Plataformas de soporte computacional: Programación con OpenMP (rev. 1.1)

Diego R. Llanos, Belén Palop
Departamento de Informática
Universidad de Valladolid
{diego,b.palop}@infor.uva.es

Índice

1. Un algoritmo secuencial

Un algoritmo secuencial: multiplicación de matrices

- Algoritmo con complejidad $O(n^3)$ y sin dependencias: buen candidato para su paralelización:
- Funcionamiento:

```
float a[TAM][TAM];
float b[TAM][TAM];
float c[TAM][TAM];

for (i=0;i<TAM;i++)
   for (j=0;j<TAM;j++) {
      c[i][j]=0;
      for (k=0;k<TAM;k++)
         c[i][j]= c[i][j]+a[i][k]*b[k][j];
```

- Vamos a implementar este algoritmo, y a medir su tiempo secuencial y paralelo.

2. Medición del tiempo

Medición del tiempo en Solaris

- Para medir el tiempo invertido por una aplicación en su conjunto puede usarse el comando `time -p ./aplicacion`
- Para medir el tiempo invertido por *una parte* de la aplicación debe utilizarse un *profiler*. Solaris proporciona el "Performance Analyzer".
- Pasos para su uso:
  1. Insertar al principio del código "`#include<libcollector.h>`"
  2. Utilizar las funciones "`collector_resume()`" y "`collector_pause()`" para iniciar y terminar la medición de tiempos.
  3. Compilar activando el flag "`-g`", que genera la tabla de símbolos necesaria para el *profiler*.
  4. Ejecutar el programa como `collect ./aplicacion`. Esto genera un directorio llamado `test.1.er` con los datos de la medición.
- Para ver los resultados de la medición del tiempo puede utilizarse la aplicación `analyzer test.1.er`.
- Existe también la alternativa de usar un comando interactivo en modo texto: `er_print test.1.er`
- Comandos del `er_print`:
  - `metrics i.%wall`: Trabaja con el tiempo total de las funciones, medido en "wall clock time".
  - `sort i.%wall`: Ordena las funciones por el tiempo de ejecución.
  - `functions`: Muestra los tiempos globales de cada función.
  - `source fuente.c`: Muestra los tiempos invertidos en cada línea del código.

Un ejemplo

- `$ cat script-erprint`

```
metrics i.%wall
sort i%wall
functions
source mult-par.c
```

- `$ collect ./mult-par`
- `$ er_print -script script-erprint test.1.er > results.txt`
- `$ cat results.txt` Esto produce algo así:

```
Current metrics: e.%wall:name
Current Sort Metric: Exclusive Wall Time ( e.%wall )
Current Sort Metric: Exclusive Wall Time ( e.%wall )
Functions sorted by metric: Exclusive Wall Time

Excl. Wall      Name
  sec.    %
51.066 100.00   <Total>
48.924  95.81   main -- MP doall from line 30 [_$d1A30.main]
 2.141   4.19   <OMP-implicit_barrier>
 0.      0.     _start
 0.      0.     main
```

```
Source file: ./mult-par.c
Object file: ./mult-par
Load Object: ./mult-par

   Excl. Wall
     sec.      %
                     1. #include<stdio.h>
                     2. #include<libcollector.h>
                     3. #include<omp.h>
                    ...
    1.361   2.7     34.          for (k=0;k<TAM;k++)
## 47.403  92.8     35.             c[i][j]= c[i][j]+a[i][k]*b[k][j];
                    ...
```

3. Opciones de compilación de SunOS para sistemas paralelos

Opciones de compilación de SunOS para sistemas paralelos

`-xopenmp=parallel` Activa el reconocimiento de directivas OpenMP. Si el nivel de optimización es menor que `-O3`, lo sube a `-O3`.

`-xopenmp=noopt` Activa el reconocimiento de directivas OpenMP, sin subir el nivel de optimización (útil para ejecutar el programa con el depurador `dbx`).

`-xopenmp=none` Desactiva el reconocimiento de directivas OpenMP (comportamiento por defecto).

`-xautopar` Activa la paralelización automática. No hace caso de las directivas de paralelización OpenMP. Eleva el nivel de optimización a `-O3`.

4. Arquitectura de nuestro sistema paralelo

Arquitectura de nuestro sistema paralelo

- Dos sistemas:
  - Frontend: nodoyuna, 2 núcleos, arquitectura Intel Dual Core.
  - Backend: geopar, 4x4 núcleos, arquitectura Intel Xeon.
- La medición de rendimientos en el frontend está muy condicionada por la carga del sistema.
- El backend, en cambio, funciona por un sistema de colas:
  - Los usuarios envían su trabajo a través de un *script* que indica cuántos threads necesita.
  - El backend encola los trabajos y los va ejecutando en orden, garantizándoles el 100 % de CPU a cada uno de ellos.
  - Cuando termina, los resultados de la aplicación se guardan en ficheros con nombres del tipo `script.o1234` y `script.e1234`, donde el número indica el número asignado a ese trabajo y "script" es el nombre del script con el que se ha lanzado.

Un ejemplo

```
$ cat lanza-paralelo
#! /usr/bin/bash
#$ -cwd
#$ -pe threads 02
/usr/bin/time -p /opt/SUNWspro/bin/collect -o mult-par.er ./mult-par
```

```
$ qsub lanza-paralelo
Your job 4586 ("lanza-paralelo") has been submitted

$ qstat
job-ID  prior   name       user   state submit/start at     queue                          slots
-----------------------------------------------------------------------------------------------
 4586   0.00000 lanza-para diego   qw   11/22/2009 14:28:29                                 2

$ qstat
job-ID  prior   name       user   state submit/start at     queue                          slots
-----------------------------------------------------------------------------------------------
 4586   0.55500 lanza-para diego    r   11/22/2009 14:29:17 all.q@geopar.dcs.fi.uva.es     2

$ qstat
```

(No queda ningún trabajo en la cola)

5. Ejercicios

Ejercicios a realizar

1. Escribir un programa en C llamado `mult-par` que multiplique dos matrices de tamaño TAM=2000: una rellena con "unos" y otra rellena con "doses".
2. Comprobar que funciona correctamente.
3. Medir el tiempo secuencial en el frontend (nodoyuna).
4. Medir el tiempo secuencial en el backend (geopar).
5. Paralelizar cada uno de los tres bucles por separado con directivas OpenMP
6. Calcular los speedups de cada versión para 2, 4, 6, 8, 10, 12, 14 y 16 procesadores.
7. Hacer una gráfica bonita con las tres curvas de speedup en función del número de procesadores
8. Enviarla por e-mail al profesor, junto con un informe de los problemas encontrados (subject: "Práctica OpenMP").

Una sugerencia

- Aunque las gráficas de speedup se pueden hacer con Excel u OpenOffice, los "profesionales" lo hacemos con `gnuplot` :-)
- Para ello, basta con guardar las coordenadas X e Y a dibujar en un fichero de texto, y preparar un script para `gnuplot` como el siguiente:

```
# script.gnp: Generacion de un fichero .EPS con las curvas de speedup
set terminal postscript 24  # Formato del fichero de salida.
set size 30/30 , 1.         # Proporciones del dibujo generado
set pointsize 1             # Tamaño de los puntos de las curvas
set title "Speedup, matrix multiplication"
set xlabel "Processors"     # Etiqueta para el eje X
set xtics 0, 2, 16          # Escala para el eje X (inicio,stride,fin)
set ytics 0, 1, 4           # Escala para el eje Y (inicio,stride,fin)
set ylabel "Speedup"        # Etiqueta para el eje X
set output "speedup-MM.eps" # Nombre del fichero de salida
plot [0:18][0:4] \          # Comando de dibujo: atentos a las "\" !
"speedup-MM-exterior.dat" title "Outer loop" with linespoints,\
```

```
"speedup-MM-medio.dat" title "Middle loop" with linespoints, \
"speedup-MM-interior.dat" title "Inner loop" with linespoints
```

- Luego, basta con ejecutar `gnuplot script.gnp`.