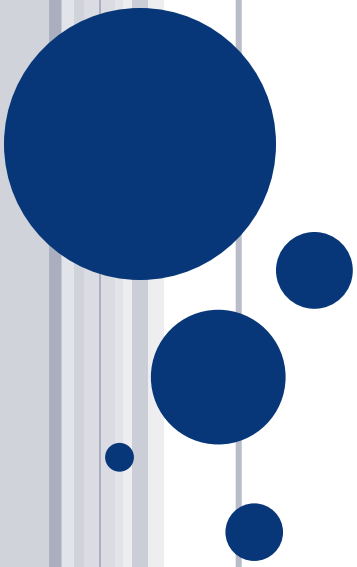


PARALLEL PROGRAMMING USING SHARED MEMORY

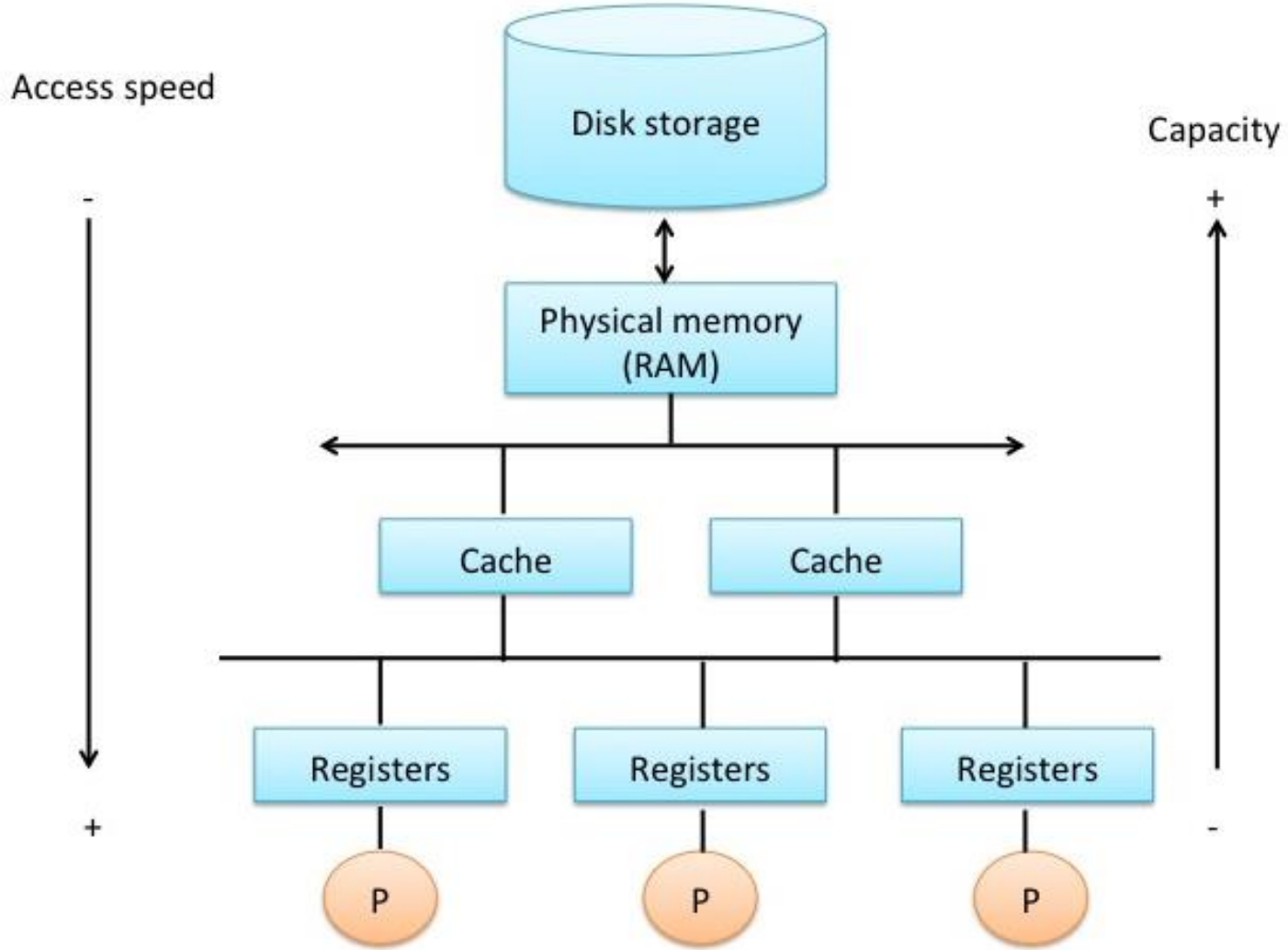
lidia.sanchez@unileon.es

Index

Index

INTRODUCTION

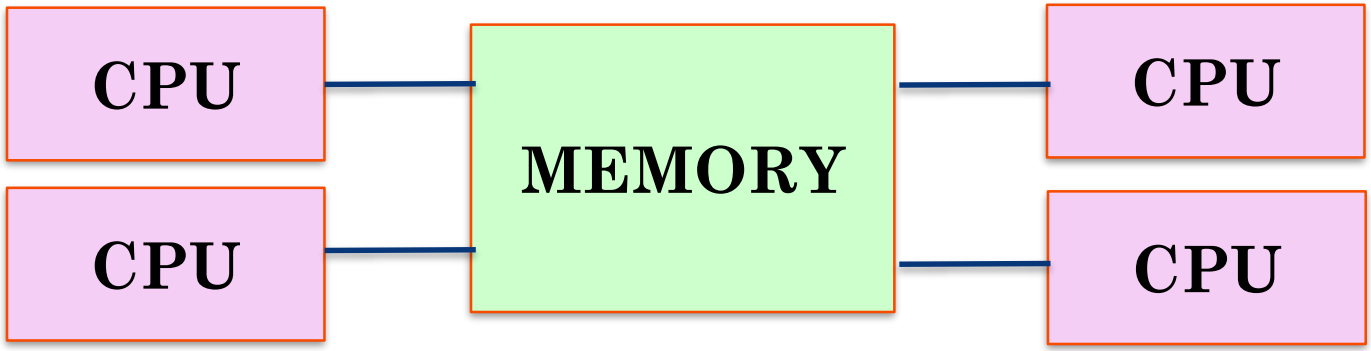
MEMORY HIERARCHY



Index

Shared Memory

ARQUITECTURE



Shared Memory

PROCESSORS WITH 1 ADDRESS SPACE

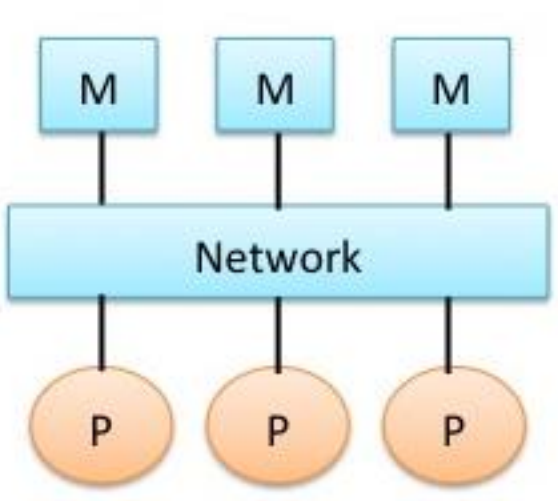
Distributed shared memory

- Physically separate memories
- Addressed as one logically shared address space
- Fast networks: low access times

Uniform Memory Access

- Constant access time

UMA Organization



Shared Memory

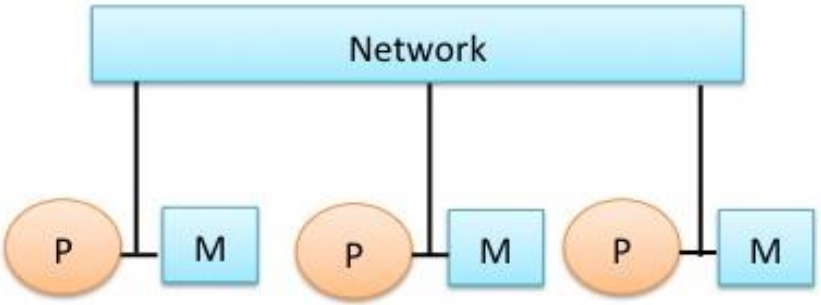
PROCESSORS WITH 1 ADDRESS SPACE

Distributed shared memory

- Physically separate memories
- Addressed as one logically shared address space
- Fast networks: low access times

Non-Uniform Memory Access

- Access time depends on the address



NUMA Organization

Index

Shared Memory Programming

Thread of execution

- Smallest sequence of program instructions managed independently by a OS

Multithread program

- If the main program has several independent procedures,they can be executed at once

Shared Memory Programming

Threads of the same process share

- Process instructions
- Data
- Files
- etc

That means…

- If a thread changes a resource, all of them see it
- Pointers to the same address point to the same data
- Synchronization is required (read/write)

