

# Calcul Parallèle avec OpenMP

J. Koko

LIMOS - Cours ED-SPI

16-17/01/19

# OpenMP Overview

OpenMP (Open specifications for MultiProcessing) API for multithreaded applications

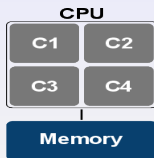
- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 20 years of Symmetric Multi-Processing (SMP) practice

Suitable architectures

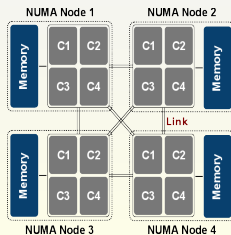
- Multi-threading processors
- Multi-core machines
- Shared-memory machines

# Shared memory architecture for SMP

Architecture **UMA** (Uniform Memory Access) : Equal access time



Architecture **NUMA** (Non-Uniform Memory Access) : Different access time



# OpenMP Syntax

- OpenMP compiler directives

```
#pragma omp construct[clause[]...]
```

Example : #pragma omp parallel for

- Function prototypes and types in the file :

```
#include "omp.h"
```

- OpenMP constructs apply to a structured block :
  - a block of one or more statements with one point of entry at the top and one point of exit at the bottom.
  - a structured block can contain an `exit()`

# Check if your environment works

## hello\_omp.c

```
#include <stdio.h>
#include "omp.h"
int main()
{
    int mytid=-1, nb_ths=0;
    #pragma omp parallel
    {
        nb_ths=omp_get_num_threads(); mytid=omp_get_thread_num();
        printf("Hello World from tid %d  %d \n", mytid, nb_ths);
    }
    return 0;
}
```

Compile : gcc -fopenmp hello\_omp.c

Set the number of threads : export OMP\_NUM\_THREADS=8

Run : ./a.out

# Check if your environment works

## hello\_omp.c 2

```
#include <stdio.h>
#include "omp.h"
int main()
{
    int mytid=-1, nb_ths=0;
    omp_set_num_threads(8);
    #pragma omp parallel
    {
        nb_ths=omp_get_num_threads(); mytid=omp_get_thread_num();
        printf("Hello World from tid %d  %d \n", mytid, nb_ths);
    }
    return 0;
}
```

Compile : gcc -fopenmp hello\_omp.c

Run : ./a.out

# Fixing the number of threads

## Environment variable

```
export OMP_NUM_THREADS=8
```

## Runtime library

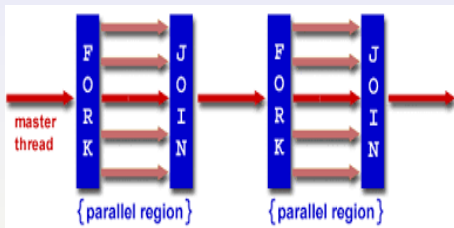
```
omp_set_num_threads(8);
```

## Clause of parallel region

```
#pragma omp parallel num_threads(8)
```

# Threads interaction

## Fork-Join model



## OpenMP is a multi-threading, shared address model

- Threads communicate by sharing variables
- Sharing of data causes data conflicts
- Synchronization protects data conflicts
- Synchronization is expensive



# Parallel sections

```
#pragma omp parallel sections (+clauses)
{
    #pragma omp section
    {
        block 1
    }
    ...
    #pragma omp section
    {
        block n
    }
} /* synchronization */
```

The blocks defined by `#pragma omp section` are performed in parallel by the threads in the team.

# Parallel loop

```
#pragma omp parallel shared(x,y,z)
{
    #pragma omp for private(i)
    for (i=0; i<DIM; i++)
        z[i] = x[i]*y[i];
}
```

Equivalent combined version

```
#pragma omp parallel for shared(x,y,z) private(i)
for (i=0; i<DIM; i++)
    z[i] = x[i]*y[i];
```

# Programming example $C = A + B$ serial

```
#include <stdio.h>
#include <stdlib.h>
#define N 4096

int main ()
{
    double A[N][N], B[N][N], C[N][N];
    int i,j;

    for (i = 0; i < N; i++) /* Initialization */
        for (j = 0; j < N; j++){
            A[i][j] = i*N+j; B[i][j] = i+j;
        }

    for (i = 0; i < N; i++) /* Computing */
        for (j = 0; j < N; j++){
            C[i][j] = A[i][j] + B[i][j];
        }

    return 0;
}
```

# Programming example $C = A + B$ parallel

```
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"
#define N 4096

int main ()
{
    double A[N][N], B[N][N], C[N][N];
    int i,j;

    #pragma omp parallel for shared(A,B) private(i,j)
    for (i = 0; i < N; i++) /* Initialization */
        for (j = 0; j < N; j++){
            A[i][j] = i*N+j; B[i][j] = i+j;
        }

    #pragma omp parallel for shared(A,B,C) private(i,j)
    for (i = 0; i < N; i++) /* Computing */
        for (j = 0; j < N; j++){
            C[i][j] = A[i][j] + B[i][j];
        }
    return 0;
}
```

# OpenMP Programming summary

## Programming

- Incrementally parallelization
- Multi-task/worksharing
- Variable management (shared/private, initial/final values, etc)
- Threads synchronization

## Issues

- Reliability of results
- Deadlocks

## omp parallel clauses

Clause	Do
<code>num_threads(nt)</code>	the number of threads to use
<code>private(list)</code>	declare private variable to each thread
<code>shared(list)</code>	variables shared by the threads in the team
<code>firstprivate(list)</code>	private + pre-initialization
<code>lastprivate(list)</code>	private + the final value of variable is the last one in the last iteration
<code>reduction(operator:list)</code>	Perform a reduction on all scalar variables in list using the specified operator

## omp for clauses

Clause	Do
<code>private(list)</code>	declare private variables to each thread
<code>firstprivate(list)</code>	<code>private</code> + pre-initialization
<code>lastprivate(list)</code>	<code>private</code> + the final value of variable is the last one in the last iteration
<code>reduction(operator:list)</code>	Perform a reduction on all scalar variables in <code>list</code> using the specified operator
<code>schedule(type)</code>	Specify how iterations are divided ( <code>static</code> , <code>dynamic</code> , <code>guided</code> )
<code>nowait</code>	avoid the implicit barrier at the end of the <code>for-lo</code>

# Dot product $x^T y = \sum x_i y_i$

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"
int main(){
    int i,n=10000;
    double u[n], v[n], s,sx, ts, te;

    for (i=0; i<n; i++){
        u[i] = 1.0/3.0; v[i] = 1.0/3.0;
    }
    sx = (double) n/9.0;

    s = 0.0; /* put "#pragma omp single" before in parallel region */
    ts = omp_get_wtime();
    #pragma omp parallel for schedule(static) reduction(+:s)
    for (i=0; i<n; i++) s += u[i]*v[i];
    te = omp_get_wtime()-ts;

    printf("s-sx=%15.8 elapsed time = %f \n",s-sx,te);
    return 0;
}
```



# Jacobi's method

Solve

$$Ax = b$$

by the Jacobi iteration :

$$\begin{aligned} x_i^{(k+1)} &= \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1} a_{ij} x_j^{(k)} \right), \\ &= \frac{1}{a_{ii}} \left( b_i + a_{ii} x_i^{(k)} - \sum_{j=1}^n a_{ij} x_j^{(k)} \right) \end{aligned}$$

Convergence if

$$\| Ax^{(k)} - b \| < \varepsilon \| b \|$$

THANK YOU FOR YOUR ATTENTION