DESARROLLO DE SOFTWARE

EN

ARQUITECTURAS PARALELAS

Grado en Ingeniería Informática

CURSO 2015/2016

Prácticas

Curso 2015/2016

Dpnt. de Ciència de la Computació i Intel·ligència artificial
Dpto. de Ciencia de la Computación e Inteligencia artificial

Universitat d'Alacant
Universidad de Alicante

Grupo de Computación de
Altas Prestaciones y Paralelismo

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS
PRÁCTICAS y PONDERACIÓN

Las prácticas a desarrollar junto con su ponderación son:

Práctica	**Peso**	**Observaciones**
CONTAR	20 %	–
MVPI TCOM	20 %	Elegir una
MMmalla	30 %	–
FW MANDELBROT	30 %	Elegir una

Las prácticas se entregan tal y como se detalla en la explicación de las mismas.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS
PROCEDIMIENTO

El procedimiento que se sigue para que el alumnado pueda trabajar en el laboratorio es el siguiente:

1. Apertura de cuenta:

   Todo el alumnado debe disponer de una cuenta propia que servirá para todas las prácticas que ha de realizar. El nombre de la cuenta se corresponde con el usuario EPS y el mismo password.

2. Publicación de información referente a la asignatura:

   Procedimientos habituales (colgar anuncios en CV, ...)

3. Directorios de prácticas:

   Todo el alumnado dispone de un directorio de prácticas que corresponde con su HOME. En concreto para esta asignatura será:

   /home/CUENTA

   donde CUENTA es el nombre de la cuenta personal.

   Dentro del directorio de prácticas se deberá entregar cada práctica en un subdirectorio concreto que contendrá lo establecido en el enunciado de prácticas y todo aquello que sea imprescindible para su funcionamiento. Esto permitirá que el alumnado cree otros directorios para almacenar ficheros de pruebas, etc, que no son de interés a la hora de la corrección.

   Todas las cuentas del alumnado de esta asignatura están almacenadas en un servidor virtual. Estas cuentas se montan en los clientes cuando se accede al laboratorio. Aunque la información se mantiene de una sesión a otra, es conveniente, como siempre, realizar copias de seguridad.

4. Entrega de prácticas:

   La entrega de la práctica se formalizará creando en el directorio de la misma un fichero llamado ENTREGA (en mayúsculas). Este fichero contendrá una línea que identifica a la persona que haya realizado la práctica; el formato ha de ser:

   APELLIDO1 APELLIDO2, NOMBRE

   El alumnado que no vaya a entregar la práctica no debe crear el fichero ENTREGA.

5. Calificación de las prácticas:

   Cada práctica se calificará en función de la implementación realizada, rendimiento y, en su caso, memoria entregada. La escala de calificación será de 0 a 10 con la ponderación previamente estableciada para cada una de ellas. Toda práctica que no compile o no se ajuste a la implementación solicitada será calificada con un 0.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS
PRÁCTICA 1: CONTAR

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/CONTAR (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas CONTAR.

El objetivo de esta práctica es comparar el comportamiento de un algoritmo paralelo y su versión secuencial, obteniendo ciertas medidas de paralelismo como son el speed-up y la eficiencia. El problema que nos interesa es un algoritmo secuencial que cuenta el número de veces que un determinado número aparece en un conjunto grande de elementos almacenados en un vector.

Para simular una búsqueda en un conjunto mayor de datos sin necesidad de utilizar tanta memoria, lo que haremos es buscar un número determinado de veces en una misma estructura de datos (array de enteros). La versión secuencial se invoca con un parámetro "tamanyo" que determina el tamaño del vector. Si fijamos un tamaño de 200.000, tendremos un vector de ese tamaño donde se colocan números generados al azar comprendidos entre 0 y 1.000 (MAXENTERO). En este arreglo se busca el número indicado en la constante NUMBUSCADO. Todo ello se repite un número determinado de veces representado por la constante REPETICIONES

La implementación paralela debe usar un esquema de variables similar a la implementación secuencial suministrada. El proceso número 0 (proceso root) se encargará de comprobar la entrada de parámetros (el tamaño del vector), y la generación de los datos al azar. Posteriormente, el proceso root realizará el cálculo secuencial midiendo tiempos.

