

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos): Daniel Díaz Pareja

Grupo de prácticas: A2

Fecha de entrega: 10/03/2016

Fecha evaluación en clase: 11/03/2016

Ejercicios basados en los ejemplos del seminario práctico

1. En el primer ejemplo de ejecución en atcgrid usando TORQUE se ejecuta el ejemplo `HelloOMP.c` usando la siguiente orden: `echo 'hello/HelloOMP' | qsub -q ac`. El resultado de la ejecución de este código en atcgrid se puede ver en el seminario. Conteste a las siguientes preguntas:

- a. ¿Para qué se usa en `qsub` la opción `-q`?

RESPUESTA: Para mandar el trabajo indicado a la cola indicada justo después de dicho comando. En este caso, se manda a la cola “ac”.

- b. ¿Cómo sabe el usuario que ha terminado la ejecución en atcgrid?

RESPUESTA: Con el comando `qstat`.

- c. ¿Cómo puede saber el usuario si ha habido algún error en la ejecución?

RESPUESTA: Mirando el fichero de salida de error, cuya extensión comienza por `.e`

- d. ¿Cómo ve el usuario el resultado de la ejecución?

RESPUESTA: Mirando el fichero de salida estándar, cuya extensión comienza por `.o`

- e. ¿Por qué en el resultado de la ejecución aparecen 24 saludos “`!!!Hello World!!!`”?

RESPUESTA: Porque el nodo de cómputo donde se ha realizado el trabajo tiene 24 procesadores lógicos.

2. En el segundo ejemplo de ejecución en atcgrid usando TORQUE el script `script_helloomp.sh` usando la siguiente orden: `qsub script_helloomp.sh`. El script ejecuta varias veces el ejecutable del código `HelloOMP.c`. El resultado de la ejecución de este código en atcgrid se puede ver en el seminario. Conteste a las siguientes preguntas:

- a. ¿Por qué no acompaña a al orden `qsub` la opción `-q` en este caso?

RESPUESTA: Porque especificamos la cola a la que se enviará el trabajo en el mismo script, con la línea `PBS -q ac`

- b. ¿Cuántas veces ejecuta el script el ejecutable `HelloOMP` en atcgrid? ¿Por qué lo ejecuta ese número de veces?

RESPUESTA: Se ejecuta 4 veces, ya que en el bucle `for`, el contador inicial es el límite de threads que puede crear un nodo, es decir, 12, ya que cada nodo tiene 12 cores lógicos.

- c. ¿Cuántos saludos “`!!!Hello World!!!`” se imprimen en cada ejecución? (indique el número exacto) ¿Por qué se imprime ese número?

RESPUESTA: 12, 6, 3 y 1 respectivamente. Se imprime ese número porque se crean 12, 6, 3 y 1 hebras en cada iteración del bucle.

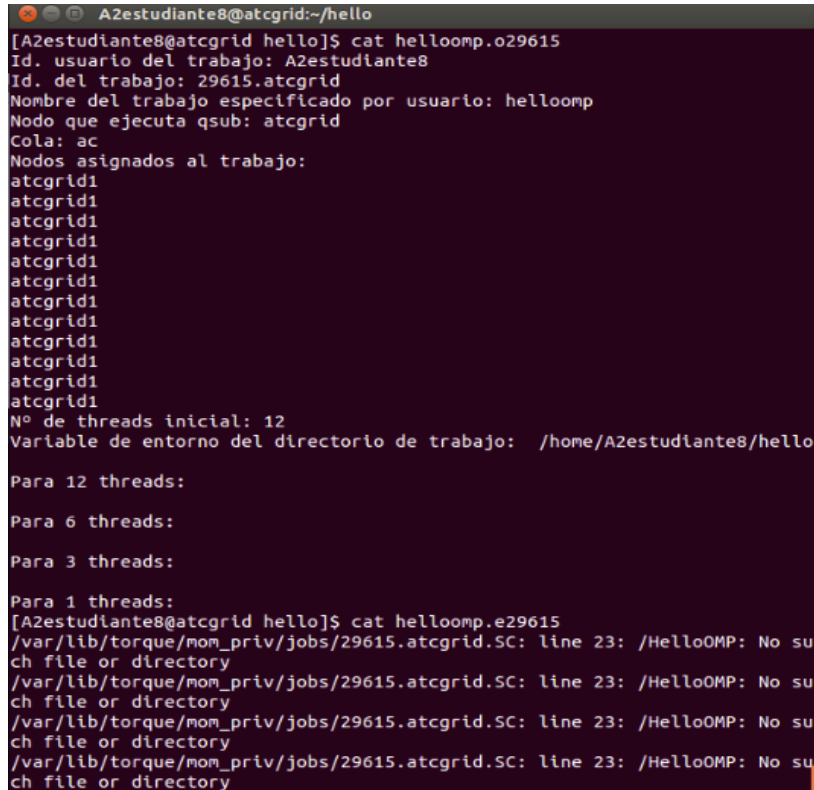
3. Realizar las siguientes modificaciones en el script “`!!!Hello World!!!`”:

- Eliminar la variable de entorno `$PBS_O_WORKDIR` en el punto en el que aparece.
- Añadir lo necesario para que, cuando se ejecute el script, se imprima la variable de entorno `$PBS_O_WORKDIR`.

Ejecutar el script con estas modificaciones. ¿Qué resultados de ejecución se obtienen en este caso?

Incorporar en el cuaderno de trabajo volcados de pantalla que muestren estos resultados.

RESPUESTA: En este caso, al eliminar la variable de entorno del directorio de trabajo, el script no ejecuta el programa en /HelloOMP, ya que el programa no existe en esa dirección, por lo que no imprime su resultado. En el fichero de salida .e obtenemos el error de que no se ha encontrado dicha ubicación.



```
A2estudiante8@atcgrid:~/hello
[A2estudiante8@atcgrid hello]$ cat helloomp.o29615
Id. usuario del trabajo: A2estudiante8
Id. del trabajo: 29615.atcgrid
Nombre del trabajo especificado por usuario: helloomp
Nodo que ejecuta qsub: atcgrid
Cola: ac
Nodos asignados al trabajo:
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
atcgrid1
Nº de threads inicial: 12
Variable de entorno del directorio de trabajo: /home/A2estudiante8/hello

Para 12 threads:

Para 6 threads:

Para 3 threads:

Para 1 threads:
[A2estudiante8@atcgrid hello]$ cat helloomp.e29615
/var/lib/torque/mom_priv/jobs/29615.atcgrid.SC: line 23: /HelloOMP: No su
ch file or directory
/var/lib/torque/mom_priv/jobs/29615.atcgrid.SC: line 23: /HelloOMP: No su
ch file or directory
/var/lib/torque/mom_priv/jobs/29615.atcgrid.SC: line 23: /HelloOMP: No su
ch file or directory
/var/lib/torque/mom_priv/jobs/29615.atcgrid.SC: line 23: /HelloOMP: No su
ch file or directory
```

Resto de ejercicios

4. Incorporar en el fichero .zip que se entregará al profesor el fichero /proc/cpuinfo de alguno de los nodos de atcgrid (atcgrid1, atcgrid2, atcgrid3), y del PC del aula de prácticas o de su PC. Indique qué ha hecho para obtener el contenido de /proc/cpuinfo en atcgrid.

RESPUESTA: Para obtener el contenido de /proc/cpuinfo usamos el comando `cat 'cpu /proc/cpuinfo' | qsub -q ac`, el fichero de salida nos muestra la ejecución de dicho comando en un nodo atcgrid.

Teniendo en cuenta el contenido de cpuinfo conteste a las siguientes preguntas (justifique las respuestas):

a. ¿Cuántos cores físicos y cuántos cores lógicos tiene el PC del aula de prácticas o su PC?

RESPUESTA: Tiene 2 cores físicos, ya que hay 2 id diferentes en el campo **core id** (0 y 1). Tiene 4 cores lógicos, ya que el número de entradas total de procesadores es 4.

b. ¿Cuántos cores físicos y cuántos cores lógicos tiene un nodo de atcgrid?

RESPUESTA: Siguiendo el mismo criterio que antes, el nodo atcgrid tiene 6 cores físicos por cada procesador, y como tiene 2 procesadores (physical id), tenemos un total de 12 cores físicos. Además, tiene 24 cores lógicos, ya que hay 24 entradas de procesadores totales. Es

decir, 12 cores lógicos por cada procesador o threads que la máquina puede ejecutar al mismo tiempo.

5. En el Listado 1 se puede ver un código fuente C que calcula la suma de dos vectores y en el Listado 2 una versión con C++:

```
v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,...N-1
```

Los códigos utilizan directivas del compilador para fijar el tipo de variable de los vectores (v1, v2 y v3). En los comentarios que hay al principio de los códigos se indica cómo hay que compilarlos. Los vectores pueden ser:

- Variables locales: descomentando en el código `#define VECTOR_LOCAL` y comentando `#define VECTOR_GLOBAL` y `#define VECTOR_DYNAMIC`
- Variables globales: descomentando `#define VECTOR_GLOBAL` y comentando `#define VECTOR_LOCAL` y `#define VECTOR_DYNAMIC`
- Variables dinámicas: descomentando `#define VECTOR_DYNAMIC` y comentando `#define VECTOR_LOCAL` y `#define VECTOR_GLOBAL`. Si se usan los códigos tal y como están en Listado 1 y Listado 2, sin hacer ningún cambio, los vectores (v1, v2 y v3) serán variables dinámicas.

