# Параллельная реализация итерационного метода решения СЛАУ на основе OpenMP

# Важные предварительные замечания

- Лаба № 2 выполняется в двух вариантах: распараллеливание всего кода сразу, и по отдельным циклам
- Основное отличие от лабы № 1
  - Распараллеливание на общей памяти
  - В MPI процессы, в OpenMP – **потоки**
    **(не путать!!! в №5 будет и то, и другое)**
- В связи с дистанционным обучение требования меняются: теперь, пожалуйста, высылайте и тексты программ тоже

# Реализация метода Якоби (1 шаг)

```
for i = 1 : n
  x_new(i) = b(i);
  for j = 1 : n
    if ( j ~= i )
      x_new(i) = x_new(i) - a(i,j) * x(j);
    end
  end
  x_new(i) = x_new(i) /a(i,i);
end
```

# Вычисление невязки

```
eps = abs(x_new - x)
```

# Необходимо проверить, как и в лабе № 1

- **Сходимость и правильность решения** для произвольной размерности матрицы
- И для любого вида матрицы с выраженным диагональным преобладанием (диагональный элемент любой строки по модулю больше суммы недиагональных)
- Для проверки задаем решение в виде вектора из 1 и вычисляем для этого решения правую часть
- Матрицу задаем в виде диагональной малой размерности (3x3, 5x5) и постепенно увеличиваем плотность заполнения. Размерность увеличиваем до 1000 или 10000, пока время счета не достигнет 5-10 минут.
-

# Измерение времени
## Два вызова функции gettimeofday

- #include <sys/time.h>
  struct timeval tv1,tv2;

  **gettimeofday(&tv1,NULL);**
  DoSomeThing();

  **gettimeofday(&tv2,NULL);**
- double dt_sec  = (tv2.tv_sec – tv1.tv_sec);
  double dt_usec = (tv2.tv_usec – tv1.tv_usec);
  dt = dt_sec + 1e-6*dt_usec;
  printf(“time diff %e \n”,dt);

**Параллельная реализация алгоритма начинается с правильно работающей *последовательной* программы**

# Перед тем, как переходить к распараллеливанию, зафиксируйте (для малой размерности, e.g. 3x3)

- Ход сходимости (перенаправлением вывода в файл):
  <Номер итерации> <значение невязки>
- Решение, которое получается в итоге (не желаемый вектор из всех единиц, а именно то, что получается)
- Параллельный алгоритм
  **должен выдать то же самое**

# Распараллеливание

- Распределение итераций циклов между потоками с помощью #**pragma omp parallel for**
- Невязка вычисляется всеми потоками по тем итерациям, которые они выполняют, поэтому нужно вычислить единое для всех значение с помощью
  - **omp reduction**(+,r) – сложение переменной по всем потокам
  - omp reduction(max,r) – вычисление максимального по всем потокам значения переменной
- Существует возможность оптимизации распределения итераций цикла между потоками с помощью директивы **omp schedule**

# OpenMP schedule static

static – блочно-циклическое распределение итераций цикла; размер блока – chunk. Первый блок из chunk итераций выполняет нулевая нить, второй блок — следующая и т.д. до последней нити, затем распределение снова начинается с нулевой нити. Если значение chunk не указано, то всё множество итераций делится на непрерывные куски примерно одинакового размера (конкретный способ зависит от реали зации), и полученные порции итераций распределяются между нитями:

```
#pragma omp for schedule (static, 2)
```

# OpenMP schedule dynamic

- динамическое распределение итераций с фиксированным
- размером блока: сначала каждая нить получает chunk итераций (по
- умолчанию chunk=1), та нить, которая заканчивает выполнение своей
- порции итераций, получает первую свободную порцию из chunk ите-
- раций. Освободившиеся нити получают новые порции итераций до тех
- пор, пока все порции не будут исчерпаны. Последняя порция может
- содержать меньше итераций, чем все остальные.
-
    - omp for schedule (dynamic, 10)

# OpenMP schedule guided

динамическое распределение итераций, при котором размер порции уменьшается с некоторого начального значения до величины chunk (по умолчанию chunk=1) пропорционально количеству ещё не распределённых итераций, делённому на количество нитей, выполняющих цикл. Размер первоначально выделяемого блока зависит от реализации. В ряде случаев такое распределение позволяет аккуратнее разделить работу и сбалансировать загрузку нитей.

Количество итераций в последней порции может оказаться меньше значения chunk.

- .

omp for schedule (guieded, 10)

# Задание по omp schedule

- Выберите один из циклов for в вашей программе (желательно самый затратный)
- Проведите несколько запусков программы с разными значениями chunk
  - По 3 значения chunk для каждого из вариантов: static, dynamic, guided
  - Эти три значения chunk должны быть
    - Заметно (напр. в 10 раз )меньше числа итераций в цикле
    - Одинаковыми для всех параметров директивы schedule ( для static, и для dynamic, и для guided)
  - Измеряйте только время работы этого цикла (влияние на время счета программы в целом может быть незначительным)
  - По результатам нарисуйте диаграмму в Excel (или OpenOffice Calc или ...)

# Два варианта распараллеливания в лабе № 2

- распараллеливание всего кода сразу,
- по отдельным циклам

# Вариант 1

- **# pragma omp parallel private ( i )**
- {
- /*
- Set up the right hand side.
- */
- # **pragma omp for**
- for ( i = 0; i < n; i++ )
- {
- b[i] = 0.0;
- }
-
- b[n-1] = ( double ) ( n + 1 );

…..

# Вариант 2

```
{
/*
  Set up the right hand side.
*/
# pragma omp parallel for
    for ( i = 0; i < n; i++ )
    {
      b[i] = 0.0;
    }

    b[n-1] = ( double ) ( n + 1 );


.....
```

# Количество OpenMP-потоков

- set_omp_num_threads(n)
- #pragma opm parallel for num_threads(n)
- Через переменную окружения