Shared Memory Programming

PROBLEMS TO DEAL WITH

Race conditions

- Multithread program behaviour depends on the order in which the threads are executed

```
i=0;//Sequential
i=i+1;//Parallel
```

Shared variable i=0;
Thread1 reads i (i = 0)
Thread1 adds 1 to i (i = 1)
Thread1 stores i (i = 1)
Thread2 reads i (i = 1)
Thread2 adds 1 to i (i = 2)
Thread2 stores i (i = 2)

Shared variable i=0;
Thread1 reads i (i = 0)
Thread2 reads i (i = 0)
Thread1 adds 1 to i (i = 1)
Thread2 adds 1 to i (i = 1)
Thread1 stores i (i = 1)
Thread2 stores i (i = 1)

Shared Memory Programming

PROBLEMS TO DEAL WITH

Race conditions

- Multithread program behaviour depends on the order in which the threads are executed
- If it is an **atomic action**, it can not be executed concurrently

Shared variable i=0;
Thread1 reads i (i = 0)
Thread1 adds 1 to i (i = 1)
Thread1 stores i (i = 1)
Thread2 reads i (i = 1)
Thread2 adds 1 to i (i = 2)
Thread2 stores i (i = 2)

Shared variable i=0;
Thread1 reads i (i = 0)
Thread2 reads i (i = 0)
Thread1 adds 1 to i (i = 1)
Thread2 adds 1 to i (i = 1)
Thread1 stores i (i = 1)
Thread2 stores i (i = 1)

Shared Memory Programming

PROBLEMS TO DEAL WITH

Critical sections

- **Piece of code** that accesses a shared resource that cannot be accessed by more than one thread of execution
- A synchronization mechanism is required (semaphore)

```
if (var1 > var2)
      var2 = var2 – var1;
```

Shared Memory Programming

PROBLEMS TO DEAL WITH

Barrier synchronization

- Thread synchronization required before executing more instructions

1. Parallel evaluation of vector components (threads)
2. Thread synchronization (barrier)
3. Compute some operations with the vector

Shared Memory Programming

PROBLEMS TO DEAL WITH

Thread safety

- Shared data are manipulated in a manner that guarantees safe execution by multiple threads at the same time

Index

OpenMP

Application Program Interface (API)

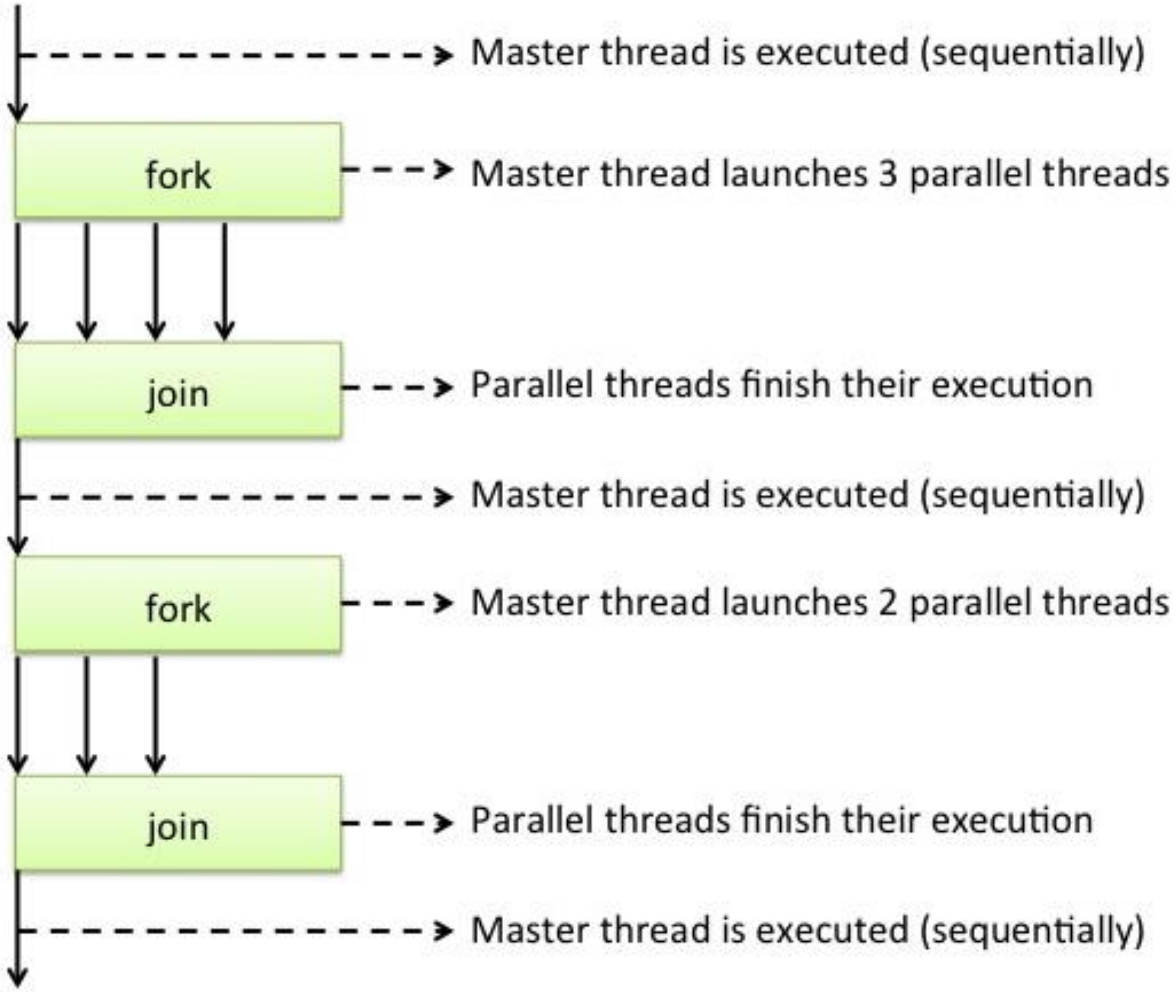
- Use to design parallel multithread programs for shared memory
- Portable: C, C++, Fortran…
- Standard

It is formed by

- Directives
- Runtime library routines
- Environment variables

OpenMP

PROGRAMMING MODEL



OpenMP

PROGRAMMING MODEL

Use directives

- Directives are included in the C/C++/Fortran code
- Directives specify where the threads are launched and joint
- A sequential compiler will ignore the directives and produce normal sequential code
- An OpenMP compiler will produce parallel code

OpenMP

DIRECTIVE FORMAT

Including omp library

- All OpenMP programs have to include the library:

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

OpenMP

DIRECTIVE FORMAT

Directive syntax

- We just can use a directive name in each directive
- The directive is applied at least to the following instruction (can be a block)
- Long directives are broken into different lines by \

  #pragma omp directive-name [clauses,…] new-line

OpenMP

DIRECTIVE FORMAT

Directive syntax

- We just can use a directive name in each directive
- The directive is applied at least to the following instruction (can be a block)
- Long directives are broken into different lines by \

#pragma omp directive-name [clauses,…] new-line

Required for all OpenMP directives

OpenMP

DIRECTIVE FORMAT

Directive syntax

- We just can use a directive name in each directive
- The directive is applied at least to the following instruction (can be a block)
- Long directives are broken into different lines by \

#pragma omp directive-name clauses,…] new-line

A valid directive name is required

OpenMP

DIRECTIVE FORMAT

Directive syntax

- We just can use a directive name in each directive
- The directive is applied at least to the following instruction (can be a block)
- Long directives are broken into different lines by \

#pragma omp directive-name [clauses,...] new-line

Optional. They can be in any order and can be repeated

OpenMP

DIRECTIVE FORMAT

Directive syntax

- We just can use a directive name in each directive
- The directive is applied at least to the following instruction (can be a block)
- Long directives are broken into different lines by \

#pragma omp directive-name [clauses,… new-line

It is mandatory to separate the pragma line from the parallel block

OpenMP

EXAMPLE: HOLA.C

```
#include <omp.h>
#include <stdio.h>
int main(){
        int nthreads, tid;
        #pragma omp parallel private(tid) //Create a group of threads
        {
                //Each thread has a copy of the variable tid
                tid=omp_get_thread_num();   //Get the thread id
                nthreads=omp_get_num_threads();    //Thread number
                printf("Hello from thread %d of %d threads\n", tid,
                        nthreads);
        }
        //All threads ends and only exists the master thread (tid==0)
        return 0;
}
```

OpenMP

EXAMPLE: HOLA.C

```
#include <omp.h>
#include <stdio.h>
int main(){
      int nthreads, tid;
      #pragma omp parallel private(tid) //Create a group of threads
      {
            //Each thread has a copy of the variable tid
            tid=omp_get_thread_num();   //Get the thread id
            nthreads=omp_get_num_threads();    //Thread number
            printf("Hello from thread %d of %d threads\n", tid,
                  nthreads);
      }
      //All threads ends and only exists the master thread (tid==0)
      return 0;
}
```

Each thread has a copy of its identifier

OpenMP

COMPILING

```
[username@frontend1 ~]$
        gcc -O2 -fopenmp -o output source.c
```

EXECUTING

```
[username@frontend1 ~]$ ./output
```

OpenMP

OUTPUT

```
Hello from thread 3 of 8 threads
Hello from thread 7 of 8 threads
Hello from thread 0 of 8 threads
Hello from thread 5 of 8 threads
Hello from thread 6 of 8 threads
Hello from thread 4 of 8 threads
Hello from thread 1 of 8 threads
Hello from thread 2 of 8 threads
```

OpenMP

SET NUMBER OF THREADS

```
[username@frontend1 ~]$
        export OMP_NUM_THREADS=4
```

EXECUTING

```
[username@frontend1 ~]$ ./output
```

OpenMP

OUTPUT

```
Hello from thread 0 of 4 threads
Hello from thread 1 of 4 threads
Hello from thread 2 of 4 threads
Hello from thread 3 of 4 threads
```

OpenMP

DIRECTIVES

Clauses

- Conditional parallelization if(expression):

  #pragma omp parallel if(variable>0)

OpenMP

CLAUSE IF - EXAMPLE

```
#include <omp.h>
#include <stdio.h>
int main(){
        int nthreads, tid, i=7;
        #pragma omp parallel private(tid) if (i>5)
        {
                tid=omp_get_thread_num();
                nthreads=omp_get_num_threads();
                printf("Hello from thread %d of %d threads\n", tid,
                        nthreads);
        }
        return 0;
}
```

OpenMP

CLAUSE IF - EXAMPLE

```
[username@frontend1 ~]$ ./outputIF_true
Hello from thread 5 of 8 threads
Hello from thread 0 of 8 threads
Hello from thread 2 of 8 threads
Hello from thread 3 of 8 threads
Hello from thread 4 of 8 threads
Hello from thread 1 of 8 threads
Hello from thread 6 of 8 threads
Hello from thread 7 of 8 threads
```

OpenMP

CLAUSE IF - EXAMPLE

```
#include <omp.h>
#include <stdio.h>
int main(){
        int nthreads, tid, i=2;
        #pragma omp parallel private(tid) if (i>5)
        {
                tid=omp_get_thread_num();
                nthreads=omp_get_num_threads();
                printf("Hello from thread %d of %d threads\n", tid,
                        nthreads);
        }
        return 0;
}
```

OpenMP

CLAUSE IF - EXAMPLE

```
[username@frontend1 ~]$ ./outputIF_false
Hello from thread 0 of 1 threads
```

OpenMP

DIRECTIVES

Clauses

- Concurrency degree *num_threads(expression)*:

  #pragma omp parallel num_threads(5)

OpenMP

Clause num_threads - Example

```
#include <omp.h>
#include <stdio.h>
int main(){
      int nthreads, tid, i=2;
      #pragma omp parallel num_threads(5) private(tid)  if (i==2)
      {
            tid=omp_get_thread_num();
            nthreads=omp_get_num_threads();
            printf("Hello from thread %d of %d threads\n", tid,
                  nthreads);
      }
      return 0;
}
```

OpenMP

CLAUSE NUM_THREADS - EXAMPLE

```
[username@frontend1 ~]$ ./NumberThreads
Hello from thread 3 of 5 threads
Hello from thread 0 of 5 threads
Hello from thread 1 of 5 threads
Hello from thread 2 of 5 threads
Hello from thread 4 of 5 threads
```

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *private(list-of-variables)*: Each thread has a copy of its identifier #pragma omp parallel private(var1, var2)
  - *firstprivate(list-of-variables)*: Each thread has a copy of its identifier and its initial value is the previous one
  - *lastprivate(list-of-variables)*: Each thread has a copy of its identifier and in the last iteration the value is copied to the original variable (i.e. Last thread in a *for* loop stores its value in the variable)

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *copyin(list-of-variables)*: =firstprivate.
  - *shared(list-of-variables)*: Threads share variables. They are the same (check the right use)
  - *default(shared/private/none)*: it notes the default type of variables. If it is none, the type of all variables has to be specified

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *reduction(operator:list-of-variables)*: combines the variables of the list using the specified operator in just one variable in the master thread. Operators: +, *, -, &, |, ^, &&, ||

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *schedule(type[,size])*: assigns iterations to threads.
    The type can be:
    - *static*: iteration space is divided into blocks of the specified size and are assigned to threads using round-robin. If there is no size, iteration space is divided into as many blocks as threads and the assignment is 1 block, 1 thread.
    - *dynamic*:
    - *guided:*
    - *runtime:*

