Arquitectura de computadoras 2015

Laboratorio - MPI

Descripción general

El objetivo del proyecto es resolver un problema de simulación física, basado en un problema real (aunque simplificado por fines didácticos.

Se dispone de una placa cuadrada plana de un material uniforme (ej: una plancha de metal). En algunos puntos de esta placa se aplica calor con una fuente de calor (ej: una llama) de modo de mantener la temperatura constante en ese punto. Si bien la aplicación de calor es puntual, el calor tiende a distribuirse alrededor de la fuente y luego de cierto tiempo, los lugares de la placa cercanos a la fuente estaran más calientes que los lejanos. En esta simulación modelamos como varía la temperatura en distintos puntos a lo largo del tiempo.

Modelado del problema

En primer lugar, si bien la temperatura varía continuamente a lo largo de la placa, nuestro modelo va a ser discreto, es decir, separar la placa en una cantidad finita de áreas, donde mediremos la temperatura promedio en cada una de esas áreas. La división de la placa se hara en una grilla uniforme de $N \times N$ bloques, donde para cada área se almacenará la temperatura en un `double`.

Llamaremos j a la cantidad de fuentes de calor. La posición de la i-ésima fuente de calor (con $0 \leq i < j$) estará dada por valores enteros (x_i, y_i), donde $0 \leq x_i < N$ y $0 \leq y_i < N$. El valor x representa la columna de la grilla donde esta situada la fuente, y el valor y representa la fila. La temperatura de la i-ésima fuente de calor sera denominada t_i, y será un valor de punto flotante (de tipo `double`).

Si T es la matriz de temperaturas, su estado inicial será:

- $T_{pq} = t_i$ cuando $(q, p) = (x_i, y_i)$ para algún i
- $T_{pq} = 0$ caso contrario

La simulación a realizar será iterativa, en k pasos ($k \geq 0$). En cada paso se construye un T' a partir de T, que al final de la iteración sustituye al *T original*:

- $T'_{pq} = t_i$ cuando $(q, p) = (x_i, y_i)$ para algún i
- $T'_{pq} = promedio([T_{pq}] + + vecinos_T(q,p))$ caso contrario

En la definición anterior, *promedio* tiene su definición usual (*promedio(xs) = sum(xs) / len(xs)*). La función auxiliar $vecinos_T(x, y)$ devuelve intuitivamente los vecinos adyacentes de una posición (x, y) (arriba, abajo, izquierda, derecha) en T, teniendo cuidado con los bordes. Por ejemplo, si $N = 10$:

- $vecinos_T(0, 0)$ tiene 2 valores: $[T_{1\,0}, T_{0\,1}]$
- $vecinos_T(9, 3)$ tiene 3 valores: $[T_{8\,3}, T_{9\,2}, T_{9\,4}]$
- $vecinos_T(6, 8)$ tiene 4 valores: $[T_{6\,7}, T_{7\,8}, T_{6\,9}, T_{5\,8}]$

0 0	1 0								
0 1									

									9 2
								8 3	**9 3**
									9 4
						6 7			
					5 8	**6 8**	7 8		
						6 9			

Un poco mas formalmente:

- $vecinos_T(x, y)$ contiene a $T_{y\ x-1}$ cuando $x > 0$
- $vecinos_T(x, y)$ contiene a $T_{y\ x+1}$ cuando $x < N-1$
- $vecinos_T(x, y)$ contiene a $T_{y-1\ x}$ cuando $y > 0$
- $vecinos_T(x, y)$ contiene a $T_{y+1\ x}$ cuando $y < N-1$

En resumen, y desde muy alto nivel, el problema a calcular sin paralelismo y en seudocódigo es:

```
Leer datos de entrada N, k, j, (x[0], y[0], t[0]) ... (x[j-1], y[j-1], t[j-1])
Construir T inicial
repetir k veces:
    Dado T, construir T'
    T := T'
Mostrar T
```

Propuesta de paralelización

Puede notarse que el cálculo de cada celda de *T'* sólo depende de las celdas adyacentes, con lo cual si particionamos *T* en bloques contiguos entre los `comm_sz` procesos, solo deberemos comunicar las celdas del borde.

