UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Computación en la Nube y de Altas Prestaciones

Seminario 2: Ejercicios de OpenMP

Índice

1 Introducción

Este conjunto de actividades trata de la paralelización de algoritmos en memoria compartida utilizando OpenMP y están pensados para ser realizados en una solo clase de seminario. Cada ejercicio contiene instrucciones sobre su entrega, que puede ser nada, el código o una pequeña memoria con resultados y reflexiones sobre los mismos. El conjunto de los ejercicios de todos los seminarios será entregado como respuesta a una única Tarea de PoliformaT.

El código se diseña, implementa y se ejecuta sobre el clúster de cálculo `kahan` mediante `ssh`, tal como se comenta en el Apartado 2. El acceso a dicho clúster es directo desde los ordenadores del aula o desde cualquier otro ordenador (portátiles del alumno) que esté conectado a la red de la UPV. También es posible conectarse accediendo al mismo entorno a través del escritorio remoto DSIC-LINUX desde https://polilabsvpn.upv.es.

Es recomendable crear una carpeta para la práctica en la unidad `W` (se asume que el disco de red de la UPV está conectado a esta unidad). También es aconsejable que la ruta de dicha carpeta sea corta y sin espacios, porque más adelante necesitaremos teclearla a menudo. Por ejemplo, podría ser `~/W/cna/ejemplo`. Guardaremos en esa carpeta los ficheros `.c` de la práctica (directamente, sin subcarpetas) proporcionados en la carpeta `material`. Sin embargo, los ejecutables que se lanzarán a los nodos del clúster no pueden estar en esta unidad, dado que los nodos no "ven" el disco de red. Será, por tanto, necesario mover el ejecutable al disco local del *font-end*.

2 Ejecución en el clúster

`kahan` es un clúster compuesto por 4 nodos de cálculo, cada uno de ellos con 32 cores, y un nodo *front-end* al cual se conectan los usuarios para compilar y lanzar ejecuciones. Todos los *cores* de un mismo nodo

comparten la memoria de dicho nodo, pero no pueden acceder a la memoria de otros nodos. En el material de la asignatura puedes encontrar más detalles sobre el clúster `kahan`.

Para trabajar con `kahan` hay que conectarse al nodo *front-end* mediante `ssh`:

```
$ ssh -l login@alumno.upv.es kahan.dsic.upv.es
```

donde `login` es tu nombre de usuario UPV y la contraseña, obviamente, también la de la UPV. Tras ejecutar la orden se entra en el directorio personal `home` de `kahan`.

Si se ejecuta el comando `ls` puede comprobarse que existe un directorio `W` que corresponde a la unidad `W` personal de la UPV. Hay que recordar que los ficheros de la práctica deberían estar en una carpeta de `W`, por ejemplo `~/W/cna/ejemplo`, tal como se indicó en el apartado de Introducción.

A continuación habrá que compilar el programa. Es importante tener en cuenta que el ejecutable generado no debe estar en la carpeta `W`, ya que dicha carpeta no es accesible desde los nodos de cálculo de `kahan`. Por ello, es necesario crear una carpeta en la que dejar los ejecutables.

```
$ mkdir ejer1
$ cd ejer1
$ gcc -Wall -fopenmp -o ejemplo ~/W/cna/ejemplo/ejemplo.c -lm
```

donde, al compilar, indicamos la ruta del fichero `ejemplo.c`.

Ahora, desde la carpeta que contiene el fichero, se puede ejecutar este, por ejemplo:

```
$ OMP_NUM_THREADS=4 ./ejemplo
```

Se acaba de ejecutar el programa en el *front-end*, **no en los nodos de cálculo** de `kahan`. Aunque es posible ejecutar un programa en el *front-end*, solo debe hacerse para ejecuciones muy cortas, es decir, para desarrollar código. El *front-end* solo debe usarse para conectarse por `ssh`, compilar y lanzar trabajos sobre el clúster de la manera que se explica a continuación.

La ejecución de trabajos en el clúster se debe llevar a cabo mediante el **sistema de colas SLURM**. Para ello crearemos un **fichero de trabajo**, el cual es un *script* con diferentes opciones del sistema de colas seguidas por los comandos que se desea ejecutar. En la Figura 1 se expone un ejemplo de fichero de trabajo en el que se ejecuta un programa OpenMP con 3 hilos de ejecución (última línea del fichero). Las líneas que comienzan por `#SBATCH` especifican distintas opciones del sistema de colas. En este caso, el trabajo utilizará la cola (partición) denominada `cna`, usando un nodo del clúster (con sus 32 cores) y con un tiempo máximo de 5 minutos para su ejecución.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=5:00
#SBATCH --partition=lpp

OMP_NUM_THREADS=3 ./ejemplo
```

Figura 1: Fichero de trabajo (`jobopenmp.sh`) para ejecutar en el sistema de colas.

Ahora hay que copiar el texto de la Figura 1 en un fichero (por ejemplo `jobopenmp.sh`) y guardarlo en la carpeta `~/W/cna/ejemplo`.

A continuación se debe lanzar el trabajo al sistema de colas, para lo cual se utiliza la orden `sbatch`. Suponiendo que el fichero de trabajo se llama, por ejemplo, `jobopenmp.sh` bastaría con ejecutar en el *front-end*:

```
$ sbatch ~/W/cna/ejemplo/jobopenmp.sh
```

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

int main(int argc, char *argv[]) {
   double t1, t2;
   ...
   t1 = omp_get_wtime();
   ...  /* Fragmento de codigo a cronometrar */
   t2 = omp_get_wtime();
   printf("Tiempo: %f\n", t2-t1);
}
```

Figura 2: Toma de tiempo de un fragmento de código.

