**Упражнение 6**

**Директиви *CRITICAL, MASTER,*  *BARRIER* , *ATOMIC* и *ORDERED***

***Директива MASTER***

Указва регион, който трябва да се изпълни само от „мастер” нишката. Всички други нишки от групата пропускат тази сеция от кода

С тази директива по подразбиране не се прилагат синхронизиращи бариери

#pragma omp master newline

structured\_block

***Директива CRITICAL***

Указва, че регионът от кода трябва да бъде изпълняван само от една нишка в даден момент от време

#pragma omp critical [ name ] newline

structured\_block

Ако дадена нишка извършва своето изпълнение вътре в критичната секция и друга нишка достигне тази секция и се опита да я изпълни, то тя ще бъде блокирана докато първата нишка не напусне критичния регион

Опционалното име позволява едновременно съществуването на различни критични региони:

* Имената са глобални идентификатори. Различните критични региони с еднакви имена се третират като един и същ регион
* Всички критини секции, които са неименовани се третират като една секция

***Директива BARRIER***

Синхронизира всички нишки от групата

Когато директивата BARRIER е достигната – нишката трябва да изчака в тази точка докато всички нишки от групата достигнат тази бариера. След това всички нишки продължават паралелното изпълнение на код, който следва бариерата.

#pragma omp barrier newline

* Всички нишки (или никоя от тях) трябва да изпълнят BARRIER
* Последователността на изпълнението на work-sharing региони и бариери трбва да бъде едн и съща за всяка една от нишките в групата

***Директива TASKWAIT***

Отнася се за OpenMP 3.0

Указва, че трябва да се изчака приключванена генерирани задачи, поради започването на текуща задача.

#pragma omp taskwait newline

Възможни се ограничанията при използване в С. Виж спецфикациите на OpenMP 3.0

***Директива ATOMIC***

Директивата указва, че конкретен регион от паметта трябва да бъде обновяван само от една нишка в даден момент от време (атомично), отколкото множество от нишки да се опитва да записва в него. В същност директивата осигурава мини-критична секция.

#pragma omp atomic newline

statement\_expression

* Директивата се прилага единствено върху следващия я израз
* Следва специализиран синтаксис.

***Директива ORDERED***

Директивата указва, че итерациите в даден цикъл трябва да бъдат изпълнени в реда, все едно че се изпълняват последователно

Нишките ще трябва да изчакват преди да започнат изпълнениет на тяхното „парче” от работата, ако предишните итерации не са изпълнени

Използва се при for цикли с клаузата ORDERED

Директивата ORDERED предоставя начин за „фино настройване” на кода, там където поседователното зпълнение трябва да се прилож в цикъла. В противен случай използването му не е необходимо.

#pragma omp for ordered [clauses...]

(loop region)

#pragma omp ordered newline

structured\_block

(endo of loop region)

* Може да се използва само с директиви for и рarallel for (C/C++)
* Само една нишка изпълнява ORDERED секция в даден момент от време
* Итерацията от цикъла не тряба да изпълнява една и съща ORDERED директива повече от веднъж, както не трябва и да изпълнява повече от една ORDERED директива.
* **Цикълът, който съдържа директивата ORDERED, трябва да е цикъл със клаузата ORDERED**

**Задачa: Напишете примерна/и програма/и, които да използват две директиви по ваш избор.**

//barrier

#include <stdio.h>

#include <omp.h>

void main() {

int x;

x = 2;

#pragma omp parallel num\_threads(2) shared(x)

{

if (omp\_get\_thread\_num() == 0) {

x = 5;

}

else {

/\* Print 1: the following read of x has a race \*/

printf("1: Thread# %d: x = %d\n", omp\_get\_thread\_num(), x);

}

#pragma omp barrier

if (omp\_get\_thread\_num() == 0) {

/\* Print 2 \*/

printf("2: Thread# %d: x = %d\n", omp\_get\_thread\_num(), x);

}

else {

/\* Print 3 \*/

printf("3: Thread# %d: x = %d\n", omp\_get\_thread\_num(), x);

}

}

}

//atomic

#include <stdio.h>

float average(float a, float b, float c) {

float r = (a + b + c) / 3;

return r;

}

float work1(int i)

{

return 1.0 \* i;

}

float work2(int i)

{

return 2.0 \* i;

}

void atomic\_example(float \*x, float \*y, int \*index, int n)

{

int i;

#pragma omp parallel for shared(x, y, index, n)

for (i = 0; i<n; i++) {

#pragma omp atomic

x[index[i]] += work1(i);

y[i] += work2(i);

}

}

void main()

{

float x[1000];

float y[10000];

int index[10000];

int i;

for (i = 0; i < 10000; i++) {

index[i] = i % 1000;

y[i] = 0.0;

}

for (i = 0; i < 1000; i++)

x[i] = 0.0;

atomic\_example(x, y, index, 10000);

}