3.3 Algoritmos paralelos para operaciones vectoriales y matriciales

Computación de Altas Prestaciones

Grado en Ingeniería de Computadores

Raúl Durán Díaz

Curso académico 2018–2019

Índice

1. Introducción

Introducción

- Los algoritmos matriciales presentan varias características:
  - Aparecen con frecuencia es diversos tipos de problemas.
  - Se prestan naturalmente a descomposición de datos por la regularidad de los cómputos.
  - Usan una tarea por proceso.
  - Cada proceso actúa sobre un conjunto de datos distintos.
- Vamos a considerar dos operaciones:
  - El **producto matriz-vector**.
  - El **producto matriz-matriz**.
- Suponemos una matriz cuadrada $n \times n$.

2. Producto matriz-vector

2.1. Algoritmo 1-D

Algoritmo serie

```
int MatVec(double **A, double *x, double *y, int n)
{
int i, j;

   for (i = 0; i < n; i++)
   {
      y[i] = 0.0;
      for (j = 0; j < n; j++)
         y[i] += A[i][j]*x[j];
   }
}
```

Tiempo de ejecución serie

- Con el algoritmo anterior, es claro que

$$T_S = \Theta\left(n^2\right).$$

Partición 1-D por filas

- Supongamos que
  - tenemos n procesos y
  - cada proceso almacena una fila de la matriz A y un elemento del vector x.
- En otras palabras, el proceso P_i es dueño de $A_{i,*}$ y de x_i.
- Cada proceso es responsable de computar y_i.

Pasos del algoritmo paralelo, con $p = n$

- Como cada proceso necesita todo el vector x, hay que hacer una difusión todos-con-todos de x.
- Cada P_i realiza
$$y_i = \sum_{j=0}^{n-1} A_{ij} \times x_j.$$
- El vector y queda almacenado de modo que el proceso i dispone de la componente y_i (igual que la distribución original de x).

Tiempo de ejecución ($p = n$)

1. La difusión todos-con-todos lleva un tiempo $\Theta(n)$.
2. La computación en cada proceso lleva también $\Theta(n)$. El tiempo paralelo total es:
$$T_P = \Theta(n).$$
3. El coste es, claramente, $\Theta\left(n^2\right)$. Como el tiempo del algoritmo serie es el mismo, el algoritmo es óptimo.

Ejecución con $p < n$

- En el caso (más real) en que $p < n$ (incluso $p \ll n$), repartimos de modo que cada proceso disponga de n/p filas y de n/p elementos del vector x. Los pasos son como antes:
  1. Difusión del vector x entre todos los procesos.
  2. Cómputo de n/p elementos de y por parte de cada proceso.

Tiempo de ejecución ($p < n$)

1. La difusión todos-con-todos entre p procesos de un mensaje de tamaño n/p en una red hipercubo se lleva un tiempo
$$t_0 \log p + \frac{n}{r_\infty}\frac{p-1}{p} \approx t_0 \log p + \frac{n}{r_\infty}.$$
2. La parte de cómputo toma un tiempo $\frac{n^2}{p}$.
3. Por tanto, el tiempo paralelo es:
$$T_P = \frac{n^2}{p} + t_0 \log p + \frac{n}{r_\infty}.$$

2.2. Algoritmo 2-D

Partición 2-D

- Ahora suponemos una distribución de datos por bloques en 2-D, es decir, cada proceso tiene un trozo de fila y un trozo de columna.
- Empezamos por el caso en que $p = n^2$ y suponemos que
  - cada proceso solo tiene un elemento de la matriz A, y que
  - los n elementos de x están en los últimos n procesos (a razón de un elemento por proceso).

Pasos del algoritmo con $p = n^2$

- Envío del vector x a los procesos "dueños" de la diagonal.
- Difusión uno-a-muchos del elemento x_i a lo largo de la columna de procesos i.
- Cómputo del producto.
- Reducción-suma por filas del resultado, que queda almacenado en los mismos procesos que contenían el vector x.

Tiempo de ejecución ($p = n^2$)

- El tiempo de computación es fijo para cada proceso, pues consiste en una sola multiplicación.
- Los tiempos de todas las comunicaciones son $\Theta(\log n)$. Por tanto,

$$T_P = \Theta(\log n).$$

- El coste es $\Theta\left(n^2 \log n\right)$, que ahora no resulta óptimo, como en el caso 1-D.

