The Second Biomedical Engineering Workshop

A brief introduction to OpenMP

Dr. Hisham Mohamed

Sensirion AG, Switzerland

SENSIRION
THE SENSOR COMPANY

Overview

- OpenMP programming model
- Work sharing
  - Parallel for loops
  - Parallel sections
  - Parallel tasks
- Data scoping & synchronization

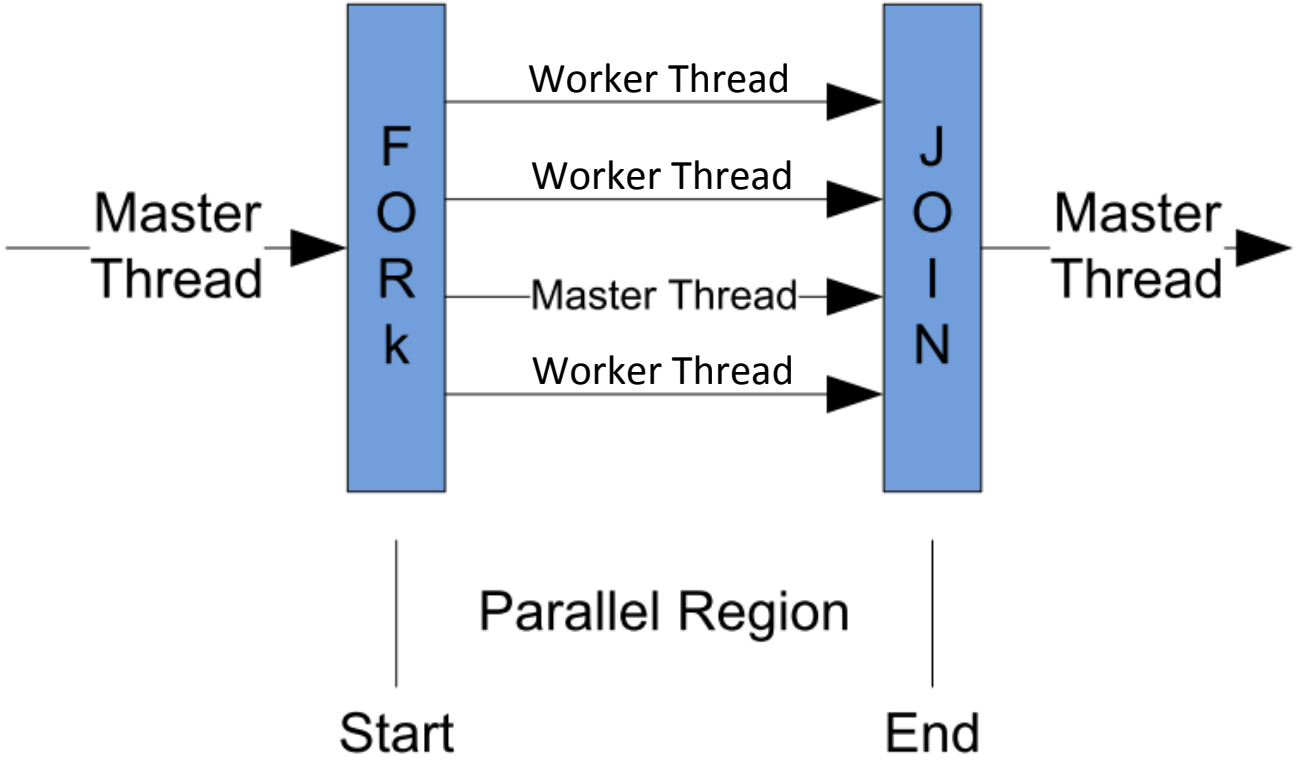
OpenMP

- Stands for : Open Multi-Processing

- API:
  - Allows shared memory parallelism
  - Specified for C/C++ and Fortran
  - Not distributed parallelism

- Released in 2005

OpenMP: Programming Model



Fork-Join model

OpenMP: Directives

#pragma omp parallel [clause ...] newline

Required for all OpenMP C/C++ directives.

Optional Clauses:
if (scalar_expression)
private (list)
shared (list)
reduction (operator: list)
copying (list)
num_threads (integer-expression)

OpenMP directive. After the "**pragma**" and before any clauses.

