncy, ... В нем подряд для всех строк матрицы хранятся номера столбцов с ненулевыми значениями. Пусть для определенности номера столбцов каждой строки упорядочены по возрастанию.

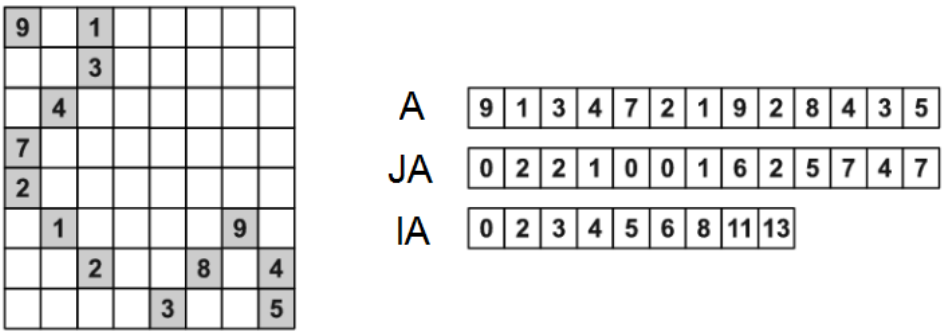
Как нам, например, перебрать все ненули i-й строки? Что то же самое, что перебрать всех соседей i-й вершины. Вот так, например:

```
for(int col=IA[i]; col<IA[i+1]; ++col){
    j = JA[col]; // номер соседней вершины, т.е. соединенной с вершиной i ребром.
    ...
}
```

Это был портрет матрицы. Для того, чтобы это стало матрицей, нам нужны коэффициенты. Они хранятся в третьем массиве:

A – размера 2E, в нем подряд по всем строкам хранятся коэффициенты матрицы.

Пример:



Доступ к ненулевым коэффициентам i-й строки:

```
for(int _j=IA[i]; _j<IA[i+1]; _j++){
      int j = JA[_j]; // номер столбца
      double a_ij = A[_j]; //значение коэффициента aij
      . . .
}
```

Постановка задачи

Работа программы, которую будем делать, состоит из 4 этапов:

1. **Generate** - Генерация графа/портрета по тестовой сетке
2. **Fill** - заполнение матрицы по заданному портрету
3. **Solve** - решение СЛАУ с полученной матрицей
4. **Report** - проверка корректности программы и выдача измерений.

Сделать задание можно на языках C, C++.

С новыми стандартами C++11 и выше могут быть проблемы на кластерах ВМК.

Этап 1 задания 1: генерация портрета на основе тестовой сетки (Generate)

Для простоты будем иметь дело с двухмерной неструктурированной смешанной сеткой, состоящей из треугольников и четырехугольников.

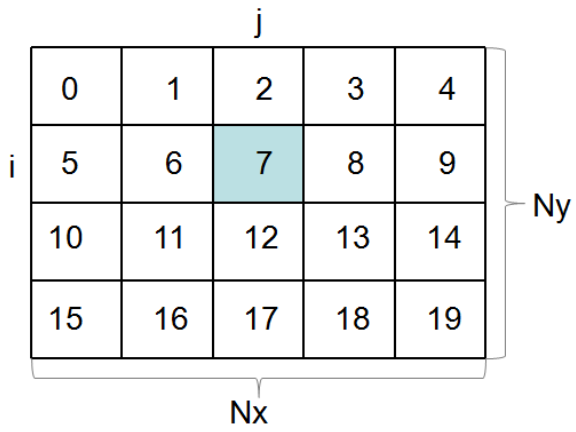
Есть два варианта определения сеточных функций

A) в узлах;

B) в элементах.

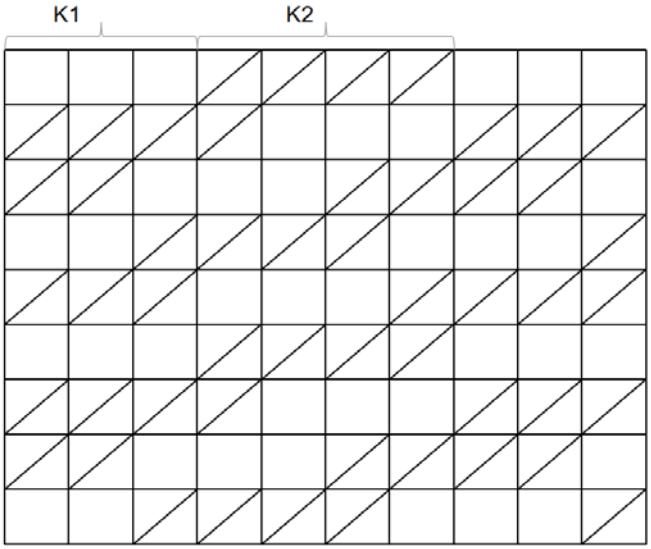
Для упрощения генерации тестовой сетки будем использовать самое просто, что только может быть – двухмерную решетку.