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *schedule(type[,size])*: assigns iterations to threads.
    The type can be:
    - *static:*
    - *dynamic:* iteration space is divided into blocks of the specified size and are assigned to threads as soon as they finish. It avoids load unbalance.
    - *guided:*
    - *runtime:*

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *schedule(type[,size])*: assigns iterations to threads. The type can be:
    - *static:*
    - *dynamic:*
    - *guided:* the block size is reduced as long as the iterations that have not been distributed decrease. The minimun block size is the size specified in the clause. Default size is 1.
    - *runtime:*

OpenMP

DIRECTIVES

Clauses

- Data management:
  - *schedule(type[,size])*: assigns iterations to threads.
    The type can be:
    - *static:*
    - *dynamic:*
    - *guided:*
    - *runtime:* the assignment is postponed until the execution time. OMP_SCHEDULE has the type of assignment. The size is not specified

OpenMP

OMP LIBRARY

Functions

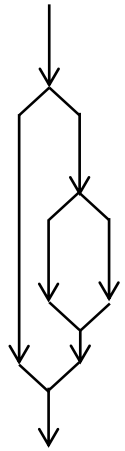
- *void omp_set_num_threads (int num_threads)* Specifies the number of threads. It has to be called from the sequential parts of the code.
- *int omp_get_num_threads (void)* Gives the number of current threads.
- *int omp_get_max_threads(void)* Maximum number of threads.
- *int omp_get_thread_num(void)* Thread number (id)
- *int omp_get_num_procs(void)* Number of processors
- *void omp_set_nested(int nested)* Enables or disables nested paralellism (deprecated use *void omp_set_max_active_levels(int max_levels)*

OpenMP

ENVIRONMENTAL VARIABLES

Variables

- *OMP_NUM_THREADS* Specifies the number of threads
- *OMP_SCHEDULE* Type of assignment in a parallel loop (for runtime schedule)
- *OMP_DYNAMIC* (TRUE/FALSE) Dynamic adjustment of the number of threads used in the parallel regions
- *OMP_NESTED* (TRUE/FALSE) Nested parallelism



EXAMPLE

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

int main(){
      int nthreads, tid;
      printf("Set 4 threads\n");
      //Set number of threads to 4
      //Obtain the number of threads
      printf("Number of threads = %d\n", nthreads);

      #pragma omp parallel private(tid)
      {
            //Get the id of the thread
            printf("Hello from thread = %d\n",tid);
            if (….){ //If I am the master
                  //Obtain the number of threads
                  printf("Number of threads = %d\n", nthreads);
            }
      }
```

EXAMPLE

```
        printf("Set 3 threads\n");
        //Set number of threads to 3
        //Obtain the number of threads
        printf(" Number of threads = %d\n",nthreads);
        #pragma omp parallel private(tid)
        {
                //Get the id of the thread
                printf("Hello from thread = %d\n",tid);
                if (….){ //If I am the master
                        //Obtain the number of threads
                        printf(" Number of threads = %d\n", nthreads);
                }
        }
}
```

OpenMP

EXAMPLE OUTPUT

```
Set 4 threads
Number of threads = 1
Hello from thread = 3
Hello from thread = 1
Hello from thread = 2
Hello from thread = 0
Number of threads = 4
Set 3 threads
Number of threads = 1
Hello from thread = 1
Hello from thread = 2
Number of threads = 3
Hello from thread = 0
```

OpenMP

PARALLEL LOOPS

Directive for

- Loop iterations after directive *for* are executed in parallel.
- Threads have to be previously started (Directive *parallel*).
- Loop has to be cannonical:

  for (i=INICIO; i OP FINAL; INCREMENTO)
- Iterations have to be independent
- Syntax:

```
#pragma omp for [clauses…]
for-loop
```

OpenMP

PARALLEL LOOPS

Directive for

- Clauses:
  - schedule (type [,size])
  - ordered
  - private (list-of-variables)
  - shared (list-of-variables)
  - firstprivate (list-of-variables)
  - lastprivate (list-of-variables)
  - reduction (operator: list-of-variables)
  - nowait

EXAMPLE

```
#include <omp.h>
#include <stdio.h>
#define CHUNKSIZE 2
#define N 10

int main (){
        int i, chunk,nthreads, tid;
        int a[N],b[N],c[N];

        for (i=0;i<N;i++)
                a[i]=b[i]=i*1.0;
        chunk=CHUNKSIZE;
        //Variables a, b, c, chunk are shared
        //Variables i, tid are private in each thread