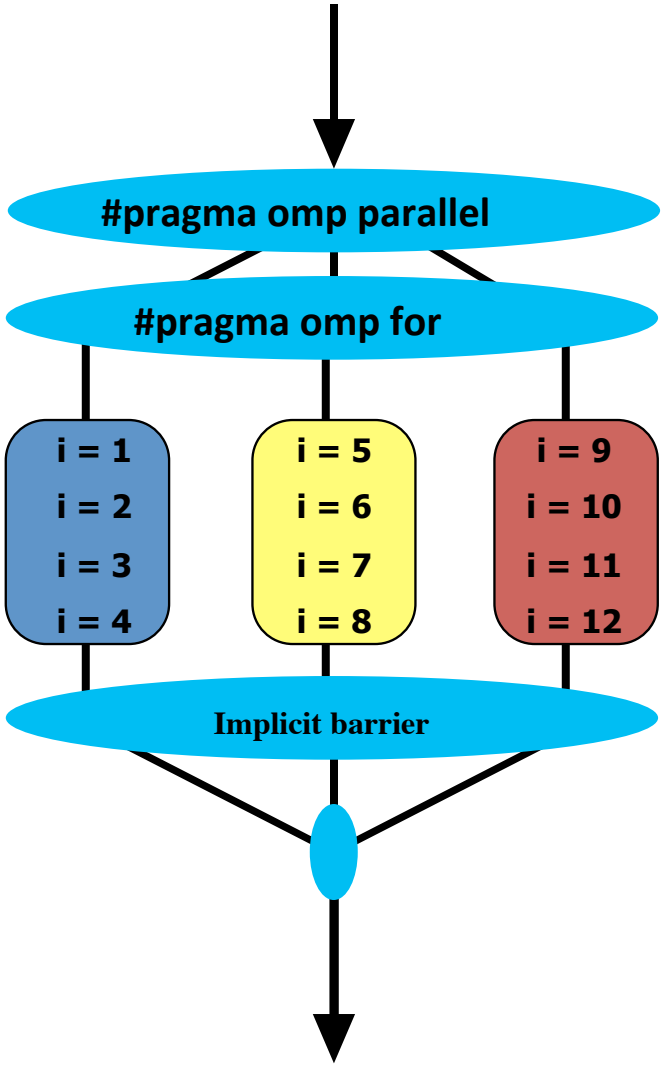
OpenMP: Pseudo Code

```
#include <omp.h>

main () {

    int var1, var2, var3;

    Serial code

    Beginning of parallel section. Fork a team of threads.
    Specify variable scoping

    #pragma omp parallel private(var1, var2) shared(var3)
    {
        Parallel section executed by all threads
          .
          .
       All threads join master thread and disband
    }
    Resume serial code
}
```

Parallel “for” loops

Parallel “for” loops

```
int N=12

#pragma omp parallel
#pragma omp for
for(i = 1, i < N+1, i++){

    c[i] = a[i] + b[i];

}
```



Parallel “for” loops

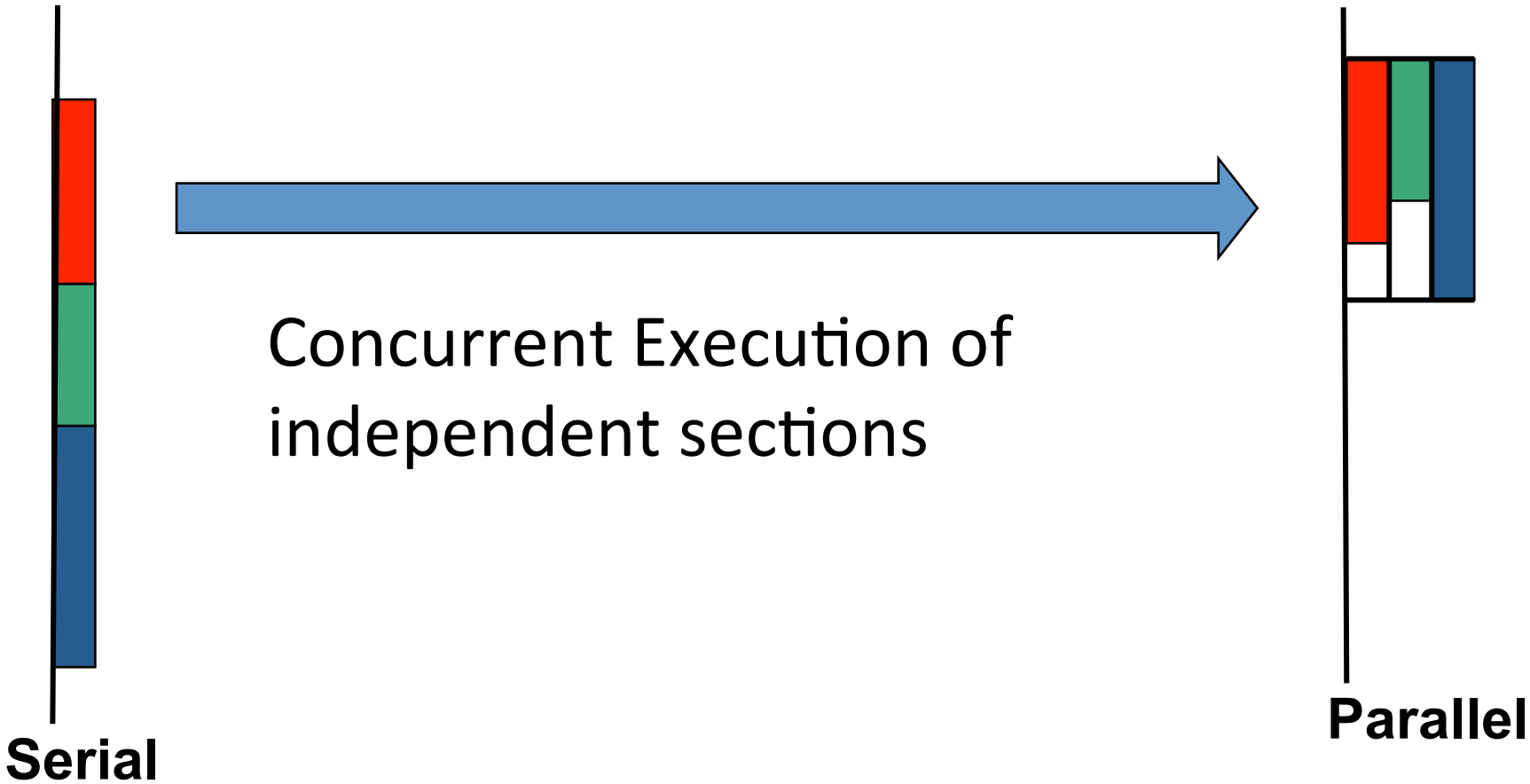
```
#pragma omp parallel
#pragma omp for
for(i = 1, i < N+1, i++){

    c[i] = a[i] + b[i];

}
```

```
#pragma omp parallel for
for(i = 1, i < N+1, i++) {

    c[i] = a[i] + b[i];

}
```

