Metodología de la Programación Paralela 2017-2018
Solución de la prueba práctica, 15 enero 2018

SOLUCIONES parte escrita

CUESTIÓN 1-a

No podemos paralelizar al mayor nivel en el bucle de la función `sec`, pues el código tiene un esquema síncrono en el que los valores de una iteración dependen de la anterior. Por tanto, paralelizamos dentro de las funciones `par` e `impar`, en cada una en el bucle que llama a la función `filapar` o `filaimpar`, poniendo #pragma omp parallel for private(i) antes del bucle. Se podrían paralelizar los otros bucles de idéntica forma, pero como la computación en los otros bucles tienen coste lineal mientras que las de los que estamos paralelizando lo tienen cuadrático, su paralelización no repercutirá de forma significativa en el tiempo de ejecución.

CUESTIÓN 1-b

Se asignan $\frac{N}{p}$ (p es el número de procesos) filas contiguas a cada proceso. No nos preocupamos por ahora del considerar tamaños distintos en cada proceso.

Antes de empezar las iteraciones se hará un broadcast del tamaño del problema, N.

Los procesos distintos del cero reservan espacio para $\frac{N}{p} * N$ enteros.

Se distribuyen los datos de la matriz A a los procesos, utilizando para eso un Scatter o comunicaciones punto a punto.

Antes de cada iteración cada proceso comunica su primera y última fila a los procesos vecinos:

En P_i, $i = 0, \ldots, p-1$:

si $i \neq 0$
    enviar primera fila a P_{i-1}
si $i \neq p-1$
    enviar última fila a P_{i+1}
si $i \neq 0$
    recibir última fila de P_{i-1}
si $i \neq p-1$
    recibir primera fila de P_{i+1}

Para la recepción de las filas de procesos adyacentes se puede utilizar vectores de tamaño N, o se puede reservar al principio espacio para $\left(\frac{N}{p} + 2\right) * N$ datos en cada proceso y utilizar la fila 0 para los datos recibidos del proceso anterior y la $\frac{N}{p} + 1$ para los recibidos del proceso siguiente. Hay que modificar los códigos del tratamiento de la primera y última fila en cada proceso, pues ahora, salvo en las filas extremas (primera fila de proceso 0 y última del $p-1$), los elementos se calculan utilizando los de las filas recibidas.

Tras acabar las iteraciones se acumulan los resultados en el proceso 0. Se puede utilizar un Gather o comuniaciones punto a punto.

Si el tamaño no es múltiplo del número de procesos cada proceso tiene un número de filas distinto. A los procesos con identificador $i < N\%p$ se les asigna $\frac{N}{p} + 1$ filas, y al resto $\frac{N}{p}$.

Las modificaciones al esquema son mínimas:

Antes de empezar las iteraciones se hará un broadcast del tamaño del problema, N.

Los procesos distintos del cero reservan espacio para $\left(\frac{N}{p} + 1\right) * N$ o $\frac{N}{p} * N$ enteros según su identificador de proceso.

Se distribuyen los datos de la matriz A a los procesos. Se puede utilizar un Scatterv o comunicaciones punto a punto. El proceso 0 tiene que calcular los tamaños asignados a todos los procesos, pero el resto de procesos solo necesita conocer su tamaño.

Las iteraciones son idénticas al caso anterior.

Tras acabar las iteraciones se acumulan los resultados en el proceso 0. Se puede utilizar un Gatherv o comunicaciones punto a punto.

CUESTIÓN 1-c

La combinación de MPI con OpenMP se hace como siempre. Dentro de cada proceso MPI se lanzan hilos OpenMP como en la cuestión 1-a, paralelizándose los mismos bucles de antes, que en este caso realizan $\frac{N}{p}$ pasadas (o una más) en vez de N.

CUESTIÓN 2-a

Se puede paralelizar cada multiplicación de matrices como se hace usualmente, paralelizando el bucle más externo de la multiplicación con #pragma omp parallel for. Pero esto produciría paralelismo de grano fino, y dado el reducido tamaño de las matrices a multiplicar puede que no se produjera una reducción importante en el tiempo de ejecución.

Para una paralelización de grano más grueso se puede hacer que cada hilo realice algunas de las multiplicaciones y las acumule en una matriz resultado asociada a ese hilo. Se puede declarar un array de matrices resultado, con una entrada (una matriz resultado) para cada hilo.

Se paraleliza el bucle de la función `sec` con `#pragma omp parallel for private(i) schedule(static,1)`, donde se usa *chunk*=1 para tener una asignación cíclica, con la que se balancea el trabajo. Cada hilo dejará el resultado en la matriz de resultados que tenga asociada.

Al finalizar la zona paralela quedará solo el hilo maestro, que sumará las matrices resultado de todos los hilos. Dado que la suma de matrices es de orden cuadrático y la multiplicación de orden cúbico, no influirá mucho en el tiempo de ejecución el no paralelizar la suma de los resultados parciales.

CUESTIÓN 2-b

El esquema es como en el apartado anterior, pero en este caso trabajando procesos en vez de hilos. Dado el alto coste de las comunicaciones es más interesante explotar paralelismo de grano grueso.

Inicialmente se hace un broadcast de N. Cada proceso distinto del 0 recibe toda la matriz, para lo que se hace un broadcast de A.

Cada proceso reserva espacio para su matriz resultado local, y realiza las iteraciones de las multiplicaciones empezando con el índice igual a su identificador (*nodo*) y aumentándolo con paso p.

Al final se acumulan todos los resultados en la matriz resultado en P_0. Se puede hacer con comunicaciones punto a punto y P_0 va haciendo las sumas tras cada recepción, o se puede utilizar la función Reduce con la operación de suma.

CUESTIÓN 2-c

La combinación de MPI con OpenMP se puede hacer usando el esquema de MPI de la CUESTIÓN 2-b y paralelizando con OpenMP cada multiplicación de matrices, con `#pragma omp parallel for` antes del primer bucle de la multiplicación. De esta forma tenemos paralelismo de grano grueso con MPI, y de grano fino con OpenMP. Como las multiplicaciones no son de gran tamaño, podría interesar que OpenMP explotara también paralelismo de grano grueso. Para esto, cada proceso *nproc* de los $NPROC$ procesos MPI lanzará $NTHRE$ hilos. Como el proceso *nproc* hace las multiplicaciones *nproc*, $nproc + NPROC$, $nproc + 2 * NPROC$..., cada hilo *nthre* (entre 0 y $NTHRE$-1) del proceso realizará parte de sus iteraciones, empezando en $nproc + nthre * NPROC$ e incrementando cada vez $NTHRE * NPROC$. Por ejemplo, si $NPROC = 3$, al proceso $nproc = 2$ le corresponden las pasadas 2, 5, 8, 11, 14, 17..., y si se lanzan 3 hilos, el 0 empieza en 2, el 1 en 5, y el 2 en 8, y sus siguientes iteraciones se obtienen sumando 9: el 0 hace las iteraciones 2, 11, 20..., el 1 las 5, 14, 23... Los resultados parciales de los hilos las suma el hilo maestro en cada proceso, y esas sumas parciales por proceso se acumulan en el proceso 0.

SOLUCIONES parte sobre ordenador

CUESTIÓN 1-a

La versión secuencia da un tiempo de 12038, y la OpenMP con 4 hilos (calisto tiene 4 cores) 4520. El speed-up no llega a 3, pero se puede considerar aceptable.

CUESTIÓN 1-b

La versión secuencial tarda 191. Se utilizan entradas menores que con OpenMP por problemas con los tamaños de los buffers de comunicación, y como eso no se había visto en las prácticas se decide reducir el tamaño. El paralelo con 4 procesos tarda 49, con lo que el speed-up es casi 4, más cercano al óptimo que en el caso de OpenMP.

CUESTIÓN 1-c

Con 2 procesos MPI y dentro de cada uno 2 hilos el tiempo es 62, con las mismas entradas que en la cuestión anterior. El uso de OpenMP vuelve a hacer que emperoren las prestaciones.

CUESTIÓN 2-a

La versión secuencial tarda 24069 y la OpenMP con 4 hilos 6420. El speed-up es muy cercano al máximo teórico.

CUESTIÓN 2-b

Con 4 procesos tarda 6658, prácticamente lo mismo que OpenMP con 4 hilos (las diferencias pueden ser circunstanciales), con lo que las comunicaciones no añaden mucha sobrecarga, lo que es normal pues no hay comunicaciones intermedias, sólo el broadcast del principio y la acumulación al final de los resultados.

CUESTIÓN 2-c

Una versión que combina grano grueso con grano fino (procesos que trabajan haciendo simultáneamente multiplicaciones distintas, y dentro de cada proceso los hilos trabajando en la paralelización de cada multiplicación) da un tiempo de 6518, y la de grano grueso con MPI y OpenMP (a cada proceso e hilo se asignan multiplicaciones distintas para hacerlas en paralelo) da 6509. Las diferencias son mínimas y los tiempos son similares a los de OpenMP y MPI. Debido a que el sistema es muy pequeño (4 cores), no da mucho juego a la hora de obtener configuraciones óptimas de ejecución.