Arquitectura de Computadores

Tema 5-3. Programación Paralela con OpenMP

Sumario

- Fundamentos de la Programación Paralela en multiprocesadores de memoria compartida
- Interfaz de programación OpenMP
- Ejecución de un programa OpenMP ejemplo en un multiprocesador real con evaluación de prestaciones
- Monitorización de la ejecución paralela de los hilos en Linux

Introducción

- Se quieren sumar 64000 números `A[i], i=0,1,…,63999` con un multiprocesador UMA de 64 núcleos. Identificadores de los núcleos: Pn = 0, 1, …, 63
- Objetivos
  - Carga computacional (número de sumas) por procesador balanceada -> división del conjunto de números en 64 grupos de igual tamaño -> cada núcleo opera sobre 1000 números
  - Los 64000 números se almacenan en el espacio direccionamiento compartido
  - Cada núcleo empieza a sumar desde una posición distinta de memoria
- Método
  - La carga computacional se divide en dos partes
  - Parte 1: cada procesador ejecuta la suma de 1000 números, se generan 64 sumas parciales
  - Parte 2: las sumas parciales se reducen a una suma acumulada final

Parte 1 de la carga computacional

- Programa C que ejecuta cada núcleo Pn = 0, 1, …, 63: suma parcial de 1000 números

```
sum[Pn]=0;
for(i=1000*Pn; i<1000*(Pn+1); i+=1)
    sum[Pn]+=A[i];
```

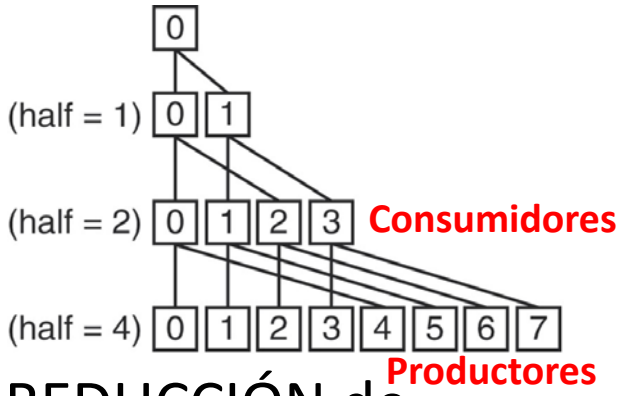
Cada núcleo conoce su Pn

En zona compartida de memoria

Cada núcleo tiene una posición de memoria “privada” donde se actualiza el valor del índice i

Copyright © 2014 Elsevier Inc. All rights reserved.

Parte 2 de la carga computacional



- Programa C que ejecuta selectivamente cada núcleo: REDUCCIÓN de los 64 números en 1

```
mitad=64;
while(mitad > 1){
   sincronizar();
   if(mitad%2!=0 && Pn==0) sum[0]+=sum[mitad-1];
   mitad=mitad/2;
   if(Pn < mitad) sum[Pn]+=sum[Pn+mitad];
}
```