Para la versión paralela, la distribución de datos debe realizarse de forma equitativa entre todos los procesos (contando el proceso root). Si el número de procesos no divide al número de elementos en el vector, el exceso de elementos se le asociará al proceso root.

Finalizado el cálculo, el proceso root debe recibir la cuenta realizada por cada proceso para obtener la cuenta total. Adicionalmente obtendrá el speed-up y la eficiencia y los mostrará por pantalla.

Deben existir unos parámetros fijos en el programa:

MAXENTERO Valor máximo de los datos donde buscar (=1000).

REPETICIONES Número de repeticiones (=10000).

NUMBUSCADO Valor buscado en el conjunto de datos (=10).

Todas las comunicaciones necesarias deben realizarse mediante funciones de comunicación punto a punto (MPI_Send, MPI_Recv).

Memoria a entregar: Se debe entregar una memoria que contenga:

1. Explicación de los pasos seguidos.

2. Listado comentado del programa.

3. Cálculo y gráficas del speed-up obtenido con respecto a la versión secuencial y análisis del resultado para distintos conjuntos de datos y número de procesos. Podéis hacer variar el número de procesos entre 2 y 8, por ejemplo, 2, 4, 6 y 8. Asimismo, el tamaño del vector puede variar entre 1000000 y 4000000 como máximo.

4. Cálculo y gráficas de la eficiencia obtenida con respecto a la versión secuencial y análisis del resultado para distintos conjuntos de datos y número de procesos.

Ficheros a entregar:

contar_mpi.c Unidad principal.

makefile Makefile utilizado.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS
PRÁCTICA 2: MVPI

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/MVPI (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas MVPI.

En esta práctica se desea que se realice la implementación paralela, sobre MPI, del producto matriz vector, utilizando el algoritmo basado en el producto interno, con distribución equitativa de todas las filas de la matriz entre los procesos. Si el número de procesos no divide al número de filas, el exceso de filas se le asociará al proceso número 0 (proceso root) (ver ejemplo psdot.c). En la implementación se debe crear una unidad llamada matvec que multiplique una matriz por un vector y devuelva su resultado. Deben existir unos parámetros fijos en el programa:

maxm Máximo número de filas en la matriz (=500).

maxn Máximo número de columnas en la matriz (=500).

maxnumprocs Máximo número de procesos (=8).

La matriz con la que se trabaje será de orden `m` × `n`, y el vector `x` de orden `n`, ambos reales doble precisión. Los valores de `m` y `n` deben ser variables y controlados de manera que no sobrepasen los valores máximos. La matriz y el vector serán dimensionados de forma estática y definidos por el proceso 0 (proceso root) como:

```
double a[maxm][maxn], x[maxn], sol[maxm];
```

```
a[i][j]=(i+j), i=0,1,2,...,m-1, j=0,1,2,...,n-1,
```

```
x[j]=j, j=0,1,2,...,n-1,
```

debiendo ser enviado cada bloque de filas correspondiente a cada proceso distinto del root.

Cuando todos los procesos hayan finalizado sus cálculos, estos deben ser enviados al proceso root, el cual escribirá la solución global. Los procesos distintos del root también deben escribir su solución parcial, junto con la identificación del número de proceso.

Ficheros a entregar:

mvpi.c Contendrá las siguientes unidades:

- Unidad principal.
- Unidad vermatriz ya suministrada para visualizar una matriz.
- Unidad vervector ya suministrada para visualizar un vector.
- Unidad matvec que multiplique una matriz por un vector y devuelva su resultado.

makefile Makefile utilizado para la compilación.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

PRÁCTICA 3: TCOM

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/TCOM (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas TCOM.

El objetivo de este ejercicio es evaluar los valores de los parámetros que determinan el modelo de coste de las comunicaciones en la plataforma del laboratorio. El coste de las comunicaciones entre dos procesadores determinados viene dado por la expresión:

$$t_{com} = \beta + \tau \cdot \text{Tamaño\_Mensaje}.$$

Los parámetros β y τ indican respectivamente la latencia necesaria para el envío de un mensaje y el tiempo necesario para enviar un byte. Estos parámetros se pueden estimar de la siguiente forma:

- El valor de β será el tiempo necesario en enviar un mensaje sin datos.
- El valor de τ será el tiempo requerido para enviar un mensaje menos el tiempo de latencia, dividido por el numero de bytes del mensaje.

Dado que `MPI_Send` y `MPI_Recv` no admiten un mensaje sin datos, lo que haremos para estimar el valor de la latencia β es comunicar un solo byte, usando el tipo `MPI_BYTE`, a partir de por ejemplo una variable del tipo `char`. El tipo de dato `MPI_BYTE` no se corresponde con un tipo en C y es un dato de 8 bits sin interpretación alguna.

Teniendo en cuenta que se utiliza el protocolo TCP/IP como medio de transmisión de los mensajes, realmente el valor del tiempo de transferencia será diferente para mensajes pequeños y grandes, ya que el protocolo de comunicaciones fragmenta los mensajes en bloques. Se pide estimar el valor del parámetro τ para tamaños de mensaje variando entre los siguientes tamaños: 256 bytes (por ejemplo, $256/8 = 32 = 2^5$ reales en doble precisión, `double` en C y `MPI_DOUBLE` con MPI), 512 bytes (2^6 `double`'s), 1K (1024 bytes, 2^7 `double`'s), 2K (2^8 `double`'s), 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K, 1MB (1024K), 2MB, 4MB (2^{19} `double`'s).

Como sería imposible sincronizar los relojes perfectamente entre ambos procesos, para la medición del tiempo de comunicaciones, deberemos utilizar dos mensajes (uno de ida y otro de vuelta, usualmente conocido como ping/pong) y dividir el tiempo entre dos. Se deberán realizar varias repeticiones y tomar la media de ellas, descartando, en su caso, los valores atípicos.

Para la medición de tiempos se utilizará la función `MPI_Wtime`. Expresar los resultados en microsegundos. Teniendo en cuenta que la función `MPI_Wtime` devuelve segundos, no tendremos más que multiplicar por 10^6 para expresarlo en microsegundos.

Memoria a entregar: Se debe entregar una memoria que contenga:

1. Explicación de los pasos seguidos.
2. Listado comentado del programa.
3. Valores de β y τ (este último para los diferentes tamaños de mensaje).
4. Gráficas comentadas de las mediciones obtenidas.

Ficheros a entregar:

tcom.c Unidad principal.

makefile Makefile utilizado.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

PRÁCTICA 4: MMmalla

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/MMmalla (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas MMmalla.

En esta práctica se desea que se realice la implementación paralela, sobre MPI, del producto matriz por matriz, utilizando el algoritmo desarrollado para una arquitectura de malla (algoritmo de Cannon). Las matrices `A` y `B` deben estar particionadas en `r` $\times$ `r` bloques (`r` variable y no superior a `rmax=4`). Todos los bloques deben ser del mismo tamaño dado por la variable `bloqtam` que no debe ser superior a `maxbloqtam=100`. Todos los bloques de matrices que son necesarios definir: `A,B,C,ATMP` se considerarán almacenados en un vector. Por ejemplo, si `A`$= [a_{ij}]_{0\leq i,j\leq m-1}$, los elementos de esta matriz estarán almacenados en el vector:

$$a = (a_{0,0}, a_{0,1}, \ldots, a_{0,m-1}, a_{1,0}, a_{1,1}, \ldots, a_{1,m-1}, \ldots, a_{m-1,0}, a_{m-1,1}, \ldots, a_{m-1,m-1}).$$

Esto permite trabajar más fácilmente con las comunicaciones.

Podemos establecer el producto `c = a·b`, cuando las matrices implicadas están almacenadas en forma de vector y de orden `m`. La siguiente función obtiene este producto y adicionalmente lo acumula en `c` (`c = c + a·b`, operación que se necesita en el algoritmo):

```
void mult(double a[], double b[], double *c, int m) {
int i,j,k;
for (i=0; i<m; i++)
    for (j=0; j<m; j++)
       for (k=0; k<m; k++)
          c[i*m+j]=c[i*m+j]+a[i*m+k]*b[k*m+j];
return; }
```

Esta función puede ser utilizada para obtener los distintos productos por bloques que calculan los procesos.

Vamos a dar ahora unas nociones sobre cómo programar este producto matriz por matriz sobre MPI sin utilizar topologías de procesos. Con MPI no hay restricciones sobre qué tareas pueden comunicarse con otras. Sin embargo, para este algoritmo deseamos ver los procesos como situados en una malla abierta. Utilizaremos para ello la numeración que cada proceso obtiene cuando llama a `MPI_Comm_rank(MPI_COMM_WORLD, &myrank)`. El proceso 0 (proceso `root`) se encargará de obtener por el input estándar el único dato necesario: el orden de cada bloque, `bloqtam`. Además, comprobará que el número de procesos, `nproc`, sea un cuadrado perfecto de manera que se ajuste a la estructura de una malla cuadrada. Posteriormente enviará el parámetro necesario a los otros procesos, `bloqtam`. Las tareas con número de orden distinto de

cero recibirán dicho parámetro. Notar que ésta es la única diferencia que hay entre lo que debe hacer el proceso 0 y el resto.

Ahora, cada tarea necesita conocer la `fila` y la `columna` en la que se encuentra localizada en la malla. Esto es sencillo, ya que la división entera `myrank/r` nos da la `fila` de la tarea y el resto de esa división entera nos da la `columna` (`fila, columna = 0,1,2,r-1`). Ahora, usando las variables `myrank`, `nproc` y `r` podemos conocer sin demasiada dificultad los números de orden de los procesos situadas `arriba` y `abajo` en la misma columna. Estos números de orden serán utilizados para la posterior rotación de los bloques de `B` entre las columnas.

En cada etapa del algoritmo se envía un bloque de matriz `A` a todas las tareas de una misma fila. Así pues, es conveniente almacenar en un array, que llamaremos `mifila[]`, los números de orden de las tareas situadas en la fila de la tarea en cuestión.

Como este algoritmo supone que cada bloque está almacenado ya en cada tarea, cada proceso definirá su bloque `a` como:

$$\texttt{a[i]=i*(float)(fila*columna+1)}^2\texttt{/bloqtam}^2\texttt{, i=0,1,2,...,bloqtam*bloqtam-1},$$

y los bloques de `b` de manera que `B` sea la matriz identidad. Esto permitirá chequear al final del algoritmo si `C=A`.

Finalmente, como ya conocemos todas las variables necesarias se procede con el bucle principal del algoritmo.

Se aconseja que se usen distintas etiquetas de mensaje en cada iteración del algoritmo y que se especifique el número de orden en `MPI_Recv()`, es decir, que no se use como número de orden el valor `MPI_ANY_SOURCE`.

Cuando los cálculos estén terminados se comprobará que `A=C` para verificar que la multiplicación se ha efectuado correctamente.

Cada proceso debe contar el número de errores que ha cometido (si todo funciona correctamente serán cero errores) y enviarle la información al proceso `root`, el cual escribirá por la salida estándar el número de errores para cada proceso identificado por su número de orden.

