

OpenMP 11

Sistemas Operativos y Distribuidos

Prof. Javier Echaiz
D.C.I.C. – U.N.S.
<http://cs.uns.edu.ar/~jechaiz>
je@cs.uns.edu.ar

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Programación de aplicaciones paralelas

- Como sabemos, los sistemas paralelos MIMD presentan dos arquitecturas diferenciadas: **memoria compartida y memoria distribuida.**
- ▶ El modelo de memoria utilizado hace que la programación de aplicaciones paralelas para cada caso sea esencialmente diferente.

2

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Programación de aplicaciones paralelas

- Para los sistemas de memoria distribuida (MPP), el “estándar” de programación, mediante pasaje de mensajes, es MPI.
- Para los sistemas de memoria compartida, básicamente SMP, la opción a considerar es OpenMP.
- Otras opciones:
UPC (Unified Parallel C).
shmem (Cray).

3

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Modelos de programación paralela

EARTH SIMULATOR
Development Environment for Application

Level	Hardware	Unit	Max	Memory	Choice of Programming
1	AP	Vector Register Length	256		Automatic Vectorization
2	Intra-Node	AP	8	Shared	OpenMP Microtasking MPI2 HPF2/JA
3	Inter-Nodes	Node	640	distributed	MPI2 HPF2/JA

OS : SUPER-UX

4

OpenMP

Una pequeña introducción

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

OpenMP (introd.)

- OpenMP trata de ofrecer un estándar para programar aplicaciones paralelas en sistemas de memoria compartida, que sea portable, que permita paralelismo incremental , y que sea independiente del hardware que lo soporte.
- Participan en su desarrollo los fabricantes más importantes: HP-Compaq, IBM, SUN, SG, ...

6

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

OpenMP (introd.)

- No se trata de un nuevo lenguaje de programación, sino de un API (*application programming interface*) formado por:
 - directivas al compilador (C) `#pragma omp <directiva>`
 - unas pocas funciones de librería
 - algunas variables de entorno

7

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

OpenMP (introd.)

- El modelo de programación paralela que aplica OpenMP es Fork - Join.
- En un determinado momento, el thread "master" genera p threads que se ejecutan en paralelo.

8

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

OpenMP (introd.)

- Los threads creados por el thread master van a ejecutar todos la misma copia de código. Cada thread lleva un identificador.
- La diferenciación de las tareas a ejecutar será:
 - mediante un modelo SPMD (`if id == ...`).
 - mediante constructores específicos de reparto de tareas (Work Sharing).

9

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

OpenMP (introd.)

En resumen, partiendo de un programa serie se obtiene un programa paralelo OpenMP:

- añadiendo directivas que especifican una región paralela (código replicado), reparto de tareas (específicas para cada thread), o sincronización entre threads.
- utilizando unas pocas funciones de librería (`include <omp.h>`) para gestionar los threads o para sincronizarlos.

10

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

un ejemplo

```
main () {
#pragma omp parallel private(tid)
{
  tid = omp_get_thread_num();
  printf ("thread %d en marcha \n", tid);
  #pragma omp for schedule(static) reduction(+:B)
  for (i=0; i<1000; i++)
  {
    A[i] = A[i] + 1;
    B = B + A[i];
  }
  if (tid==0) printf(" B = %d \n", B);
}
}
```

11

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

ideas básicas

- Procesos paralelos (threads).
- REGIONES PARALELAS. Ámbito de las variables.
- REPARTO DE TAREAS. Bucles for. Planificación de iteraciones. Sections / Single / ...
- SINCRONIZACIÓN: secciones críticas, locks (cerrojos), barreras.

12

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

ideas básicas

- 0 Número de procesos
- estático, una variable de entorno:
 - > export OMP_NUM_THREADS = 10
- dinámico, mediante una función:
 - omp_set_num_threads(10);

13

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

región paralela

- 1 REGIÓN PARALELA (parallel regions)
 - #pragma omp parallel (cláusulas)
- Una región paralela es un trozo de código que se va a repetir en todos los threads.
- Las variables de una región paralela pueden ser compartidas (de todos los threads, shared) o privadas (diferentes para cada thread).
- El ámbito de las variables se define mediante cláusulas específicas.