        .....
         {
                        //Static schedule for the iterations
                        //Block size is fixed = chunk
                        .....
                        for (i=0;i<N;i++){
                                //Get ID of thread
                                //Get number of threads
                                c[i]=a[i]+b[i];
                                printf("Thread %d, of %d threads, computes
                                        iteration i = %d\n", tid, nthreads, i);
                        }
        } //Parallel region ends
}
```

OpenMP

Directive for

EXAMPLE OUTPUT

```
Thread 0, of 8 threads, computes iteration i = 0
Thread 0, of 8 threads, computes iteration i = 1
Thread 3, of 8 threads, computes iteration i = 6
Thread 3, of 8 threads, computes iteration i = 7
Thread 1, of 8 threads, computes iteration i = 2
Thread 1, of 8 threads, computes iteration i = 3
Thread 4, of 8 threads, computes iteration i = 8
Thread 4, of 8 threads, computes iteration i = 9
Thread 2, of 8 threads, computes iteration i = 4
Thread 2, of 8 threads, computes iteration i = 5
```

EXAMPLE – USING NOWAIT

```
#include <omp.h>
#include <stdio.h>
#define CHUNKSIZE 2
#define N 10

int main (){
        int i, chunk,nthreads, tid;
        int a[N],b[N],c[N];

        for (i=0;i<N;i++)
                a[i]=b[i]=i*1.0;
        chunk=CHUNKSIZE;
        //Variables a, b, c, chunk are shared
        //Variables i, tid are private in each thread


        .....
         {
                        //Static schedule for the iterations
                        //Block size is fixed = chunk
                        .....
                        for (i=0;i<N;i++){
                                //Get ID of thread
                                //Get number of threads
                                c[i]=a[i]+b[i];
                                printf("Thread %d, of %d threads, computes
                                        iteration i = %d\n", tid, nthreads, i);
                        }
                        printf("Thread %d ends\n", tid);
        } // Parallel region ends
        printf("Parallel region ends");
}
```

OpenMP

EXAMPLE – WITHOUT NOWAIT

Directive for

```
Thread 0, of 8 threads, computes iteration i = 0
Thread 0, of 8 threads, computes iteration i = 1
Thread 1, of 8 threads, computes iteration i = 2
Thread 1, of 8 threads, computes iteration i = 3
Thread 4, of 8 threads, computes iteration i = 8
Thread 4, of 8 threads, computes iteration i = 9
Thread 2, of 8 threads, computes iteration i = 4
Thread 2, of 8 threads, computes iteration i = 5
Thread 3, of 8 threads, computes iteration i = 6
Thread 3, of 8 threads, computes iteration i = 7
Thread 1 ends
Thread 0 ends
Thread 3 ends
Thread 4 ends
Thread 0 ends
Thread 0 ends
Thread 2 ends
Thread 0 ends
Parallel region ends
```

OpenMP

EXAMPLE – USING NOWAIT

Directive for

```
Thread 3, of 8 threads, computes iteration i = 6
Thread 3, of 8 threads, computes iteration i = 7
Thread 3 ends
Thread 2, of 8 threads, computes iteration i = 4
Thread 2, of 8 threads, computes iteration i = 5
Thread 2 ends
Thread 1, of 8 threads, computes iteration i = 2
Thread 1, of 8 threads, computes iteration i = 3
Thread 4, of 8 threads, computes iteration i = 8
Thread 4, of 8 threads, computes iteration i = 9
Thread 4 ends
Thread 0, of 8 threads, computes iteration i = 0
Thread 0 ends
Thread 0, of 8 threads, computes iteration i = 1
Thread 1 ends
Thread 0 ends
Thread 0 ends
Thread 0 ends
Parallel region ends
```

OpenMP

PARALLEL REGIONS

Directive section

- Piece of code that will be distributed
- Threads have to be previously started (Directive *parallel*).
- There is a barrier at the end of the directive unless *nowait* is used
- If num_threads > num_sections some threads will not work
- If num_threads < num_sections the implementation will decide the distribution

OpenMP

PARALLEL REGIONS

Directive section

```
#pragma omp sections [clauses …]
{
    #pragma omp section
        block
    #pragma omp section
        block
}
```

Clauses:

- private (list-of-variables)
- firstprivate (list-of-variables)
- lastprivate (list-of-variables)
- reduction (operator: list-of-variables)
- nowait

EXAMPLE

```
#include <omp.h>
#include <stdio.h>
int main(){
        int nthreads, tid;
        //Variable tid is private to each thread
        …..
        {
                #pragma omp sections
                {
                        //A section

