Curso 2022-2023

Práctica 3: Producto Matricial Completo

17/10/2022
Prácticas de Programación Concurrente y Paralela
Pablo Revuelta Sanz & José Ranilla

Tipo de Práctica: Abierta
Entrega Documentación: Ver sección "*¿Cuándo entregar la práctica?*"
Duración: 2 sesiones (17 al 28 de octubre)

ÍNDICE

Objetivos

En combinación con la 1ª práctica, en estas prácticas el alumnado debe implementar soluciones secuenciales y paralelas en memoria compartida para el problema descrito en la 1ª práctica.

El objetivo, la motivación de estas prácticas, es el rendimiento. Por ello, el alumnado debe utilizar los recursos OpenMP, y las técnicas propuestas para alcanzar el máximo rendimiento.

En cualquier caso, el tipo de almacenamiento para las matrices que se debe usar es el ya conocido 1D *column-major layout.*

OpenMP

Como el alumnado ya conoce, son varios los mecanismos OpenMP para abordar la paralelización de códigos secuenciales. Dada la naturaleza del problema no se recomienda el constructor de carga *sections* (aún así, ¿se podría?). Se pedirá implementar distintas funciones y obtener sendos resultados con otros constructores.

Las técnicas

En la 1ª práctica se ha visto que explotar la **localidad espacial** (usar A^t) reduce el tiempo de la versión estándar. El resultado al comparar el incremento de rendimiento usando A^t (explotar la localidad espacial) con el obtenido usando los *flags* de optimización del compilador (p. ej. –O3) es “decepcionante”. El rendimiento estará lejos del obtenido por las librerías de alto rendimiento (usadas, por ejemplo, por *python*).

Además de las estrategias basadas en explotar la localidad espacial, hay otras con un mayor potencial. Una habitual es la **técnica por bloques (*tiling*)**. La técnica por bloques busca explotar la **localidad temporal**: “realizar el mayor número de operaciones con un dato subido a la jerarquía de memorias caché antes de que sea expulsado”. Piénsese, por ejemplo, en la primera fila de A y la primera columna de B. Tanto una como otra son usadas reiteradamente al calcular los elementos de la primera columna y fila de C, respectivamente. Si las dimensiones de las matrices son grandes, entre un uso y el siguiente de dicha columna/fila, ésta habrá sido expulsada de la caché para “hacer hueco a otros datos”.

Para conseguir el objetivo perseguido se debe realizar un cambio conceptual: “*no plantear la resolución del producto matricial como operaciones a nivel de vector; plantearlo como la resolución de múltiples subproblemas matriciales*”:

$$\begin{pmatrix} A1 & B1 \\ C1 & D1 \end{pmatrix} \times \begin{pmatrix} A2 & B2 \\ C2 & D2 \end{pmatrix} =$$

$$\begin{pmatrix} A1A2 + B1C2 & A1B2+B1D2 \\ C1A2 + D1C2 & C1B2+D1D2 \end{pmatrix}$$

Es obvio que, con esta estrategia, aparece un nuevo grado de libertar en el problema: el tamaño del bloque. Este parámetro, que se deja por ahora al arbitrio del alumnado, puede variar entre dos extremos: el valor 1, en cuyo caso el producto vuelve a ser el original (elemento a elemento), y el de la matriz completa, de forma que, de nuevo, el producto vuelve a ser el original, solo que por el motivo contrario al anterior. Valores intermedios empezarán a explotar la localidad temporal. No obstante, no hay a priori un valor establecido, pues éste estará íntimamente relacionado con el tamaño de las caches: si es demasiado pequeño, un mismo dato entrará y saldrá de la caché, con lo que no sacaremos el máximo partido al diseño. Si es demasiado grande, no cabrá, y el propio sistema operativo necesitará meterlo y sacarlo de las cachés en trozos, perdiéndose rendimiento.

Los prototipos suministrados

Se deben reutilizar, a excepción del fichero ***Makefile*** y ***Prototipos.h***, los archivos de la 1ª práctica concernientes al alumnado (escritos a partir del fichero *PRAC01.tgz* de */opt/PracticasPCP2022_2023/Practica01/*), es decir: *PRAC01.c*, *PRAC01.py* y *EjecutaPrac01.sh*. El fichero *Makefile* y el nuevo *Prototipos.h* que se deben usar ahora están en */opt/PracticasPCP2022_2023/Practica03*. **Es importante renombrar el fichero *PRAC01.c* como *PRAC03.c* para que el *Makefile* compile correctamente. El resto de ficheros reutilizados pueden renombrarse si se desea por claridad, pero no es obligatorio para que todo funcione y pueda corregirse.**

Las tareas a realizar

Análisis teórico

A lo ya pedido en la practica 1 (TPP_{dp}, t_c…), añadir estimar la complejidad temporal y espacial de todas y cada una de las soluciones propuestas, así como el *speedup* teórico para cada máquina disponible.

Plano empírico

1º. Implementar en *PRAC03.c*, usando almacenamiento *column-major layout*, las siguientes funciones para resolver el problema (además de o modificando la ya hecha en la primera práctica *MyDGEMM*), consultar el archivo *Prototipos.h* para ver las cabeceras a implementar:

- *MyDGEMM*: extiende la resolución del problema de forma paralela mediante *parallel for*.
- *MyDGEMMT*: resuelve el problema de forma paralela mediante *tasks*.
- *MyDGEMMB*: resuelve el problema mediante bloques, secuencial y paralela (implementación paralela que se desee).

Cuando una misma función deba funcionar como secuencial y paralela, se codificará de tal forma que mediante variables de entorno pueda funcionar de uno u otro modo. Todas ellas realizan cálculo con la matriz transpuesta.
Las tres deberán poder ser llamadas desde el código de *python* según las cabeceras provistas en *Prototipos.h*.

En las implementaciones por bloques se tendrán en cuenta las siguientes asunciones:
- Las matrices serán cuadradas.
- El tamaño de bloque será un submúltiplo del tamaño total.
- Los bloques serán cuadrados.

El prototipo que maneje bloques debe obedecer al siguiente patrón:

MyDGEMMB(tipo, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc, blk)

Se destaca en rojo el último argumento, que es nuevo respecto a la 1ª práctica. Es el tamaño de bloque, número de elementos en cada dimensión del bloque.

2º. Completar el fichero *PRAC01.py* para poder realizar los cálculos solicitados con las nuevas funciones. No obstante, este fichero no se entrega.

En resumen

Se trata de completar el estudio de la primera práctica (entorno secuencial) con el procedente del uso de la programación paralela en memoria compartida (uso del OpenMP).

En la memoria no se deben separar las prácticas. Se deben abordar los apartados del análisis teórico de forma conjunta, secuencial y paralelo, uno detrás del otro.

En el plano empírico el fichero ***PRAC03.c*** debe incluir todas las funciones especificadas, en sus versiones paralelas. Si alguna se deja en blanco, debe tener bien la cabecera para permitir compilar y corregir el resto.

El sistema de corrección compilará usando el *PRAC03.c* de cada alumnx (los demás serán los proporcionados por los profesores en los *.tgz* y un *script* de *python* propio), y ejecutará las prácticas. Todo lo que el alumnado desee hacer debe estar encapsulado en sus *wrapper*, que obedecen a los prototipos anteriores.

¿Qué entregar?

El fichero *PRAC03.c* debe estar en la carpeta ***$HOME/PRAC03***. Se usará una herramienta automatizada que recorrerá las carpetas *PRAC03* de todas las cuentas que:

1. Copiará el fichero citado. Si el archivo no existe o tiene otro nombre se suspende el proceso de corrección.
2. Compilará. Si la compilación NO es correcta se suspende el proceso de corrección.
3. Construirá un fichero de órdenes para OGE/SGE y enviará el trabajo al gestor.
4. Si la ejecución NO es correcta (violación de segmento, bloqueo...) se suspende el proceso de corrección.
5. Comparará la solución del alumnado con la del profesor.

Si alguno de los pasos anteriores falla o la comparativa es inconsistente, la práctica será considerada incorrecta (no apta, suspensa) y no se corregirá la documentación.

Para la documentación, el alumnado debe subir al Campus Virtual un único fichero en formato PDF. Cualquier otro formato no será válido y, nuevamente, la práctica se considerará incorrecta. En la documentación se debe incluir:

1. La respuesta a las preguntas formuladas explícitamente en “Análisis teórico” de este enunciado.
2. La tabla del ANEXO I cubierta tantas veces como se solicite. **SOLO compilador icc** (ver *Makefile* proporcionado) **y cálculo con transpuesta**. Se pueden añadir más tablas con otros parámetros probados (como tamaño de bloque, o dónde se generan las *tasks*, por ejemplo).
3. Una discusión de los resultados obtenidos. Ésta debe incluir un análisis de los resultados (no una mera lectura de los mismos), poniéndolos en comparación con lo obtenido teóricamente, contrastando distintas soluciones empíricas entre ellas y

con la teoría (eficiencia, *speedup*...), avanzando hipótesis que expliquen resultados inesperados (a partir de lo aprendido en teoría), explicando mediante el HW el comportamiento observado, cualquier gráfica extra que ilustre lo discutido, etc.

El alumnado debe ser consciente de que el problema a resolver es, respecto al tiempo de ejecución, cúbico y que la experimentación planteada es exhaustiva: a) 3 tamaños, b) 3 prototipos y c) 3 potenciales entornos de experimentación (en total, 12 tablas a rellenar como mínimo). Se estima que, para una programación libre de errores y relativamente eficiente, la duración de todas las ejecuciones está acotada en, aproximadamente, 28 minutos, contando las tres máquinas (aprox: 10 min en i3, 4,5 min en el ryzen y 13,5 min en xeon). Compilaciones, bloqueos por bucles mal construidos, etc. pueden alargar estas estimaciones. Si todo el alumnado (unas 70 personas) lanza una ejecución completa y correcta a la vez, el tiempo de respuesta estará en los i3 entre 0 y 3h, mientras que para el ryzen oscilará entre 0 y 2h50 y entre 0 y 15h30 para el xeon.
Es, por tanto, responsabilidad de cada cual planificar sus experimentos para poder entregar a tiempo.

¿Cuándo entregar?

La documentación será subida al Campus Virtual y el código el definitivo dejado en las correspondientes carpetas, como muy tarde, antes de las 23:55:00 horas del día:

- Grupo de Prácticas de los lunes 6 de noviembre de 2022
- Grupos de Prácticas de los miércoles 8 de noviembre de 2022
- Grupos de Prácticas de los viernes 10 de noviembre de 2022

ANEXO I

Tabla 1 Tiempos experimentales. Función: Máquina: Versión:

m	n	k	Nº flop	t_c	Teórico	Empírico O0	Empírico O3
1000	1001	999					
2000	2001	1999					
3000	3001	2999					

La tabla 1 debe ser rellenada para los siguientes casos, para α=1.3 y β=1.7:

- Solución en python. Una tabla para cada máquina (i3, ryzen y xeon).
- *MyDGEMM*. Una tabla para cada máquina (i3, ryzen y xeon) y versión (secuencial/paralela).
- *MyDGEMMT*. Sólo máquina ryzen. Solo versión paralela.
- *MyDGEMMB*. Sólo máquina ryzen. Una tabla para cada versión (secuencial/paralela). En este caso, *m=n=k={1000, 2000, 3000}*.