Практическое задание №2
Распределенная реализация солвера BiCGSTAB для СЛАУ с разреженной матрицей, заданной в формате CSR

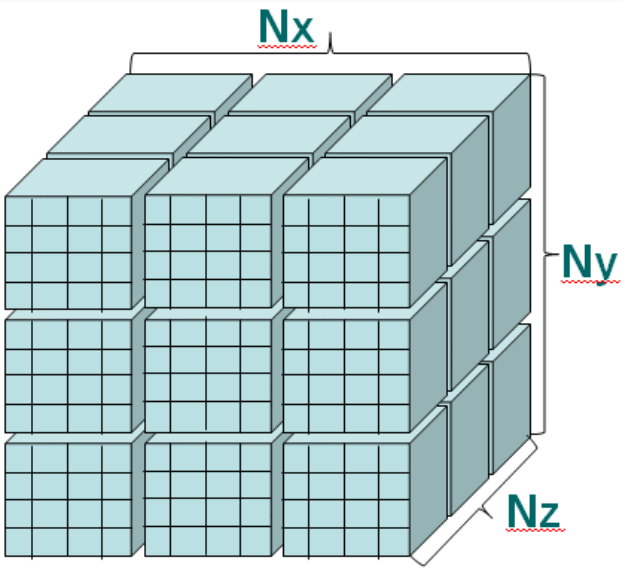
В этом задании необходимо расширить реализацию, сделанную в задании 1, на параллельные вычисления в рамках параллельной модели с распределенной памятью. Распределенная реализация подразумевает, что расчетная область задачи разделяется на части – подобласти. Эти подобласти распределяются между параллельными процессами. Каждый процесс работает **со своей частью расчетной области** и обменивается с соседними процессами только информацией по интерфейсным ячейкам, граничащим с ячейками других подобластей. Для обмена сообщениями используется MPI.

Этот тип распараллеливания рассматривается в первой части лекции 3:
http://caa.imamod.ru/files/vmk18/vmklection3.pdf

Генерация распределенной матрицы - этап препроцессора

Разделим нашу модельную расчетную область, представленную декартовой решеткой из N = Nx*Ny*Nz ячеек, на P = Px*Py*Pz подобластей (см. слайд 23) путем декомпозиции по каждому из трех направлений.

Рис. 1. Пример декомпозиции для Px=3, Py=3, Pz=3



Каждый процесс работает только **со своей частью** расчетной области и с ее гало (т.е. приграничными ячейками из соседних подобластей).

Рассмотрим для простоты на примере двухмерной решетки.

Рис. 2. Расчетная область:

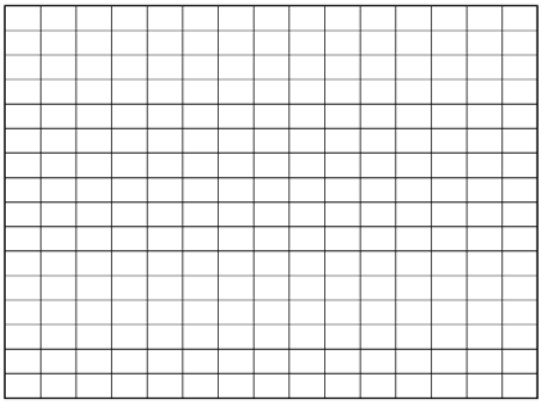


Рис. 3. Поделим ее, например, на 9 частей, части нумеруем по тому же принципу, что и ячейки (показан номер процесса и курсивом его позиция в решетке процессов px,py):

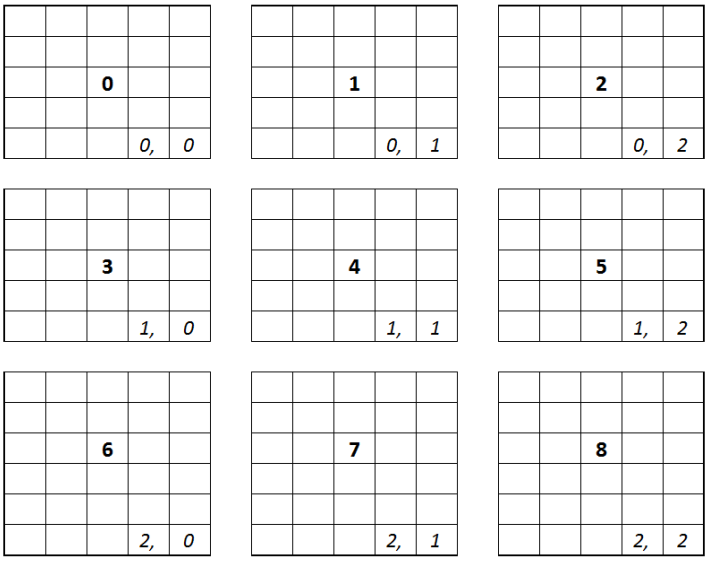


Рис. 4. Рассмотрим один из процессов, например, который по середине - №4. Ячейки его подобласти отмечены серым. Другими цветами отмечены гало ячейки этой подобласти, принадлежащие чужим подобластям.

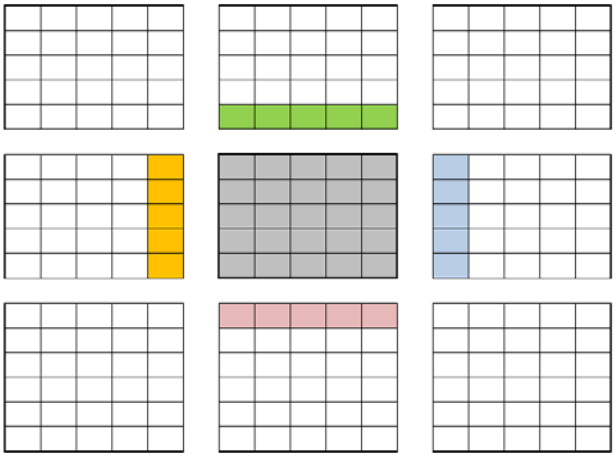
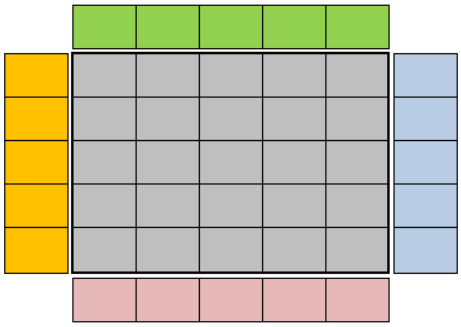


Рис. 5. Так выглядит часть расчетной области, с которой имеет дело 4-й процесс:



О существовании других ячеек этот процесс не знает.

Расширенная подобласть процесса = свои ячейки + гало ячейки.

Она состоит из **n = ni + nh** ячеек, из которых ni своих ячеек (серые), и nh гало ячеек (цветные).

Для начала по заданным Nx, Ny, Nz и Px, Py, Pz определим принадлежность ячеек. Для простоты можно сделать следующим образом. Каждый процесс создаст себе интовый массив Part размера N = Nx*Ny*Nz в который для каждого элемента запишет номер его процесса-владельца.

Далее нужно сгенерировать матрицу в распределенном виде. То есть так, чтобы у процессов были только строки, соответствующие своим ячейкам. Для этого нужно внести небольшие изменения в процедуру генерации матрицы.

Нужно найти границы циклов для данного процесса, зная его позицию в решетке процессов. Вместо исходного варианта

```
for(int k=0; k<Nz; k++){
     for(int j=0; j<Ny; j++){
               for(int i=0; i<Nx; i++){
```

будет что-то типа

```
for(int k=kb; k<ke; k++){
     for(int j=jb; j<je; j++){
               for(int i=ib; i<ie; i++){
```

Диапазоны kb, ke; jb, je; ib, ie следуют из позиции данного процесса в решетке процессов, т.е. его порядковых номеров по трем осям (что такое решетка процессов см. рис. 3). Только на картинке 2 индекса, поскольку нарисована 2D решетка, а в нашем случае их 3.

Заполняя строки матрицы, также заполним и список строк, т.е. список глобальных номеров строк, которым соответствуют наши строки матрицы. Это будет отображением из локального номера строки в глобальный. Пусть этот массив номеров будет Rows. Номер строки в локальной нумерации будет соответствовать ее порядковому номеру, т.е. будет что-то типа

```
int I = 0; // текущий номер строки
for(int k=kb; k<ke; k++){
      for(int j=jb; j<je; j++){
            for(int i=ib; i<ie; i++, I++){
                Rows[I] = i + j*Nx + k*Nx*Ny;
                ...
```

Номера столбцов в строках заполняются в глобальной нумерации, т.е. напрямую вычисляя глобальный номер по индексам i,j,k, как в исходной версии.

На выходе препорцессора имеем

1) Rows - список глобальных номеров строк матрицы, принадлежащих данному процессу

2) IA, JA, A - сами эти строки матрицы, принадлежащие данному процессу (т.е. локальный кусок матрицы, соответствующий маленькому кубику), в формате CSR. номера столбцов в глобальной нумерации.

3) Массив Part, задающий принадлежность ячеек процессам.

Распределенное решение СЛАУ - вычислительное ядро

На вход получены данные 1), 2), 3), описанные выше. Вычислительное ядро ничего не знает о топологии расчетной области, о том, что это была решетка и т.д. (теперь данные представлены в обобщенном виде).

Нужно сделать проход по всем полученным на вход строкам, чтобы составить списки ячеек на отправку и прием и преобразовать номера столбцов в локальную нумерацию.

Для этого создадим массив Glob2Loc размера N, заполним значениями -1.

Затем, используя массив Rows (отображение из локальной нумерации в глобальную для своих ячеек), заполним в Glob2Loc локальные номера своих ячеек.

Получим отображение из глобальных номеров в свою локальную нумерацию, значение -1 в Glob2Loc обозначает, что эта ячейка не принадлежит подобласти.

Определим гало-ячейки и создадим списки ячеек на отправку и получение. Делаем проход по строкам матрицы и смотрим номера столбцов в JA. Если Glob2Loc от этого номера столбца = -1, то это гало ячейка, т.е. чужая ячейка, нужная нашей строке. Добавим этот номер в список входящего сообщения от ее владельца.

Если в строке попался хотя бы один чужой столбец, внесем нашу ячейку, которой соответствует строка, в список отправки каждому из владельцев чужих ячеек этой строки.

Пройдя по всем своим строкам, мы получим для каждого соседнего процесса, т.е. процесса, ячейки которого попались в строках, список номеров ячеек на отправку этому процессу и список номеров на получение от этого процесса. На отправку - наши ячейки, на прием - гало.

В списках сообщений не должно быть повторяющихся номеров ячеек.

В каждом сообщении упорядочим список номеров по возрастанию глобального номера (в сообщениях на отправку мы его узнаем через Rows, на получение он и так глобальный).

Для проверки корректности можем распечатать эти списки, выдавая глобальные номера ячеек, и проверить, что список на отправку от данного процесса соседу в точности совпадает со списком на прием у этого соседа от данного процесса. И так можно проверить по всем процессам.

Теперь, объединив списки сообщений на прием в порядке возрастания номеров процессов-владельцев, получим список всех гало ячеек в нужном нам порядке - упорядоченно по возрастанию номера соседа-владельца и для каждого соседа упорядоченно по возрастанию глобального номера.