Variable privada por núcleo

Consumidor debe esperar a que el productor genere su resultado

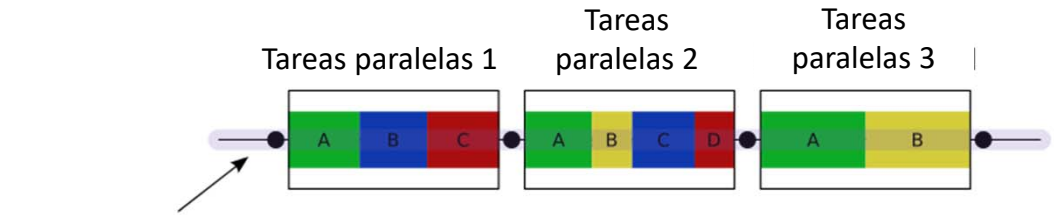
Cada núcleo conoce su Pn

Necesario cuando mitad es impar; no se ejecuta cuando mitad=64

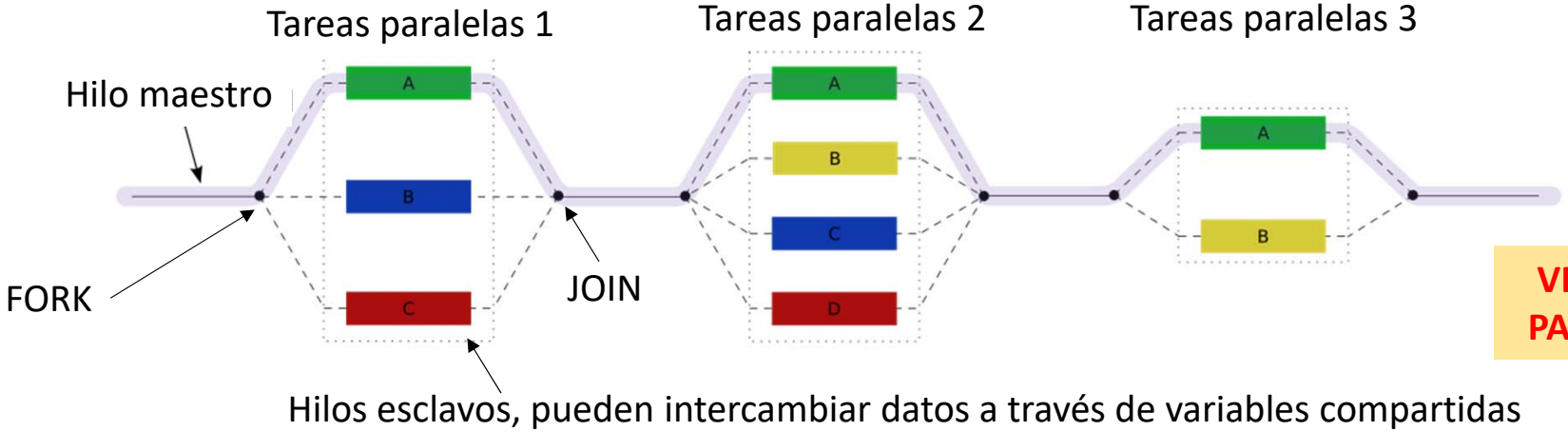
OpenMP

- Sistema de programación paralela tipo API (Application Programmer Interface) que extiende lenguajes de programación estándar (C, Fortran). Incluye:
  - Directivas del compilador (pragmas)
  - Librería de programas ya compilados
  - Variables de entorno
- Genera múltiples hilos para ser ejecutados en un multiprocesador de memoria compartida
- Especializado en bucles y sus reducciones
- Fácil de programar

Modelo FORK/JOIN de generación de hilos en OpenMP



VERSIÓN SERIE



VERSIÓN PARALELA

OpenMP

- ¿Cómo se activa la compilación de programas C que usan OpenMP?
  - `$ gcc -fopenmp programa.c`
- ¿Cómo se activan los núcleos del multiprocesador?
  - `#define P 64`
  - `#pragma omp parallel num_thread(P)` ← se crean 64 hilos de instrucciones
- Programa C+OpenMP que ejecuta el multiprocesador: sumas paralelas

```
#pragma omp parallel for private(Pn)
for (Pn=0; Pn<P; Pn+=1){
  sum[Pn]=0;
  int i;
  for(i=1000*Pn; i<1000*(Pn+1); i+=1) sum[Pn]+=A[i];
}
```

OpenMP

- Programa C+OpenMP que ejecuta el multiprocesador: Reducción

```
#pragma omp parallel for reduction(+: SumaFinal)
for (i=0; i<P; i+=1) SumaFinal+=sum[i];
```

OpenMP: programa real

```
#include <stdio.h>
#include <omp.h>
#define P 16
void main() {
  int Pn, sum[P], A[P*1000], SumaFinal=0;
  for(i=0; i<P*1000; i++) A[i]=i%1000;  // A[i] = 0,…,999
// Parte 1
  #pragma omp parallel for private(Pn)
  for (Pn=0; Pn<P; Pn+=1){
    sum[Pn]=0;
    int k;
    for(k=1000*Pn; k<1000*(Pn+1); k+=1) sum[Pn]+=A[k];
    printf("Suma parcial de procesador P%2i: %i\n", omp_get_thread_num(), sum[Pn]);
  }
// Parte 2
  #pragma omp parallel for reduction(+: SumaFinal)
  for (Pn=0; Pn<P; Pn+=1) SumaFinal+=sum[Pn];
  printf("Suma Final: %i\n", SumaFinal);
}
```

```
dbenitez@vector:~/grado$ ./suma16000
Suma parcial de procesador P 0: 499500
Suma parcial de procesador P 1: 499500
Suma parcial de procesador P 7: 499500
Suma parcial de procesador P 9: 499500
Suma parcial de procesador P15: 499500
Suma parcial de procesador P14: 499500
Suma parcial de procesador P12: 499500
Suma parcial de procesador P 3: 499500
Suma parcial de procesador P 5: 499500
Suma parcial de procesador P 8: 499500
Suma parcial de procesador P 2: 499500
Suma parcial de procesador P11: 499500
Suma parcial de procesador P 6: 499500
Suma parcial de procesador P13: 499500
Suma parcial de procesador P10: 499500
Suma parcial de procesador P 4: 499500
Suma Final: 7992000
dbenitez@vector:~/grado$
```

OpenMP: programa real + evaluación de prestaciones

Paralelo

```c
#include <stdio.h>
#include <omp.h>
#define P 16
#define Niteraciones 1e5
void main()
{
   int Pn, sum[P], A[P*1000], SumaFinal=0, i;
// Inicializacion del vector A
   for(i=0;i<P*1000;i++) {
      A[i]=i%10;
   }
// Parte 1: suma parcial reiterada de 16 numeros en grupos de 1000 numeros
   int N = Niteraciones;
   #pragma omp parallel for private(Pn)
   for (i=0; i<N; i+=1) {
     Pn = omp_get_thread_num();
     sum[Pn]=0;
     int k;
     for(k=1000*Pn; k<1000*(Pn+1); k+=1) {
        sum[Pn]+=A[k];
     }
   }
// Parte 2
   #pragma omp parallel for reduction(+: SumaFinal)
   for (Pn=0; Pn<P; Pn+=1) {
     SumaFinal+=sum[Pn];
   }
   printf("Iteraciones totales: %i, Suma Final: %i\n", N, SumaFinal);
}
```

Secuencial

```c
#include <stdio.h>
#include <omp.h>
#define P 16
#define Niteraciones 1e5
void main()
{
   int Pn, sum[P], A[P*1000], SumaFinal=0, i;
// Inicializacion del vector A
   for(i=0;i<P*1000;i++) {
      A[i]=i%10;
   }
// Parte 1: suma parcial reiterada de 16 numeros en grupos de 1000 numeros
   int N = Niteraciones;
   for (i=0; i<N; i+=1) {
     for (Pn=0; Pn<P; Pn+=1){
       sum[Pn]=0;
       int k;
       for(k=1000*Pn; k<1000*(Pn+1); k+=1) {
        sum[Pn]+=A[k];
       }
     }
   }
// Parte 2
   for (Pn=0; Pn<P; Pn+=1) {
     SumaFinal+=sum[Pn];
   }
   printf("Iteraciones totales: %i, Suma Final: %i\n", i, SumaFinal);
}
```

Ejecución de programa paralelo

Ejecución de programa secuencial

```
dbenitez@vector:~/grado$ time ./suma16000
Iteraciones totales: 100000, Suma Final: 72000

real    0m1.704s
user    0m25.210s
sys     0m0.000s
dbenitez@vector:~/grado$ time ./suma16000secuencial
Iteraciones totales: 100000, Suma Final: 72000

real    0m5.637s
user    0m5.635s
sys     0m0.000s
```

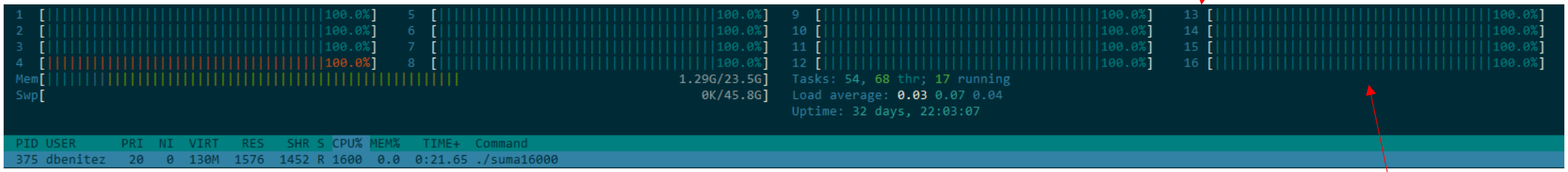
Speed-Up = 5,637/1,704 = 3,31

Eficiencia Paralelismo = 3,31 / 16 = 21%

Monitorización Linux con **htop** en un multiprocesador de 16 núcleos

16 Núcleos

Ejecución de programa paralelo



Ocupación por núcleo

Ejecución de programa secuencial