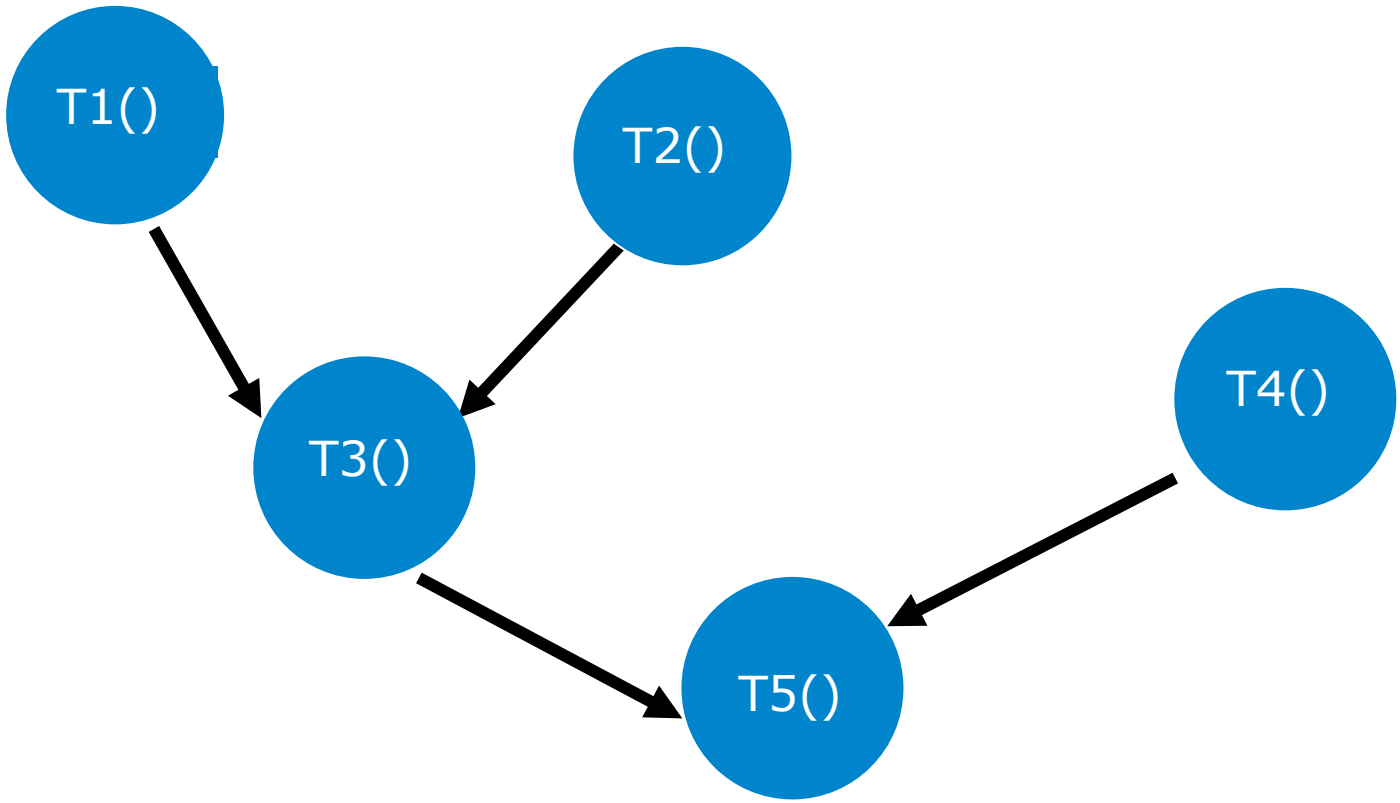
The same!

