PROGRAMACIÓN PARALELA Y COMPUTACIÓN DE ALTAS PRESTACIONES

TRABAJO AUTÓNOMO: ENTORNOS DE PROGRAMACIÓN PARALELA

PRÁCTICAS DE PPCAP 17/18

José Luis Cánovas Sánchez

16 OCTUBRE

Algoritmos Matriciales por Bloques

CUESTIÓN 1

Comparar los tiempos de las multiplicaciones matriciales por bloques y sin bloques.

Tabla de tiempos (seg):

Algoritmo	N=500	N=1000	N=2000	N=3000	N=4000
matriz_matriz	0.251404	4.951881	56.249824	167.854739	396.104500
matriz_matriz_ld	0.311325	2.727537	29.544091	102.771928	551.153559
matriz_matriz_bloques					
Tamaño de bloque: 25	0.177865	1.453277	11.751641	39.685550	93.973494
Tamaño de bloque: 50	0.180716	1.361554	10.562337	35.292006	83.937251
Tamaño de bloque: 100	0.188849	1.376489	10.777013	36.056393	84.532981
matriz_matriz_bloquesdobles					
Bloque 50 Sub-bloque 10	**0.143431**	**1.131428**	**9.062817**	**30.563007**	**72.788488**
Bloque 50 Sub-bloque 25	0.189209	1.478096	11.823844	39.859929	95.002231

Estas pruebas se realizaron en `marte`.

Como era de esperar, la versión sin bloques con leading dimension tiene mejores resultados que una matriz bidimensional.

Las versiones con un nivel de bloque difieren poco entre sí, hasta ~10segundos de diferencia en N=4000 según el tamaño de bloque. Para los tests obtenidos, la mejor elección es un bloque de tamaño 50. En todos los casos se mejora notablemente respecto a las versiones sin bloques.

Tomamos esa mejor elección de bloque 50, y lo dividimos en sub-bloques de 25, que nos da tiempos esperados un poco mayores que la versión de un nivel de bloques de tamaño 25 (por el overhead de dos niveles de bloques). Utilizando sub-bloques de tamaño 10 mejoramos aún más los tiempos, obteniendo la mejor marca en todos los casos.

CUESTIÓN 2

Quitar la restricción de múltiplo del tamaño de bloque.

Basta con sustituir las dimensiones de las submatrices (bloques) multiplicadas en `multiplicar_acumular_matrices()` y `copiar_matriz()`, que hasta ahora eran tb, el tamaño de bloque fijo, por los siguientes valores, que hacen tomar bloques menores en los extremos de las matrices:

- Filas bloque en A:

  ```
  fblocka = ( (i+tb) < fa ) ? tb : (fa-i);
  ```

- Columnas bloque en A == Filas bloque en B:

  ```
  cblocka = ( (k+tb) < ca ) ? tb : (ca-k);
  ```

- Columnas bloque en B:

  ```
  cblockb = ( (j+tb) < cb ) ? tb : (cb-j);
  ```

CUESTIÓN 3

Programar LU con bloques y comprobar con LU sin bloques.

En el fichero `LU-bloques.c` está implementada la descomposición con bloques y sin bloques (la primera depende de ella), y ejecuta ambas, comprobando, si se compila con `-D DEBUG` que ambas son iguales a la original, por lo que la ejecución es correcta.

CUESTIÓN 4

Comparar tiempos de LU con y sin bloques.

Tabla de tiempos (seg):

Algoritmo	N=500	N=1000	N=2000	N=3000	N=4000
LU sin bloques	0.042273	0.422934	5.582038	19.640315	46.243277
LU por bloques					
Tamaño de bloque: 10	**0.041593**	**0.337875**	**2.746687**	**9.299434**	**21.883269**
Tamaño de bloque: 25	0.060312	0.537237	4.440736	15.637062	44.447150
Tamaño de bloque: 50	0.065794	0.541200	5.052546	18.529195	45.904555
Tamaño de bloque: 100	0.064633	0.611730	5.578424	18.794480	46.560913

Estas pruebas se realizaron en `marte`.

Los mejores tiempos se obtienen con el tamaño de bloque 10 en todos los casos. A partir de un tamaño de bloque 50 los tiempos obtenidos están al nivel de la descomposición sin bloques. Esta

información junto con la de los bloquesdobles nos indica que para la máquina marte el mejor tamaño de bloque es 10, y podremos utilizarlo para la siguiente cuestión.

CUESTIÓN 5

Comparar tiempos de LU con bloques y diferentes multiplicaciones matriciales.

En `LU-bloques2.c` cambiamos la multiplicación con resta de las matrices que era una versión `matriz_matriz_ld`, por la multiplicación con bloques que modificamos antes. Tomamos bloques para la multiplicación de matrices más pequeños que los bloques de la LU. Si fueran iguales, sólo tendríamos un overhead respecto de la cuestión anterior. Además, tras las pruebas, vamos a comprobar si la mejor configuración para la máquina `marte` corresponde a tomar bloques para LU de tamaño 50 (que mejoraban por poco la versión sin bloques), y sub-bloques para la multiplicación de tamaño 10 (es el tamaño que obtenía los mejores tiempos tanto en la multiplicación de matrices como la descomposición LU).

Tabla de tiempos (seg):

Algoritmo	N=500	N=1000	N=2000	N=3000	N=4000
LU por bloques: 25					
Mult de bloque: 10	0.051613	0.433742	3.622686	12.313919	29.648840
LU por bloques: 50					
Mult de bloque: 10	**0.050091**	**0.422087**	**3.466454**	**11.591401**	**27.938498**
Mult de bloque: 25	0.070367	0.518026	4.237743	14.091616	34.104282
LU por bloques: 100					
Mult de bloque: 10	0.052033	0.451865	3.580422	11.700435	29.033573
Mult de bloque: 25	0.063045	0.532620	4.322647	14.246308	35.328427
Mult de bloque: 50	0.062132	0.516972	3.954549	12.982711	32.558991

Estas pruebas se realizaron en `marte`.

Comprobamos la hipótesis anterior. Al tomar tamaños de bloque donde la descomposición LU se beneficiaba, y al tomar sub-bloques que optimizan la operación más pesada (multiplicación de matrices), obtenemos el mejor tiempo al utilizar multiplicación por bloques.

Sin embargo, no mejora a la versión ld anterior con bloques de tamaño 10 para la descomposición LU.

Veamos ahora otra multiplicación alternativa que en los algoritmos básicos obtuvo los mejores resultados. La multiplicación con transpuesta. En `LU-bloques3.c` utilizamos esta multiplicación, y repetimos las pruebas:

Tabla de tiempos (seg):

Algoritmo	N=500	N=1000	N=2000	N=3000	N=4000
LU por bloques					
Tamaño de bloque: 10	**0.041172**	**0.333281**	**2.737566**	**9.302499**	**21.875436**
Tamaño de bloque: 25	0.059080	0.486190	3.895753	13.138739	31.618844
Tamaño de bloque: 50	0.055878	0.462909	3.729899	12.489604	30.229669
Tamaño de bloque: 100	0.055859	0.472999	3.821283	12.495737	31.301160

Los mejores tiempos los obtiene de nuevo un tamaño de bloque de 10, con poco mejores tiempos que los de la cuestión 4. Para bloques tan pequeños, el método por la matriz transpuesta apenas añade mejora.

Para bloques mayores, no mejoramos al de tamaño 10, pero al comparar con los casos previos, los tiempos mejoran notablemente, debido a que las matrices de hasta 100x100 en los casos previos, que era la mayor carga de trabajo del algoritmo LU, ahora se hacen con transposición de matrices, que ya vimos que era la mejor opción para matrices densas.