Решетка состоит из N = Nx×Ny клеточек. У каждой клеточки есть позиция (i,j), где i – номер строки, j – номер столбца. Нумерация клеточек построчно слева направо, сверху вниз. Номер клеточки в единой общей нумерации I = i*Nx + j.



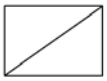
Итак, мы задали два параметра Nx, Ny, получили решетку. Теперь получим из решетки сетку. Чтобы в сетке были разные элементы, часть квадратных клеток поделим на треугольники. Для этого введем еще два параметра K1, K2, чтобы регулировать соотношение числа треугольников и четырехугольников. Будем делить клетки следующим образом. K1 клеток не делим, K2 следующих клеток делим, K1 следующих клеток снова не делим, K2 следующих клеток снова делим, и так далее. Легко догадаться, что клетка поделена на треугольники, если ее нормер I%(... впрочем, вы сами легко напишете формулку.

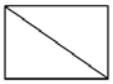
Пример для Nx=10, Ny=9, K1=3, K2=4:



Да, поделить четырехугольник можно двумя способами:

Способ 1 Способ 2





Получились следующие варианты:

A) ячейки, то есть вершины графа = узлы

B) ячейки, то есть вершины графа = элементы

1) делим клетки способом 1

2) делим клетки способом 2.

Итого 4 варианта: A1, A2, B1, B2.

Вариант определяем по списку группы по алфавиту. Берем порядковый номер в списке, находим остаток от деления на 4, и берем один из этих вариантов в указанном выше порядке.

На этапе 1 нужно сгенерировать "топологию" связей ячеек, то есть построить графа, то есть сгенерировать портрет матрицы смежности графа. И еще надо дополнить этот портрет диагональными элементами.

Входные данные:

Nx, Ny – число клеток в решетке по вертикали и горизонтали;

k1, k2 – параметры для количества ~~однопалубных и двухпалубных кораблей~~ треугольных и четырехугольных элементов.

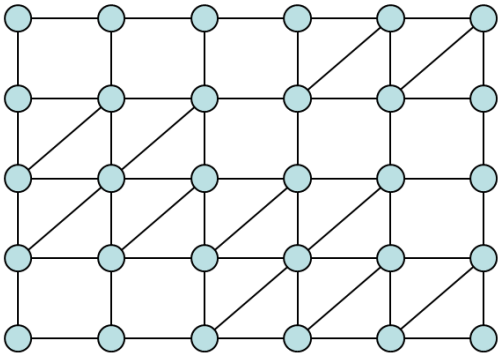
Выходные данные:

N – размер матрицы = число вершин в графе;

IA, JA – портрет разреженной матрицы смежности графа, дополненный главной диагональю (в формате CSR).

Разберемся еще подробнее с разными вариантами, что там будет графом, и как что нумеруется.

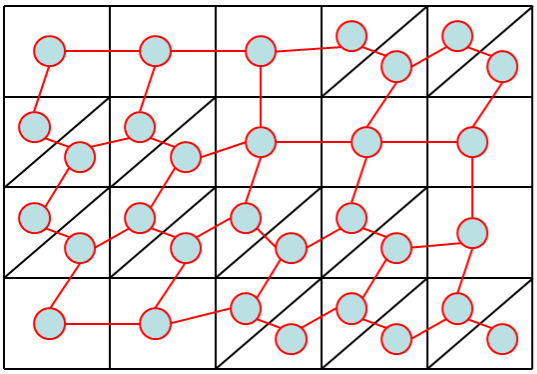
Вариант А) – вершины графа = узлы сетки. Узлы сетки - это точки пересечения сеточных линий. Вот они кружочками отмечены:



Nx = 5
Ny = 4
K1 = 3
K2 = 4

Граф связей узлов совпадает с сеткой, он тут и нарисован на этой картинке. Вершин в графе N = (Nx+1)(Ny+1). Нумерация узлов точно такая же – слева направо, сверху вниз. Удобство этого варианта – сразу понятно число вершин, оно не меняется от разбиения на треугольники. Неудобство – чтобы посчитать число соседей у вершины, надо смотреть, что делается в соседних клетках.

Вариант В) – вершины графа = элементы сетки, то есть четырехугольники и треугольники. Граф связей – двойственный граф сетки. Элементы считаем смежными, если у них есть общая грань. Граф будет такой:



Nx = 5
Ny = 4
K1 = 3
K2 = 4

Нумерация элементов такая же, слева направо, сверху вниз. Что там левее/правее/выше/ниже – определяем по центру масс элемента. Удобство этого варианта – сразу понятно, сколько соседей у ячеек. Неудобство – число ячеек зависит от параметров разбиения на треугольники.

Во всех вариантах не забываем, что к портрету **надо добавлять главную диагональ.**

Генерацию надо сделать в виде функции с входными и выходными данными в качестве аргументов.

В мэйне только считываются параметры пользовательского ввода (из командной строки или файла) и вызывается эта функция. Интерактивный ввод не использовать. Ну то есть не надо никаких cin >> и прочих scanf.

Программа должна брать хотя бы один аргумент командной строки. Если ввод параметров через файл, то это имя входного файла. Если программа запущена без аргументов, она должна напечатать хелп, как ей пользоваться.

Программа должна иметь проверку корректности пользовательского ввода. В случае ошибки она должна выйти с информативным сообщением, а не по какому-нибудь сегфолту.

Для проверки результатов программа должна уметь печатать выходные данные в текстовый файл или в stdout. Для этого должен быть параметр, включающий отладочную печать. По-умолчанию печать выключена.

Для проверки задания можно

1) напечатать выходные данные для маленькой сетки, которую можно просто нарисовать на бумаге и проверить;

2) сравнить выходные данные с результатами других авторов с тем же вариантом задания;

3) проверить, что расход памяти и время работы программы растут линейно с ростом числа ячеек;

4) сравнить время работы с другими авторами.

Срок выполнения этапа 28.09

Сдавать этап можно будет на занятии 28.09, в систему сдавать пока не надо, туда целиком задание пойдет.

Этап 2 задания 1: построение СЛАУ по заданному портрету матрицы (Fill)

Входные данные:

N – размер (квадратной матрицы NxN);

IA, JA – портрет матрицы.

Выходные данные:

A – массив ненулевых коэффициентов матрицы (размера IA[N]);

b – вектор правой части (размера N).

На этом этапе мы уже ничего не знаем о том, что сетка когда-то была решеткой. Теперь мы работаем только с топологией (то есть связями между ячейками/объектами/вершинами графа) в общем виде.

Для заполнения матрицы и правой части лучше использовать одинаковые формулы, чтобы можно было сравнивать результаты программы друг с другом для проверки (для одинаковых вариантов, естественно).

Например, для внедиагональных элементов строки можно взять какой-нибудь косинус от индексов строки и столбца, $a_{ij} = cos(i * j + i + j)$, $i \neq j$, $\boldsymbol{j} \in \boldsymbol{Col(i)}$, где *Col(i)* – множество номеров столбцов для строки i, которые входят в портрет матрицы (это то, что лежит в JA[IA[i],...,IA[i+1]–1]).

Диагональный элемент в каждой строке будем вычислять как сумму модулей внедиагональных элементов строки, умноженную на больший единицы коэффициент:

$$a_{ii} = 1.234 \sum_{j,j \neq i} |a_{ij}|$$

Все остальные коэффициенты матрицы равны нулю (К.О.). Домножаем диагональ на коэффициентик, чтобы у матрицы было диагональное преобладание. Это нужно, чтобы солвер, который мы дальше будем делать, устойчиво сходился к решению, а не болтался в проруби.

Заполняем матрицу так: идем по строкам, перебираем номера столбцов, если внедиагональный коэффициент – заполняем по формуле и прибавляем значение в сумму по строке, если диагональный – запоминаем позицию и пропускаем. Дошли до конца строки, домножили полученную сумму по строке на коэффициент и записали в диагональный элемент. Перешли к следующей строке, занулив сумму.

Вектор правой части заполняем по формуле $b_i = sin(i)$.

Когда функции для этапов 1 и 2 сделаны и корректно работают, нужно внедрить многопоточное распараллеливание средствами OpenMP. В связи с этим у программы появляется еще один входной параметр:

T – число нитей (threads).

Функции этапов 1 и 2 нужно распараллелить. Построение портрета, заполнение коэффициентов – всё должно выполняться параллельно T нитями.

Чтобы оценить эффект от распараллеливания, нужно добавить замеры времени. Для этого можно использовать функцию *omp_get_wtime*.

Замечания по выдаче.

Программа может сообщать в лог о прохождении этапов, выдавать контрольные величины (например, число вершин графа, число ребер, число ненулей в портрете, ...), выдавать табличку замеров времени. По замерам будем проверять параллельное ускорение.

По умолчанию, без запроса пользователя, программа не должна никуда печатать никаких массивов, ни портрет, ни коэффициенты, ни в страуд, ни в файл. Выдача отладочной информации – только в режиме отладки. Размеры сетки, которые надо будет тестировать, это 1 ~ 10 млн ячеек (К.О. сообщает: на таких размерах печатать портрет– плохая идея).