Parallel Sections

Parallel Sections



Parallel Sections

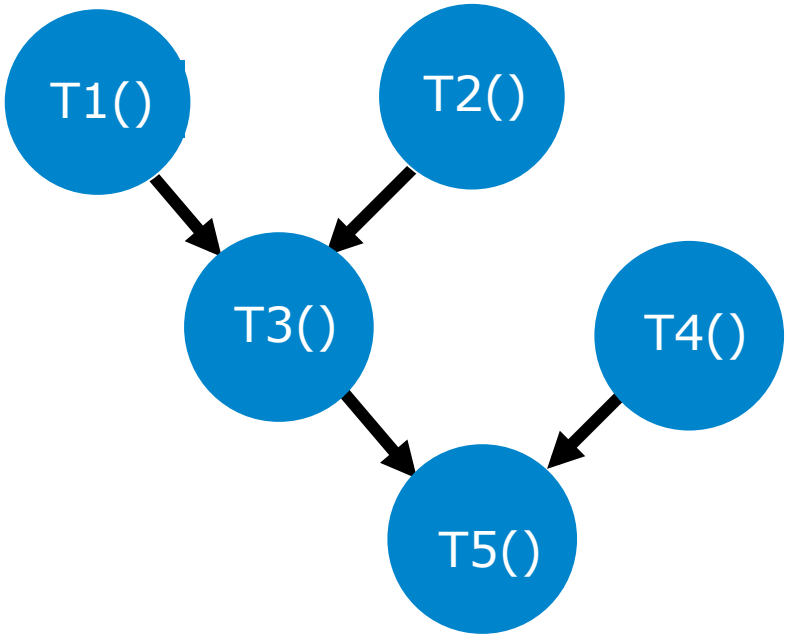


T1(),T2(), and T4() can be computed in parallel

Parallel Sections

```
#pragma omp parallel sections
{
   #pragma omp section
    x = T1()
   #pragma omp section
    y = T2()
   #pragma omp section
   z = T4();
}
 s = T3(x, y);
 printf ("%d",T5 (s,z));
```

Implicit barrier

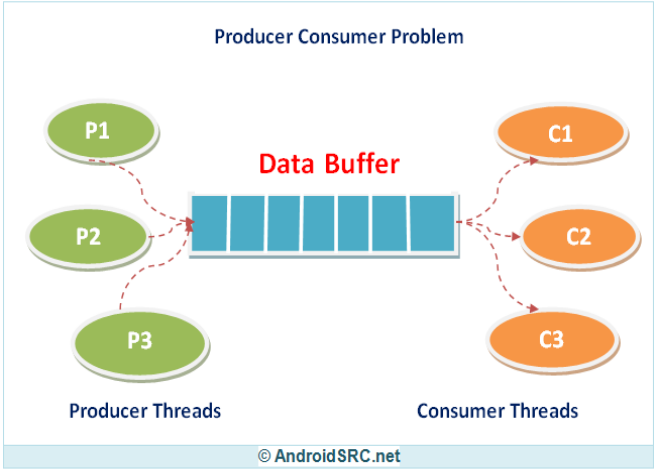


Parallel Tasks

Parallel Tasks

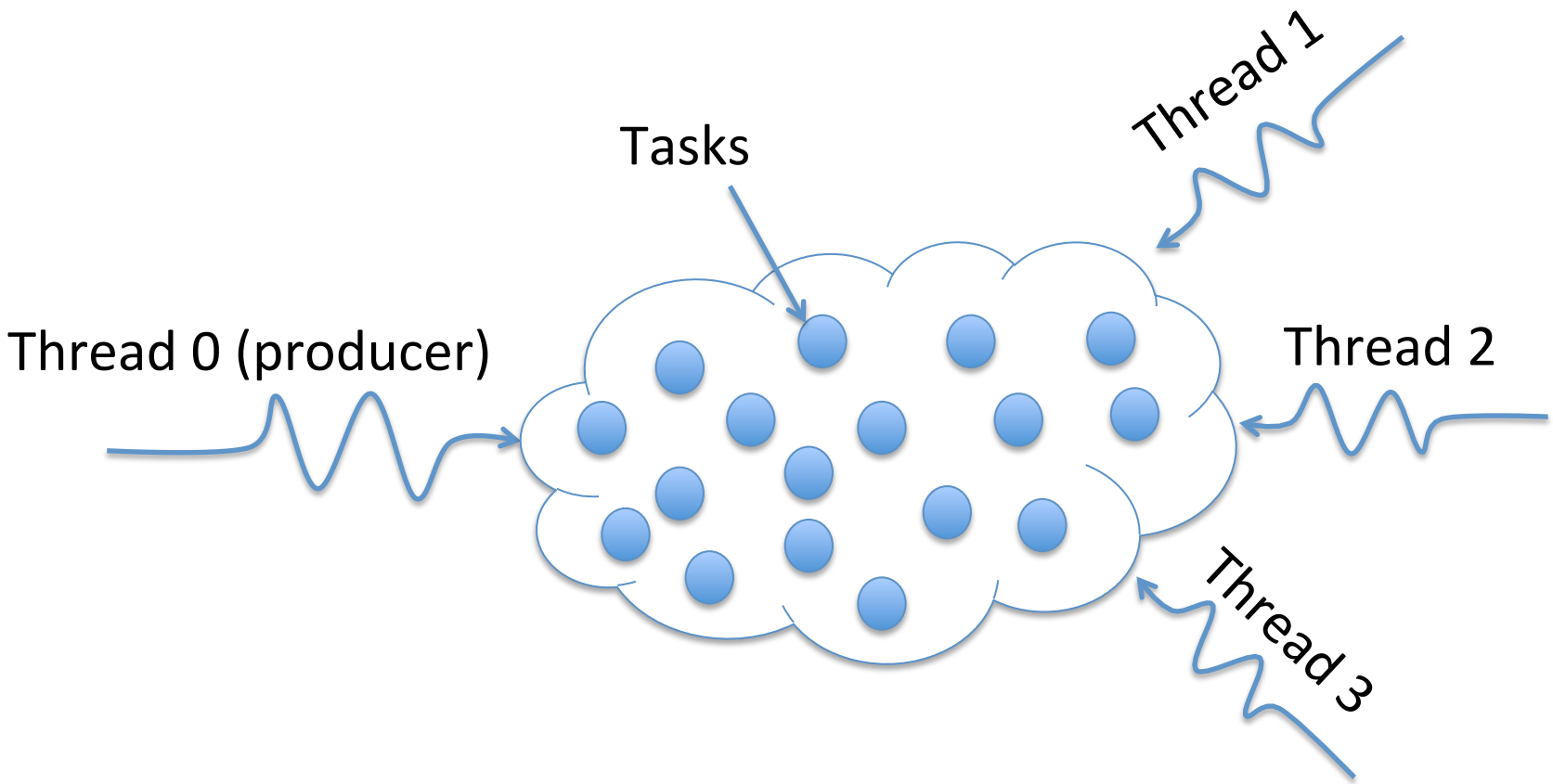
- Available from OpenMP 3.0

- Suitable for:
  - Unbounded loops (e.g. while(x!=0))
  - Recursive algorithms
  - Producer/consumer

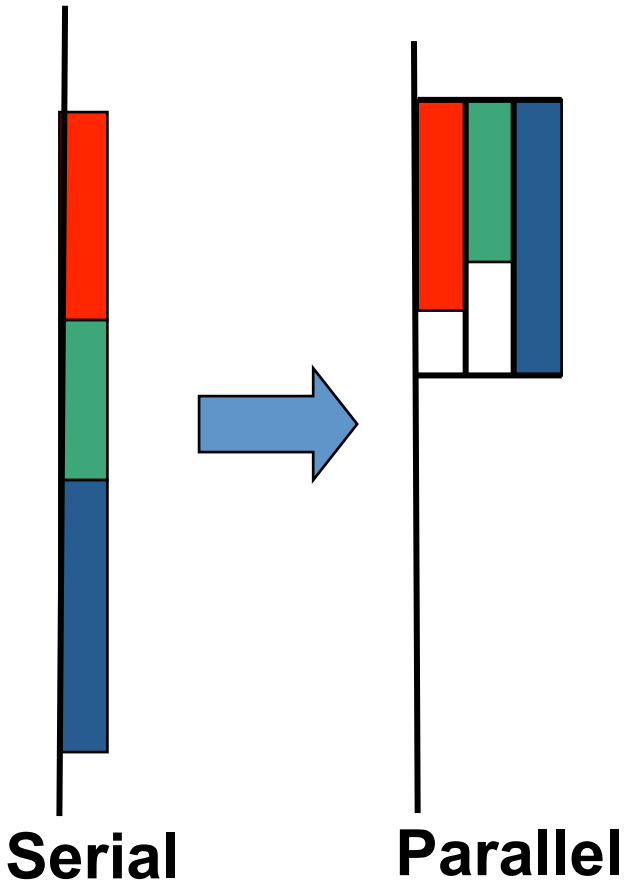


http://androidsrc.net/producer-consumer-problem-java-detailed-explaination/

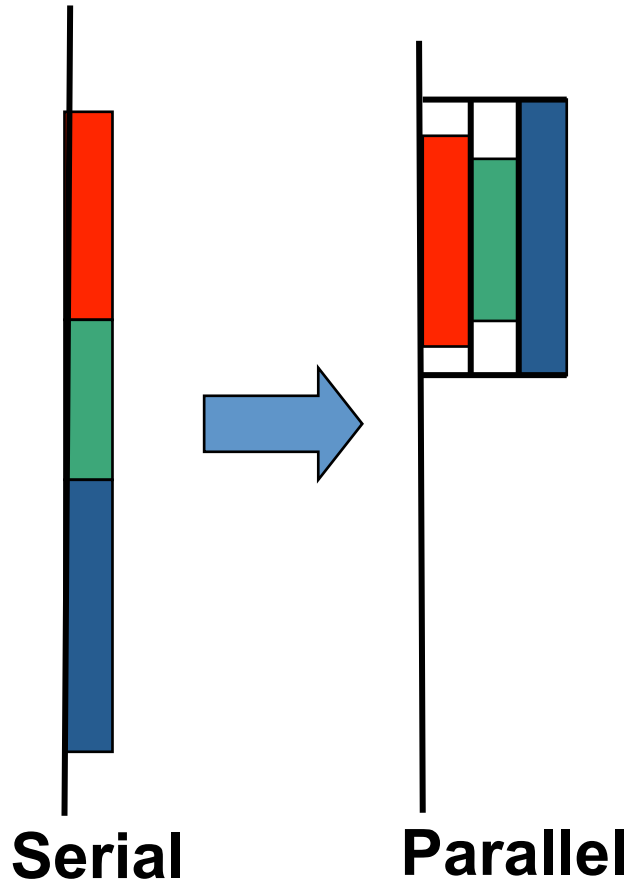
Parallel Tasks



Parallel Sections vs. Parallel Tasks

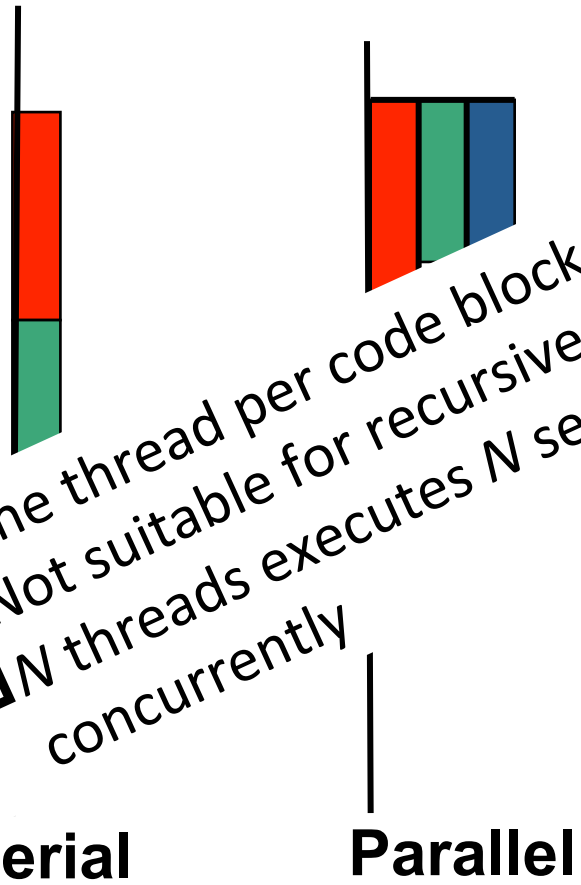


Parallel Sections



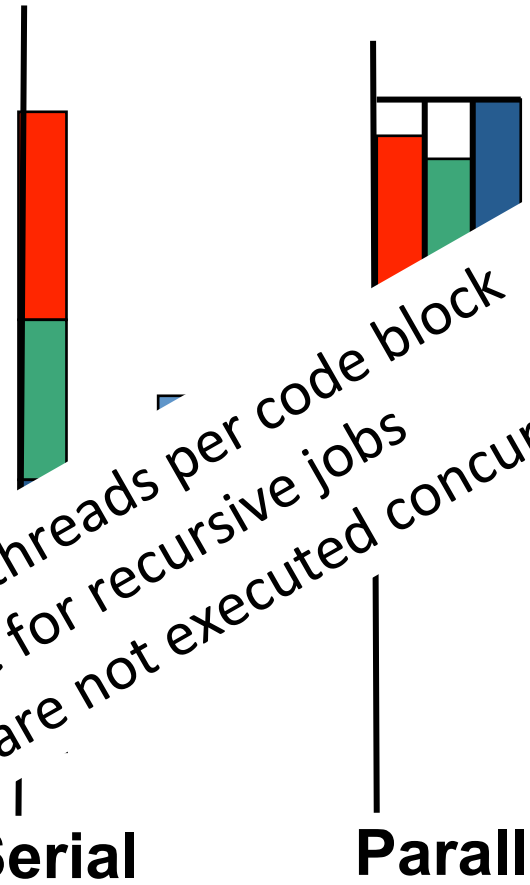
Parallel Tasks

Parallel Sections vs. Parallel Tasks



- One thread per code block
- Not suitable for recursive jobs
- *N* threads executes *N* sections concurrently

Parallel Sections



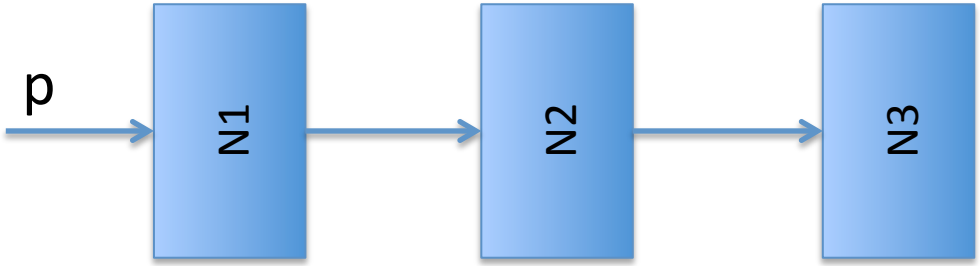
- Multiple threads per code block
- Suitable for recursive jobs
- Tasks are not executed concurrently

Parallel Tasks

Parallel Tasks

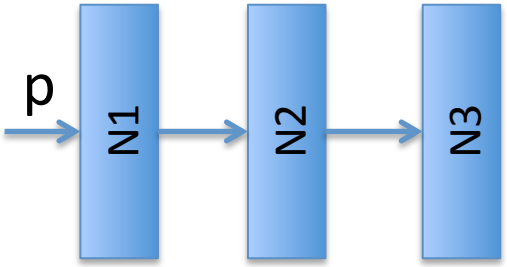
```
node *p = head;

while (p) {

      process(p);

      p = p->next;
}
```

One section can not be parallelized with OpenMP sections!



Parallel Tasks

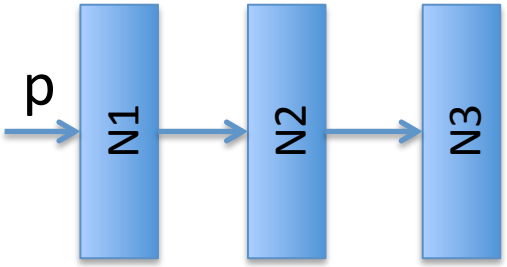
```
node *p = head;

while (p) {

    process(p);
                    } Independent !
    p = p->next;
}
```



