Введение в параллельные вычисления

Лекция 5. OpenMP (часть 3)

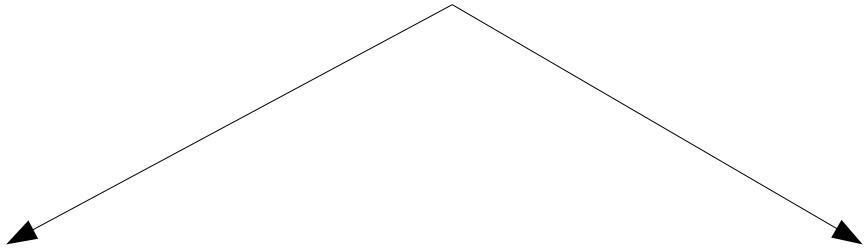
КС-40, КС-44
РХТУ

Преподаватель
Митричев Иван Игоревич, к.т.н.

2017

Средства синхронизации в OpenMP

Синхронизация (от греч. συνχρόνος — одновременный) — процесс приведения к одному значению одного или нескольких параметров разных объектов.



Синхронизация процессов (потоков)

Синхронизация данных

- barrier
- ordered
- critical

Директива critical

```
#pragma omp parallel
{
	//.. параллельное выполнение
	#pragma omp critical
	{
		// критическая секция
	}
}
```

l04_29.cpp

Критическая секция – область программы, которая **выполняется одновременно только одним потоком**. Все потоки проходят критическую секцию по очереди.

Не путать с master/single/ordered!

Атомарная операция

```
srand(time(NULL));
int n;
// just explicitly name it shared (without it - the same sense)
#pragma omp parallel num_threads(3) shared(n)
{
// one-line critical to update shared variable
                        #pragma omp atomic
                                n++;
}

cout<<"n= "<<n<<endl;
```

l04_30.cpp

Атомарные операции служат для обновления общих переменных

Атомарная операция – однострочная критическая секция

Мьютексы (блокировки, замки)

void omp_init_lock(omp_lock_t *lock) ← инициализация простого замка

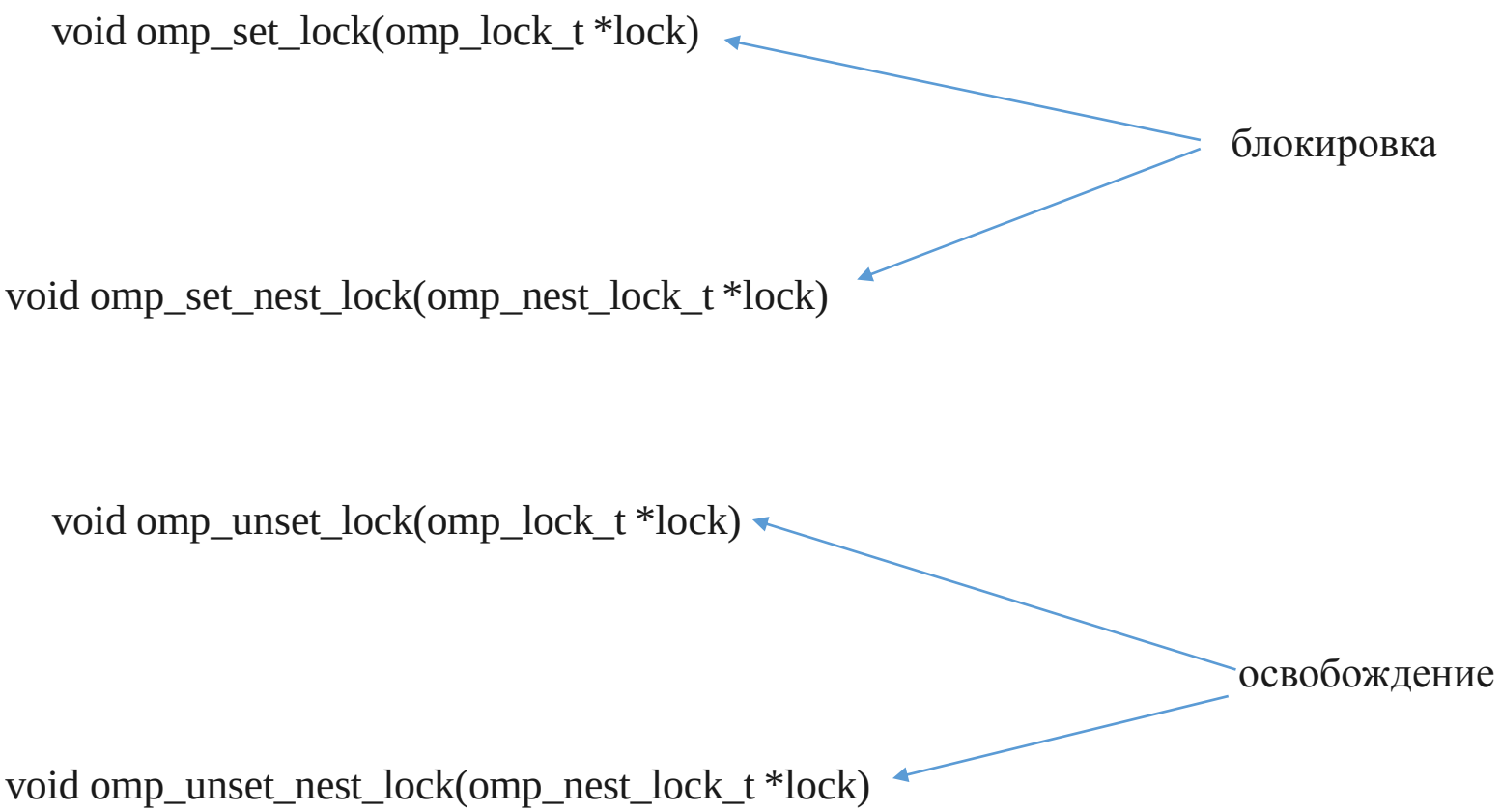
void omp_init_nest_lock(omp_nest_lock_t *lock) ← инициализация множественного замка (замок с коэффициентом захваченности)