                                //Get ID of thread
                                //Get number of threads
                                printf("Thread %d, of %d, computes section 1\n", tid, nthreads);

                        //Another section
                                //Get ID of thread
                                //Get number of threads
                                printf(" Thread %d, of %d, computes section 2\n", tid, nthreads);

                        //Another section

                                //Get ID of thread
                                //Get number of threads
                                printf(" Thread %d, of %d, computes section 3\n", tid, nthreads);

                        //Another section

                                //Get ID of thread
                                //Get number of threads
                                printf(" Thread %d, of %d, computes section 4\n", tid, nthreads);

                } //Sections' end
        } //Parallel region ends
}
```

OpenMP

Directive sections

EXAMPLE OUTPUT

```
Thread 3, of 8, computes section 3
Thread 2, of 8, computes section 2
Thread 4, of 8, computes section 1
Thread 1, of 8, computes section 4
```

OpenMP

DIRECTIVE COMBINATION

Directive parallel-for

- Regions with just one *for* directive.
- By default, iterations are distributed into blocks of the same size for each thread.
- Syntax

```
#pragma omp parallel for [clauses …]
for-loop
```

- Clauses: any of the parallel or for directives.

OpenMP

EXAMPLE

Directive parallel-for

- Matrix * Vector multiplication

EXAMPLE

```
#include <omp.h>
…
#define N 4
#define M 4

int main(){
        int i,j, nthreads, tid, n, m, sum, a[M],c[N],b[M][N];
        srand(time(NULL));
        m=M;
        n=N;
        for (i=0; i<M; i++) {
                for (j=0; j<N; j++) {
                        b[i][j]=rand()%100;
                }

        }
        for (i=0; i<N; i++)
                c[i]=rand()%100;

…
```

EXAMPLE

```
	//Variables a,b,c,m,n,nthreads are shared
	//Variables i,j,sum,tid are private

	.....
	for (i=0;i<m;i++){
		//Get ID of thread
		//Get number of threads
		sum=0;
		for(j=0;j<n;j++)
			sum+=b[i][j]*c[j];
		a[i]=sum;
		printf(”Thread %d, of %d threads, computes iteration i=%d\n", tid,
nthreads,i);
	}
	for (i=0; i<M; i++) {
		printf("a[%d]=%d\n",i,a[i]);
	}
}
```

OpenMP

EXAMPLE OUTPUT

Directive parallel-for

```
Thread 0, of 8 threads, computes iteration i=0
Thread 3, of 8 threads, computes iteration i=3
Thread 1, of 8 threads, computes iteration i=1
Thread 2, of 8 threads, computes iteration i=2
a[0]=5723
a[1]=16593
a[2]=11369
a[3]=8093
```

OpenMP

EXERCISE

Directive parallel-for

- PI computation

$$\pi = \int_0^1 \frac{4}{1+x^2}$$

Adding areas of n rectangles:

$$\pi = \frac{1}{n}\sum_{i=1}^{n} \frac{4}{1+\left(\frac{i-0.5}{n}\right)^2}$$

OpenMP

EXERCISE

Directive parallel-for

- Adding n random numbers stored in a vector

OpenMP

EXERCISE

Directive parallel-for

- Matrix multiplication

OpenMP

DIRECTIVE COMBINATION

Directive parallel-sections

- Regions with just one *sections* directive.
- Syntax

```
#pragma omp parallel sections [clauses …]
for-loop
```

- Clauses: any of the parallel or sections directives.

OpenMP

EXERCISE

Directive parallel-sections

- Change the code of EjemploSectionsVector.c to use the directive parallel sections.

OpenMP

EXERCISE

Directive parallel-sections

- Change the code of EjemploSectionsVector.c to use the directive parallel sections.
- Change this for loop as follows:

```
for (i=0;i<THREADS;i++){
        sumaparcial(&a[i*n/THREADS],n/THREADS);
        printf("Soy el thread %d",omp_get_thread_num());
}
```

- What happens?
- What do we need to do?

OpenMP

EXERCISE

Directive parallel-sections

- Change the code of EjemploSectionsVector.c to use the directive parallel sections.
- Add a line in the *for* loop as follows:

```
for (i=0;i<THREADS;i++){
        sumaparcial(&a[i*n/THREADS],n/THREADS);
        printf("Soy el thread %d",omp_get_thread_num());
}
```

- What happens?
- What do we need to do?
- Execute export OMP_NESTED=TRUE and try again

OpenMP

API CALLS

Other useful functions

- *double omp_get_wtime()* Returns the number of seconds since a point in the past