En el terminal se mostrará el número del trabajo.

Una vez lanzado, el sistema de colas se encarga de asignar al trabajo los recursos que necesite (en este caso un nodo del clúster) cuando estos estén disponibles, manteniendo el trabajo en espera hasta entonces. De esta manera, se asegura que los nodos asignados a un trabajo no son utilizados por ningún otro trabajo.

¿Qué ocurre con los mensajes que debería mostrar el programa? Esos mensajes no se muestran en el terminal, sino que se almacenan en un fichero. Por ejemplo, si el número de trabajo 620, tras su ejecución se creará el fichero `slurm-620.out` (salida estándar). Podemos mostrar su contenido con la orden `cat`. También se puede copiar el fichero generado a la carpeta `~/W/cna/ejemplo`, y una vez allí visualizarlo con cualquier editor de texto.

Se puede consultar el estado de las colas con la orden `squeue`. Para cada trabajo se muestra su estado (ST), entre los que destacan: en cola o pendiente (PD), en ejecución (R) o finalizado con éxito (CD). También se puede cancelar un trabajo con la orden `scancel` seguida del número de trabajo.

Ejercicio 0: Comienzo

Escribe un programa en OpenMP que se ejecute con un número variable de threads indicados por línea de comandos, identifique los threads que se están ejecutando, el número total de threads, los que se están ejecutando y con los que se ha iniciado el programa, y saque los resultados por pantalla.

El programa también mostrará la fecha de la versión de OpenMP utilizada (variable `_OPENMP`) con una frase como la siguiente:

```
Versión de OpenMP de Noviembre de 2015.
```

Ejecutad el códido en un nodo de `kahan`.

Entrega: De este ejercicio no se entrega nada.

3 Toma de tiempos

En muchos casos interesa medir el tiempo de ejecución del programa, o de una parte de él, para poder compararlo con el tiempo del programa secuencial, y determinar la mejora obtenida.

Para medir el tiempo de un fragmento de un programa utilizaremos la función `omp_get_wtime()` de OpenMP, que devuelve el tiempo transcurrido (en segundos) desde un instante inicial fijo. La forma de usar esta función se ilustra en la Figura 2.

4 Ejercicio 1: Multiplicación de matrices

Este ejercicio trata del estudio del algoritmo básico para la multiplicación de matrices, tanto en secuencial como en paralelo. El código proporcionado (`matrixproduct.c`) realiza la multiplicación de dos matrices cuadradas, es decir, $C = A \times B + C$ de tamaño $n \times n$ siguiendo el pseudocódigo siguiente:

```
para i = 1,...,n
  para j = 1,...,n
    para k = 1,...,n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    fin para
  fin para
fin para
```

Si nos fijamos bien podremos observar que los tres bucles son intercambiables dando lugar a 3! algoritmos distintos pero correctos todos. Esto es así porque las matrices `A` y `B` solo se leen mientras que la matriz `C` solo se escribe acumulando el nuevo valor calculado al anterior. Para que esto último sea correcto es necesario inicializar a cero la matriz C.

Trabajo a realizar

Tomando como base el código proporcionado, el ejercicio consta de los siguientes programas:

Programa 1: Realizar una implementación secuencial que contenga las seis variantes en un mismo código pero que permita obtener el tiempo de cada versión cuando termine el programa. Por simplicidad se utilizarán matrices cuadradas y su orden será pasado como argumento por línea de comandos. El programa debe comprobar que las seis variantes devuelven el mismo resultado.

Programa 2: En otro programa diferente hay que analizar cada variante y, en caso de que sea posible, hay que paralelizar dicha variante de la forma más eficiente. Hay que introducir directivas OpenMP minimizando la modificación de código para paralelizar. El programa debe proporcionar el tiempo de ejecución de las versiones paralelas. También debe comprobar que las variantes paralelizadas devuelven el mismo resultado.

5 Ejercicio 2: El problema de la mochila

El *problema de la mochila* unidimensional o, en inglés, *the one-dimensional 0/1 knapsack problem*, se puede definir de la siguiente manera. Se dispone de una mochila con capacidad c y una serie de n objetos numerados como $1, 2, \ldots, n$. Cada objeto tiene un peso w_i y un valor p_i. Sea $v = [v_1, v_2, \ldots, v_n]$ un vector solución en el que $v_i = 0$ si el objeto i no está presente en la mochila, o $v_i = 1$ si lo está. El objetivo es el de encontrar un subconjunto de objetos tal que

$$\sum_{i=1}^{n} w_i v_i \leq c,$$

y

$$\sum_{i=1}^{n} p_i v_i,$$

sea máximo.

Existen diferentes métodos para resolver este problema. El método que se proporciona en el fichero `knapsackBF.c` es el más simple, conocido como de *Fuerza Bruta*, y consiste en generar las 2^n posibles soluciones y elegir la mejor. El objetivo es el de paralelizar dicho código con OpenMP. Hacedlo sin cambiar código, simplemente añadiendo las directivas de OpenMP adecuadas.