

OpenMP: parallel for loop

Erik Saule

`esaule@uncc.edu`

Intro to Parallel Programming

Learning Outcomes

After this lecture you will be able to

- give an example of how middleware help programming parallel system
- write parallel loops in OpenMP
- convert an OpenMP program using loops and a scheduling policy to a graph

The assignment will make you

- use OpenMP loop construct in practice
- implement complex algorithm using OpenMP
- see cases where loop constructs do not get the schedule expected

Outline

- 1 Basic OpenMP
- 2 Parallel for construct
- 3 From OpenMP to scheduling
- 4 Further

Middleware

Essentially a software layer that sits between the operating system and the application to provide higher level abstraction.

Language Extension

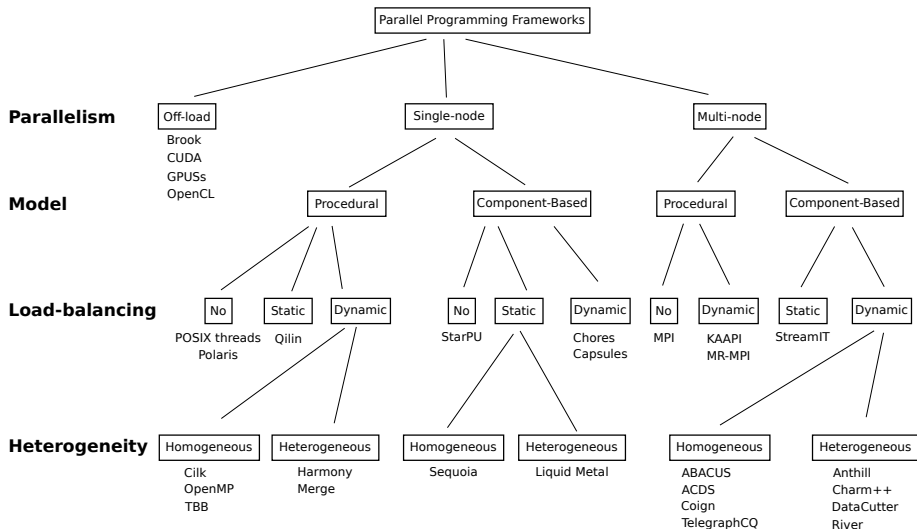
In parallel computing, it often comes with introducing a new language, a language extension, or a complex set of API.

Runtime System

A runtime system, a library or a set of process or threads, that implements and manages the abstractions.

Examples of middleware for parallel computing

Examples of middleware for parallel computing



source: Tim Hartley, PhD dissertation

Compiler Extension

Compiler Extension for Fortran and C. In C, they are preprocessor directives `#pragma omp`. Declare blocks of code to be parallel and how they should be scheduled. And insert callbacks to the runtime.

Runtime System

Mostly a library that will manage thread pools, thread synchronization, schedule tasks, perform reductions, ...

Include `omp.h` and compile with `-fopenmp`.

OpenMP changed quite a bit recently, so pay attention to which OpenMP version is supported by compilers.

parallel section

code

```
#include <omp.h>
#include <iostream>

int main () {

    std::cout<<"Before"<<std::endl;

    #pragma omp parallel
    {
        std::cout<<"During"<<std::endl;
    }

    std::cout<<"After"<<std::endl;

    return 0;
}
```

Produces

Before
DuringDuring
During
During
After

Can we nest parallel sections? hum... It's complicated.

- Variables declared out of a parallel section are shared among threads.
- Variables declared in a parallel section are local to each threads.
- Programmer can control scoping with `#pragma parallel shared(var1, var2)`
- Support different scoping: `shared`, `private`, `threadprivate`, `lastprivate`, `firstprivate`, (Check documentation.)

Synchronization

master and single

- Section of code executed only once.
- If it is a master section, it is executed only by the master thread (Thread 0).
- If it is a single section, it is by any of them (Often, the first).

critical

Makes a section that only one thread can execute at a time.

atomic

protects just a single statement. Often to make a atomic instruction call in the backend.

lock

Pretty much a mutex.

Outline

- 1 Basic OpenMP
- 2 Parallel for construct
- 3 From OpenMP to scheduling
- 4 Further

Worksharing manually a for loop?

Simply

One can get the thread id `omp_get_thread_num()` between 0 and `omp_get_num_threads()`. (Confusing names!)

