2º curso / 2º cuatr.

Grado Ingeniería Informática

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 0. Entorno de programación

Estudiante (nombre y apellidos): Raúl Rodríguez Pérez
Grupo de prácticas y profesor de prácticas: C1
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Parte I. Ejercicios basados en los ejemplos del seminario práctico

Crear el directorio con nombre `bp0` en atcgrid y en el PC local.

NOTA: En las prácticas se usa slurm como gestor de colas. Consideraciones a tener en cuenta:

- Slurm está configurado para asignar recursos a los procesos (llamados *tasks* en slurm) a nivel de core físico. Esto significa que por defecto slurm asigna un core a un proceso, para asignar más de uno se debe usar con `sbatch/srun` la opción `--cpus-per-task`.
- En slurm, por defecto, `cpu` se refiere a cores lógicos (ej. en la opción `--cpus-per-task`), si no se quieren usar cores lógicos hay que añadir la opción `--hint=nomultithread` a `sbatch/srun`.
- Para asegurar que solo se crea un proceso hay que incluir `-n1` en `sbatch/srun`.
- Para que no se ejecute más de un proceso en un nodo de atcgrid hay que usar `--exclusive` con `sbatch/srun` (se recomienda no utilizarlo en los `srun` dentro de un script).
- Los `srun` dentro de un script heredan las opciones fijadas en el `sbatch` que se usa para enviar el script a la cola slurm.

1. Ejecutar `lscpu` en el PC y en un nodo de cómputo de atcgrid. (Crear directorio `ejer1`)

(a) Mostrar con capturas de pantalla el resultado de estas ejecuciones.

RESPUESTA:

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 0$ lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
Address sizes:                       39 bits physical, 48 bits virtual
CPU(s):                              12
Lista de la(s) CPU(s) en línea:      0-11
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              6
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              158
Nombre del modelo:                   Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
Revisión:                            10
CPU MHz:                             800.052
CPU MHz máx.:                        4500,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5199.98
Virtualización:                      VT-x
Caché L1d:                           192 KiB
Caché L1i:                           192 KiB
```

```
[c1estudiante4@atcgrid ~]$ srun lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU           E5645  @ 2.40GHz
Stepping:              2
CPU MHz:               1600.000
CPU max MHz:           2401,0000
CPU min MHz:           1600,0000
BogoMIPS:              4800.38
Virtualization:        VT-x
L1d cache:             32K
```

(b) ¿Cuántos cores físicos y cuántos cores lógicos tienen los nodos de cómputo de atcgrid y el PC? Razonar las respuestas

RESPUESTA:

Un nodo de cómputo del atcgrid tiene, como podemos ver en la segunda imagen, 12 cores físicos, los cuales vienen de hacer el producto del número de sockets (2 sockets) por el número de cores por sockets (6 cores por socket) y 24 cores lógicos, los cuales viene de hacer el producto del número de cores físicos por el número de threads por core (2 threads por core), y el PC, si observamos la primera imagen y hacemos los mismos cáculos que en el caso anterior, podemos ver que tiene 6 cores físicos y 12 cores lógicos.

2. Compilar y ejecutar en el PC el código `HelloOMP.c` del seminario (recordar que se debe usar un directorio independiente para cada ejercicio dentro de `bp0` que contenga todo lo utilizado, implementado o generado durante el desarrollo del mismo, para el presente ejercicio el directorio sería `ejer2`, como se indica en las normas de prácticas).

(a) Adjuntar capturas de pantalla que muestren la compilación y ejecución en el PC.

RESPUESTA:

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 0/0/ejer2$ gcc -O2 -fopenmp -o HelloOMP HelloOMP.c
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 0/0/ejer2$ ./HelloOMP
(2:!!!Hello world!!!)(4:!!!Hello world!!!)(7:!!!Hello world!!!)(5:!!!Hello world
!!!)(6:!!!Hello world!!!)(10:!!!Hello world!!!)(0:!!!Hello world!!!)(9:!!!Hello
world!!!)(8:!!!Hello world!!!)(3:!!!Hello world!!!)(11:!!!Hello world!!!)(1:!!!H
ello world!!!)usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda
Adjudicación/P
ráctica 0/0/ejer2$
```

(b) Justificar el número de "Hello world" que se imprimen en pantalla en ambos casos teniendo en cuenta la salida que devuelve `lscpu`.

RESPUESTA:

Como podemos ver en la imagen, se han impreso por pantalla 12 "Hello world", ya que se imprime un mensaje por cada hebra que esté trabajando, es decir por cada core lógico, como vimos anteriormente en la ejecución de la

orden lscpu, nuestro PC tiene 12 cores lógicos numerados del 0 al 11, los cuales coinciden con los de la ejecución de HelloOMP.

3. Copiar el ejecutable de `HelloOMP.c` que ha generado anteriormente y que se encuentra en el directorio `ejer2 del` PC al directorio `ejer2` de su home en el *front-end* de atcgrid. Ejecutar este código en un nodo de cómputo de atcgrid a través de cola ac del gestor de colas (no use ningún *script*) utilizando directamente en línea de comandos:

(a) `srun -p ac –n1 --cpus-per-task=12 --hint=nomultithread HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

```
[c1estudiante4@atcgrid ejer2]$ srun -p ac -n1 --cpus-per-task=12 --hint=nomulti
thread HelloOMP
(0:!!!Hello world!!!)(10:!!!Hello world!!!)(4:!!!Hello world!!!)(9:!!!Hello wor
ld!!!)(7:!!!Hello world!!!)(6:!!!Hello world!!!)(5:!!!Hello world!!!)(11:!!!Hel
lo world!!!)(8:!!!Hello world!!!)(1:!!!Hello world!!!)(3:!!!Hello world!!!)(2:!
!!Hello world!!!)[c1estudiante4@atcgrid ejer2]$
```

(b) `srun -p ac -n1 --cpus-per-task=24 HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

```
[c1estudiante4@atcgrid ejer2]$ srun -p ac -n1 --cpus-per-task=24 HelloOMP
(22:!!!Hello world!!!)(7:!!!Hello world!!!)(11:!!!Hello world!!!)(19:!!!Hello w
orld!!!)(23:!!!Hello world!!!)(13:!!!Hello world!!!)(8:!!!Hello world!!!)(18:!!
!Hello world!!!)(12:!!!Hello world!!!)(14:!!!Hello world!!!)(10:!!!Hello world!
!!)(15:!!!Hello world!!!)(4:!!!Hello world!!!)(9:!!!Hello world!!!)(3:!!!Hello
world!!!)(1:!!!Hello world!!!)(0:!!!Hello world!!!)(21:!!!Hello world!!!)(16:!!
!Hello world!!!)(5:!!!Hello world!!!)(2:!!!Hello world!!!)(20:!!!Hello world!!!
)(6:!!!Hello world!!!)(17:!!!Hello world!!!)[c1estudiante4@atcgrid ejer2]$
```

(c) `srun -p ac –n1 HelloOMP`

Adjuntar capturas de pantalla que muestren el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

```
[c1estudiante4@atcgrid ejer2]$ srun -p ac -n1 HelloOMP
(0:!!!Hello world!!!)(1:!!!Hello world!!!)[c1estudiante4@atcgrid ejer2]$
```

(d) ¿Qué orden `srun` usaría para que HelloOMP utilice los 12 cores físicos de un nodo de cómputo de atcgrid (se debe imprimir un único mensaje desde cada uno de ellos, en total, 12)?

RESPUESTA:

Utilizaría la orden “ `srun -p ac –n1 --cpus-per-task=12 --hint=nomultithread HelloOMP`”.

4. Modificar en su PC `HelloOMP.c` para que se imprima “world” en un `printf` distinto al usado para “Hello”, en ambos `printf` se debe imprimir el identificador del thread que escribe en pantalla. Nombrar al código resultante `HelloOMP2.c`. Compilar este nuevo código en el PC y ejecutarlo. Copiar el fichero ejecutable resultante al front-end de atcgrid (directorio `ejer4`). Ejecutar el código en un nodo de cómputo de atcgrid usando el script `script_helloomp.sh` del seminario (el nombre del ejecutable en el script debe ser `HelloOMP2`).

(a) Utilizar: `sbatch -p ac –n1 --cpus-per-task=12 --hint=nomultithread script_helloomp.sh`. Adjuntar capturas de pantalla que muestren el nuevo código, la compilación, el envío a la cola de la ejecución y el resultado de esta ejecución tal y como la devuelve el gestor de colas.

RESPUESTA:

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

int main(void) {

#pragma omp parallel
  printf("(%d:!!!Hello!!!)",
         omp_get_thread_num());

  printf("(%d:!!!world!!!)",
         omp_get_thread_num());

return(0);

}
```

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/Práctica 0/0/ejer4$ gcc -O2 -fopenmp -o HelloOMP2 HelloOMP2.c
```

```
[c1estudiante4@atcgrid ejer4]$ cat slurm-8157.out
############################################################################
Id. usuario del trabajo: c1estudiante4
Id. del trabajo: 8157
Nombre del trabajo especificado por usuario: helloomp
Directorio de trabajo (en el que se ejecuta el script): /home/c1estudiante4/ejer4
Cola: ac
Host de envío del trabajo:atcgrid
Nº de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 24
############################################################################
Nº de threads inicial: 12
Directorio de trabajo: /home/c1estudiante4/ejer4

1.Para 12 threads:
(0:!!!Hello!!!)(3:!!!Hello!!!)(6:!!!Hello!!!)(2:!!!Hello!!!)(8:!!!Hello!!!)(10:!!!Hello!!!)(7:!!!Hello!!!)(9:!!!Hello!!!)(11:!!!Hello!!!)(5:!!!Hello!!!)(4:!!!Hello!!!)(1:!!!Hello!!!)(0:!!!world!!!)
1.Para 6 threads:
(1:!!!Hello!!!)(2:!!!Hello!!!)(3:!!!Hello!!!)(4:!!!Hello!!!)(0:!!!Hello!!!)(5:!!!Hello!!!)(0:!!!world!!!)
1.Para 3 threads:
(1:!!!Hello!!!)(0:!!!Hello!!!)(2:!!!Hello!!!)(0:!!!world!!!)
1.Para 1 threads:
(0:!!!Hello!!!)(0:!!!world!!!)[c1estudiante4@atcgrid ejer4]$
```

(b) ¿Qué nodo de cómputo de atcgrid ha ejecutado el script? Explicar cómo ha obtenido esta información.

RESPUESTA:

El script ha sido ejecutado por el nodo 1 del atcgrid, esto lo sabemos gracias a la captura de pantalla de la ejecución del script en la cual se nos muestra que el nodo asignado al trabajo es el nodo "atcgrid1".

NOTA: Utilizar siempre con `sbatch` las opciones `–n1` y `--cpus-per-task`, `--exclusive` y, para usar cores físicos y no lógicos, no olvide incluir `--hint=nomultithread`. Utilizar siempre con `srun`, si lo usa fuera de un script, las opciones `–n1` y `--cpus-per-task` y, para usar cores físicos y no lógicos, no olvide incluir `--hint=nomultithread`. Recordar que los `srun` dentro de un script heredan las opciones utilizadas en el `sbatch` que se usa para enviar el script a la cola slurm. Se recomienda usar `sbatch` en lugar de `srun` para enviar trabajos a ejecutar a través slurm porque éste último deja bloqueada la ventana hasta que termina la ejecución, mientras que usando `sbatch` la ejecución se realiza en segundo plano.

Parte II. Resto de ejercicios

5. Generar en el PC 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`). El comentario inicial del código muestra la orden para compilar (siempre hay que usar `–O2` al compilar como se indica en las normas de prácticas). Incorporar volcados de pantalla que demuestren la compilación y la ejecución correcta del código en el PC (leer lo indicado al respecto en las normas de prácticas).

RESPUESTA:

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 0/0/ejer5$ gcc -O2 SumaVectoresC.c -o SumaVectoresC -lrt
```

```
usuario@usuario-GF63-Thin-9SC:~/Escritorio/Segundo GII/AC/Segunda Adjudicación/P
ráctica 0/0/ejer5$ ./SumaVectoresC 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000000440       / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.0
00000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
```

6. En el código del Listado 1 se utiliza la función `clock_gettime()` para obtener el tiempo de ejecución del trozo de código que calcula la suma de vectores. El código se imprime la variable `ncgt`,
(a) ¿qué contiene esta variable?
RESPUESTA:
La variable ncgt contiene la duración en segundos del cálculo de la suma de los vectores.

(b) ¿en qué estructura de datos devuelve `clock_gettime()` la información de tiempo (indicar el tipo de estructura de datos, describir la estructura de datos, e indicar los tipos de datos que usa)?

RESPUESTA:

clock_gettime() devuelve la información en la estrucuta de datos "timespec", la cual encapsula un intervalo de tiempo en segundos y nanosegundos, y además usa los tipos de datos "time_t" para representar los segundos y "syscall_slong_t" para representar los nanosegundos.

(c) ¿qué información devuelve exactamente la función `clock_gettime()`en la estructura de datos descrita en el apartado (b)? ¿qué representan los valores numéricos que devuelve?

RESPUESTA:

La función clock_gettime() devuelve el instante de tiempo actual en el que fue ejecutada dicha función, representada en segundos y nanosegundos.

7. Rellenar una tabla como la Tabla 1 en una hoja de cálculo con los tiempos de ejecución del código del Listado 1 para vectores locales, globales y dinámicos. Obtener estos resultados usando scripts (partir del script que hay en el seminario). Debe haber una tabla para atcgrid y otra para su PC en la hoja de cálculo. En la columna "Bytes de

un vector" hay que poner el total de bytes reservado para un vector. (NOTA: Se recomienda usar en la hoja de cálculo el mismo separador para decimales que usan los códigos al imprimir. Este separador se puede modificar en la hoja de cálculo.)

RESPUESTA:

Tabla 1 . Copiar la tabla de la hoja de cálculo utilizada

Nº de Componentes	Bytes de un vector	Tiempo para vect. locales	Tiempo para vect. globales	Tiempo para vect. dinámicos
65536				
131072				
262144				
524288				
1048576				
2097152				
4194304				
8388608				
16777216				
33554432				
67108864				

1. Con ayuda de la hoja de cálculo representar **en una misma gráfica** los tiempos de ejecución obtenidos en atcgrid y en su PC para vectores locales, globales y dinámicos (eje y) en función del tamaño en bytes de un vector (por tanto, los valores de la segunda columna de la tabla, que están en escala logarítmica, deben estar en el eje x). Utilizar escala logarítmica en el eje de ordenadas (eje y). ¿Hay diferencias en los tiempos de ejecución?

   RESPUESTA:

2. **(a)** Cuando se usan vectores locales, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

   RESPUESTA:

   (b) Cuando se usan vectores globales, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

   RESPUESTA:

   No se obtiene ningún error.

   (c) Cuando se usan vectores dinámicos, ¿se obtiene error para alguno de los tamaños?, ¿a qué cree que es debido lo que ocurre? (Incorporar volcados de pantalla como se indica en las normas de prácticas)

   RESPUESTA:

   No se obtiene ningún error.

3. **(a)** ¿Cuál es el máximo valor que se puede almacenar en la variable N teniendo en cuenta su tipo? Razonar respuesta.
   RESPUESTA:
   Como N es una variable de tipo unsigned int el valor máximo que podrá tomar es 2^32-1, ya que una variable int ocupa 4B que serían 32 bits, por lo tanto podríamos representar desde el 0 hasta el 2^32-1.

   (b) Modificar el código fuente C (en el PC) 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 y generar el ejecutable. ¿Qué ocurre? ¿A qué es debido? (Incorporar volcados de pantalla que muestren lo que ocurre)
   RESPUESTA:

Entrega del trabajo

Leer lo indicado en las normas de prácticas sobre la entrega del trabajo del bloque práctico en SWAD.

Listado 1 . Código C que suma dos vectores

```
/* SumaVectoresC.c
 Suma de dos vectores: v3 = v1 + v2

 Para compilar usar (-lrt: real time library, no todas las versiones de gcc necesitan que se incluya
-lrt):
         gcc  -O2 SumaVectores.c -o SumaVectores –lrt
         gcc  -O2 –S SumaVectores.c –lrt   //para generar el código ensamblador

 Para ejecutar use: SumaVectoresC longitud
*/

#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>       // biblioteca donde se encuentra la función clock_gettime()

//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL    // descomentar para que los vectores sean variables ...
                          // locales (si se supera el tamaño de la pila se ...
                          // generará el error "Violación de Segmento")
//#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
                          // globales (su longitud no estará limitada por el ...
                          // tamaño de la pila del programa)
#define VECTOR_DYNAMIC    // descomentar para que los vectores sean variables ...
                          // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432         //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){

   int i;
   struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución

   //Leer argumento de entrada (nº de componentes del vector)
   if (argc<2){
      printf("Faltan nº componentes del vector\n");
      exit(-1);
   }

   unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
   #ifdef VECTOR_LOCAL
   double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
                                 // disponible en C a partir de actualización C99
   #endif
   #ifdef VECTOR_GLOBAL
   if (N>MAX) N=MAX;
   #endif
   #ifdef VECTOR_DYNAMIC
```

```
   double *v1, *v2, *v3;
   v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
   v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
   v3 = (double*) malloc(N*sizeof(double));
      if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
      printf("Error en la reserva de espacio para los vectores\n");
      exit(-2);
   }
   #endif

   //Inicializar vectores
   for(i=0; i<N; i++){
      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
   }

   clock_gettime(CLOCK_REALTIME,&cgt1);
   //Calcular suma de vectores
   for(i=0; i<N; i++)
      v3[i] = v1[i] + v2[i];

   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
          (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

   //Imprimir resultado de la suma y el tiempo de ejecución
   if (N<10) {
   printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
   for(i=0; i<N; i++)
      printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
                i,i,i,v1[i],v2[i],v3[i]);
   }
   else
      printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /
                V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
                ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);


   #ifdef VECTOR_DYNAMIC
   free(v1); // libera el espacio reservado para v1
   free(v2); // libera el espacio reservado para v2
   free(v3); // libera el espacio reservado para v3
   #endif
   return 0;
}
```