1.

What is the problem with the following code?

struct T {

int a;

size\_t b;

};

T array[arraySize];

size\_t i;

#pragma omp parallel sections num\_threads(2)

{

#pragma omp section

{

for (i = 0; i != arraySize; ++i)

array[i].a = 1;

}

#pragma omp section

{

for (i = 0; i != arraySize; ++i)

array[i].b = 2;

}

}

The i variable is shared, because it is declared locally, outside of the parallel section. Thus, both sections will be modifying the same i variable and the array objects will not be modified as expected.

2.

Use OpenMP to parallelize the following code. What would happen if this was a one-dimensional array, in a single for loop, and the same parallelization was used?

int i, j;

#pragma omp parallel for private(j)

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

for(j=1; j<n; j++) {

array[i][j]+= array[i][j-1];

}

1D case:

#pragma omp parallel for

for(j=1; j<n; j++)

array[j]+= array[j-1];

This would not be a valid solution because the threads need to be created for each row because each row is independent of one another – this implementation does not make specify where the threads will start at.

3.

Optimize the following code, using OpenMP.

void hello(long \*old, long \*new, int n) {

int i;

double sumWeights=0, sum=0;

sum = n\*old[0];

#pragma omp parallel for reduction(+:sumWeights)

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

new[i] = old[i] \* exp(100.0f/old[i]);

sumWeights += new[i];

}

sumWeights /= sum;

#pragma omp parallel for

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

new[i] = new[i]/sumWeights;

}

4.

What are the differences between dynamic and static linking? What are some advantages and disadvantages?

In static linking, all dependent libraries of a program will be combined together into one large executable at compile time. Symbols are resolved immediately. In dynamic linking, the executable will not contain the whole library, but the relevant parts of the library will be loaded at run time through pointers to the necessary symbols. Static linking: faster, but larger executable size. Less issues. Dynamic linking: saves memory.

5.

What type of exception would each of the following lead to? Are they synchronous or asynchronous exceptions? What is their return behavior?

1. Dividing by 0

Synchronous exception (floating point exception), unintentional fault, aborts

1. Tired of waiting for your “optimized” code for the OpenMP lab, you terminate your process by pressing Ctrl-C at the keyboard

Asynchronous exception, interrupt, returns to next instruction

1. The MMU fetches a PTE from the page table in memory, but the valid bit is zero

Synchronous exception (page fault), unintentional fault, must fetch from disk, then return to execution

1. You create a file using the open() system call

Synchronous exception (trap), intentional, will return to execution after finished

6. (Textbook 9.3)

Given a 32 bit virtual address space and a 24-bit physical address, determine the number of bits in the VPN, VPO, PPN, and PPO for the following page sizes P:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | VPN bits | VPO bits | PPN bits | PPO bits |
| 1 KB | 8 |  |  |  |
| 2 KB |  |  |  |  |
| 4 KB |  |  |  |  |
| 8 KB |  |  |  |  |