Sistemas Complejos en Máquinas Paralelas

Clase 1: OpenMP

Francisco García Eijó

Departamento de Computación - FCEyN
UBA

15 de Mayo del 2012

Memoria compartida

- Las mas conocidas son las máquinas tipo Symmetric Multiprocessor(SMP)

Memoria compartida

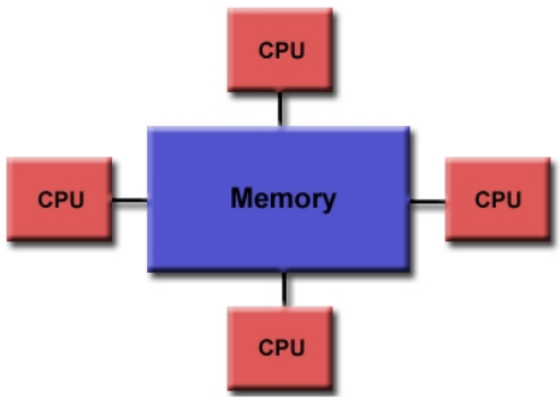
- Las mas conocidas son las máquinas tipo Symmetric Multiprocessor(SMP)
- Todos los procesadores son identicos entre si.

Memoria compartida

- Las mas conocidas son las máquinas tipo Symmetric Multiprocessor(SMP)
- Todos los procesadores son identicos entre si.
- Acceso uniforme a toda la memoria y mismo tiempo de acceso para llegar a cualquier parte del mapa de memoria.

Memoria compartida

- Las mas conocidas son las máquinas tipo Symmetric Multiprocessor(SMP)
- Todos los procesadores son identicos entre si.
- Acceso uniforme a toda la memoria y mismo tiempo de acceso para llegar a cualquier parte del mapa de memoria.
- A veces se las llama CC-UMA (Cache Coherent UMA). La coherencia de cache (Cache coherency) significa que si un procesador actualiza el dato almacenado en la memoria, todos los procesadores se enteran del cambio. La coherencia de cache se implementa a nivel hardware.



Threads (Hilos)

- Un hilo se define como un flujo de instrucciones independientes que puede ser puesto a correr por el sistema operativo... pero ¿que significa esto?

Threads (Hilos)

- Un hilo se define como un flujo de instrucciones independientes que puede ser puesto a correr por el sistema operativo... pero ¿que significa esto?
- Es algo así como un procedimiento que puede correr independientemente del programa principal.

Threads (Hilos)

- Un hilo se define como un flujo de instrucciones independientes que puede ser puesto a correr por el sistema operativo... pero ¿que significa esto?
- Es algo así como un procedimiento que puede correr independientemente del programa principal.
- Imaginen un programa principal que contiene una serie de procedimientos que pueden ponerse a correr simultaneamente y/o independientemente por el sistema operativo. Esto lo que podríamos llamar un programa multi-hilo.

Threads (Hilos)

- Un hilo se define como un flujo de instrucciones independientes que puede ser puesto a correr por el sistema operativo... pero ¿que significa esto?
- Es algo así como un procedimiento que puede correr independientemente del programa principal.
- Imaginen un programa principal que contiene una serie de procedimientos que pueden ponerse a correr simultaneamente y/o independientemente por el sistema operativo. Esto lo que podríamos llamar un programa multi-hilo.
- Si todavía no entiendieron nada y miran con cara de "¿donde estoy metido?":
  - Pueden pensar en la tarea a realizar por un programa como la accion de tejer.
  - Si uno tuviera una sola aguja en la maquina, entonces estamos en el caso de un solo hilo.
  - En el caso de existir mas agujas, todas colaboran para realizar la tarea sobre el mismo tejido. Este caso es similar a un programa multi-hilo.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.
- Los hilos comparten naturalmente la memoria, por lo que acceder a un dato que genero otro hilo es leer directamente una variable.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.
- Los hilos comparten naturalmente la memoria, por lo que acceder a un dato que genero otro hilo es leer directamente una variable.
- La ganancia obtenida programando con hilos en una maquina SMP supera ampliamente lo que se puede sacar utilizando MPI.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.
- Los hilos comparten naturalmente la memoria, por lo que acceder a un dato que genero otro hilo es leer directamente una variable.
- La ganancia obtenida programando con hilos en una maquina SMP supera ampliamente lo que se puede sacar utilizando MPI.
- Las dos tecnicas mas utilizadas para programar con hilos son:

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.
- Los hilos comparten naturalmente la memoria, por lo que acceder a un dato que genero otro hilo es leer directamente una variable.
- La ganancia obtenida programando con hilos en una maquina SMP supera ampliamente lo que se puede sacar utilizando MPI.
- Las dos tecnicas mas utilizadas para programar con hilos son:
  - pthreads: POSIX threads, es un standard que esta soportado en Unix, Linux y Windows. Solo para gente con muuuuchas ganas.