void omp_destroy_lock(omp_lock_t *lock)

void omp_destroy_nest_lock(omp_nest_lock_t *lock)

деструкторы замков

Мьютексы (блокировки, замки)



Неблокирующий захват мьютекса

```
int omp_test_lock(omp_lock_t *lock)
```

```
int omp_test_nest_lock(omp_nest_lock_t *lock)
```

Мьютекс (пример)

```
omp_lock_t lock;
int main()
{
        int n;
        omp_init_lock(&lock);
        double s=omp_get_wtime();
        #pragma omp parallel private (n) num_threads(3)
        {
                n = omp_get_thread_num();
                omp_set_lock(&lock);
                sleep(1);
                omp_unset_lock(&lock);
        }
        omp_destroy_lock(&lock);

// затраченное время – 3 секунды – из-за блокировки мьютекса (mutex)
каждым из потоков на 1 секунду
cout<<omp_get_wtime()-s<<endl;
```

l04_31.cpp
l04_32.cpp

Какое время будет выведено при использовании omp_test_lock?

Синхронизация состояния памяти

#pragma omp flush (список переменных)

Синхронизировать состояние памяти (явно записать значение переменной в общую память)

Неявно flush выполняется при:
- barrier;
- parallel;
- critical;
- ordered;
- omp_set_lock();
- omp_unset_lock();
- omp_test_lock();
- atomic

Директива for без collapse

```
int A[20],B[20],C[20],i,j,n;
for (i=0;i<20;i++)
{
        A[i]=i; B[i]=2*i;
}
#pragma omp parallel shared (A,B,C) private (i,j,n) num_threads(10)
{
        n = omp_get_thread_num();
// on my PC 3 - chunk size by default
        #pragma omp for schedule (static,1) //collapse(2)
        for (j=0;j<2;j++)
        for (i=0;i<10;i++)
        {
                C[j*10+i]=A[j*10+i]+B[j*10+i];
                #pragma omp critical
                cout<<i<<" (C) by "<<n<<endl;
        }
}
```

2 потока выполняют по 1 итерации внешнего цикла.
Внутренней цикл выполняет каждый из двух потоков для своего значения j