14

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

región paralela

Un ejemplo sencillo:

```
#include <stdio.h>
#include <omp.h>
#define N 12
int i, tid, A[N];
main () {
    omp_set_num_threads(4);
    #pragma omp parallel private(tid) shared(A)
    { tid = omp_get_thread_num();
      printf("Thread %d en marcha\n", tid);
      A[tid] = 10 + tid;
      printf("El thread %d ha terminado\n", tid);
    }
    for (i=0; i<N; i++) printf("A[%d] = %d\n", i, A[i]);
}
```

15

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

reparto de tareas: bucles for

- 2 REPARTO DE TAREAS: bucles
- Los bucles son uno de los puntos de los que extraer paralelismo de manera "sencilla" (paralelismo de grano fino).
- Obviamente, la simple replicación de código no es suficiente. Por ejemplo,

```
#pragma omp parallel shared(A) private(i)
{ for (i=0; i<100; i++)
  A[i] = A[i] + 1;
}
```

16

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

bucles for

- Tendríamos que hacer algo así:

```
#pragma omp parallel shared(A)
private(tid, ini, nth, fin, i)
{ tid = omp_get_thread_num();
  nth = omp_get_num_threads();
  ini = tid * 100 / nth;
  fin = (tid+1) * 100 / nth;
  for (i=ini; i<fin; i++) A[i] = A[i] + 1;
}
```

17

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

bucles for

```
#pragma omp parallel
{
  #pragma omp for (cláusulas)
  for (i=0; i<100; i++)
    A[i] = A[i] + 1;
}
```

18

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

bucles for

```

for (i=0; i<N; i++)
  for (j=0; j<M; j++)
  {
    X = B[i][j] * B[i][j];
    A[i][j] = A[i][j] + X;
    C[i][j] = X * 2 + 1;
  }
    
```

⇒

```

#pragma omp parallel for
private (j, X)
for (i=0; i<N; i++)
  for (j=0; j<M; j++)
  {
    X = B[i][j] * B[i][j];
    A[i][j] = A[i][j] + X;
    C[i][j] = X * 2 + 1;
  }
    
```

Estamos paralelizando el bucle externo: los threads ejecutan el bucle interno. Paralelismo de grano "medio". X y j deben declararse como privadas!

19

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

bucles for

```

for (i=0; i<N; i++)
  for (j=0; j<M; j++)
  {
    X = B[i][j] * B[i][j];
    A[i][j] = A[i][j] + X;
    C[i][j] = X * 2 + 1;
  }
    
```

⇒

```

for (i=0; i<N; i++)
  #pragma omp parallel for
  private (X)
  for (j=0; j<M; j++)
  {
    X = B[i][j] * B[i][j];
    A[i][j] = A[i][j] + X;
    C[i][j] = X * 2 + 1;
  }
    
```

Estamos paralelizando el bucle interno: el paralelismo es de grano fino. X debe declararse como privada.

20

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

planificación de las iteraciones

- ¿Cómo se reparten las iteraciones de un bucle entre los threads?
- Puesto que el **for** termina con una **barrera**, si la carga de los threads está mal equilibrada tendremos una pérdida (notable) de eficiencia.
- ▶ La cláusula **schedule** permite definir diferentes estrategias de reparto, tanto estáticas como dinámicas.

21

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

planificación de las iteraciones

- Ejemplo

```

#pragma omp parallel for shared(A) schedule(static, 4)
{ for (i=0; i<100; i++)
    A[i] = A[i] + 1;
}
    
```

Recordar

- estático: menos costo / mejor localidad datos
- dinámico: más costo / carga más equilibrada

22

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

reparto de tareas

- Sections: tareas diferentes

```

#pragma omp parallel sections
{clausulas}
{
  #pragma omp section
  fun1();
  #pragma omp section
  fun2();
  #pragma omp section
  fun3();
}
    
```

▶ **Single**: una tarea específica para un único thread dentro de una región paralela

23

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

- 3 SINCRONIZACIÓN
- Cuando no pueden eliminarse las dependencias de datos entre los threads, entonces es necesario sincronizar su ejecución.

