HPC Seminar

Parallelization on a Single Node – Part II

Sergey Yakubov - Maxwell Team - DESY IT
Hamburg, 28.05.2018

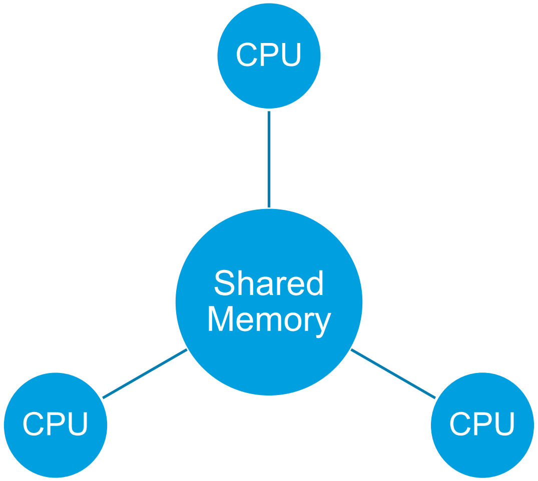
Agenda

- Recap
- Parallelization using OpenMP – Part II
- Summary

Software development

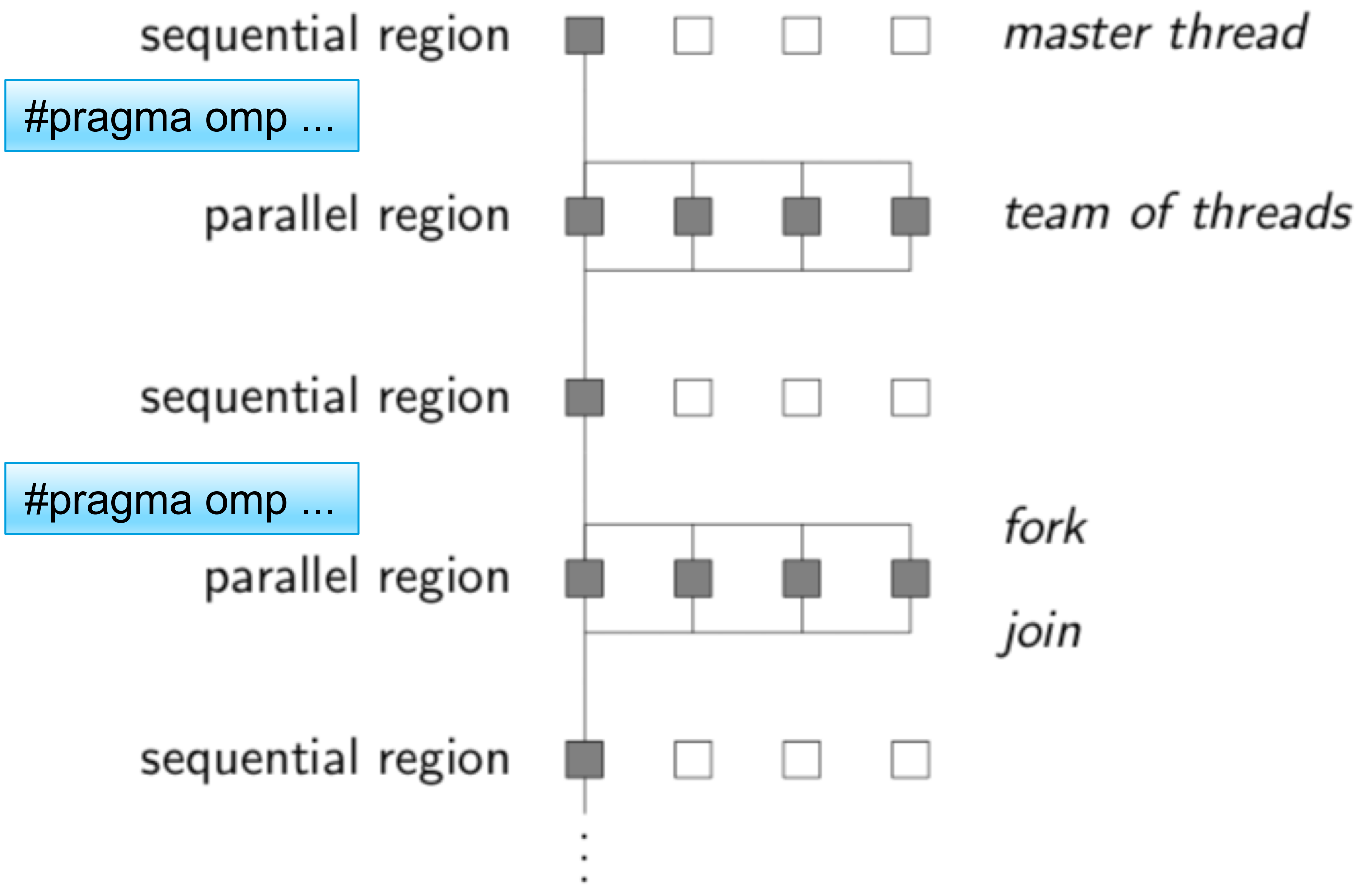
Main Challenges

- Single node = multi-core node -> multithreaded programming
    - All threads have access to the same resources (memory, I/O, ...)
  - Main challenges
    - Synchronization between threads
      - Race condition /shared variables
      - Deadlocks
    - False sharing
    - Be aware of “thread-safety” of 3rd-party libraries
    - Performance (is always a challenge)



Software development

OpenMP – Concepts



sequential region

#pragma omp ...

parallel region

sequential region

#pragma omp ...

parallel region

sequential region

master thread

team of threads

fork

join

Summary

From Part I

- Prefer C++ threads for C++ programs
  - somewhat easier
  - platform-independent
- Use OpenMP for scientific software
  - usually you have loops there
  - also platform independent
  - even more easier to program
- Be careful with shared variables
  - data race, ...
- Be careful with false sharing
- Don’t print from threads
- Always check performance

OpenMP – Parallelizing Loops

Simple loops

- simplest and most often used strategy

```
for (int i=0; i < 300; i++) {
     a[i] = b[i];
}
```

```
#pragma omp parallel for
for (int i=0; i < 300; i++) {
     a[i] = b[i];
}
```

- work is decomposed by the compiler and run-time library. E.g., for 3 threads:

```
for (int i=0; i < 100; i++)
{
     a[i] = b[i];
}
```

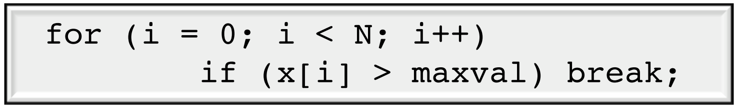
```
for (int i=101; i < 200;
i++) {
     a[i] = b[i];
}
```

```
for (int i=201; i < 300;
i++) {
     a[i] = b[i];
}
```

OpenMP – Parallelizing Loops

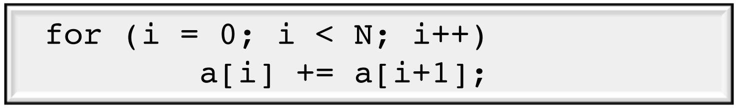
Problematic loops

- Some loops cannot be parallelized (by simply adding pragma)
  - break inside the loop

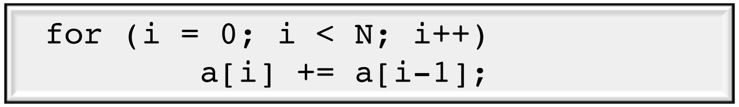


```
for (i = 0; i < N; i++)
    if (x[i] > maxval) break;
```

  - (anti)dependencies

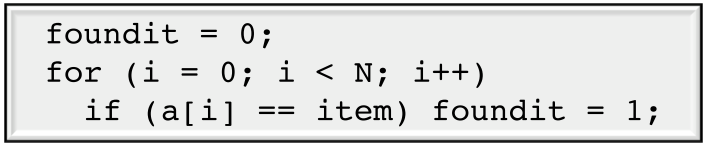


```
for (i = 0; i < N; i++)
    a[i] += a[i+1];
```

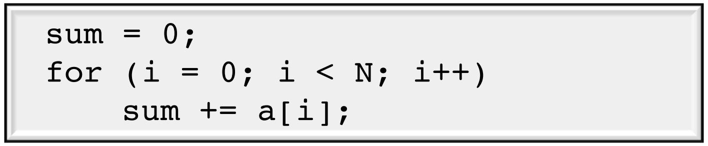


```
for (i = 0; i < N; i++)
    a[i] += a[i-1];
```

  - shared variables



```
foundit = 0;
for (i = 0; i < N; i++)
 if (a[i] == item) foundit = 1;
```



```
sum = 0;
for (i = 0; i < N; i++)
  sum += a[i];
```

OpenMP – Problematic Loops

Loop break

```
 for (i = 0; i < N; i++)
             if (x[i] > maxval) break;
```

Solution:

Don’t break ☺

OpenMP – Problematic Loops

Anti dependency

```
 for (i = 0; i < N - 1; i++)
         a[i] += a[i+1];
```

Solution:

split in two loops

```
#pragma omp parallel for
for(i = 0; i < N-1; i++)
   b[i] = a[i + 1];

#pragma omp parallel for
for(i = 0; i < N-1; i++)
   a[i] += b[i];
```