Ejecución con $p < n^2$

- Supongamos que tenemos $p = q^2$ procesos y distribuimos la matriz A de modo que cada proceso tiene un bloque de $n/q \times n/q$ datos. El vector x está distribuido en porciones de n/q elementos en los últimos q procesos. Los pasos son:
  1. Difusión de las porciones de n/q elementos de x a los procesos "dueños" de la diagonal.
  2. Difusión uno-a-muchos de n/q elementos de x a lo largo de la columna de procesos.
  3. Cómputo del producto. Cada proceso realiza n^2/p multiplicaciones y n/q sumas locales.
  4. Reducción-suma por filas de los n/q valores, que quedan almacenados en los mismos procesos que contenían el vector x.

Tiempo de ejecución ($p < n^2$)

1. La difusión de las porciones de x a los procesos de la diagonal equivale a enviar un mensaje de tamaño n/q, que lleva un tiempo $t_0 + \frac{n}{qr_\infty}$.
2. La difusión uno-a-muchos lleva un tiempo $\left(t_0 + \frac{n}{qr_\infty}\right) \log q$. La reducción final (ignorando el tiempo de la suma, que es muy pequeño) necesita el mismo tiempo.
3. El cómputo necesita $\frac{n^2}{p}$.

Tiempo de ejecución ($p < n^2$)

El tiempo paralelo total es:

$$\begin{aligned} T_P &= \overbrace{n^2/p}^{\text{computación}} + \overbrace{t_0 + \frac{n}{qr_\infty}}^{\text{difusión diagonal}} + \\ &\quad \overbrace{\left(t_0 + \frac{n}{qr_\infty}\right)\log q}^{\text{difusión columna}} + \overbrace{\left(t_0 + \frac{n}{qr_\infty}\right)\log q}^{\text{reducción}} \\ &= \frac{n^2}{p} + \left(t_0 + \frac{n}{qr_\infty}\right)(1 + \log p)\,. \end{aligned}$$

Comparación 1-D – 2-D

- Para el mismo número de procesos, 1-D es más lento que 2-D.
- El método 1-D no puede usarse con más de n procesos, mientras que 2-D sí.
- El método 2-D es más escalable, es decir, mantiene mejor el rendimiento conforme aumentamos el número de procesos.

3. Producto matriz-matriz

3.1. Algoritmo simple

Algoritmo serie

```
void MatMat(double **A, double **B, double **C, int n)
{
int i, j, k;

   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         C[i][j] = 0.0;
         for (k = 0; k < n; k++)
            C[i][j] += A[i][k]*B[k][j];
      }
}
```

Tiempo de ejecución serie

- Con el algoritmo anterior, es claro que

$$T_S = \Theta\left(n^3\right).$$

Operaciones por bloque

- Para lo sucesivo es útil considerar la estructura de la matriz como si estuviera compuesta de $q \times q$ bloques cada uno de tamaño n/q (naturalmente, q debe dividir a n).

- En forma de bloques, la multiplicación de las matrices sigue siendo la misma de antes, es decir:

$$C_{i,j} = \sum_{k=0}^{q} A_{i,k} \times B_{k,j}. \tag{1}$$

- El número de operaciones es igual que antes: hay q^2 bloques, cada uno de los cuales exige $q \times (n/q)^3$ operaciones, lo que supone un total de $q^2 \times \left(q \times (n/q)^3\right) = n^3$ operaciones.

Partición 2-D

- Suponemos una distribución de datos por bloques en 2-D, es decir, cada proceso tiene un bloque como se ha descrito antes.
- Se tiene un número de procesos $p = q^2$ y matrices $n \times n$, tal que q divide a n.
- Podemos nombrar los procesos como $P_{i,j}$, con $0 \leq i < q$, $0 \leq j < q$.
- El proceso $P_{i,j}$ es "dueño" de $A_{i,j}$ y $B_{i,j}$ y responsable de calcular el resultado parcial $C_{i,j}$.

Pasos del algoritmo simple

- Observemos que el proceso $P_{i,j}$, para calcular $C_{i,j}$, necesita los bloques $A_{i,k}$ y $B_{k,j}$, con $0 \leq k < q$. Luego necesitamos una difusión de todos-con-todos de los bloques de A entre los procesos de la misma "fila" y de los bloque de B entre los procesos de la misma "columna".
- A continuación, debe realizar el cómputo del producto, de acuerdo a la fórmula (1).

Tiempo de ejecución del algoritmo simple

- Se requieren dos difusiones todos-con-todos entre grupos de q procesos, cada una con q mensajes de tamaño n^2/p. El tiempo total de comunicación es:

$$2\left(t_0 \log q + \frac{n^2}{p r_\infty}(q-1)\right)$$

- El tiempo de computación de un bloque, como hemos visto arriba, es n^3/p.
- Por tanto, el tiempo paralelo es, aproximadamente,

$$T_P = \frac{n^3}{p} + t_0 \log p + 2\frac{n^2}{q r_\infty}.$$

Coste y optimalidad

- Observemos que el coste del algoritmo simple es

$$pT_P = n^3 + p t_0 \log p + 2\frac{n^2 q}{r_\infty}.$$

- Por tanto, el coste es óptimo si $q = O(n)$, es decir, $p = O(n^2)$.

3.2. Algoritmo de Fox

Algoritmo de Fox

- El algoritmo simple es poco eficiente en cuanto al uso de memoria, pues cada proceso debe albergar todos los bloques de su fila y su columna para realizar el producto, lo que supone $2q$ bloques con n^2/p elementos cada uno.
- En el algoritmo de Fox veremos que cada proceso necesita solamente dos copias de un bloque de A, un bloque de B y uno de C. Con ello, la cantidad de memoria que necesita es cuatro veces el tamaño de un bloque, o sea, $4n^2/p$, que es $\Theta(n^2)$ para un número fijado de procesos.

Pasos del algoritmo de Fox

1. Inicializamos una variable $\ell = 0$.
2. Cada proceso $P_{i,i+\ell}$ difunde su bloque $A_{i,i+\ell \text{ mód } q}$ a todos los procesos $P_{i,k}$, $0 \leq k < q$, de su misma fila.
3. Cada proceso multiplica el bloque recibido $A_{i,i+\ell \text{ mód } q}$ por $B_{i+\ell \text{ mód } q,j}$ y lo acumula en $C_{i,j}$.
4. Todos los procesos mueven su bloque B al proceso de arriba, módulo q.
5. Se incrementa la variable $\ell = \ell + 1$ y se vuelve al paso 2.

Una sucesión de q difusiones y multiplicaciones completan la operación.

Disposición inicial

$A_{0,0}$ $B_{0,0}$	$A_{0,1}$ $B_{0,1}$	$A_{0,2}$ $B_{0,2}$	$A_{0,3}$ $B_{0,3}$
$A_{1,0}$ $B_{1,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{1,2}$	$A_{1,3}$ $B_{1,3}$
$A_{2,0}$ $B_{2,0}$	$A_{2,1}$ $B_{2,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{2,3}$
$A_{3,0}$ $B_{3,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{3,2}$	$A_{3,3}$ $B_{3,3}$

Figura 1: Bloques iniciales

Desplazamiento I

Desplazamiento II

Desplazamiento III

Desplazamiento IV

$A_{0,0}$ $B_{0,0}$	$A_{0,0}$ $B_{0,1}$	$A_{0,0}$ $B_{0,2}$	$A_{0,0}$ $B_{0,3}$
$A_{1,1}$ $B_{1,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,1}$ $B_{1,2}$	$A_{1,1}$ $B_{1,3}$
$A_{2,2}$ $B_{2,0}$	$A_{2,2}$ $B_{2,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,2}$ $B_{2,3}$
$A_{3,3}$ $B_{3,0}$	$A_{3,3}$ $B_{3,1}$	$A_{3,3}$ $B_{3,2}$	$A_{3,3}$ $B_{3,3}$

Figura 2: Primer desplazamiento

$A_{0,1}$ $B_{1,0}$	$A_{0,1}$ $B_{1,1}$	$A_{0,1}$ $B_{1,2}$	$A_{0,1}$ $B_{1,3}$
$A_{1,2}$ $B_{2,0}$	$A_{1,2}$ $B_{2,1}$	$A_{1,2}$ $B_{2,2}$	$A_{1,2}$ $B_{2,3}$
$A_{2,3}$ $B_{3,0}$	$A_{2,3}$ $B_{3,1}$	$A_{2,3}$ $B_{3,2}$	$A_{2,3}$ $B_{3,3}$
$A_{3,0}$ $B_{0,0}$	$A_{3,0}$ $B_{0,1}$	$A_{3,0}$ $B_{0,2}$	$A_{3,0}$ $B_{0,3}$

Figura 3: Segundo desplazamiento

$A_{0,2}$ $B_{2,0}$	$A_{0,2}$ $B_{2,1}$	$A_{0,2}$ $B_{2,2}$	$A_{0,2}$ $B_{2,3}$
$A_{1,3}$ $B_{3,0}$	$A_{1,3}$ $B_{3,1}$	$A_{1,3}$ $B_{3,2}$	$A_{1,3}$ $B_{3,3}$
$A_{2,0}$ $B_{0,0}$	$A_{2,0}$ $B_{0,1}$	$A_{2,0}$ $B_{0,2}$	$A_{2,0}$ $B_{0,3}$
$A_{3,1}$ $B_{1,0}$	$A_{3,1}$ $B_{1,1}$	$A_{3,1}$ $B_{1,2}$	$A_{3,1}$ $B_{1,3}$