Threads (Hilos)

- Se utilizan hilos para sacar provecho de las maquinas con memoria compartida.
- Con MPI cada proceso tiene su propio espacio de memoria: cuando se necesita un dato, debe copiarselo de la memoria de un proceso a la del otro... esto es caro.
- Los hilos comparten naturalmente la memoria, por lo que acceder a un dato que genero otro hilo es leer directamente una variable.
- La ganancia obtenida programando con hilos en una maquina SMP supera ampliamente lo que se puede sacar utilizando MPI.
- Las dos tecnicas mas utilizadas para programar con hilos son:
  - pthreads: POSIX threads, es un standard que esta soportado en Unix, Linux y Windows. Solo para gente con muuuuchas ganas.
  - OpenMP: se basa en directivas de compilador, mucho mas barato para programar, aunque se paga un costo por la facilidad.

OpenMP

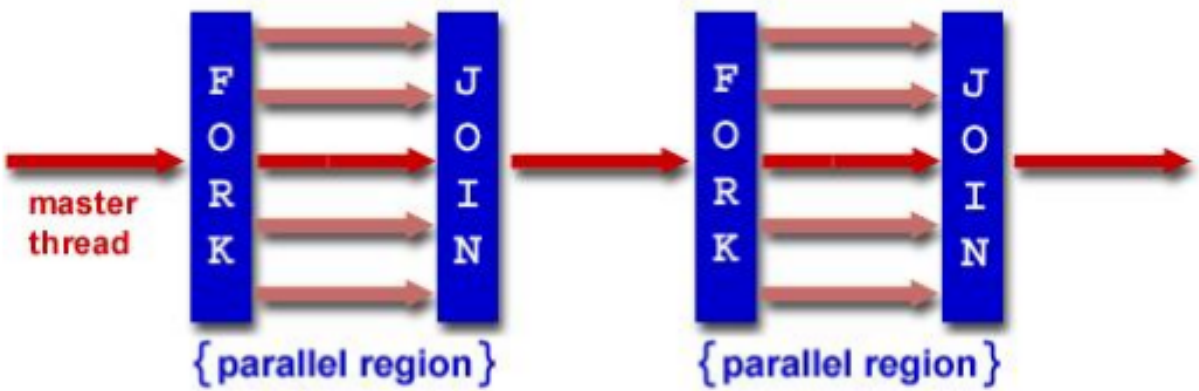
- El nombre viene de Open Multi-Processing.
- Es una API (Application Program Interface) que se usa para paralelismo basado en multiples hilos en entornos de memoria compartida.
- Es muy portable, funciona para C/C++ y para Fortran.
- Hay implementaciones en el mundo Linux/Unix y en Windows.

OpenMP

- No realiza la distribucion de trabajo en los hilos por si mismo.
- No esta implementado exactamente igual por todas las empresas.
- No garantiza hacer el uso mas eficiente de los recursos (para eso esta pthreads).
- No programa por uno, hay que sentarse a preparar el codigo para asegurarse que funciona bien.

Cómo se programa en OpenMP

- Paralelismo basado en memoria compartida y multiples hilos.
- Paralelismo explícito (ojo, no automatico), dando opciones al programador para tener control de la paralelizacion.
- Modelo Fork-Join:



Cómo se programa en OpenMP

- FORK: el master thread crea un equipo de hilos (la cantidad se puede controlar desde programa o con una variable de entorno). La parte paralela del codigo es ejecutada por los hilos creados.
- JOIN: cuando los hilos terminan de la parte marcada como paralela del codigo, se sincronizan y terminan, quedando solo en ejecucion el master thread hasta la proxima region paralela.