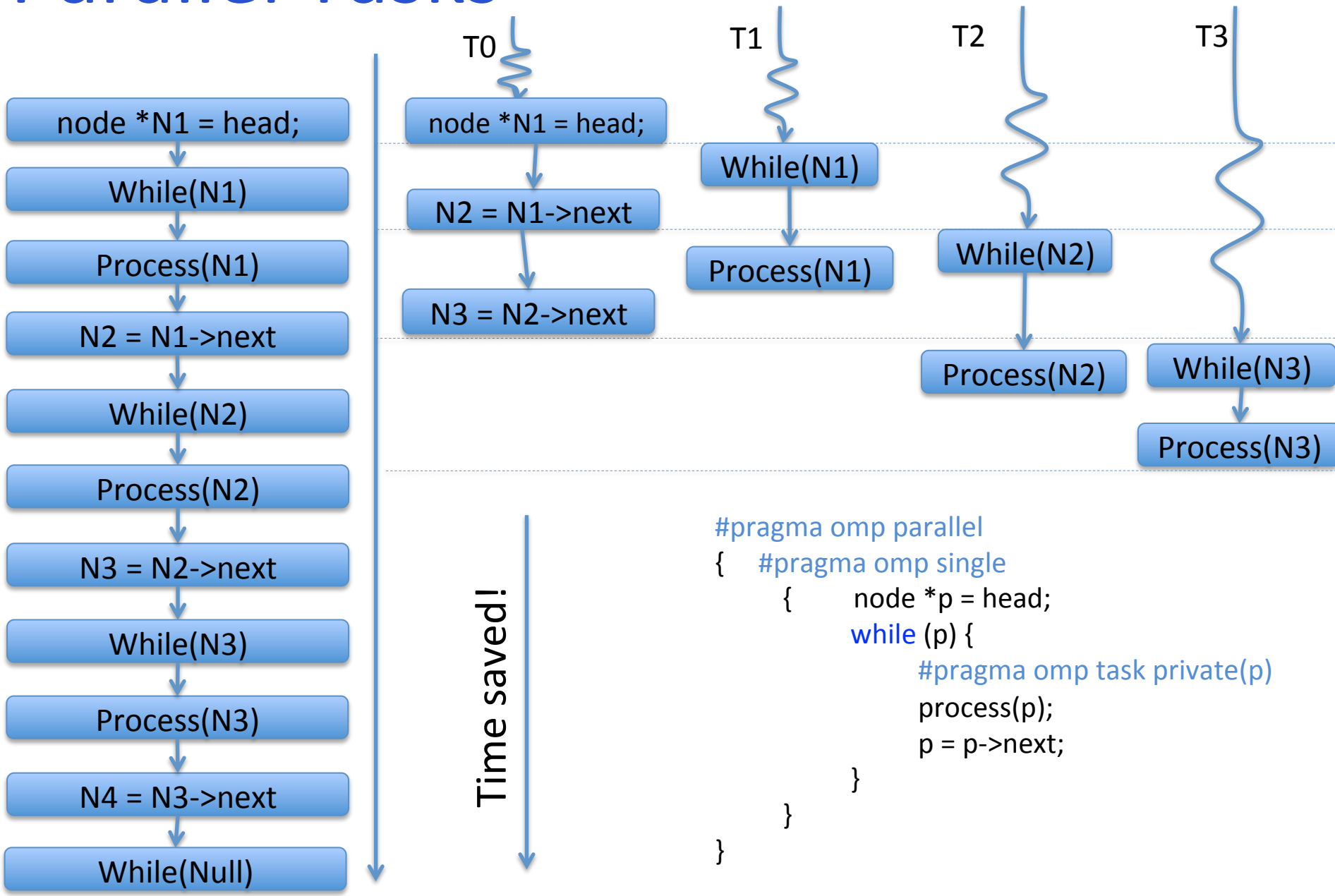
Parallel Tasks

```
node *p = head;

while (p) {

    process(p);
                    Independent !
    p = p->next;
}
```

```
#pragma omp parallel
{
    #pragma omp single
    {
        node *p = head;
        while (p) {
            #pragma omp task private(p)
            process(p);
            p = p->next;
        }
    }
}
```



Parallel Tasks




```
#pragma omp parallel
{   #pragma omp single
        {       node *p = head;
                while (p) {
                        #pragma omp task private(p)
                        process(p);
                        p = p->next;
                }
        }
}
```

Data scoping

Data Scoping

- Shared

#pragma omp parallel shared(j,k)

- Private
  - Private, firstprivate, threadprivate

#pragma omp parallel private(j,k)

Data Scoping: firstprivate

```
Incr=1;

#pragma omp parallel for firstprivate(incr)

for (I=0;I<=MAX;I++) {

    if ((I%2)==0) incr++;

    A(I)=incr;

}
```

incr = 1 for all the threads

Data Scoping: threadprivate

```
int init_value = 9
#pragma omp threadprivate(init_value)
int main()
 {

    #pragma omp parallel {
        int final = 2 * init_value
       // do something
     }

 }
```

Final = 18

Synchronization

Race Condition

- Two or more threads access a shared variable

```
float dot_prod(float* a, float* b, int N) {

float sum = 0.0;
#pragma omp parallel for shared(sum)
  for(int i=0; i<N; i++) {

    sum += a[i] * b[i];

  }
 return sum;
}
```

Race Condition: Critical Regions

```
float dot_prod(float* a, float* b, int N) {

  float sum = 0.0;
  #pragma omp parallel for
   for(int i=0; i<N; i++) {

    #pragma omp critical
     sum += a[i] * b[i];

   }
  return sum;
}
```

Overhead!

Seq. comp. inside a parallel region

```
#pragma omp parallel
{
  Foo();

#pragma omp single
  {
   SeqCompu();
  }

  Bar();
}
```

```
#pragma omp parallel
{
   Foo();

  #pragma omp master
  {
     SeqCompu(x);
  }

  Bar();
}
```

Barriers

```
#pragma omp parallel shared (A, B, C)
{
    Foo(A,B);

    printf("Processed A and B\n");

    #pragma omp barrier

    Bar(B,C);

    printf("Processed B and C\n");
}
```

Conclusion

- Parallel “for” loops
- Parallel tasks vs. parallel sections
- Data scoping
- Thread synchronization

References

- http://www.openmp.org/mp-documents/OpenMP3.1.pdf
- http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf