Practicas de Multiprocesadores

Primavera 2007

Introducción a la paralelización con OpenMP

0. Objetivo de la práctica

El objetivo de esta práctica es familiarizarse con el entorno de ejecución y el proceso básico de paralelización de una aplicación secuencial, como paso previo a las paralelización de un programa usando OpenMP y/o MPI. Al final de esta práctica debereis enviar un mensaje a jesus.labarta@bsc.es con la ficha de entrega disponible al final de este enunciado.

1. Acceso a bscsmp01 y material para la práctica

- Grupos:
  - Las prácticas se pueden hacer por parejas o individual.
  - Se os asignará número de grupo en la primera sesión.
  - Tenéis que enviar un mail al profesor (jesus.labarta@bsc.es) con copia a support@bsc.es incluyendo el número de grupo, nombre de los integrantes y direcciones de e-mail de contacto.
- Desde vuestra cuenta en máquinas de la FIB entrar en bscsmp01.bsc.es
  - `ssh acmpXX@bscsmp01.bsc.es` (XX: 01..20)
  - password: `acmplabXX`, siendo XX el identificador del número de grupo
  - La primera vez que entreis se os pedirá cambiar el password.
- Del directorio `/home/cursos/mp/` cuelga:
  - `acmp`: directorio en el que encontrareis material para las prácticas
  - `acmpXX`: directorio de trabajo para el grupo XX.
- Copiar `../acmp/intro_MP.tar` en vuestro directorio de trabajo
  - Desplegar: `tar –xvf intro_MP.tar`
  - Veréis que contiene, además del makefile para compilar y el script run.sh para ejecutar en las colas, varios ficheros fuente en C: `redBlackSOR.c` y versiones de `relax.YY.c` (siendo YY `serial`, `omp1` y `omp2`). Además también contiene el fichero rbInput.dat que da valores de entrada a la ejecución del binario que se generará.
- Editores: tenéis accesible vi y vim

- We encourage you to read the updated Altix User Guide document, that you will find attached to this mail. Inside you will find information about submitting jobs to Moab/Slurm, porting your job scripts and the basic commands available. The last version can always be found at the following link:

  `http://www.bsc.es/media/1487.pdf`

2. Ejecución del programa secuencial

- Copiar `relax.serial.c` sobre `relax.c`

```
#include <math.h>

extern double rho2;

void initialize( int n, double rhs, double f[n][n], double
u[n][n] ) {
    int i, k;

    for (k=0; k<n; k++) {
      for (i=0; i<n; i++) {
           u[i][k] = 0.0;
           f[i][k] = rhs;
           }
      }
}

void updateOmega( double *omega ) {
    if ( *omega == 1.0)
        *omega = 1.0 / (1.0 - 0.5 * rho2);
    else
        *omega = 1.0 / (1.0 - 0.25 * rho2 * *omega );
}

void relax( int n, double *omega, double f[n][n], double
u[n][n], double *nm ) {
    int i, k;
    double resid;
    double norm=0.0;

    for ( i = 1; i < n-1; i++ ) {
        for ( k = 1; k < n-1; k++ ) {
            resid = u[i+1][k] + u[i-1][k] +
                    u[i][k+1] + u[i][k-1] -
                    4 * u[i][k] - f[i][k];
            norm += fabs(resid);
            u[i][k] -= *omega * resid / (-4);
        }
    }

    *nm = norm;
}
```

-
- Compilación del fuente secuencial original (`make`)
- Ejecución: `mnsubmit run.sh` envía el job a la cola acmp reservada para prácticas de Multiprocesadores. El fichero `run.sh` especifica el nombre del job, ficheros de salida y error, recursos solicitados y la ejecución propiamente dicha que se quiere realizar.
- To check the status/manage a job:
  - `mnq`: shows all the jobs submitted.
  - Checkjob <job_id>: shows detailed information about a specific job.
  - `mncancel <job_id>:` removes a job fromhte queue, cancelling the execution if it had already started.

- Mirar los ficheros generados por el sistema de colas: `*.o<job_id>`, `*.e<job_id>` (en este caso no debería haber errores)

Preguntas

P1. Rellenar la Tabla 1 con el tiempo de ejecución del programa, indicando el tiempo de CPU en modo usuario y en sistema

3. Profile con comando gprof

- Modificar el fichero `makefile` para que compile con el flag `-p`
- Ejecución normal del binario obtenido
- Se genera `gmon.out`
- Análisis: el comando `gprof redBlackSOR > profile.txt` genera a partir de `gmon.out` y del fichero ejecutable `redBlackSOR` un profile texto con información del tiempo consumido por cada rutina. Permite analizar por separado los tiempos cuando una rutina es invocada desde distintos puntos.

Preguntas

P2. Completar la Tabla 1 con el número de invocaciones de cada rutina y desde quien es invocada y con los tiempos de ejecución inclusivos y exclusivos.

4. Autoparalelización

- Modificar el `makefile` para que se compile con el flag `-parallel`. El compilador intentará paralelizar de forma automática los bucles en el programa secuencial original. Vereis que fracasa totalmente en el intento.

Preguntas

P3. ¿Son ciertas todas las dependencias que detecta el compilador? Indicad las que son realmente ciertas.

5. Paralelización con OpenMP

- Suponed que, a pesar de las dependencias reales, el programador decide paralelizar con OpenMP, tal como se muestra en `relax.omp1.c`. Copiad `relax.omp1.c` sobre `relax.c`.
- Modificar el `makefile` para que compile con el flag `-openmp`.
- Ejecución normal del binario obtenido, inicializando la variable de entorno OMP_NUM_THREADS con el número de procesadores a utilizar para la ejecución.

Preguntas

P4. Completad la Tabla 2 en la que se muestra el tiempo de CPU en modo usuario y sistema y el porcentaje de utilización de la CPU para las ejecuciones con 1, 2, 4 y 8 procesadores.

P5. Completad la Grafica 1 en la que se muestra el speedup respecto a la ejecución secuencial (no a la paralela con 1 sólo procesador) para la parte del programa en que se hace el cálculo de la función `relax`.

P6. ¿Es normal que la ejecución de resultados distintos al ejecutar con un número de procesadores distintos? ¿Podría incluso pasar que dos ejecuciones con el mismo número de procesadores dieran resultados distintos?.

6. Paralelización RedBlack con OpenMP

- Una versión más estable del código anterior se incluye en `relax.omp2.c`. Copiad `relax.omp2.c` sobre `relax.c`.

- Ejecución normal del binario obtenido, inicializando la variable de entorno `OMP_NUM_THREADS` con el número de procesadores a utilizar para la ejecución.

Preguntas

P7. Realizar el análisis de dependencias en los bucles en la función relax. Para ello, contestar:
   a. ¿Que secuencia de valores toma la variable `lsw` en los dos bucles?
   b. ¿Que elementos de la matriz `u` son modificados en el primer bucle? ¿Cuales son modificados en el segundo bucle?

P8. Completad la Grafica 2 en la que se muestra el speedup de esta versión (omp2) respecto a la ejecución secuencial (no a la paralela con 1 sólo procesador). En esta versión se utiliza como política de planificación de iteraciones schedule(runtime), lo que significa que antes de ejecutar la aplicación hay que definir la variable de entorno `OMP_SCHEDULE` con el valor adecuado (por ejemplo `export OMP_SCHEDULE="static,1"`). En caso de no definirse, utiliza "static".

P9. Paralelizar la función `initialize` y volver a ejecutar para completar la gráfica de escalabilidad (versión omp3). ¿A que se debe tanta mejora en el tiempo de ejecución, si esta función tiene un peso prácticamente despreciable en la ejecución del programa? ¿Mejora aún más si se intercambia el orden de los bucles `i` y `k`? Completad la grafica de escalabilidad con los datos obtenidos (versión omp4) ¿A que se debe la mejora adicional?

P10. Modificar la estrategia de asignación de iteraciones a procesadores y completad la Gráfica 3. Para ello utilizar sólo 4 procesadores y modificar la planificación modificando la variable de entorno `OMP_SCHEDULE` con los valores siguientes. Comentad los resultados obtenidos.
   a. `"static"` es la que utiliza por defecto (es decir, cuando se omite la cláusula)
   b. `"static, chunk"` con valores de `chunk` 1, 10 y 100
   c. `"dynamic, chunk"` con valores de `chunk` 1, 10 y 100
   d. `"guided,10"`

Ficha de entrega de la práctica 1. Número de grupo de prácticas: …….