OpenMP proporciona los mecanismos de sincronización más habituales: exclusión mutua y sincronización por eventos.

24

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

a. Secciones Críticas

```

MAX = -infinito;
MIN = infinito;

for (i=0; i<N; i++)
{
  if (A[i] > MAX) MAX = A[i]
  if (A[i] < MIN) MIN = A[i]
}
    
```

```

MAX = -infinito;
MIN = infinito;
#pragma omp parallel for
for (i=0; i<N; i++)
{
  #pragma omp critical (M1)
  { if (A[i] > MAX) MAX = A[i] }
  #pragma omp critical (M2)
  { if (A[i] < MIN) MIN = A[i] }
}
    
```

25

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

- atomic

Una sección crítica especial, para una operación simple de tipo RMW.
Por ejemplo:

```

...
#pragma omp atomic
  X = X + 1;
...
    
```

26

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

b. Funciones con locks (cerrojos)

- omp_set_lock (&C)
toma el lock o espera a que esté libre.
- omp_unset_lock (&C)
libera el lock.
- omp_test_lock (&C)
testea el valor del lock; devuelve T/F.

27

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

- ejemplo

```

#pragma omp parallel private(mi_it)
{
  omp_set_lock(&C1);
  mi_it = i;
  i = i + 1;
  omp_unset_lock(&C1);
  while (mi_it < N)
  {
    A[mi_it] = A[mi_it] + 1;
    omp_set_lock(&C1);
    mi_it = i;
    i = i + 1;
    omp_unset_lock(&C1);
  }
}
    
```

28

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

sincronización de los threads

c. Sincronización global: barreras

```

#pragma omp parallel private(tid)
{
  tid = omp_get_thread_num();
  A[tid] = func1(tid);
  #pragma omp barrier
  #pragma omp for
  for (i=0; i<N; i++) B[i] = func2(A, i);
  #pragma omp for nowait
  for (i=0; i<N; i++) C[i] = func3(A, B, i);
  A[tid] = func4(tid);
}
    
```

29

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Un ejemplo: cálculo de pi

```

static long num_iter = 100000;
double paso;
void main ()
{
  int i;
  double x, pi, sum = 0.0;
  paso = 1.0 / (double) num_iter;

  for (i = 1; i <= num_iter; i++)
  {
    x = (i - 0.5) * paso;
    sum = sum + 4.0 / (1.0 + x*x);
  }
  pi = sum * paso;
}
    
```

30

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Solución mediante threads: Win32 API

```

#include <windows.h>
#define NUM_THREADS 8
HANDLE thread_handles[NUM_THREADS];
CRITICAL_SECTION hUpdateMutex;
static long num_iter = 100000;
double paso;
double global_sum = 0.0;
void Pi (void *arg)
{
    int i, start;
    double x, sum = 0.0;
    start = *(int *) arg;
    paso = 1.0 / (double) num_iter;
    for (i = start; i <= num_iter; i = i + NUM_THREADS)
    {
        x = (i - 0.5) * paso;
        sum = sum + 4.0 / (1.0 + x*x);
    }
    EnterCriticalSection(&hUpdateMutex);
    global_sum += sum;
    LeaveCriticalSection(&hUpdateMutex);
}

```

```

void main ()
{
    double pi; int i;
    DWORD threadID;
    int threadArg[NUM_THREADS];
    for (i = 0; i < NUM_THREADS; i++)
        threadArg[i] = i + 1;
    InitializeCriticalSection(&hUpdateMutex);
    for (i = 0; i < NUM_THREADS; i++)
    {
        thread_handles[i] = CreateThread(0, 0,
        (LPTHREAD_START_ROUTINE) Pi,
        &threadArg[i], 0, &threadID);
    }
    WaitForMultipleObjects(NUM_THREADS,
    thread_handles, TRUE, INFINITE);
    pi = global_sum * paso;
    printf (" pi = %f\n", pi);
}

```

El código crece!

31

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Solución: simplicidad

- Librerías de threads:
 - El programador tiene control sobre todo el proceso.
 - Cuidado: el programador debe(!) controlarlo todo.

En general, a mayor control, mayor complejidad para el programador. En muchas ocasiones es más útil un enfoque más sencillo.