Thread i executes all iterations between $i \frac{N}{P}$ and $(i + 1) \frac{N}{P} - 1$.

not so simply

With locks, reimplement the scheduler from assignment 3.

parallel for

```
#include <omp.h>

int main () {
    int n = 10000;
    int* arr = new int[n];

    #pragma omp parallel for
        for (int i=0; i<n; ++i)
            arr[i] = 0;

    return 0;
}
```

parallel for

```
#include <omp.h>

int main () {
    int n = 10000;
    int* arr =new int[n] ;

#pragma omp parallel
    {
        //all threads execute before

#pragma omp for
        for (int i=0; i<n; ++i)
            arr[i] = 0; //workshared

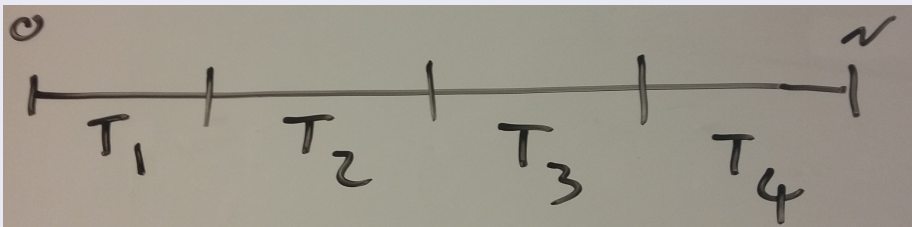
        //all threads execute after
    }
    return 0;
}
```

`#pragma omp for reduction(+:a)` will sum the local values of `a` at the end of the for loop.

different operators are available: `+`, `-`, `*`, `|`, `&`, `^`, `&&`, `||`. `min` and `max` are supported since openmp 3 (.1 ?), but were supported in Fortran since 2.0.

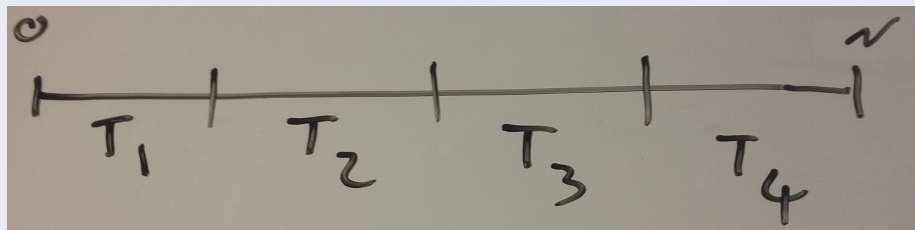
Scheduling - static

```
#pragma omp for schedule(static)
```

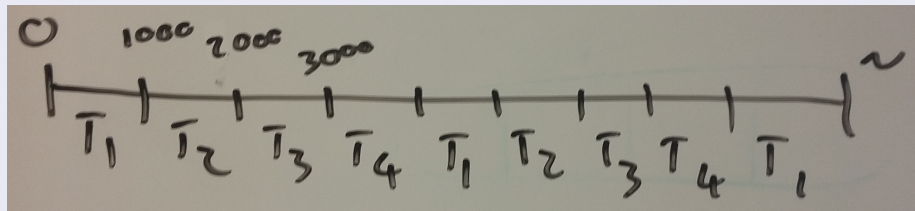


Scheduling - static

```
#pragma omp for schedule(static)
```

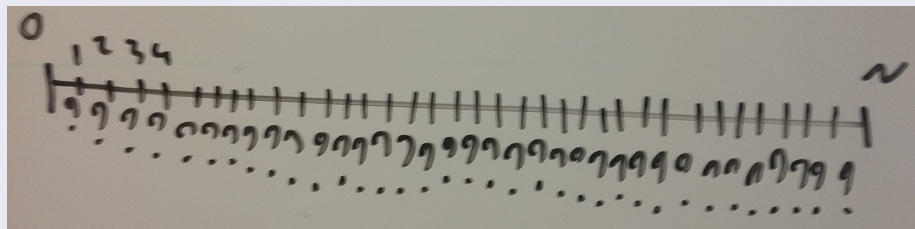


```
#pragma omp for schedule(static,1000)
```



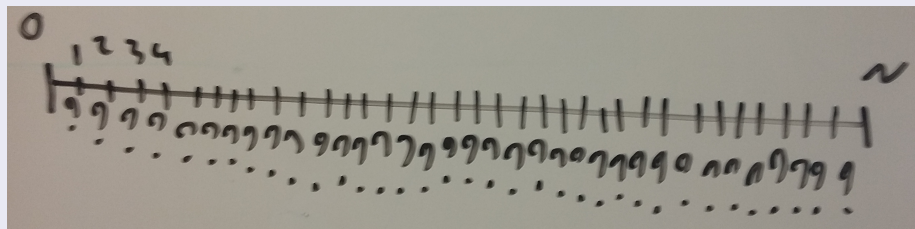
Scheduling - dynamic

```
#pragma omp for schedule(dynamic)
```

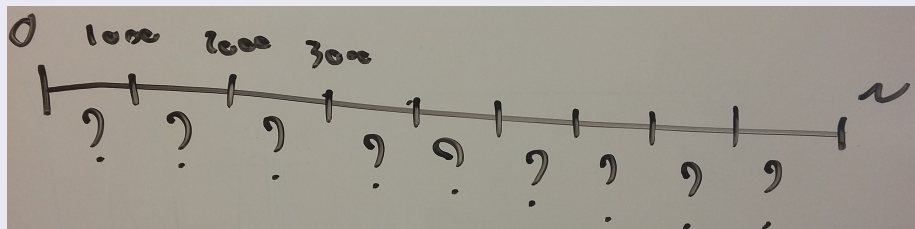


Scheduling - dynamic

```
#pragma omp for schedule(dynamic)
```

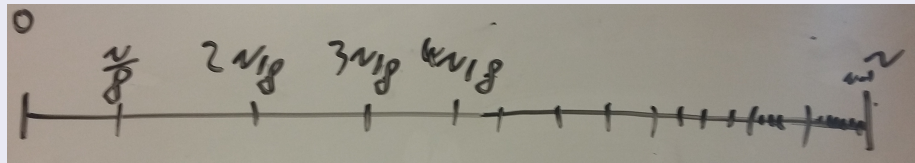


```
#pragma omp for schedule(dynamic,1000)
```



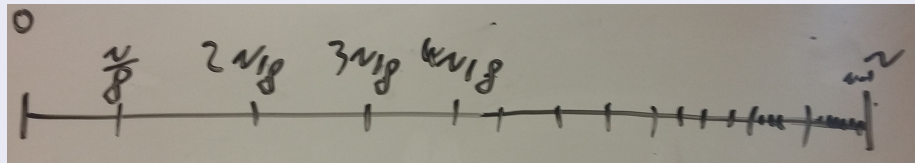
Scheduling - guided

```
#pragma omp for schedule(guided)
```

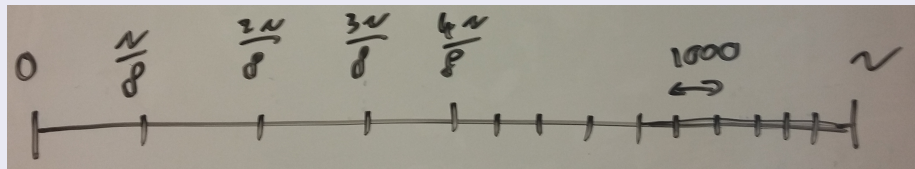


Scheduling - guided

```
#pragma omp for schedule(guided)
```



```
#pragma omp for schedule(guided,1000)
```

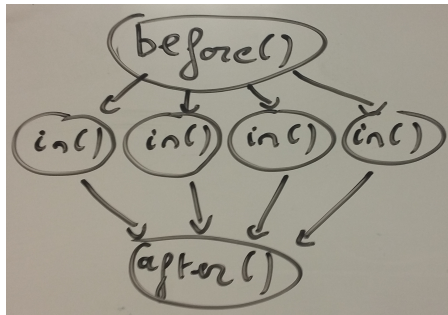


Outline

- 1 Basic OpenMP
- 2 Parallel for construct
- 3 From OpenMP to scheduling
- 4 Further

omp parallel

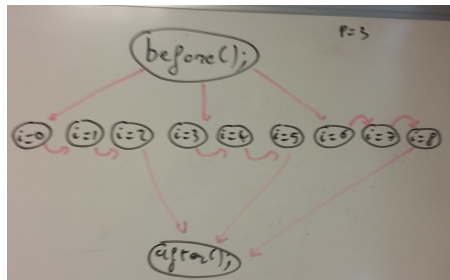
```
before();  
#pragma omp parallel  
{  
    inc()  
}  
after();
```



(assuming the code does not put more constraint such as locks.)

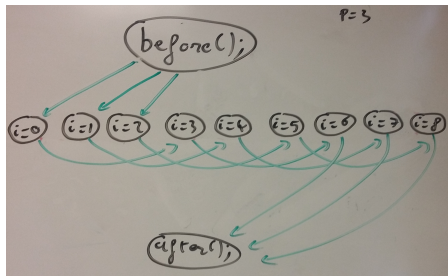
parallel for - static

```
int a[9];  
before();  
#pragma omp parallel  
{  
  #pragma omp for schedule(static)  
  for (int i=0; i<9; i++)  
    a[i]=0;  
}  
after();
```