OpenMP – Problematic Loops

Flow dependency

```
for (i = 0; i < N - 1; i++)
        a[i] += a[i-1];
```

Solution:

difficult, don’t care unless really needed

OpenMP – Problematic Loops

Shared variables

```
foundit = 0;
for (i = 0; i < N; i++)
  if (a[i] == item) foundit = 1;
```

```
sum = 0;
for (i = 0; i < N; i++)
   sum += a[i];
```

Solution:

we had it last time, but for simple operations there is a better one

OpenMP – Reduction

```
int sum = 0;
#pragma omp parallel for reduction(+:sum)
{
  for (i = 0; i < N; i++)
        sum += a[i];
}
```

- Allowed operations: +,∗,−,max,min, &, &&, |, ||, ^
- Fallback to solution with local variables/locks for other cases

Hands-On

Problematic loops

- Look at /data/netapp/hpc-seminars/SingleNodeParallelization/6_problematic_loops
- Make it work in parallel

Finished? https://goo.gl/forms/QfmZL1jw6g5OE2Aq2

Hands-On - Solution

Problematic loops

```
// Problem 1
    int found_it = 0;
    #pragma omp parallel for reduction(max:found_it)
    for (int i = 0; i < size; i++)
        if (a[i] == item) found_it = 1;
    std::cout<<"Found: "<<(found_it?"yes":"no")<<std::endl;

// Problem 2
    int i,ind;
    ind = size;
    #pragma omp parallel for reduction(min:ind)
    for (i = 0; i < size; i++)
        if (a[i] == item) {ind = i;}
    std::cout<<"finished loop at "<<ind<<std::endl;

// Problem 3
    std::vector<int> b(size);
    #pragma omp parallel for
    for (int i = 0; i < size-1; i++)
        b[i] = a[i+1];
    #pragma omp parallel for
    for (int i = 0; i < size-1; i++)
        a[i] += b[i];
    std::cout<<"value: "<<a[size/2]<<std::endl;
```

Hands-On

Shared Counter with OpenMP (reduction)

- Look at /data/netapp/hpc-seminars/SingleNodeParallelization/7_openmp_shared_counter_reduction
- Rewrite it using reduction

Finished? https://goo.gl/forms/QfmZL1jw6g5OE2Aq2

Hands-On - Solution

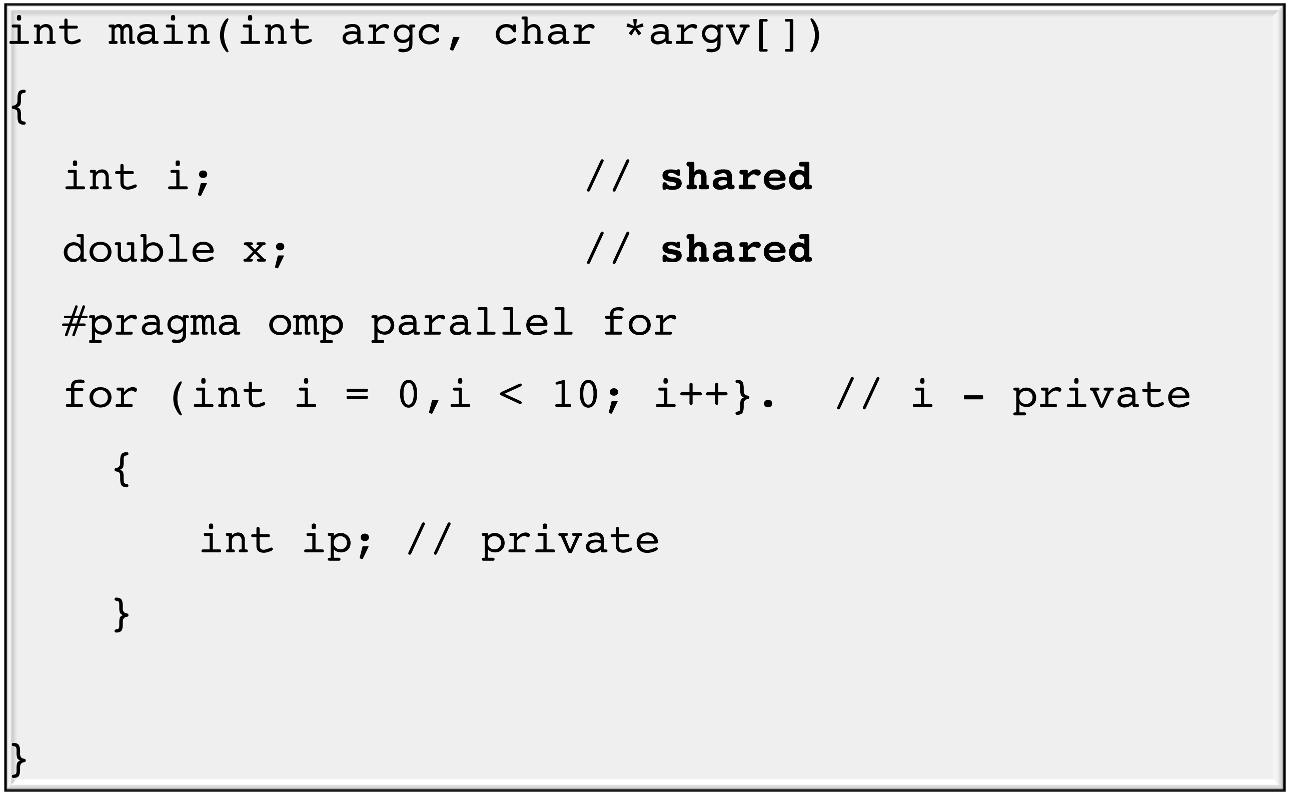
Shared Counter with OpenMP (reduction)

```
int sharedCounter = 0;

#pragma omp parallel for reduction(+:sharedCounter)
for (int n = 0; n < 10; n++)
    for (int i = 0; i < 10000000; i++) {
        sharedCounter++;
        dummy(&sharedCounter);
    }
```

OpenMP – Shared and Private Variables

Once again



```
int main(int argc, char *argv[])
{
  int i;                    // shared
  double x;                 // shared
  #pragma omp parallel for
  for (int i = 0,i < 10; i++}.  // i - private
    {
        int ip; // private
    }


}
```

OpenMP – Shared and Private Variables

Inside function

```
int main(int argc, char *argv[])
{
  int n;                // shared
  double x;             // shared
  #pragma omp parallel for
  for (int i = 0;i < 10; i++}  // i - private
       called(true);
  printf (“counted %d\n“,counted(false));
}
int called (bool update) {
  int i; // private
  static int counter = 0; // shared!
  if (update) counter++;
  return counter;
}
```

Hands-On

Static variable inside a function

- Look at /data/netapp/hpc-seminars/SingleNodeParallelization/8_variable_scope
- Make it work

Finished? https://goo.gl/forms/QfmZL1jw6g5OE2Aq2

Hands-On - Solution

Static variable inside a function

```
void called(bool update,int *counter) {
  // do something
    if (update) {
        (*counter)++;
    }
}

int main(int argc, char* argv[]) {
    int counter = 0;
    #pragma omp parallel for reduction(+:counter)
    for (int i = 0; i < 100000; i++)
        called(true,&counter);

    printf("counted %d\n", counter);
}
```

OpenMP – Other stuff

Which we'll not cover in details

- Syncronization
  - barrier
    - synchronisation point of all threads in the team
  - critical
    - restricts execution of a block to a single thread at a time
  - atomic
    - ensures that a storage location is updated by one thread at a time

```
#pragma omp parallel for {
   for (i = 0; i < N; i++)
          #pragma omp atomic
          s += a[i];
}
```

bad

```
#pragma omp parallel for {
   for (i = 0; i < N; i++){
          a[i] = very_long_function(i)
          #pragma omp atomic
          s += a[i]; }
}
```

better

In addition there are run-time library lock routines (similar to what we had for pthreads)

Hands-On – Solution (version 2)

Static variable inside a function

```
int called(bool update) {
    static int counter = 0; // shared!

    // do something
    if (update)  {
        #pragma omp atomic
        counter++;
    }
    return counter;
}

int main(int argc, char* argv[]) {

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++)
        called(true);

    printf("counted %d\n", called(false));
}
```

make it serial in this case, but might be ok in other situations

OpenMP – Other stuff

Which we'll not cover in details

- Loop schedules
  - assignment of loop iterations to threads
    - static: The assignment of iterations to threads is given by the number of iterations and the number of threads alone. It is determined at the beginning of the loop
    - dynamic: The assignment of iterations to threads is determined at run-time. The assignment happens chunk by chunk. It can vary from run to run (at identical parameters)

```
#pragma omp parallel for schedule (static [,chunk])
```

OpenMP – Other stuff

Which we'll not cover in details

- Functional parallelism
  - paralellization without loops

```
#pragma omp parallel sections
{
  #pragma omp section
  {
  //this code will be executed by thread 0
  }
  #pragma omp section
  {
  //this code will be executed by thread 1
  }
}
```

  - tasks (since OpenMP 3)

OpenMP – Other stuff

Which we'll not cover in details

- Functional parallelism
  - paralellization without loops

```
#pragma omp parallel sections
{
   #pragma omp section
   {
   //this
   }
   #pragma omp section
   {
   //this code will be executed by thread 1
   }
}
```

Better do it with MPI? – come in two weeks!

  - tasks (since OpenMP 3)