Результаты работы программы в параллельном и последовательном режиме должны быть полностью идентичны. Для проверки можно, например, печатать в файл массивчики и сравнивать файлы. Можно вычислять по массивам контрольные суммы, какие-нибудь хэш-функции, и сравнивать просто числа, что удобнее, особенно на больших размерах. Можно и так, и эдак, но массивы – только в режиме отладки, а контрольные величины можно и всегда выдавать.

Срок выполнения этапа 05.10

Этап 3 задания 1: Решение СЛАУ (Solve) итерационным методом

Входные данные:

N – размер (квадратной матрицы NxN);

IA, JA – портрет матрицы;

A – массив ненулевых коэффициентов матрицы (размера IA[N]);

b – вектор правой части (размера N).

tol – относительная точность решения (отношение L2 нормы невязки к норме правой части. Невязка – это r = Ax–b, где x - полученное решение)

Выходные данные:

x – вектор решения (размера N);

n – количество выполненных итераций;

res – L2 норма невязки.

Поскольку матрица симметричная, будем использовать простейший метод сопряженных градиентов CG (Conjugate Gradient) 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.

Для отладки рекомендуется на каждой итерации вычислять норму невязки и печатать в стандартный поток вывода с новой строки номер итерации и текущую норму невязки

системы на данной итерации. Этот лог итерационного процесса можно сравнивать друг с другом для проверки (у кого одинаковые варианты, естественно).

Для решателя понадобится сделать 3 кернела, то есть функции для алгебраических операций:

- скалярное произведение – dot;
- линейная комбинация двух векторов – axpby (**x** = a**x**+b**y**), **x,y** - вектора, a,b-скалярные значения;
- матрично-векторное произведение с разреженной матрицей - SpMV (sparse matrix-vector)

Рекомендуется сделать проверочные тесты для каждой из операций. Входные вектора можно заполнять по какой-то формуле от номера элемента. 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)), и т.д. Последующие параллельные реализации можно будет сравнивать с исходными последовательными реализациями по этим контрольным значениям для подтверждения корректности.

Для решателя нужно сделать замеры времени, для чего можно использовать функцию omp_get_wtime().

Для каждой из трех базовых операций тоже нужно сделать замеры.

Нужно 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 – кернелам, оценить улучшения.

Затем нужно сделать многопоточные параллельные реализации базовых операций с помощью **OpenMP**. В итоге надо получить **многопоточную корректно работающую версию решателя, дающую ускорение при сохранении корректности результата.**

Этап 4 задания 1: проверка корректности программы и выдача измерений (Report)

На этом этапе надо просто проверить, что невязка системы после работы решателя удовлетворяет заданной точности, выдать значение невязки, и распечатать табличку таймирования, сколько времени в секундах затрачено на:

1. этап генерации;
2. этап заполнения;
3. этап решения СЛАУ;
4. каждую из трех кернелов – алгебраических операций решателя СЛАУ

Чтобы эти данные получить, по вашей программе надо расставить соответствующие замеры времени.

Отчет

По результатам нужно будет подготовить отчет, в котором привести данные по исследованию производительности решателя. 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
Описание параметров компиляции под конкретную систему
3.2 Результаты измерений производительности
3.2.1 Последовательная производительность
Для всех этапов, включая генерацию и заполнение (то есть этапы инициализации), продемонстрировать линейный рост стоимости, линейный рост потребления памяти.
Solve-часть надо исследовать подробнее. 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= 10000, 100000, 1000000, 10000000.
Для повышения точности измерений, замеры времени лучше производить, выполняя операции многократно в цикле, чтобы осреднить время измерений. Суммарное время измерений чтобы получалось хотя бы порядка десятых долей секунды.
Оценить выигрыш от примененной оптимизации (по желанию).
3.2.2. Параллельное ускорение
Для этапов инициализации продемонстрировать наличие параллельного ускорения. Достигаемую производительность для этапов инициализации измерять не надо, считать ТВР не надо.
Для solve-части измерить OpenMP ускорение для различных N для каждой из 3-х базовых операций и для всего алгоритма солвера. Измерить достигаемую производительность. Можно сделать таблички или графики, показывающие GFLOPS от T, где T - число нитей. Можно представить данные по ускорению и производительности единой таблицей, в которой для разных N и T привести ускорение и достигаемую производительность.

4 Анализ полученных результатов
Оценить ТВР и получаемый процент от достижимой производительности для каждой из трех базовых операций, какой процент от пиковой производительности устройства составляет максимальная достигаемая в тесте производительность. Для этапов инициализации ничего оценивать и считать не требуется.

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.