parallel for - static

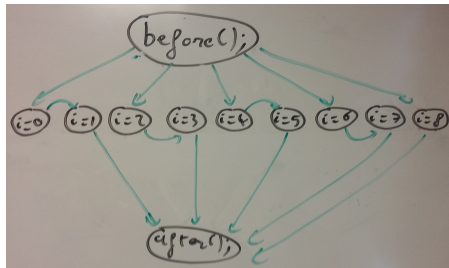
```
int a[9];  
before();  
#pragma omp parallel  
{  
    #pragma omp for schedule(static,1)  
    for (int i=0; i<9; i++)  
        a[i]=0;  
  
}  
after();
```



parallel for - dynamic

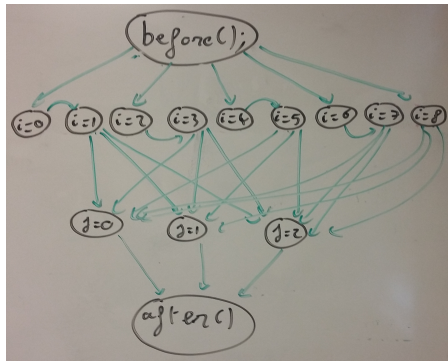
```
int a[9];  
before();  
#pragma omp parallel  
{  
    #pragma omp for schedule(dynamic,2)  
    for (int i=0; i<9; i++)  
        a[i]=0;  
}
```

after();



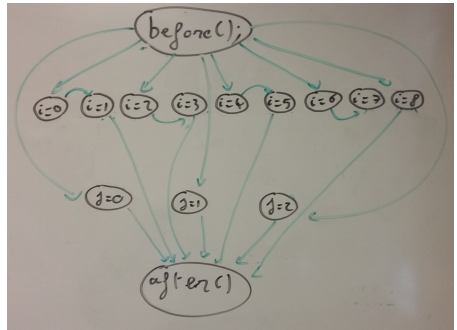
consecutive for loops

```
int a[9], b[3];  
before();  
#pragma omp parallel  
{  
  #pragma omp for schedule(dynamic,2)  
  for(int i=0; i<9; i++)  
    a[i]=0;  
  #pragma omp for schedule(dynamic,1)  
  for(int j=0; j<3; j++)  
    b[j]=0;  
}  
after();
```

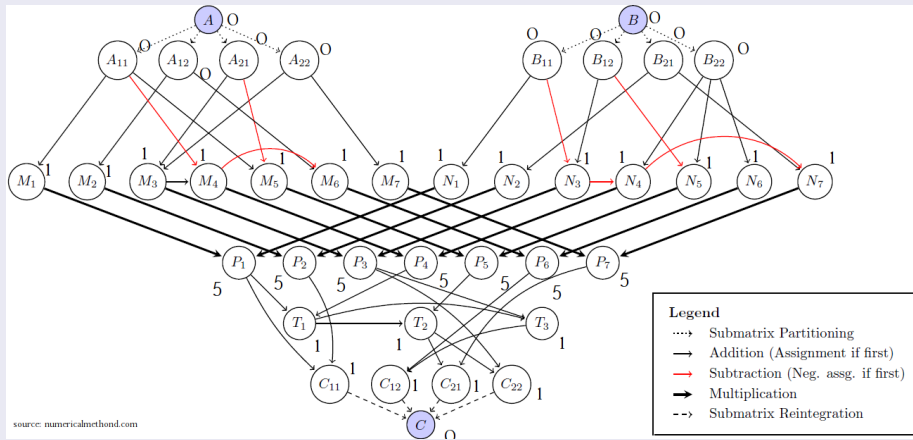


consecutive for loops : the nowait clause

```
int a[9], b[3];  
before();  
#pragma omp parallel  
{  
  #pragma omp for schedule(dynamic,2) nowait  
  for (int i=0; i<9; i++)  
    a[i]=0;  
  #pragma omp for schedule(dynamic,1)  
  for (int j=0; j<3; j++)  
    b[j]=0;  
}  
after();
```



Some dependency structure can not be represented



Outline

- 1 Basic OpenMP
- 2 Parallel for construct
- 3 From OpenMP to scheduling
- 4 Further**

Middleware:

- Wikipedia <https://en.wikipedia.org/wiki/Middleware>
- Tim Hartley. Accelerating Component-Based Dataflow Middleware with Adaptivity and Heterogeneity. PhD Dissertation. OSU. 2011.

OpenMP official

- OpenMP committee: <http://openmp.org/wp/>
- Openmp 3.1 ref card : <http://openmp.org/mp-documents/OpenMP3.1-CCard.pdf>

Learning OpenMP:

- B. Chapman, G. Jost, R van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming. 2007
- TAMU: OpenMP by Example slides <http://sc.tamu.edu/shortcourses/SC-openmp/openmp-2013.pdf>
- GCC's OpenMP runtime <https://gcc.gnu.org/onlinedocs/libgomp/>
- LLNL tutorial <https://computing.llnl.gov/tutorials/openMP/>