32

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Alternativa 1 Ejemplo de una región paralela (programa SPMD)

```

#include <omp.h>
static long num_iter = 100000; double paso;
#define NUM_THREADS 8
void main ()
{
    int i; double x, pi, sum[NUM_THREADS];
    paso = 1.0 / (double) num_iter;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        double x; int id;
        id = omp_get_thread_num();
        for (i = id, sum[id] = 0.0; i < num_iter; i = i + NUM_THREADS)
        {
            x = (i - 0.5) * paso;
            sum[id] += 4.0 / (1.0 + x*x);
        }
    }
    for (i = 0, pi = 0.0; i < NUM_THREADS; i++) pi += sum[i] * paso;
}

```

SPMD:

Los threads ejecutan el mismo código; el identificador del thread permite elegir tareas específicas.

33

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Alternativa 2: Variables privadas y sección crítica

```

#include <omp.h>
static long num_iter = 100000; double paso;
#define NUM_THREADS 8
void main ()
{
    int i; double x, sum, pi=0.0;
    paso = 1.0 / (double) num_iter;
    omp_set_num_threads(NUM_THREADS)
    #pragma omp parallel private (x, sum)
    {
        id = omp_get_thread_num();
        for (i=id, sum=0.0; i < num_iter; i=i+NUM_THREADS) {
            x = (i - 0.5) * paso;
            sum += 4.0 / (1.0 + x*x);
        }
        #pragma omp critical
        pi += sum * paso
    }
}

```

No es necesario crear un array para almacenar sumas locales.

34

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

Solución OpenMP Parallel for con reduction

```

#include <omp.h>
static long num_iter = 100000; double paso;
#define NUM_THREADS 8
void main ()
{
    int i; double x, pi, sum = 0.0;
    paso = 1.0 / (double) num_iter;
    omp_set_num_threads(NUM_THREADS)
    #pragma omp parallel for reduction(+:sum) private(x)
    for (i = 1; i <= num_iter; i++)
    {
        x = (i - 0.5) * paso;
        sum = sum + 4.0 / (1.0 + x*x);
    }
    pi = sum * paso;
}

```

El código OpenMP es sencillo y "corto"

35

Sistemas Operativos y Distribuidos – OpenMP Prof. Javier Echaiz

MPI vs. OpenMP

- No existe la mejor aproximación para escribir códigos paralelos. Cada uno tiene sus ventajas y desventajas:
 - **MPI** - poderoso, general, y universal, compuesto de rutinas de paso de mensaje las cuales proveen un control muy fino sobre las comunicaciones, pero obliga al programador a operar en un relativo bajo nivel de abstracción.
 - **OpenMP** – conceptualmente es una sencilla aproximación para desarrollar códigos paralelos sobre máquinas de memoria compartida, pero no es aplicable a plataformas de memoria distribuida.

36

MPI vs. OpenMP

- MPI es más general y portable (plataformas, aunque no es eficiente en sistemas SMPs).
- La arquitectura y el problema a resolver frecuentemente hacen que la decisión sea suya!!!

37

Referencias

TEXTOS

- R. Chandra et al.
Parallel Programming in OpenMP
Morgan Kaufmann, 2001.

WEB

- www.openmp.org (especificación 2.0, software, ...).
- En mi página hay links a muchos otros recursos.

COMPILADORES

- pagos (++)
- libres: p. e., el compilador de C/C++ de Intel (hay +).

38

Bola de Cristal SD

- Clustering?
 - Si, seguirá siendo importante! (incluso bajará clustering a nivel de micros)
- Grid computing?
 - Algo de cuerda todavía, especialmente temas de Seguridad, Interoperabilidad y Aplicaciones.
- P2P
 - “De moda” gracias a emule, torrents, etc... queda bastante por hacer aquí.
- Confluencia P2P y Grid.
- Sensor Networks / Wireless.
- Etc. (haciendo que la bola de cristal sea 100% efectiva ☐).

39

Dado que esta es la Ultima Clase...

Nos Vemos!!!

```
#include <stdio.h>
#include <mpi.h>
void main (int argc, char *argv[] ) {
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Bye world from %d\n", rank);
    MPI_Finalize();
}
```

40

Gracias...



41