Se propone una separación por filas. Permitimos suponer que `comm_sz` divide a *N*, con lo cual cada proceso **calcula** *N*/`comm_sz` filas. Notese que en realidad cada proceso deberá tener **almacenadas** (*2 + N*/`comm_sz`) filas, ya que ademas de las filas que calcula, necesita la fila superior e inferior.

Entre pasos consecutivos de la iteración, los procesos deberan comunicarse la primera y última fila calculada a los vecinos.

En pseudocódigo, el algoritmo paralelo debería ser algo similar a:

```
funcion reset_sources(T):
    para cada i en 0...j-1:
        si y[i] es una fila que yo calculo:
            y = fila en T que corresponde a y[i]
            T[x[i], y] = t[i]

Construir T, T' de 2 + N/comm_sz filas cada una
Inicializar T en 0
reset_sources(T)
Para cada i en 1...k:
    Intercambiar filas limite con los vecinos
    Construir T' en base a promedios de T (sin importar fuentes de calor)
    T := T'
    reset_sources(T)
```

Formatos de entrada/salida

El programa recibira datos por entrada estándar en un formato prefijado, del cual se dan algunos ejemplos. El formato consiste, línea por línea

1. Un entero *N* (tamaño en filas/columnas de la grilla)
2. Un entero *k* (cantidad de iteraciones)
3. Un entero *j* (cantidad de fuentes de calor)
4. Una tripla $x_0\ y_0\ t_0$ separada por espacios; los dos primeros valores son enteros, el tercero es real. Esto describe la fuente de calor 0
5. Una tripla $x_1\ y_1\ t_1$ separada por espacios; los dos primeros valores son enteros, el tercero es real. Esto describe la fuente de calor 1
6. ...
7. Una tripla $x_{j-1}\ y_{j-1}\ t_{j-1}$ separada por espacios; los dos primeros valores son enteros, el tercero es real. Esto describe la fuente de calor j-1

La salida del programa mostrara una fila de la matriz por línea, en orden. Cada fila sera una lista de valores separados por espacios, con cada valor impreso usando formato `"%.2f"` de C

Recomendaciones para resolver el problema

Asumase que el valor de *j* es "pequeño" en lo que respecta a performance. Es decir, no vale la pena hacer un algoritmo complicado que lleve tiempo *O(log j)*, para sustituir uno simple de tiempo *O(j)*.

Usen `double` para el tipo de los valores de punto flotante

Chequeen la salida. Comparen con los ejemplos. vean que el grafico sea "suave". Si el problema es simétrico, la matriz debería ser simétrica y el dibujo también.

Testeen con comm_sz en 1 y en N

Testeen con k=0, deberían obtener el estado inicial

Las funciones "promedio" y "vecinos" son útiles para *describir* el problema, pero no para implementarlo (porque pasar listas de tamaño variable en C es complicado/incomodo). Busquen implementar algo equivalente de una forma mas directa.

Asegurense de que ningún proceso tenga una matriz de N×N en RAM cuando p>1. El problema se puede resolver con cada proceso manejando su matriz (y al final ir imprimiendo de a un bloque a la vez en el proceso 0).

Condiciones de entrega

- El trabajo es grupal, en grupos de hasta 3 personas.
- Los grupos no pueden compartir código entre sí.
- Cada grupo debe indicar sus integrantes al docente antes de la entrega.
- El trabajo se presenta el día viernes 13 de noviembre de 2015, para una evaluación oral.
- Todos los miembros del grupo serán evaluados y deben asistir y ser capaces de responder preguntas del proyecto.
- Las personas del grupo deben encargarse de traer el proyecto en un pendrive el dia de la evaluación.
- La aprobación del laboratorio es individuales.
- Se evaluará correctitud, claridad de la solución y aspectos de performance.

- El proyecto se califica como "no aprobado" o "aprobado".
- El trabajo se podrá recuperar el día 20 de noviembre.