El HolaMundo! en OpenMP

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

int main(int argc, char* argv[]){
    #pragma omp parallel
    {
        int ID;
        ID = omp_get_thread_num();
        printf("Hola soy el Thread %d. \n", ID);
    }
    return 0;
}
```

- Para compilar: gcc -fopenmp hola omp.c -o hola

Resumiendo...

Resumiendo...

- OpenMP es una API que se utiliza para paralelismo basado en memoria compartida.
  - Threads se comunican por las variables compartidas.

Resumiendo...

- OpenMP es una API que se utiliza para paralelismo basado en memoria compartida.
  - Threads se comunican por las variables compartidas.
- Un compartir de información no deseado puede causar una condición de carrera.
  - cuando la salida depende del orden en que se hayan ejecutado cada thread.

Resumiendo...

- OpenMP es una API que se utiliza para paralelismo basado en memoria compartida.
  - Threads se comunican por las variables compartidas.
- Un compartir de información no deseado puede causar una condición de carrera.
  - cuando la salida depende del orden en que se hayan ejecutado cada thread.
- Como resolver estos problemas?
- Lo veremos más adelante...

Como creamos Threads

Si queremos crear 4 threads que ejecuten la región paralela:

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

int main(int argc, char* argv[]){
    omp_set_num_threads(4);
    #pragma omp parallel
    {
        int ID;
        ID = omp_get_thread_num();
        printf("Hola soy el Thread %d. \n", ID);
    }
    return 0;
}
```

Otra forma

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

int main(int argc, char* argv[]){
    #pragma omp parallel num_threads(4)
    {
        int ID;
        ID = omp_get_thread_num();
        printf("Hola soy el Thread %d. \n", ID);
    }
    return 0;
}
```

Ejercicio

Sincronización

- Tipos de sincronismo:
  - Critical
  - Atomic
  - Barrier
  - Ordered.

Critical

Solo un thread accede a esa sección por vez.
Un ejemplo:

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

int main(int argc, char* argv[]){
    #pragma omp parallel num_threads(4)
    {
        int id;
        float temp;
        id = omp_get_thread_num();
        ...
        ...
        #pragma omp critical
        funcionLoca(B)
    }
    return 0;
}
```

Atomic

Solo un thread puede escribir en dicha sección por vez.
Un ejemplo:

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

#define MAX 10

int main() {
   int count = 0;
   #pragma omp parallel num_threads(MAX)
   {
      #pragma omp atomic
      count++;
   }
   printf("Numero de Threads: %d\n", count);
}
```

Barrier

Se espera a que todos los threads hayan llegado a ese lugar.
Un ejemplo:

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

#define MAX 10

int main() {
   int count = 0;
   #pragma omp parallel num_threads(MAX)
   {
      #pragma omp atomic
      count++;
      #pragma omp barrier
      printf("Numero de Threads: %d\n", count);
   }
}
```

Paralelización de Ciclos

- Un ciclo es facilmente paralelizable en OpenMP.
- El master crea threads adicionales que cubren todas las iteraciones del ciclo.
- No puede haber dependencia entre la iteraciones.

Un ejemplo:

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

#define MAX 100

int main() {
   int count = 0;
   double res[MAX]; int i;
   #pragma omp parallel
   {
      #pragma omp for
      for (i=0;i<MAX; i++) {
      res[i] = i;
   }
}
```

Reduction

La variable tiene una copia local en cada thread, al finalizar dicho valor es reducido en una variable global compartida.
Un ejemplo:

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

#define MAX 100

int main() {
   int count = 0;
   double res[MAX]; int i;
   double prom;
   #pragma omp parallel for reduction (+:prom)
   for (i=0;i< MAX; i++) {
      prom += A[i];
   }
   prom = prom/MAX;
}
```

Algunos detalles

- Shared: Las variables son compartidas por todos los threads (cuidado con compartir demasiado).
- Private: Las variables quedan en la seccion privada de cada thread, no se ven los cambios realizados por uno en otro.
- `#pragma omp for nowait:`
  se distribuye el trabajo, pero se puede seguir.
- `#pragma omp master:`
  las instrucciones son realizadas unicamente por el master.
- `#pragma omp single:`
  las instrucciones son realizadas por un único thread (cualquiera).
- `omp_in_parallel():`
  te informa si estás dentro de un región paralela o no.

Algunos detalles

- `omp_get_num_procs()`:
  indica la cantidad de procesadores que hay en el sistema.
- `omp_get_max_threads()`:
  indica la cantidad máxima de threads que podés tener.
- `omp_set_num_threads(MAX)`:
  setea la cantidad de threads.
- `omp_get_wtime()`:
  para calcular tiempos de ejecución.

Ejercicio

Preguntas?