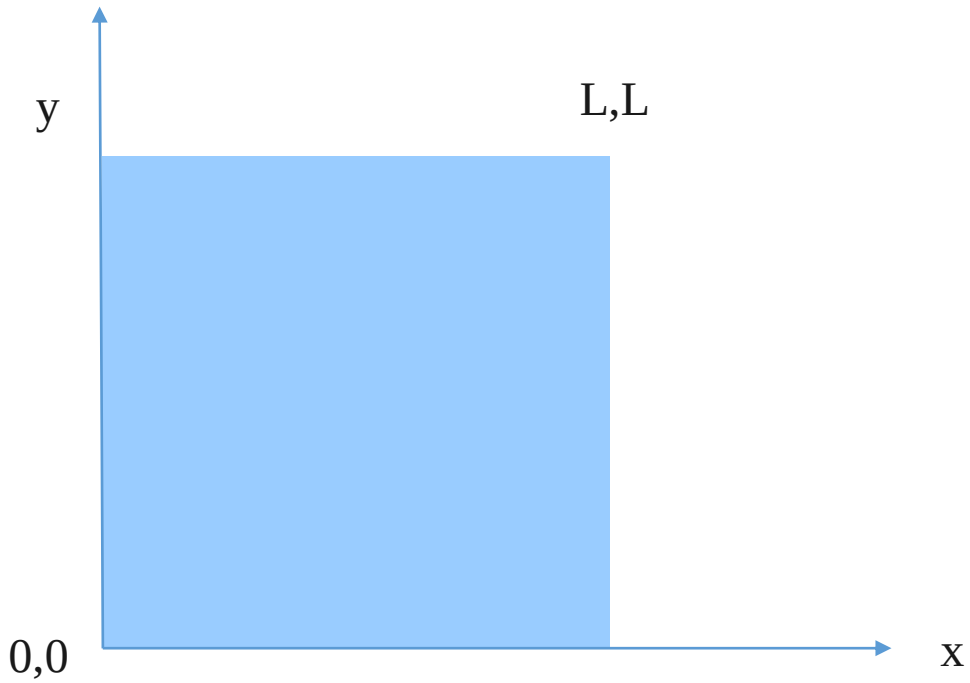
Директива for и collapse

```
int A[20],B[20],C[20],i,j,n;
for (i=0;i<20;i++)
{
        A[i]=i; B[i]=2*i;
}
#pragma omp parallel shared (A,B,C) private (i,j,n) num_threads(10)
{
        n = omp_get_thread_num();
// on my PC 3 - chunk size by default
        #pragma omp for schedule (static,1) collapse(2)
        for (j=0;j<2;j++)
        for (i=0;i<10;i++)
        {
                C[j*10+i]=A[j*10+i]+B[j*10+i];
                #pragma omp critical
                cout<<i<<" (C) by "<<n<<endl;
        }
}
```

10 потоков выполняют по 1 итерации из 20 итераций.
Итерациях двух циклов распараллеливаются независимо по потокам

Краевая задача для уравнения Лапласа

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$



Разностная схема для уравнения Лапласа

u(x=0, y) = u(x, y=0) = u(x=L, y) = u(x, y=L) = 1

Начальное приближение u(0<x<L,0<y<L)=0

Решается методом установления, **методом Якоби**

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta h_y^2} = 0$$

Пусть $h_x = h_y$

Итерационная формула (до стабилизации значения $u_{i,j}$)

$$u_{i,j} = \frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})$$

Последовательная реализация

```
#include <algorithm>
#include <math.h>
#include <omp.h>
#include <iostream>
using namespace std;
#define eps 0.001

int main()
{
double start = omp_get_wtime();

int N = 200;
int i,j;
int iter=0;
double ** grid = new double * [N+1];
double ** newgrid = new double * [N+1];

for (i=0;i<N+1;i++)
{
        grid[i] = new double[N+1];
        newgrid[i] = new double [N+1];
}

for (i=0;i<N+1;i++)
{
        grid[i][0]=1;
        grid[0][i]=1;
        grid[N][i]=1;
        grid[i][N]=1;
        newgrid[i][0]=1;
        newgrid[0][i]=1;
        newgrid[N][i]=1;
        newgrid[i][N]=1;
}

for (i=1;i<N;i++)
        for (j=1;j<N;j++)
                grid[i][j]=0;

double maxdiff=0;
do{
for (i=1;i<N;i++)
for (j=1;j<N;j++)
{
        newgrid[i][j]= 0.25 * (grid[i-
1][j]+grid[i+1][j]+grid[i][j-1]+grid[i][j+1]);
}
maxdiff=0;
for (i=1;i<N;i++)
for (j=1;j<N;j++)
{
        maxdiff = max(maxdiff,
fabs(newgrid[i][j]-grid[i][j]));
}
for (i=1;i<N;i++)
for (j=1;j<N;j++)
{
        grid[i][j]=newgrid[i][j];
}
iter++;
}
while (maxdiff > eps);

cout<<"Iter:"<<iter<<endl;

/*for (i=0;i<N+1;i++)
{
        for (j=0;j<N+1;j++)
    cout<<grid[i][j]<<" ";
        cout<<endl;
}*/

for (i=0;i<N+1;i++)
{
        delete [] grid[i];
        delete [] newgrid[i];
}
delete grid;
delete newgrid;
double end = omp_get_wtime();
std::cout<<"Total time = "<<end-start<<endl;
return 0;
}
```