Ficheros a entregar:

matriz.c Contendrá la unidad principal y la función *mult* que calcula el producto de dos matrices almacenadas como vectores.

makefile Makefile utilizado para la compilación.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

PRÁCTICA 5: FW

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/FW (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas FW.

En esta práctica se desea que se realice la implementación paralela, sobre MPI, del algoritmo de Floyd-Warshall para el cálculo de los caminos más cortos entre todos los pares de vértices en un grafo ponderado.

La implementación paralela del algoritmo de Floyd-Warshall estará basada en una descomposición unidimensional por bloques de filas consecutivas de las matrices intermedias en cada iteración. Si el número de procesos no divide al número de filas, el exceso de filas se le asociará al proceso número 0 (proceso root) (ver ejemplo psdot.c). Cada proceso será responsable de la actualización de una o más filas adyacentes de la matriz de pesos y de la matriz de caminos.

La implementación debe usar un esquema de variables similar a la implementación secuencial suministrada. En particular, deben existir unos parámetros fijos en el programa:

maxn Máximo número de vértices en el grafo (=1000).

maxnumprocs Máximo número de procesos (=8).

La generación del grafo (definición de la matriz de pesos) se realizará también de la misma forma que en la versión secuencial. Esta generación se realizará por el proceso root, debiendo ser enviado cada bloque de filas correspondiente a cada proceso distinto del root. Para este envío debe usarse la función `MPI_Scatter()`.

Cuando se finalice el algoritmo, el proceso root recibirá del resto de procesos, las filas evaluadas de la matriz de distancias y de la matriz de caminos. Esta comunicación debe realizarse con la función `MPI_Gather()`.

De forma similar a como se hace en la versión secuencial, el proceso root debe mostrar la matriz de pesos y la matriz de distancias final solo cuando el número de vértices sea menor o igual que 10. Adicionalmente, debe ser capaz de mostrar el peso y el camino más corto entre dos vértices dados.

Ficheros a entregar:

fw.c Contendrá la unidad principal y todas las funciones auxiliares.

makefile Makefile utilizado para la compilación.

DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

PRÁCTICA 6: MANDELBROT

La práctica se ENTREGARÁ en el directorio:

/home/CUENTA/MANDELBROT (CUENTA=cuenta personal).

Los ficheros o directorios que sea necesario crear pero no vayan a ser entregados al profesor deberán crearse directamente en el directorio /home/CUENTA y no dentro del subdirectorio de prácticas MANDELBROT.

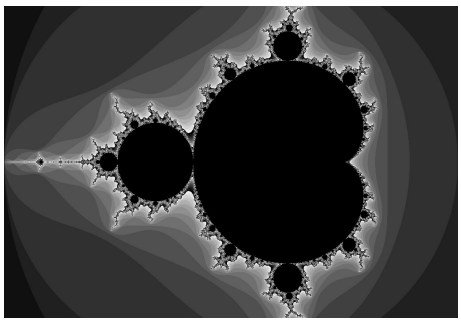
Los fractales son representaciones gráficas de funciones matemáticas que suelen parecerse a paisajes y formas de la naturaleza, donde pueden observarse muchos ejemplos de estructuras repetitivas que configuran hojas, dunas, etc. Como el objetivo no es dominar la teoría de fractales, lo que se abordará en esta práctica es la representación parcial de un fractal conocido como conjunto de Mandelbrot. Dicho conjunto está compuesto por los números complejos, $c = a + bi$, para los que la relación de recurrencia:

$$z_{k+1} = z_k^2 + c, \quad k = 0, 1, 2, \ldots,$$

converge a un número complejo finito, siendo $z_0 = 0$ la condición inicial.

Se puede demostrar que si existe un m para el cual $|z_m| > 2$, la sucesión diverge y entonces c no pertenece al conjunto de Mandelbrot. Como el valor de m, de existir, podría ser muy grande, la primera aproximación que se hace es llegar como máximo hasta el elemento $z_{IterMax}$. Si se alcanza dicho $z_{IterMax}$ sin que ningún $|z_i|$ sea mayor que 2, se supone que c pertenece al conjunto de Mandelbrot.

En esta práctica se desea que se realice la implementación paralela, sobre MPI, de la técnica para la obtención del conjunto de Mandelbrot.



La implementación debe usar un esquema de variables similar a la implementación secuencial suministrada. El proceso número 0 (proceso root) se encargará de leer, en su caso, los datos iniciales:

- El número de píxeles a considerar pixelXmax x pixelYmax. Por defecto 1024 x 1024.
- El número máximo de iteraciones IterMax. Por defecto, 1000.

- El nombre del archivo de salida con la imagen del conjunto de Mandelbrot obtenido. Por defecto imgA.pgm, imgB.pgm.
- Seleccionar un área del plano complejo de las consideradas previamente. Por defecto, la parte real varía en $[-2, 1]$ y la parte imaginaria se encuentra en $[-1{,}5, 1{,}5]$.

La implementación paralela debe basarse en la idea de establecer un pool de procesos. Definiremos una tarea como el procesado de una fila de píxeles de la imagen del conjunto de Mandelbrot. Estas tareas deben ir asignándose a los procesos conforme estos quedan inactivos.

Una vez finalizado el algoritmo, el proceso 0 guardará en archivo la imagen final del conjunto de Mandelbrot obtenido.

Memoria a entregar: Se debe entregar una memoria que contenga:

- Explicación de la implementación.
- Distintas ejecuciones con distinto número de procesos. Se presentarán tablas indicando para cada ejecución, el tiempo secuencial, el tiempo paralelo, el speed-up y la eficiencia. Para alguna ejecución se puede presentar la imagen del conjunto de Mandelbrot obtenida.

Las distintas ejecuciones pueden corresponder, por ejemplo, a los siguientes datos:

Ancho imagen	Alto imagen	Iteraciones	Dominio
1024	1024	100	0
		1000	1
		2000	3
		3000	4
		1000	6
		4000	7
		2000	8

Ficheros a entregar:

mandelbrot_mpi.c Contendrá la unidad principal y todas las funciones auxiliares.

makefile Makefile utilizado para la compilación.

Unidad mandelbrot_mpi

- Proceso 0: Lectura datos iniciales.
- Porceso 0: Se crea el archivo que contendrá la imagen y se escribe la cabecera.
- Envío y recepción de datos comunes a todos los procesos: `IterMax`, `pixelXmax`, `pixelYmax`, `RealMin`, `RealMax`, `ImMin` e `ImMax`.
- Cálculo de `AnchoPixel` y `AltoPixel`.
- Proceso 0: Crear la matriz de píxeles.
- Proceso 0: Asignar las primeras `nproc-1` filas de píxeles a los procesos distintos de 0 (filas `0,1,nproc-2`).
- Proceso 0:

  `for (pixelY=nproc-1;pixelY<pixelYmax;pixelY++)`

  Recibir resultado de un proceso identificado con `status.MPI_SOURCE`

  Realizar nueva asignación de trabajo: fila `pixelY` se asigna al proceso `status.MPI_SOURCE`
- Proceso 0: Recibir las últimas `nproc-1` filas de píxeles ya procesadas que quedarán pendientes.
- Proceso 0: Enviar señal de terminación a todos los procesos.
- Proceso 0: Escribir imagen en archivo.
- Procesos > 0: Crear el vector necesario para almacenar `pixelXmax` píxeles.
- Procesos > 0: Proceso iterativo (`while (1)`)
  - Recibir asignación de trabajo del proceso 0.
  - Comprobar si la etiqueta de envío indica que se debe parar.
  - Procesar la fila de píxeles asignada.
  - Enviar al proceso 0 la fila de píxeles obtenida.