Computación en GPU CC7515-1 - Otoño 2024

Control de Lectura 3

Sebastián Andrés Mira

P1

El problema que aborda el artículo es el mapeo del espacio de computación al dominio de un problema, en particular cuando hay threads que no caben en el dominio (pues se organizan en una matriz triangular) y deben ser descartados, es decir, el mapeo en td-problems.

Es relevante porque se desperdicia espacio de computación y porque hay una penalización en los condicionales que descartan threads dentro del kernel, entonces cualquier mejora en el problema beneficiará a otros problemas paralelizables.

P2

Es el mapeo en bloques propuesto para td-problems que lleva un problema de tamaño N a una grilla de tamaño $n' \times n'$, donde $n' = \sqrt{\frac{n(n+1)}{2}}$, en lugar de una $n \times n$ donde se desperdiciará espacio de computación.

$g(\lambda) = (i, j) = \left(\lfloor \sqrt{\frac{1}{4} + 2\lambda} - \frac{1}{2} \rfloor, \lambda - i(i+1)/2 \right)$ (pensando que se utiliza la diagonal)

Para $g(\lambda)$ λ debe tener un valor real que se encuentre las filas del bloque B_x y el B_{x+1}, entonces su fila (valor de i) debe ser el menor de esos valores, para que j represente el bloque dentro de esa fila. Acotando entonces los valores de x, se busca el valor real de $x^2 + x - 2\lambda = 0$ y se obtiene la función piso. Luego j es simplemente restar el indice lineal λ por el índice de la fila a la que representa i.

$$g(0) = \left(\lfloor \sqrt{\frac{1}{4}} - \frac{1}{2} \rfloor, \ -i(i+1)/2 \right) = (0,0)$$

$$g(3) = \left(\lfloor \sqrt{\frac{1}{4} + 6} - \frac{1}{2} \rfloor, 3 \ -i(i+1)/2 \right) = (2,0)$$

$$g(5) = \left(\lfloor \sqrt{\frac{1}{4} + 10} - \frac{1}{2} \rfloor, 5 \ -i(i+1)/2 \right) = (2,2)$$

Se puede pensar que se obtiene el mapeo correcto porque en ningún momento se obtuvo un índice que perteneciera a la matriz triangular superior, y se ve que por ejemplo g(1) y g(2), sin cálculos hechos, pertenecerán al (1,0) y (1,1).

P3

El BB es simplemente mapear a una matriz cuadrada y descartar aquellos threads sobre la diagonal. En rectangular box (RB) se rota de la matriz triangular inferior la porción triangular que queda, acomodándose en un grid rectangular, por ejemplo, los threads $t_{x,y}$ con $x > N/2$ y $y > N/2$ se llevan a posiciones en el rango $x, y \in [0, N/2]$. En REC se define el tamaño del problema como $N = m2^k$, y se llaman múltiples grids en k niveles, cada vez con grids más pequeños para poder acomodarse a la diagonal. Por último UTM es similar a LTM con las diferencias de que mapea threads y no bloques, llevando a más cálculos (y posibles errores) además de mapear a una matriz superior.

P4

Se obtiene comparando cómo los costos de mapeo de todos los bloques en BB vs LTM, pero el mapeo de BB se hace a n^2 bloques verificando que solo los de la matriz inferior trabajen (comparación de coste β), mientras que en LTM son $n(n+1)/2$ bloques los que son mapeados por un coste de τ (cálculo de raíz y el resto de ops aritméticas), entonces

$I = (\beta \cdot n^2)/(\tau \cdot n(n+1)/2) \approx 2\beta/\tau$ para n grande.

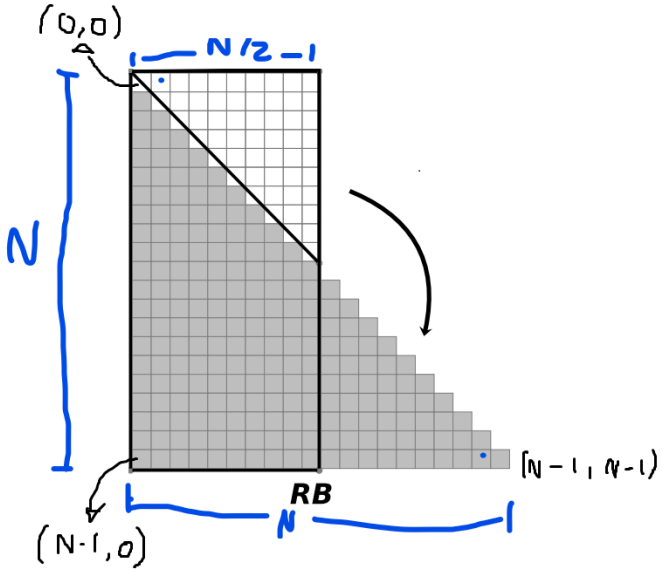
Con respecto a lo anterior entonces si $\tau \geq 2\beta$ no hay mejora, y realmente $\tau \geq \beta$ siempre, pues la raíz es más costosa que una comparación, así para una mejora factible se tiene entonces que:

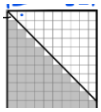
$$2\beta > \tau \geq \beta \Rightarrow \frac{1}{2\beta} < \frac{1}{\tau} \leq \frac{1}{\beta} \Rightarrow 1 < \frac{2\beta}{\tau} = I \leq 2$$

P5

RB

Para un espacio de NxN del cual solo se ocupará NxN/2 para la matriz triangular se puede hacer lo siguiente.



Los threads tendrán índices tx, ty tal que tx está en [0,N-1] y ty en [0, N/2]. Queremos llevar a los threads en la parte superior de la imagen  a que representa la parte inferior derecha de la matriz. Por eso entonces se les hace el siguiente mapeo

```
if (tx > ty) {  // si está bajo la diagonal
        i = tx-1
        j = ty-1
} else {   // si está en la diagonal o arriba, recuadros blancos de la imagen
        i = N-tx-2
        j = N - ty - 1
}
```

(sé que no aparece así en la lectura, pero tiene sentido, en serio ;-;)

UTM

En este caso para un índice de thread k, su par i,j está dado por:

$$i = \lfloor \frac{-(2n+1)+\sqrt{4n^2-4n-8k+1}}{-2} \rfloor$$

$$j = (i+1) + k - \frac{(i-1)(2n-i)}{2}$$

Caso muy similar a LTM pero mapeando threads y en una matriz superior, por lo que la fórmula se obtiene de una manera similar a UTM, pero obteniendo luego la traspuesta.

P6

RB

```
// kernel mapeado a una matriz rectangular de NxN/2

int tx = get_global_id(0);

int ty = get_global_id(1);

int i, j;

if (tx > ty) {

        i = tx-1;

        j = ty-1;

} else {

        i = N-tx-2;

        j = N - ty - 1;

}
```

UTM

```
// en este caso si es un id global no necesariamente en una matriz

int k = get_global_id(0);

int a = static_cast<int>(floor((sqrt(pow(2*n, 2)+ 2*n - 8 * k + 1) - (2*n+1))/-2));

int b = (a+1)+k-(a-1)*(2*n-1)/2;
```

P7

Más costosa: UTM 1*CostoRaiz + 7*CostoMult+ 2*CostoDivisión + 10* CostoSumaResta + 1*FuncPiso

= CostoRaiz+20*CostoAritmética

LTM 1*CostoRaiz + 2*CostoMult+ 3*CostoDivisión + 4* CostoSumaResta + 1*FuncPiso

= CostoRaiz+10*CostoAritmética

RB 1*Condicional+3*CostoSumaResta

Menos costosa BB: 1*Condicional

P8

Dummy kernel, para medir solo el costo de la función de mapeo.

Matriz de distancia euclidiana (EDM), para medir el rendimiento en problemas de memoria global.

Detección de colisiones 3D, para medir el rendimiento en problemas de memoria compartida.

Así los experimentos representan diferentes escenarios de uso y patrones de acceso a la memoria.

P9

En primer lugar porque se desligan del resto de la ejecución del kernel, pero principalmente porque no se desperdicia espacio de cálculo y por lo mismo la GPU puede ejecutar más rápidamente el espacio útil.