l05_01.cpp

Параллельная реализация

```
#include <algorithm>
#include <math.h>
#include <omp.h>
#include <iostream>
using namespace std;
#define eps 0.001

int main(){
double start = omp_get_wtime();
int N = 200;
int i,j;
int iter=0;
double ** grid = new double * [N+1];
double ** newgrid = new double * [N+1];

for (i=0;i<N+1;i++)
{
        grid[i] = new double[N+1];
        newgrid[i] = new double [N+1];
}

#pragma omp parallel shared(grid,newgrid)
private (i,j)
{
  #pragma omp single
cout<<"Num threads:
"<<omp_get_num_threads()<<endl;
        #pragma omp for
        for (i=0;i<N+1;i++)
        {
                grid[i][0]=1;
                grid[0][i]=1;
                grid[N][i]=1;
                grid[i][N]=1;
                newgrid[i][0]=1;
                newgrid[0][i]=1;
                newgrid[N][i]=1;
                newgrid[i][N]=1;
        }
        #pragma omp for //collapse(2)
        for (i=1;i<N;i++)
                for (j=1;j<N;j++)
                grid[i][j]=0;
}

double maxdiff=0;
do{
#pragma omp parallel
shared(grid,newgrid,maxdiff) private (i,j)
{
#pragma omp for //collapse(2)
for (i=1;i<N;i++)
for (j=1;j<N;j++)
{
        newgrid[i][j]= 0.25 * (grid[i-
1][j]+grid[i+1][j]+grid[i][j-1]+grid[i][j+1]);
}

#pragma omp master
{
maxdiff=0;
for (i=1;i<N;i++)
for (j=1;j<N;j++)
        maxdiff = max(maxdiff,
fabs(newgrid[i][j]-grid[i][j]));
}

#pragma omp barrier

#pragma omp parallel for //collapse(2)
for (i=1;i<N;i++)
for (j=1;j<N;j++)
{
        grid[i][j]=newgrid[i][j];
}
} //here parallel section ends
iter++;
}
while (maxdiff > eps);

cout<<"Iter:"<<iter<<endl;

for (i=0;i<N+1;i++)
{
        delete [] grid[i];
        delete [] newgrid[i];
}
delete grid;
delete newgrid;
double end = omp_get_wtime();
std::cout<<"Total time = "<<end-start<<endl;
return 0;
}
```

l05_02.cpp

Вывод программы
Num threads: 4
Iter:360
Total time = 0.233292

Тестирование (на http://coliru.stacked-crooked.com/)

Размер области	200	500	1000
Последовательная	0,325	1,962	8,227
OpenMP	0,218	1,449	6,741*

* - 6,541 – при использовании collapse(2)

l05_03.cpp

Оптимизация (избавление от присвоений и раскрутка цикла)

```
for (i=1;i<N;i++)
        for (j=1;j<N;j++)
        newgrid[i][j]= 0.25 * (grid[i1][j]+grid[i+1][j]+grid[i][j1]+grid[i][j+1]);
for (i=1;i<N;i++)
        for (j=1;j<N;j++)
        grid[i][j]= 0.25 * (newgrid[i-1][j]+newgrid[i+1][j]+newgrid[i][j-1]+newgrid[i][j+1]);
maxdiff=0;
for (i=1;i<N;i++)
        for (j=1;j<N;j++)
        maxdiff = max(maxdiff, fabs(newgrid[i][j]grid[i][j]));
```

l05_04.cpp

```
for (i=1;i<N;i++)
        for (j=1;j<N;j++)
        grid[i][j]=newgrid[i][j];
```

Размер области	200	1000
До оптимизации (OpenMP + collapse)	0,208	6,741
После оптимизации (OpenMP + collapse)	0,123	**3,656**
До оптимизации кода, но с оптимизацией компилятора gcc -O2	0,077	2,099
После оптимизации кода, с оптимизацией компилятора gcc –O2	0,043	**0,931**
После оптимизации кода, с оптимизацией компилятора, использование **parallel for**	0,047	0,976