Сделаем массив Halo и сольем туда эти номера в соответствующем порядке.

Пройдемся по Halo и запишем в Glob2Loc локальные номера для гало-ячеек, которые будут: позиция в Halo + число собственных ячеек ni (Glob2Loc[Halo[i]] = i + ni)

Теперь у нас есть отображение из глобальных номеров в локальные для своих ячеек и для гало. Пройдемся по строкам матрицы и переведем номера столбцов в JA в локальную нумерацию.

То же самое сделаем для сообщений на прием.

В итоге во внутренней нумерации процесса ячейки будут упорядочены так, что сначала идут свои ячейки (0, ..., ni-1), затем все гало ячейки (ni, ..., n-1),

упорядоченные по владельцам (см. слайд 19), а внутри набора каждого владельца ячейки упорядочены по возрастанию их глобального номера.

Мы преобразовали матрицу в локальную нумерацию и получили схему обменов, т.е. списки ячеек для отправки каждому соседу и списки для получения от каждого соседа.

Выделять вектора мы теперь будем по размеру n = ni + nh, а обрабатывать будем только первые ni из них.

Теперь надо реализовать распределенные операции. Операции SpMV, dot, axpby по сути не меняются, требуется только добавить обмены. Операция dot требует allreduce для нахождения глобальной суммы, для SpMV требуется обмен для обновления значений в гало. Только теперь операции выполняются уже над локальными данными, размер N теперь = ni (число своих ячейки).

Для обновления гало надо (слайд 25), запаковать сообщения, инициализировать асинхронные отправку и прием, дождаться завершения всех обменов и распаковать входящие сообщения.

В результате должна получиться корректно работающая программа, дающая идентичные результаты с заданием 1, но работающая в распределенном режиме. На вход относительно задания 1 добавляются 3 параметра – Px, Py, Pz.

Отчет

По результатам нужно будет подготовить отчет, в котором привести данные по исследованию производительности солвера в распределенном режиме.

Для каждой из базовых операций и для всего алгоритма:

Выполнить сравнение MPI и OpenMP распараллеливания на многоядерном процессоре, убедиться, что ускорения сопоставимы.

Исследовать масштабирование и параллельную эффективность на кластере (слайды 26, 27).

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 Краткое 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.

3 Исследование производительности
3.1 Характеристики вычислительной системы:
описание одной или нескольких систем, на которых выполнено исследование. Использование кластера в этом задании обязательно. Просьба указывать здесь или в следующих пунктах, как программа компилировалась (каким компилятором, с какими параметрами).
3.2 Результаты измерений производительности
3.2.1 Сравнение MPI с OpenMP на одном многоядерном процессоре
Для каждой из трех базовых операций и для всего алгоритма солвера сравнить ускорения на разном числе ядер, полученные в MPI и OpenMP режиме, оценить параллельную эффективность. Достаточно одного размера системы, $N \geq 10^6$. Данные представить в виде таблицы.

3.2.2. Параллельное ускорение
Измерить MPI ускорение для различных N порядка 10^5, 10^6, 10^7,… для каждой из 3-х базовых операций и для всего алгоритма солвера: при фиксированном числе N варьируется число процессов и измеряется параллельное ускорение. Построить графики ускорения.
Важно понимать принципиальную разницу между ускорением от 1 ядра и от 1 узла кластера. Графики для кластера лучше делать от 1 узла.

3.2.3. Масштабирование

Измерить масштабирование для различных фиксированных N/P порядка 10^4, 10^5, 10^6. Здесь N/P – количество неизвестных на процесс. В этом тесте число N варьируется пропорционально числу процессов, P. Данные представить в виде таблицы и графика.

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

Требования к программе:

1 Программа должна использовать MPI для распараллеливания с распределенной памятью, OpenMP или posix threads для многопоточного распараллеливания (которое уже имеется из 1-го задания)

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

3 Распараллеливание должно быть корректно