Respuestas a P1 y P2:

Tabla 1: Tiempo de ejecución / Número de llamadas

	time	**gprof**			
	tiempo	parent	#calls	self	descendents
programa					
initialize					
relax					
updateOmega					
readInput					
saveOutput					
usecs					

Respuesta a P3:

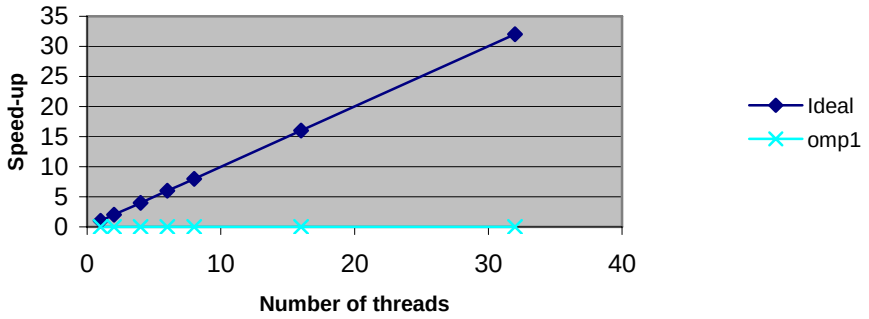
Respuesta a P4:

Tabla 2: Tiempo de ejecución (usuario y sistema) y % de utilización de las CPUs

	time		
	elapsed time	cpu time	cpu utilization
1			
2			
4			
8			

Respuesta a P5:

Gráfica 1: Speed-up respecto a la ejecución sequencial para la paralelización inicial

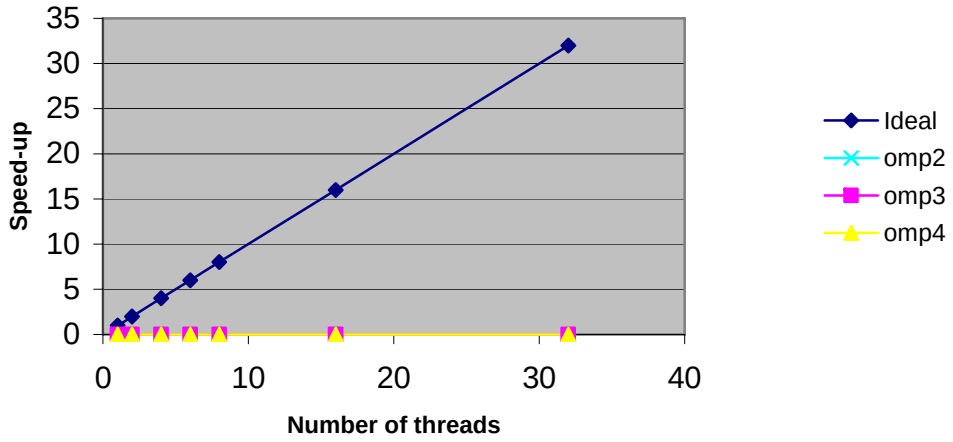


Respuesta a P6:

Respuesta a P7:

Respuestas a P8 y P9:

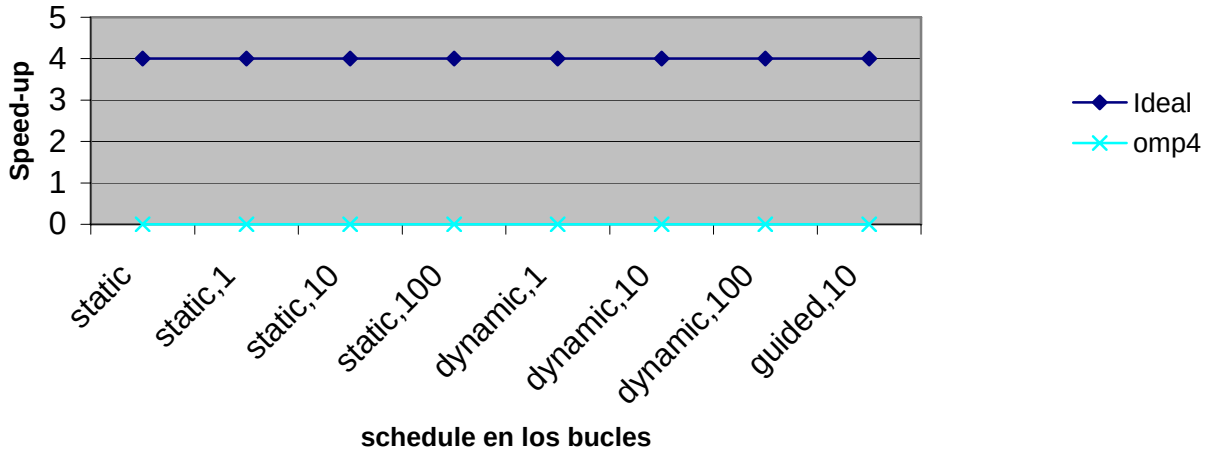
Gráfica 2: Speed-up respecto a la ejecución sequencial para la paralelización redblack



Comentarios:

Respuesta a P10:

Gráfica 3: Speed-up respecto a la ejecución sequencial para distintas planificaciones de iteraciones, con 4 procesadores



Comentarios: