2º curso / 2º cuatr.

Grado en Ing. Informática

ETSIIT
Universidad de Granada

Prácticas de Arquitectura de Computadores

Programación paralela

Material elaborado por los profesores responsables de la asignatura:
Mancia Anguita, Julio Ortega

0 Contenido

En este guion de prácticas podrá encontrar información general sobre las prácticas de la asignatura Arquitectura de Computadores (AC) y, en particular, encontrará las prácticas de programación paralela de AC. AC es una asignatura de rama del Grado de Ingeniería Informática que se imparte en el segundo cuatrimestre del segundo curso (cuarto semestre). Este guion se divide en siete secciones: una introducción (Sección 1), una sección dedicada a los cuatro bloques prácticos (del 0 al 3) que componen las prácticas de programación paralela (Secciones 2 a 5) y una última sección con la bibliografía recomendada (Sección 6).

En la sección de introducción podrá encontrar el objetivo general que se persigue con las prácticas de AC (Apartado 1.1), información sobre la plataformas que se va a utilizar en las aulas de prácticas y sobre las que en general se pueden utilizar para desarrollar las prácticas (Apartado 1.2), la planificación y estructura de las mismas (Apartado 1.3) e información sobre cómo se va a evaluar y calificar la adquisición de objetivos (Apartado 1.4).

En las secciones dedicadas a bloques prácticos (Secciones 2 a 5) encontrará un primer apartado con los objetivos que se persiguen con el bloque y que, por tanto, se van a evaluar, un segundo apartado que describe a grandes rasgos los contenidos del seminario del bloque, y un último apartado con el trabajo que debe desarrollar y entregar para su evaluación.

La Sección 6 contiene la bibliografía recomendada dividida en varios apartados. Las referencias, que pueden encontrar en esta sección, son direcciones web y algunos libros disponibles en la Biblioteca de la ETSIIT. Con estas referencias puede resolver dudas y ampliar conocimientos relacionados con las herramientas que se van a utilizar en prácticas (Apartados 6.1 a 6.5) y consultar contenidos y ampliar conocimientos sobre programación paralela (Apartado 6.6).

1 Introducción

1.1 Objetivo general de las prácticas

Las prácticas de AC persiguen como objetivo general que el estudiante sea capaz de programar aplicaciones con un buen rendimiento en computadores personales y servidores de gama baja (Figura 1); es decir, se trata de conseguir código para estas plataformas con una buena relación entre las prestaciones (tiempo de respuesta, calidad de las salidas y funcionalidades) y el coste que supone conseguirlas. La programación con un buen rendimiento del resto de computadores del mercado (Figura 1) se cubre en una parte con los contenidos abordados en AC (hay características de los componentes, que aquí se aprovechan, de PC y servidores básicos, que están presentes también en otros niveles del mercado) y, principalmente, con las dos materias de la especialidad de Ingeniería de Computadores (IC): Sistemas de Cómputo de Altas Prestaciones (asignaturas Arquitecturas y Computación de Altas Prestaciones, Centros de Procesamiento de Datos y Arquitectura de Sistemas) y Sistemas de Cómputo para Aplicaciones Específicas (asignaturas Desarrollo de Hardware Digital, Sistemas con Microprocesadores y Sistemas Empotrados). La primera materia estudia sistemas de gama media y alta y, la segunda, sistemas empotrados.

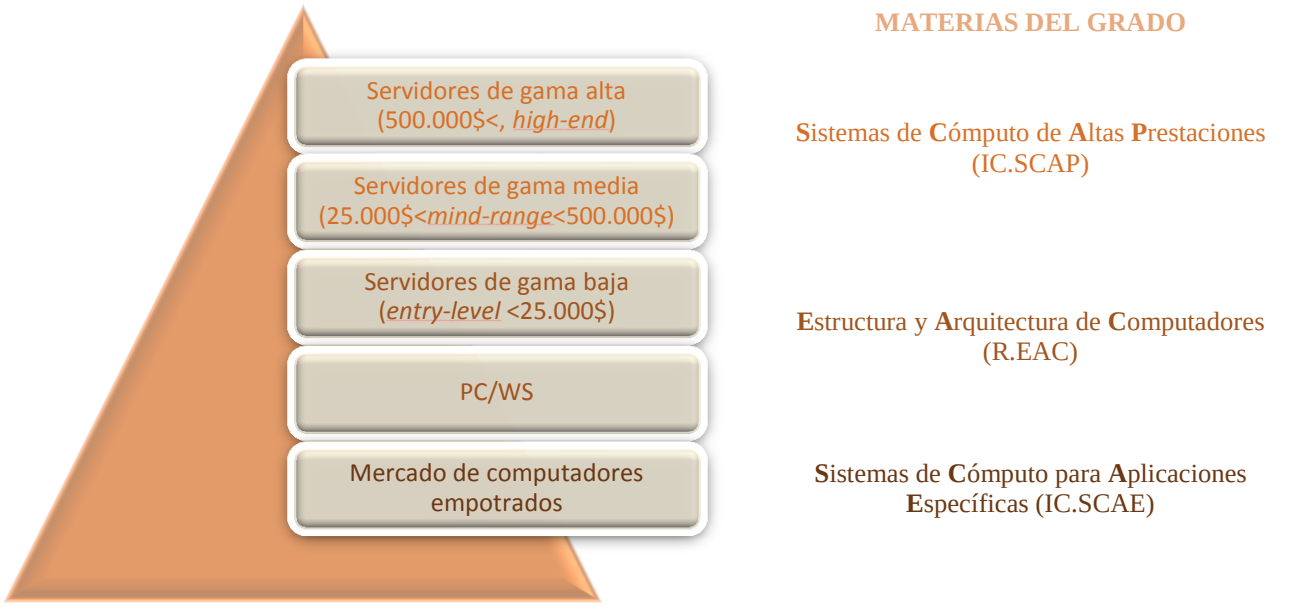

Figura 1. Pirámide del mercado de computadores (WS: *Workstation*, IC: Ingeniería de Computadores, R: Rama). Conforme se sube hacia arriba en la pirámide disminuye el volumen de venta y aumenta el precio.

1.2 Plataforma de prácticas

Para el desarrollo de las prácticas se usarán los PC del aula de prácticas y un pequeño cluster de servidores (atcgrid).

En los PC del aula de prácticas se usará la distribución de Linux ([1], [2]) Ubuntu de 64 bits ([3], [5]) con procesadores de 64 bits. Se sugiere usar el entorno de ventanas GNOME [6] instalado en este sistema operativo (para iniciar el entorno en las aulas de prácticas se usará `startx`). Se trabajará con las siguientes herramientas disponibles en el aula de prácticas bajo Ubuntu de 64 bits:

- ***C/C++.*** El código secuencial se escribirá en C ([13], [14], [15]), aunque se comparará un código sencillo en C con su equivalente en C++ ([16], [17], [18]) en el Bloque Práctico 0. Para compilar código C/C++ en el aula se usarán los compiladores de GNU: gcc o *GNU Compiler Collection* ([19], [20]). Los ejecutables deben ser de 64 bits.

- ***OpenMP***. Es un estándar de facto para la programación paralela con el estilo/paradigma de variables compartidas ([21], [22], [23], [24], [25]) que se puede utilizar con C/C++ y Fortran. Esta herramienta de programación paralela, como se mostrará a lo largo de los bloques prácticos 1, 2 y 3, se basa en directivas y funciones (en la Lección 4/Tema 2 de AC se estudian los tipos de herramientas y los estilos/paradigmas de programación paralela). Para escribir código fuente paralelo con OpenMP se usará principalmente lenguaje C más la API (*Application Programming Interface*) de OpenMP. En particular se usará la implementación de OpenMP que viene integrada con el compilador de GNU desde su versión 4.1.1 (puede encontrar el manual de OpenMP de GNU en [20]).

Para editar los códigos fuente en C se puede usar, por ejemplo, `gedit`, disponible en el entorno de ventanas que se va a utilizar en prácticas. Otra alternativa sería usar el editor de un entorno de desarrollo para C/C++, se recomienda el entorno de desarrollo Eclipse ([26], [27], [28], [29] y [30]). Una ventaja de usar un entorno es que facilitan la depuración de errores porque incluye un depurador. Eclipse permite depurar un programa a alto nivel (en nuestro caso en C) y en ensamblador.

El cluster atcgrid (Figura 2) tiene instalado Red Hat de 64 bits. Consta de dos servidores (atcgrid1 y atcgrid2) con dos procesadores Intel® Xeon® E5645 (6 cores/12 threads, 12M L2 Cache compartida, 2.40 GHz cada core, 5.86 GT/s Intel® QPI) (http://ark.intel.com/products/48768?wapkw=(E5645)). Además dispone de un nodo que actúa como *front-end, master* o *host* (atcgrid.ugr.es). El cluster atcgrid ha sido financiado por el Programa de Apoyo a la Docencia Práctica de la Universidad de Granada a petición del ámbito de Arquitectura y Tecnología de Computadores.

Se puede acceder al cluster atcgrid mediante `ssh` (***s**ecure **sh**ell*) para ejecutar comandos y con `sftp` (***s**ecure **f**ile **t**ransfer **p**rotocol*) para transferir ficheros. El usuario que accede a atcgrid.ugr.es en realidad accede a su cuenta de usuario en el nodo *front-end* (es un computador que "da la cara"). Desde el *front-end* el usuario enviará trabajos a atcgrid1 y atcgrid2 a través de un gestor de recursos distribuidos, en particular, se usará el gestor TORQUE [11][12]). TORQUE está basado, al igual que otros sistemas de colas conocidos, en PBS (*Portable Batch System*). En el Seminario 0 del Bloque Práctico 0 hay información más detallada sobre el cluster atcgrid, el sistema de colas TORQUE y el acceso remoto a atcgrid de los usuarios.

Se puede acceder al *front-end,* atcgrid.ugr.es, desde cualquier punto dentro de la red de la Universidad de Granada, como por ejemplo desde los PC de las aulas de prácticas o desde su propio portátil personal, usando el campus virtual inalámbrico https://csirc.ugr.es/informatica/RedUGR/CVI/. Si accede a través del campus virtual inalámbrico utilice cviugr-v2 o, en su defecto, cviugr estableciendo una conexión VPN (https://csirc.ugr.es/informatica/RedUGR/CVI/ConectarCVI-UGR/). También puede acceder desde fuera de la

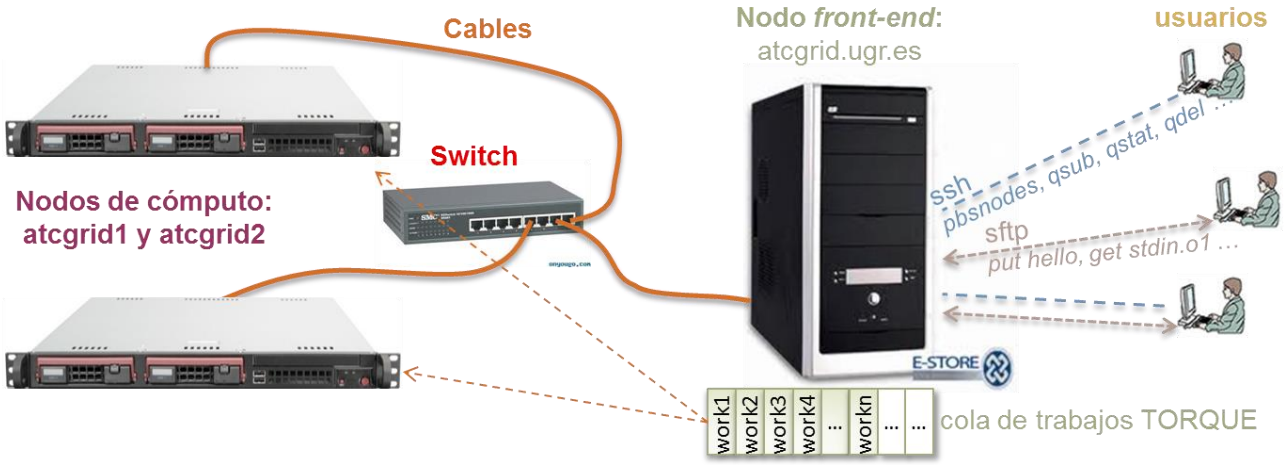

Figura 2. Componentes del cluster atcgrid (nodos de cómputo, *switch* y cables de interconexión) y nodo *front-end*. Los usuarios acceden al cluster (en particular al nodo *front-end*) de forma remota usando `ssh` y `sftp`. Desde el front-end se accede a los nodos de cómputo usando el gestor TORQUE.

red de la Universidad de Granada (desde casa, etc.) mediante conexión VPN. Para establecer una conexión VPN siga las instrucciones que se encuentran en la siguiente dirección web: https://csirc.ugr.es/informatica/RedUGR/VPN/. Para realizar una conexión a cvigur-v2 o a cviugr siga las instrucciones que se encuentran en la siguiente dirección web: https://csirc.ugr.es/informatica/RedUGR/CVI/ConectarCVI-UGR/.

Para acceder al *front-end,* atcgrid.ugr.es, desde Windows puede utilizar:

- Un cliente `ssh`, como por ejemplo `putty`, para ejecutar comandos (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html).
- Un cliente `sftp`, como por ejemplo `FileZilla` (http://filezilla-project.org/download.php) o `psftp` (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) para cargar y descargar ficheros.

Para acceder al *front-end,* atcgrid.ugr.es, desde Linux puede utilizar:

- `ssh` –X usuario@atcgrid.ugr.es (`man ssh` para más información) para ejecutar comandos (se usa –X para exportar gráficos).
- `sftp` usuario@atcgrid.ugr.es (`man sftp` para más información) para cargar y descargar ficheros.

No use en ningún caso máquinas virtuales al realizar las prácticas, los resultados obtenidos en máquinas virtuales llevan frecuentemente a interpretaciones erróneas.

1.3 Planificación y estructura de las prácticas de AC

Las prácticas se organizan en cinco bloques, del Bloque Práctico 0 al 4:

- Bloque Práctico 0. Entorno de programación: cluster atcgrid y gestor TORQUE
- Bloque Práctico 1. Programación paralela I: Directivas OpenMP
- Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP
- Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP
- Bloque Práctico 4. Optimización de código

Los bloques de 0 a 3 están incluidos en este guion de prácticas, el bloque 4 tiene su propio guion. Cada bloque práctico consta de un Seminario y de varios ejercicios. Estos últimos se realizarán en clase presencial y en casa. En la sección correspondiente a cada bloque se especifican los objetivos que se pretenden conseguir con el seminario y los ejercicios del bloque.

Al bloque práctico 0 se dedicará una semana. El resto de prácticas se pueden desarrollar en dos o tres semanas. La Figura 3 presenta una planificación que, por ser para todos los grupos de prácticas, es necesariamente **aproximada** (por ejemplo, los días festivos no coinciden ni en número ni temporalmente en todos los grupos de prácticas). Los seminarios (S en la figura) se presentarán en una clase presencial o troceados en varias clases presenciales dependiendo de la preferencia del profesor de prácticas. En clase presencial, durante las semanas dedicadas a un bloque, los estudiantes realizarán parte del trabajo del bloque práctico y el profesor de prácticas resolverá las dudas que puedan surgir en el grupo de prácticas (representado con P en la figura). El resto de trabajo lo realizarán los estudiantes en casa y el resto de dudas las resolverá el profesor en horario de tutorías (en la plataforma docente y en la web de grados se puede consultar el lugar y el horario de las tutorías de los profesores).

Todos los códigos fuente implementados, los resultados de su ejecución y la contestación a las preguntas deberán aparecer en el cuaderno de trabajo del bloque práctico que se entregará en la plataforma docente para la evaluación por parte de los profesores de prácticas.

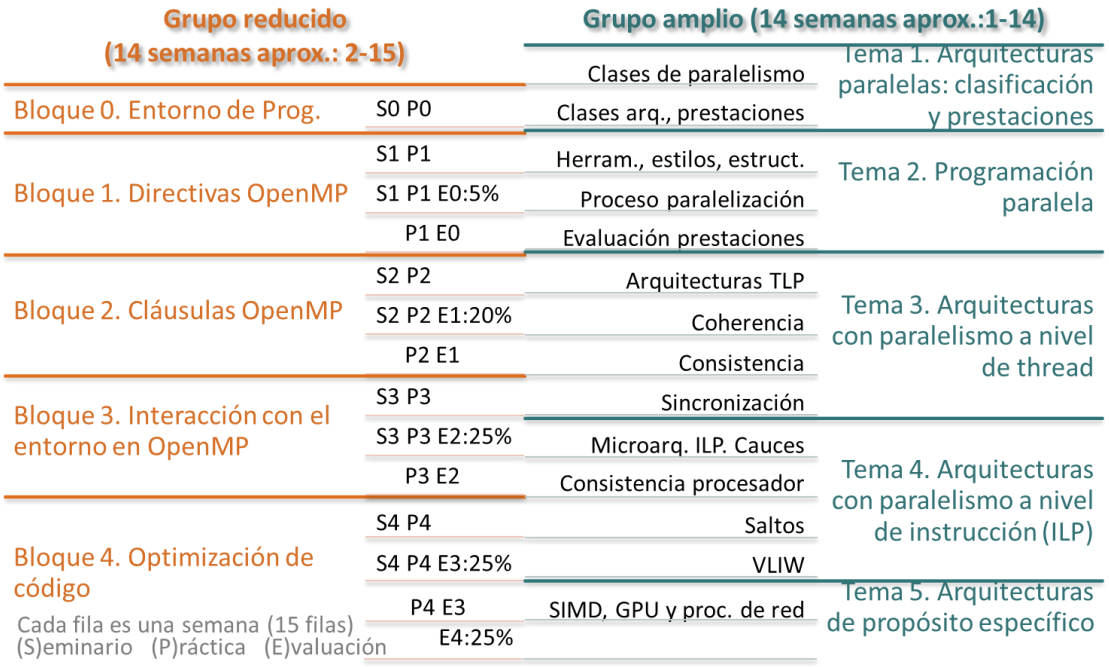

Figura 3. Planificación **aproximada** para todos los grupos. (S)eminario (P)ráctica (E)valuación. Cada fila representa una semana (total 15 semanas,1-15)

1.4 Evaluación de las prácticas

Las prácticas suponen **4 puntos** (sobre 10) de la puntuación final de la asignatura. Para poder aprobar la asignatura se necesita obtener un **mínimo 1.6 puntos** en las prácticas. Para la calificación de las prácticas se tendrá en cuenta en general las calificaciones obtenidas por:

- **Evaluación continua** durante el cuatrimestre (50%, **2 puntos**):
  - Cuaderno de trabajo en formato electrónico de los diferentes bloques.
  - Defensa del trabajo realizado en el aula (se suma a la puntuación del cuaderno).
- **Examen escrito** a realizar junto con el examen de teoría en las convocatorias de junio, septiembre y diciembre (50%, **2 puntos**).

Los estudiantes que **no se hayan sometido a evaluación continua** de los Bloques Prácticos 1 a 4 serán evaluados en **septiembre** y **diciembre** (no en junio) mediante un examen escrito que puntuará sobre **4 puntos**. Excepto en el caso que se acaba de comentar, el cuaderno y la defensa suponen un 50% y el examen escrito un 50% del total de la calificación. La proporción se justifica en base a que durante la evaluación continua el alumno está en fase de aprendizaje, por ese motivo el porcentaje de la evaluación continua no supera el 50%. La proporción en la que afecta cada una de las cinco evaluaciones continuas realizadas durante el cuatrimestre se muestra en la Figura 3 (E0 a E4): el bloque práctico 0 supone un 5 %, el 1 un 20% y el resto de bloques un 25% de los 2 puntos de la evaluación continua.

El cuaderno de trabajo de cada bloque práctico, que el estudiante ha ido confeccionando **en formato electrónico** durante las semanas de trabajo del bloque, se entregará en **pdf** en la plataforma docente SWAD (en *Evaluación->Mis Trabajos*) en la fecha que fijen los profesores de prácticas. El pdf debe contener:

- todos los códigos implementados,
- todos los resultados de ejecución y
- la contestación a todas las preguntas.

Además, aparte de este pdf, se entregarán los ficheros con los códigos fuente. El pdf y los códigos fuente se entregarán en un fichero que **necesariamente** debe ser **zip**. El nombre del fichero pdf debe ser **BloquePracticoX_Apellido1Apellido2Nombre.pdf** y el nombre del fichero zip **BloquePracticoX_Apellido1Apellido2Nombre.zip**, donde X es el número del bloque práctico (0, 1, 2, 3 o 4). Se recomienda usar un software gratuito para generar el pdf; por ejemplo, podría utilizar PDFCreator o, si usa OpenOffice para editar su cuaderno de trabajo, la opción *Archivo->Exportar en formato PDF* (OpenOffice está disponible en las aulas de prácticas).

Los profesores comunicarán las fechas de entrega en las clases presenciales y mediante avisos en la plataforma docente (aparecen a la izquierda de la página en forma de *Post-it®*). Como regla general, siempre que no se especifique otra cosa, los grupos podrán realizar la entrega del trabajo de un bloque práctico hasta seis días después de la última sesión presencial del bloque práctico.

La **evaluación/defensa** en el aula se realizará durante sesiones presenciales de trabajo (notado como E en la Figura 3). Para contestar a las preguntas puede necesitar un computador y el cuaderno de trabajo de las prácticas realizadas hasta el momento; por tanto, deberá tener accesible este material en el aula de prácticas los días de evaluación. Al menos deberá enfrentarse a la defensa en el aula una vez durante el cuatrimestre. En el examen escrito de prácticas los profesores le podrán plantear cualquier pregunta que permita comprobar si ha adquirido uno o varios de los objetivos de los bloques prácticos. Tenga en cuenta que "*evaluar el trabajo*" significa "*comprobar que se han adquirido los objetivos del mismo*" y que, en las secciones de cada bloque práctico, se especifica cuáles son sus objetivos.

2 Bloque Práctico 0. Entorno de programación

2.1 Objetivos

Una vez finalizado el bloque debería ser capaz de (si observa que no alcanza alguno de estos objetivos pregunte las dudas que tenga a su profesor en clase presencial o en su despacho de tutorías en horario de tutorías):

1. Usar `ssh` (***sec**ure **sh**ell*) para conectarse desde un PC del aula a su home del nodo *front-end* del cluster atcgrid con el objetivo de ejecutar commandos.
2. Usar `sftp` (***s**ecure **f**ile **t**ransfer **p**rotocol*) para conectarse desde un PC del aula a su home del *front-end* de atcgrid con el objetivo de cargar y descargar ficheros.
3. Enviar desde el *front-end*, usando las colas de TORQUE, trabajos a los dos nodos servidores de atcgrid usando `qsub`, comprobar el estado del trabajo en las colas con `qstat`, borrar trabajos de las colas con `qdel` y leer los dos ficheros de resultados (errores y salida del trabajo ejecutado) que devuelve `qsub`.

2.2 Seminario

El seminario presenta las características de atcgrid y el uso de `ssh`, `sftp` y TORQUE para ejecutar trabajos en atcgrid.

2.3 Práctica o trabajo a desarrollar

2.3.1 Ejercicios basados en los ejemplos del seminario práctico

En clase presencial y en casa se reproducirá los ejemplos presentados en el seminario y se trabajarán los siguientes ejercicios y cuestiones:

1. En el primer ejemplo de ejecución en atcgrid usando TORQUE se ejecuta el ejemplo `HelloOMP.c` de la página 12 del seminario usando la siguiente orden: `echo 'hello/HelloOMP' | qsub -q ac`. El resultado de la ejecución de este código en atcgrid se puede ver en la página 19 del seminario. Conteste a las siguientes preguntas:
   a. ¿Para qué se usa en `qsub` la opción `-q`?
   b. ¿Cómo sabe el usuario que ha terminado la ejecución en atcgrid?
   c. ¿Cómo puede saber el usuario si ha habido algún error en la ejecución?
   d. ¿Cómo ve el usuario el resultado de la ejecución?
   e. ¿Por qué en el resultado de la ejecución aparecen 24 saludos "¡¡¡Hello World!!!"?
2. En el segundo ejemplo de ejecución en atcgrid usando TORQUE el script `script_helloomp.sh` de la página 22 del seminario 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 la página 26 del seminario. Conteste a las siguientes preguntas:
   a. ¿Por qué no acompaña a al orden `qsub` la opción `-q` en este caso?
   b. ¿Cuántas veces ejecuta el `script` el ejecutable `HelloOMP` en atcgrid?
   c. ¿Cuántos saludos "¡¡¡Hello World!!!" se imprimen en cada ejecución? ¿Por qué se imprime ese número?
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`.

Ejecute el script con estas modificaciones. ¿Qué resultados de ejecución se obtienen en este caso? Incorpore en su cuaderno de trabajo volcados de pantalla que muestren estos resultados.

2.3.2 Resto de ejercicios

4. Incorporar en el cuaderno de prácticas el contenido del fichero `/proc/cpuinfo` de atcgrid1 o de atcgrid2 (consultar seminario), del PC del aula de prácticas y de su PC (si tiene Linux instalado). Indique qué ha hecho para obtener el contenido de `/proc/cpuinfo` en atcgrid.

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++:

   $$v3 = v1 + v2;\ \ v3(i) = v1(i) + v2(i),\ \ i=0,\ldots N-1$$

   Los códigos utilizan directivas del compilador para fijar el tipo de variable de los vectores (v1, v2 y v3). 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`.

   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 devuelve la función `clock_gettime()`?

   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.

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.

7. Ejecutar en atcgrid el código generado en el apartado anterior usando el `script` del Listado 3 . Genere el ejecutable usando la opción de optimización –O2. 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?

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?

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). Realice otra gráfica con los tiempos obtenidos en el PC local.

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

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				

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 (MAX=2^32-1). Generar el ejecutable usando variables globales. ¿Qué ocurre?

Listado 1 . Código C que suma dos vectores

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

 Para compilar usar (-lrt: real time library):
 gcc  -O2 SumaVectores.c -o SumaVectores -lrt

 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()

//#define PRINTF_ALL// comentar para quitar el printf ...
                    // que imprime todos los componentes

//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 (n° de componentes del vector)
  if (argc<2){
    printf("Faltan n° 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
  #ifdef PRINTF_ALL
  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]);
  #endif

  #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;

}
```

Listado 2 . Código C++ que suma dos vectores

```
/* SumaVectoresCpp.cpp
 Suma de dos vectores: v3 = v1 + v2

 Para compilar usar (-lrt: real time library):
 g++  -O2 SumaVectoresCpp.cpp -o SumaVectoresCpp -lrt

 Para ejecutar use: SumaVectoresCpp longitud
*/

#include <cstdlib>  // biblioteca con atoi()
#include <iostream> // biblioteca donde se encuentra la función cout
using namespace std;
#include <time.h>  // biblioteca donde se encuentra la función clock_gettime()

//#define COUT_ALL // comentar para quitar el cout ...
                   // que imprime todos los componentes

//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){

  struct timespec cgt1,cgt2;    //para tiempo de ejecución

  //Leer argumento de entrada (n° de componentes del vector)
  if (argc<2){
    cout << "Faltan n° componentes del vector\n" << endl ;
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);
  #ifdef VECTOR_LOCAL
  double v1[N], v2[N], v3[N];
  #endif
  #ifdef VECTOR_GLOBAL
  if (N>MAX) N=MAX;
  #endif
  #ifdef VECTOR_DYNAMIC
  double *v1, *v2, *v3;
  v1 = new double [N];    //si no hay espacio suficiente new genera una excepción
  v2 = new double [N];
  v3 = new double [N];
  #endif

  //Inicializar vectores
  for(int 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(int i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];

  clock_gettime(CLOCK_REALTIME,&cgt2);
  double 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
  #ifdef COUT_ALL
  cout << "Tiempo(seg.):" << ncgt << "\t/ Tamaño Vectores:" << N << endl;
```

```
  for(int i=0; i<N; i++)
  cout << "/ V1[" << i << "]+V2[" << i << "]=V3" << i << "](" << v1[i] << "+" << v2[i] << "=" <<
v3[i] << ") /\t" << endl;
    cout <<"\n"<< endl;
  #else
    cout << "Tiempo(seg.):" << ncgt << "\t/ Tamaño Vectores:" << N << "\t/ V1[0]+V2[0]=V3[0](" <<
v1[0] << "+" << v2[0] << "=" << v3[0] << ") / / V1[" << N-1 << "]+V2[" << N-1 << "]=V3[" << N-1 <<
"](" << v1[N-1] << "+" << v2[N-1] << "=" << v3[N-1] << ")/\n" << endl;
  #endif

  #ifdef VECTOR_DYNAMIC
  delete [] v1; // libera el espacio reservado para v1
  delete [] v2; // libera el espacio reservado para v2
  delete [] v3; // libera el espacio reservado para v3
  #endif
  return 0;

}
```

Listado 3 . `Script` para la suma de vectores (`SumaVectores.sh`). Se supone en el `script` que el fichero a ejecutar se llama `SumaVectorC` y que se encuentra en el directorio en el que se ha ejecutado `qsub`.

```
#!/bin/bash
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N SumaVectoresC_vlocales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
#Se ejecuta SumaVectorC que está en el directorio bp0 para N potencia de 2 desde 2^16 a 2^26
for ((N=65536;N<67108865;N=N*2))
do
   $PBS_O_WORKDIR/SumaVectoresC $N
done
```

3 Bloque Práctico 1. Programación paralela I: Directivas OpenMP

3.1 Objetivos

En este bloque práctico se utilizarán las directivas OpenMP [21] para crear y terminar una región paralela, para realizar trabajo compartido y para sincronización/comunicación ([22], [23], [24], [25]). Una vez finalizado el bloque debería ser capaz de (si observa que no alcanza alguno de estos objetivos pregunte las dudas que tenga a su profesor en clase presencial o en su despacho de tutorías en horario de tutorías):

1. Definir OpenMP y caracterizarlo dentro de las herramientas de programación.
2. Distinguir entre los tres componentes de OpenMP: directivas, funciones y variables de entorno.
3. Escribir código fuente OpenMP añadiendo, a código fuente C, directivas y funciones OpenMP.
4. Generar código ejecutable paralelo a partir de código fuente OpenMP usando el compilador `gcc` desde la línea de comandos.
5. Escribir código fuente OpenMP a partir del cual se pueda generar código ejecutable secuencial (código fuente OpenMP portable).
6. Fijar, modificando variables de entorno, el número de *threads* que se va a utilizar en la ejecución de código ejecutable paralelo.
7. Distinguir entre directiva ejecutable y directiva declarativa.
8. Distinguir entre bloque estructurado, construcción y región.
9. Describir la acción que realiza y utilizar la directiva `parallel`.
10. Describir las acciones que realizan y utilizar las directivas de trabajo compartido de OpenMP `for`, `sections` y `single`.
11. Describir las acciones que realizan y utilizar las directivas para comunicación y sincronización `critical`, `atomic` y `barrier`.
12. Distinguir entre las directivas `single` y `master`.
13. Usar la directiva `parallel` y las directivas de trabajo compartido de OpenMP para paralelizar un código paralelo.
14. Generar el código ensamblador de un programa fuente usando la opción -S de `gcc`.
15. Obtener el tiempo de ejecución (*elapsed time*), tiempo de CPU, los MIPS y los MFLOPS de un código (consultar Lección 3/Tema 1 de AC).

3.2 Seminario

El seminario de este bloque práctico presenta las principales directivas de OpenMP usadas para escribir código paralelo ([21], [22], [23], [24], [25]).

3.3 Práctica o trabajo a desarrollar

3.3.1 Ejercicios basados en los ejemplos del seminario práctico

En clase presencial y en casa se ejecutarán los códigos presentados en el seminario y se trabajarán los siguientes ejercicios y cuestiones:

1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorpore el código fuente resultante a su cuaderno de prácticas.
2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque

estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

3.3.2 Resto de ejercicios

En clase presencial y en casa, usando plataformas (procesadores + SO) de 64 bits**,** se trabajarán los siguientes ejercicios y cuestiones:

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema para el ejecutable en atcgrid y en el PC local. Obtenga los tiempos para vectores con 10000000 componentes.

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC).

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1 , que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`**,** que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1 , que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

Tabla 2 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) ¿?threads/cores	T. paralelo (versión `sections`) ¿?threads/cores
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1 . Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución (*elapsed time*) del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos.

Tabla 3 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión `for` ¿? Threads/cores		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						

11. Rellenar una tabla como la Tabla 3 para atcgrid y otra para el PC local con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7, y para el programa secuencial del Listado 1 . Ponga en la tabla el número de threads/cores que usan los códigos.

4 Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

4.1 Objetivos

En este bloque práctico se aprenderán y se utilizarán las principales cláusulas OpenMP ([21], [22], [23], [24], [25]). Una vez finalizado el bloque deberá ser capaz de (si observa que no alcanza alguno de estos objetivos pregunte las dudas que tenga a su profesor en clase presencial o en su despacho de tutorías en horario de tutorías):

1. Distinguir entre directivas y cláusulas.
2. Describir la acción que realizan las cláusulas que determinan el ámbito de variables (`shared`, `private`, `lastprivate`, `firstprivate`, `default`) y explicar las diferencias entre ellas.
3. Describir la acción que realizan las cláusulas para comunicación y sincronización `reduction` y `copyprivate`.
4. Usar cláusulas de ámbito de variables: `shared`, `private`, `lastprivate`, `firstprivate`, `default`.
5. Discernir el ámbito de una variable en un código.
6. Usar la cláusula de comunicación y sincronización `reduction`.
7. Estudiar la escalabilidad de un programa en una plataforma (consulte la lección 6/Tema 2)

4.2 Seminario

El seminario de este bloque práctico presenta las principales cláusulas de OpenMP usadas para afinar el comportamiento de las directivas ([21], [22], [23], [24], [25]).

4.3 Práctica o trabajo a desarrollar

4.3.1 Ejercicios basados en los ejemplos del seminario práctico

En clase presencial y en casa se ejecutarán los códigos presentados en el seminario y se trabajarán los siguientes ejercicios y cuestiones:

1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? Si se plantea algún problema, resuélvalo sin eliminar `default(none)`.
2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta.
3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?
4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6. ¿El código imprime siempre 6? Razone su respuesta.
5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?
6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado
7. En el ejemplo `reduction-clause.c`, elimine `for` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo.

4.3.2 Resto de ejercicios

En clase presencial y en casa se trabajarán los siguientes ejercicios y cuestiones:

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por vector, v1:

$$v2 = M \bullet v1;\ v2(i) = \sum_{k=0}^{N-1} M(i,k) \bullet v(k),\ i = 0,\ldots N-1$$

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):
    a. una primera que paralelice el bucle que recorre las filas de la matriz y
    b. una segunda que paralelice el bucle que recorre las columnas.

    Use las directivas que estime oportunas y las cláusulas que sean necesarias excepto la cláusula `reduction`. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:
    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:
    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2).

5 Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP y evaluación de prestaciones

5.1 Objetivos

En este bloque práctico aprenderá a interaccionar con el entorno con OpenMP ([22], [23], [24], [25]). Una vez finalizado el bloque deberá ser capaz de (si observa que no alcanza alguno de estos objetivos pregunte las dudas que tenga a su profesor en clase presencial o en su despacho de tutorías en horario de tutorías):

1. Definir las variables de control de OpenMP `nthreads-var`, `dyn-var`, `nest-var`, `run-sched-var`, `def-sched-var`
2. Usar las variables de entorno `OMP_NUM_THREADS`, `OMP_DYNAMIC`, `OMP_SCHEDULE` para modificar y consultar variables de control.
3. Usar las funciones de OpenMP 2.5 para modificar y consultar las variables de control: `omp_get_dynamic()`, `omp_set_dynamic()`, `omp_get_num_threads()`, `omp_set_num_threads()`, `omp_get_thread_num()`, `omp_get_max_threads()`, `omp_get_num_procs()`, `omp_in_parallel()`.
4. Usar las siguientes funciones de OpenMP 3.0 que modifican y consultan las variables de control: `omp_get_thread_limit`, `omp_get_schedule(kind,modifier)` y `omp_set_schedule(kind, modifier)`
5. Describir la diferencia entre las dos cláusulas que influyen en el número de *threads* de una región paralela (`if`, `num_threads`) y ser capaz de usarlas.
6. Describir la acción que realizan las diferentes variantes de la cláusula `schedule` de la versión 2.5 de OpenMP que determinan la planificación de bucles (`static`, `dynamic`, `guided`, `runtime`), y ser capaz de usarlas.
7. Deducir el número de *threads* que va a ejecutar un trozo de código en paralelo.

5.2 Seminario

El seminario de este bloque práctico presenta las principales variables de control, variables de entorno, funciones y cláusulas de OpenMP usadas para interaccionar con el entorno de ejecución (versión 2.5 y algunas de 3.0) ([21], [22], [23], [24], [25]). También dedica un apartado a clasificar las funciones de la biblioteca de OpenMP.

5.3 Práctica o trabajo a desarrollar

5.3.1 Ejercicios basados en los ejemplos del seminario práctico

En clase presencial y en casa se ejecutarán los códigos presentados en el seminario y se trabajarán los siguientes ejercicios y cuestiones:

1. Usar la cláusula `num_threads(x)` en el ejemplo del seminario `if_clause.c`, y añadir un parámetro de entrada al programa que fije el valor `x` que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.
2. **(a)** Rellenar la Tabla 4 (se debe poner en la tabla el *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario `schedule-clause.c`, `scheduled-clause.c` y `scheduleg-clause.c` con dos *threads* (0,1) y unas entradas de:
   - iteraciones: 16 (0,…15)
   - chunck= 1, 2 y 4

   (b) Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

3. Añadir al programa `scheduled-clause.c` lo necesario para que imprima el valor de las variables de control `dyn-var`, `nthreads-var`, `thread-limit-var` y `run-sched-var` dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

Tabla 4 . Tabla `schedule`. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-claused.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0									
1									
2									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									

5. Añadir al programa `scheduled-clause.c` lo necesario para modificar las variables de control `dyn-var`, `nthreads-var` y `run-sched-var` y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

5.3.2 Resto de ejercicios

En clase presencial y en casa se trabajarán los siguientes ejercicios y cuestiones:

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un vector.

   NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (`N`-1, `N`-1) del resultado antes de que termine el programa.

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno OMP_SCHEDULE. Obtener en atcgrid los tiempos de ejecución del código paralelo que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096

Tabla 5 . Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas

Chunk	Static ¿? threads	Dynamic ¿? threads	Guided ¿? threads
por defecto			
2			
32			
64			
2048			

componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 5 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C;\ A(i,j) = \sum_{k=0}^{N-1} B(i,k) \bullet C(k,j),\ i,j = 0,\ldots N-1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (`N`-1, `N`-1) del resultado antes de que termine el programa.

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2).

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas, en alguna gráfica se debe presentar un máximo y en al menos una no se debería presentar un máximo. Consulte la Lección 6/Tema 2.

6 Bibliografía

Esta sección se ha dividido en varios apartados: Linux, C/C++, gcc, OpenMP, Eclipse y programación paralela.

6.1 Linux

Para resolver rápidamente las dudas más usuales relacionadas con el sistema operativo se recomienda utilizar las tarjetas de referencia (*reference cards*) de Linux [2] y Ubuntu [5]. Para resolver dudas al programar *scripts* (guiones) del intérprete de comandos (*shell*) GNU bash se recomienda usar la tarjeta de referencia de *bash* [3].

[1]. Linux en Wikipedia: http://es.wikipedia.org/wiki/GNU/Linux http://en.wikipedia.org/wiki/Linux

[2]. Linux *reference card*: http://files.fosswire.com/2007/08/fwunixref.pdf

[3]. Bash *reference card* (útil para programar *scripts*): http://www.fejf.de/tips/bash.quickref.pdf, http://database.sarang.net/study/bash/bash.pdf

[4]. Ubuntu en Wikipedia: http://es.wikipedia.org/wiki/Ubuntu http://en.wikipedia.org/wiki/Ubuntu_(operating_system)

[5]. Ubuntu *reference card*: http://files.fosswire.com/2008/04/ubunturef.pdf http://tangosoft.com/refcard/ (de W. Martin Borgert, hay versión en español)

[6]. GNOME en Wikipedia: http://es.wikipedia.org/wiki/GNOME http://en.wikipedia.org/wiki/GNOME

[7]. SSH en Wikipedia: http://en.wikipedia.org/wiki/Secure_Shell

[8]. Página de manual de SSH: http://linux.die.net/man/1/ssh

[9]. SFTP en Wikipedia: http://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol

[10]. Página de manual de SFTP: http://linux.die.net/man/1/sftp

[11]. TORQUE en Wikipedia http://en.wikipedia.org/wiki/TORQUE_Resource_Manager

[12]. TORQUE *home page* http://www.adaptivecomputing.com/products/open-source/torque/. TORQUE manual http://docs.adaptivecomputing.com/torque/4-1-3/help.htm. TORQUE commands http://docs.adaptivecomputing.com/torque/4-1-3/help.htm#topics/12-appendices/commandsOverview.htm

6.2 C/C++

En este apartado puede encontrar direcciones web que pueden aclarar dudas y ampliar conocimientos de los lenguajes de programación C. Se recomienda usar, cuando se programa en C en el aula y en casa, la tarjeta de referencia de C [14], es útil para encontrar rápidamente como expresar lo que se necesita (declarar un tipo de variable o constante, definir un tipo de dato, hacer una operación, incluir librerías, usar funciones de librerías del estándar para entrada/salida, operar con cadenas, realizar operaciones matemáticas complejas, …) en el lenguaje C. Si hace falta más detalle, entonces use el tutorial de C [14]. Comparando las tarjetas de referencia de C y C++ se pueden ver rápidamente las diferencias entre ambos.

[13]. C en Wikipedia: http://es.wikipedia.org/wiki/Lenguaje_C, http://en.wikipedia.org/wiki/C_%28programming_language%29

[14]. Tutorial de programación en C http://www.cs.cf.ac.uk/Dave/C/CE.html

[15]. C *reference card*: http://refcards.com/docs/silvermanj/ansi-c/ansi-c-refcard-a4.pdf

[16]. C++ en Wikipedia: http://es.wikipedia.org/wiki/C%2B%2B, http://en.wikipedia.org/wiki/C%2B%2B

[17]. Tutorial de programación en C++ en español http://c.conclase.net/curso/index.php

[18]. C++ *reference card*: http://www.ifi.uzh.ch/vmml/teaching/lectuces/systems-hs11/C_plus_plusRef_Card.pdf

6.3 Gcc

Este apartado contiene direcciones web donde puede aclarar dudas y ampliar conocimientos sobre el compilador que se va a utilizar en las aulas (gcc o *GNU Compiler Collection*). Si necesita resolver dudas sobre opciones del compilador, sobre librerías o sobre la sintaxis del ensamblador, use la referencia [20].

[19]. Gcc (*GNU Compiler Collection)* en Wikipedia: http://es.wikipedia.org/wiki/GNU_Compiler_Collection, http://en.wikipedia.org/wiki/GNU_Compiler_Collection

[20]. The GNU Compiler Collection manuals: http://gcc.gnu.org/onlinedocs/

6.4 OpenMP

Con las direcciones web de este apartado puede aclarar dudas de OpenMP y ampliar conocimientos. Se recomienda usar en el aula y en casa la tarjeta de referencia de OpenMP [23], es muy útil para encontrar rápidamente como realizar una acción, si necesita más detalle, entonces use el tutorial de OpenMP [22] o el manual con las especificaciones de OpenMP [24] o el manual de gcc para OpenMP que se puede encontrar en [20]. Si necesita un texto más didáctico para resolver las dudas, utilice le libro de Chapman et al. [25] (se encuentra en la biblioteca), que abarca la versión 2.5. En los seminarios hay ejemplos que se han extraído o son modificaciones de algún ejemplo de este libro, como se refleja en los propios seminarios.

[21]. OpenMP en Wikipedia: http://es.wikipedia.org/wiki/OpenMP, http://en.wikipedia.org/wiki/OpenMP

[22]. Blaise Barney, OpenMP 3.0 Tutorial, Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/openMP/

[23]. C/C++ OpenMP *reference card*: http://www.openmp.org/mp-documents/OpenMP3.1-SummarySpec.pdf

[24]. Web de OpenMP: http://openmp.org/wp/: especificaciones de la API (http://openmp.org/wp/openmp-specifications/), FAQ, presentaciones, etc.

[25]. [Chapman 2008] Barbara Chapman, Grabriele Jost, Ruud van der Pas, "*Using OpenMP. Portable Shared Memory Parallel Programming*", The MIT Press. 2008. (dedicado a la versión OpenMP 2.5) http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11387. ESIIT/D.1 CHA usi

6.5 Eclipse

En este apartado puede encontrar direcciones web que pueden aclarar dudas y ampliar conocimientos sobre el uso y los conceptos [30] de Eclipse, y una de las web de la que puede descargar el IDE de Eclipse para C/C++ [29].

[26]. Eclipse en Wikipedia: http://en.wikipedia.org/wiki/Eclipse_(software)

[27]. Web de Eclipse: http://www.eclipse.org , http://www.eclipse.org/users/

[28]. Web de Eclipse CDT: http://www.eclipse.org/cdt/

[29]. Sitio de descargas de CDT: http://www.eclipse.org/cdt/downloads.php

[30]. Eclipse user interface guidelines http://wiki.eclipse.org/User_Interface_Guidelines

6.6 Programación paralela

[31]. [Ortega 2005] J. Ortega, M. Anguita, A. Prieto. "*Arquitectura de Computadores*". Thomson, 2005. ESIIT/C.1 ORT arq

[32]. [Rauger 2010] T. Rauber, G. Ründer. "*Parallel Programming: for Multicore and Cluster Systems*". Springer 2010. Disponible en línea (biblioteca UGR): http://dx.doi.org/10.1007/978-3-642-04818-0

7 Agradecimientos

Agradecemos a todos los profesores de prácticas sus comentarios.