Por tanto, se debe definir sólo una de las siguientes constantes: `VECTOR_LOCAL`, `VECTOR_GLOBAL` o `VECTOR_DYNAMIC`.

- a. En los dos códigos (Listado 1 y Listado 2) se utiliza la función `clock_gettime()` para obtener el tiempo de ejecución del trozo de código que calcula la suma de vectores. En el código se imprime la variable `ncgt`, ¿qué contiene esta variable? ¿qué información devuelve exactamente la función `clock_gettime()`? ¿en qué estructura de datos devuelve `clock_gettime()` la información (indicar el tipo de estructura de datos y describir la estructura de datos)?

RESPUESTA: La variable **ncgt** contiene el tiempo transcurrido en el bloque de código comprendido entre las variables `cgt1` y `cgt2`, en nanosegundos. Dicho bloque comprende la suma de dos vectores. A continuación se explica como se obtiene dicho tiempo detalladamente.

La función `int clock_gettime(clockid_t clock_id, struct timespec *tp)` recibe como parámetros el tiempo de reloj especificado por `clock_id` y lo pone en la posición de memoria dada por el puntero `*tp`. Devuelve 0 si ha habido éxito y -1 si ha ocurrido un error. Sólo se le puede pasar como primer argumento `CLOCK_REALTIME`.

Entonces, la estructura de datos que nos interesa es la apuntada por el puntero `*tp`, que es de tipo **timespec** (un struct). Esta estructura contiene al menos los siguientes elementos:

- `time_t tv_sec`: segundos transcurridos.
- `time_t tv_nsec`: nanosegundos transcurridos.

Así, teniendo dos variables `timespec`, podemos entro un bloque de código y restar el campo `tv_sec` de la segunda variable a la primera para calcular cuanto tiempo transcurre al ejecutarse dicho bloque de código. Para más precisión, se resta el campo `tv_nsec` de la misma manera, y se suma al resultado total, obteniendo los nanosegundos transcurridos.

- b. Escribir en el cuaderno de prácticas las diferencias que hay entre el código fuente C y el código fuente C++ para la suma de vectores.

RESPUESTA:

Descripción diferencia	En C	En C++
Se usan diferentes bibliotecas,	<code>#include <stdlib.h></code>	<code>#include <cstdlib></code>

aunque su función es la misma.	<code>#include <stdio.h></code> <code>#include <time.h></code>	<code>#include <iostream></code> <code>#include <time.h></code>
Para declarar los vectores, en c se utiliza la alocaión de un bloque de memoria, mientras que en c++ se utiliza el operador new para memoria dinámica.	<code>v1 = (double*) malloc(N*sizeof(double));</code> <code>v2 = (double*) malloc(N*sizeof(double));</code> <code>v3 = (double*) malloc(N*sizeof(double));</code>	<code>v1 = new double [N];</code> <code>v2 = new double [N];</code> <code>v3 = new double [N];</code>
Al cambiar de bibliotecas, el modo de imprimir por pantalla es de diferente sintaxis.	Se utiliza <code>printf</code>	Se utiliza <code>cout</code>
Al borrar los vectores dinámicos de memoria, en c se utiliza una función y en c++ el operador correspondiente.	<code>free(v1);</code> <code>free(v2);</code> <code>free(v3);</code>	<code>delete [] v1;</code> <code>delete [] v2;</code> <code>delete [] v3;</code>
En c tenemos una variable i declarada fuera de dentro de los propios argumentos de un bucle for, ya que no se puede hacer. En c++ se puede declarar dentro del mismo bucle for.	<code>int i;</code> . . . <code>for(i=0; i<N; i++){ }</code>	<code>for(int i=0; i<N; i++){...}</code>

6. Generar el ejecutable del código fuente C del Listado 1 para vectores locales (para ello antes de compilar debe descomentar la definición de `VECTOR_LOCAL` y comentar las definiciones de `VECTOR_GLOBAL` y `VECTOR_DYNAMIC`). Ejecutar el código ejecutable resultante en atcgrid usando el la cola TORQUE. Incorporar volcados de pantalla que demuestren la ejecución correcta en atcgrid.

RESPUESTA:

```
A2estudiante8@atcgrid:~/hello
[A2estudiante8@atcgrid hello]$ echo 'hello/SumaVectoresC 1000' | qsub -q ac
29678.atcgrid
[A2estudiante8@atcgrid hello]$ cat STDIN.o29678
Tiempo(seg.):0.000002391 / Tamaño Vectores:1000 / V1[0]+V2[0]=V3[0](100.000000+100.000000=200.000000) / V1[999]+V2[999]=V3[999](
199.900000+0.100000=200.000000) /
[A2estudiante8@atcgrid hello]$
```

7. Ejecutar en atcgrid el código generado en el apartado anterior usando el script del Listado 3. Generar el ejecutable usando la opción de optimización `-O2` tal y como se indica en el comentario que hay al principio del programa. Ejecutar el código también en su PC local para los mismos tamaños. ¿Se obtiene error para alguno de los tamaños? En caso afirmativo, ¿a qué se debe este error?

RESPUESTA: Se obtiene error de Segmentation Fault para 2^{19} en adelante. Esto quiere decir que el programa intenta acceder a lugares externos a su espacio de direcciones asignado. Esto sucede porque al utilizar variables locales, se está utilizando el tamaño de la pila para cada hilo del programa, y en 2^{19} el espacio de la pila asignado al programa ya se supera y no se puede cambiar. Este espacio es asignado al comienzo del programa y es totalmente fijo.

8. Generar los ejecutables del código fuente C para vectores globales y para dinámicos. Genere el ejecutable usando `-O2`. Ejecutar los dos códigos en `atcgrid` usando un `script` como el del Listado 3 (hay que poner en el `script` el nombre de los ficheros ejecutables generados en este ejercicio) para el mismo rango de tamaños utilizado en el ejercicio anterior. Ejecutar también los códigos en su PC local. ¿Se obtiene error usando vectores globales o dinámicos? ¿A qué cree que es debido?

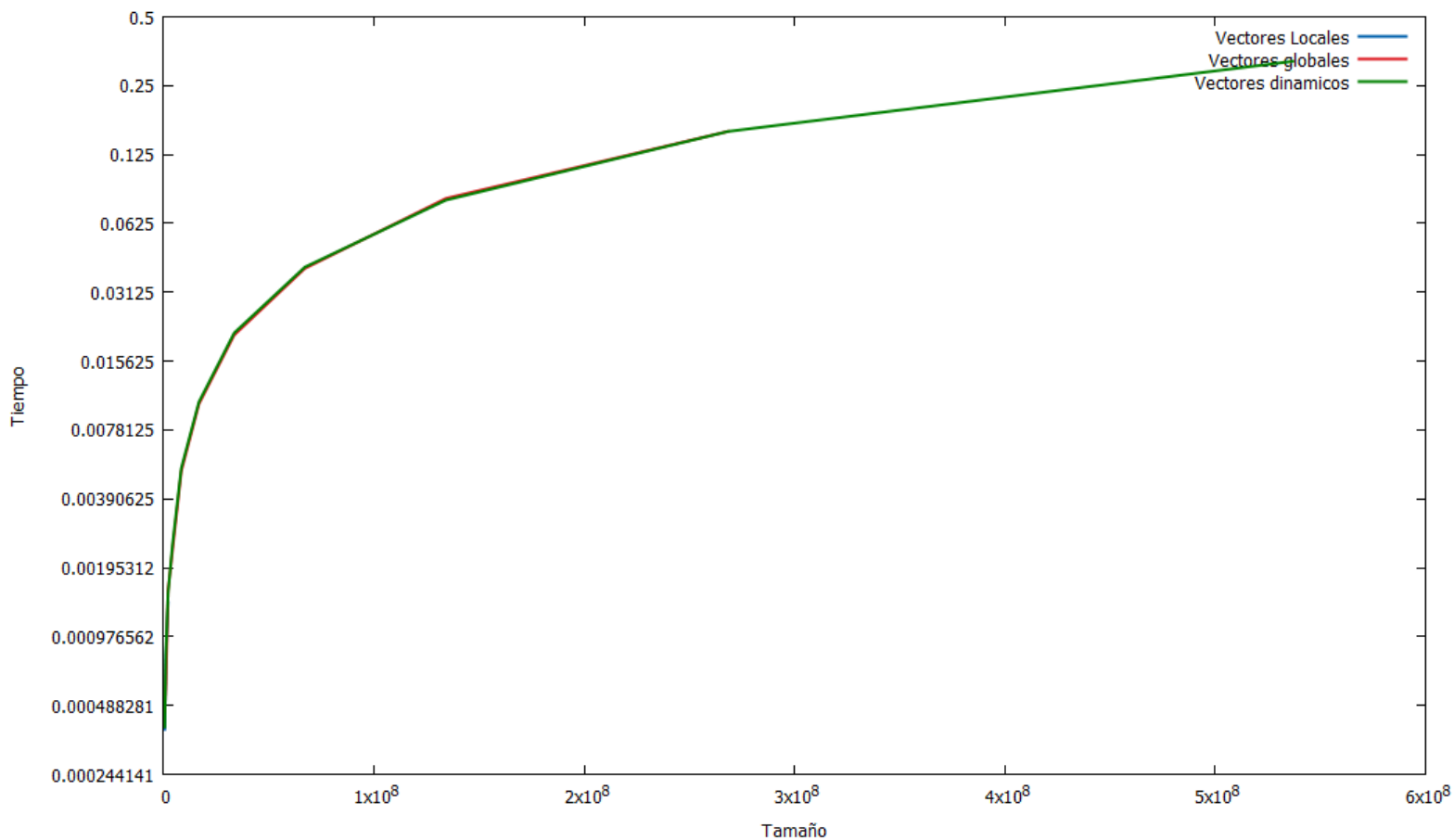
RESPUESTA: Usando vectores globales o dinámicos no se obtiene ningún error, debido a que se usa memoria dinámica para almacenar los vectores, es decir, espacio en el `heap` cuyo tamaño puede cambiar en tiempo de ejecución. El procesador puede pedir memoria en cualquier momento al sistema operativo para aumentar el espacio de direcciones del programa.

9. Rellenar una tabla como la Tabla 1 para `atcgrid` y otra para el PC local con los tiempos de ejecución obtenidos en los ejercicios anteriores para el trozo de código que realiza la suma de vectores. En la columna “Bytes de un vector” hay que poner el total de bytes reservado para un vector. Ayudándose de una hoja de cálculo represente en una misma gráfica los tiempos de ejecución obtenidos en `atcgrid` para vectores locales, globales y dinámicos (eje y) en función del tamaño en bytes de un vector (eje x). Utilice escala logarítmica en el eje de ordenadas (eje y) en todas las gráficas. ¿Hay diferencias en los tiempos de ejecución con vectores locales, globales y dinámicos?

RESPUESTA: El tamaño de los vectores se ha calculado multiplicando el tamaño de un `double` en memoria (8 bytes) por el número de componentes. El tiempo para vectores locales es ligeramente más rápido, ya que están almacenados en pila. La pila es más rápida porque el patrón de acceso es muy simple (un puntero se incrementa o se decrementa). Por contra, el `heap` es mucho más complejo en cuanto a localización, ya que puede redimensionarse continuamente. Además, cada byte de la pila tiende a ser reutilizado muy frecuentemente, por lo que suele estar mapeado por el caché del procesador, dotándolos de un acceso muy rápido. Por último, otro golpe de rendimiento en contra del `heap`, a la hora del `multi-threading`, es que normalmente cada redimensión debe estar sincronizada con todos los otros accesos al `heap` en el programa.

Tabla 1. Tiempos de ejecución de la suma de vectores para vectores locales, globales y dinámicos para `atcgrid`

Nº de Componentes	Bytes de un vector	Tiempo para vect. locales	Tiempo para vect. globales	Tiempo para vect. dinámicos
65536	524288	0.000383895	0.000459684	0.000393599
131072	1048576	0.000772432	0.000605075	0.000818920
262144	2097152	0.001375368	0.001558701	0.001515751
524288	4194304	-	0.002313120	0.002443414
1048576	8388608	-	0.005147435	0.005309428
2097152	16777216	-	0.010150372	0.010359613
4194304	33554432	-	0.020307066	0.020753875
8388608	67108864	-	0.039686577	0.040227559
16777216	134217728	-	0.080375351	0.078965181
33554432	268435456	-	0.157792891	0.157723300
67108864	536870912	-	-	0.319388779



Como vemos en la gráfica, prácticamente los tiempos se superponen unos a otros ya que las diferencias son mínimas, pero los vectores locales son más rápidos que los dinámicos y globales.

10. Modificar el código fuente C para que el límite de los vectores cuando se declaran como variables globales sea igual al máximo número que se puede almacenar en la variable N ($\text{MAX}=2^{32}-1$). Generar el ejecutable usando variables globales. ¿Qué ocurre? ¿A qué es debido? Razone además por qué el máximo número que se puede almacenar en N es $2^{32}-1$.

RESPUESTA: El máximo número que se puede almacenar en N es $2^{32}-1$ porque este es el máximo valor que puede tomar un entero con signo en arquitecturas de 32 bits, es decir, el máximo número representable con 32 bits.