Figura 4: Tercer desplazamiento

$A_{0,3}$ $B_{3,0}$	$A_{0,3}$ $B_{3,1}$	$A_{0,3}$ $B_{3,2}$	$A_{0,3}$ $B_{3,3}$
$A_{1,0}$ $B_{0,0}$	$A_{1,0}$ $B_{0,1}$	$A_{1,0}$ $B_{0,2}$	$A_{1,0}$ $B_{0,3}$
$A_{2,1}$ $B_{1,0}$	$A_{2,1}$ $B_{1,1}$	$A_{2,1}$ $B_{1,2}$	$A_{2,1}$ $B_{1,3}$
$A_{3,2}$ $B_{2,0}$	$A_{3,2}$ $B_{2,1}$	$A_{3,2}$ $B_{2,2}$	$A_{3,2}$ $B_{2,3}$

Figura 5: Cuarto desplazamiento

Tiempo de ejecución para el algoritmo de Fox

1. Cada paso del algoritmo involucra q difusiones, a razón de una por fila, y q desplazamientos, a razón de uno por columna. Por tanto, las comunicaciones suponen
$$q\left(t_0+\frac{n^2}{pr_\infty}\right)\log q+q\left(t_0+\frac{n^2}{pr_\infty}\right).$$
2. El tiempo de computación es n^3/p.
3. Por tanto, el tiempo paralelo es:
$$T_P=\frac{n^3}{p}+(1+\log q)\left(qt_0+\frac{n^2}{qr_\infty}\right).$$

Coste y optimalidad

- Observemos que el coste del algoritmo de Fox es
$$pT_P=n^3+q\,(1+\log q)\left(q^2t_0+\frac{n^2}{r_\infty}\right).$$
- Por tanto, el coste es óptimo si $p=O(\frac{n^2}{\log n})$.

3.3. Algoritmo de Cannon

Algoritmo de Cannon

- La ventaja que ofrece este algoritmo es que, en cada paso, solo se necesita que cada proceso tenga a su disposición un bloque de A y otro de B para multiplicarlos y almacenarlos acumulativamente en un bloque C. En este caso, la cantidad de memoria que necesita es tres veces el tamaño de un bloque, o sea, $3n^2/p$, que es $\Theta(n^2)$.

Pasos del algoritmo de Cannon

1. Se realiza una "alineación" inicial de modo que el bloque $A_{i,j}$ se mueve i lugares a la izquierda y el bloque $B_{i,j}$ se mueve j lugares hacia arriba todo módulo q.
2. Cada proceso $P_{i,j}$ multiplica su bloque de $A_{i,k}$ por el de $B_{k,j}$ y lo acumula en $C_{i,j}$.
3. Todos los procesos mueven su bloque A al proceso de su izquierda, módulo q.
4. Todos los procesos mueven su bloque B al proceso de arriba, módulo q.

Una sucesión de q multiplicaciones y desplazamientos completan la operación.

Disposición inicial

Primera alineación

Desplazamiento I

$A_{0,0}$ $B_{0,0}$	$A_{0,1}$ $B_{0,1}$	$A_{0,2}$ $B_{0,2}$	$A_{0,3}$ $B_{0,3}$
$A_{1,0}$ $B_{1,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{1,2}$	$A_{1,3}$ $B_{1,3}$
$A_{2,0}$ $B_{2,0}$	$A_{2,1}$ $B_{2,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{2,3}$
$A_{3,0}$ $B_{3,0}$	$A_{3,1}$ $B_{3,1}$	$A_{3,2}$ $B_{3,2}$	$A_{3,3}$ $B_{3,3}$

Figura 6: Bloques iniciales

$A_{0,0}$ $B_{0,0}$	$A_{0,1}$ $B_{1,1}$	$A_{0,2}$ $B_{2,2}$	$A_{0,3}$ $B_{3,3}$
$A_{1,1}$ $B_{1,0}$	$A_{1,2}$ $B_{2,1}$	$A_{1,3}$ $B_{3,2}$	$A_{1,0}$ $B_{0,3}$
$A_{2,2}$ $B_{2,0}$	$A_{2,3}$ $B_{3,1}$	$A_{2,0}$ $B_{0,2}$	$A_{2,1}$ $B_{1,3}$
$A_{3,3}$ $B_{3,0}$	$A_{3,0}$ $B_{0,1}$	$A_{3,1}$ $B_{1,2}$	$A_{3,2}$ $B_{2,3}$

Figura 7: Primera “alineación”

$A_{0,1}$ $B_{1,0}$	$A_{0,2}$ $B_{2,1}$	$A_{0,3}$ $B_{3,2}$	$A_{0,0}$ $B_{0,3}$
$A_{1,2}$ $B_{2,0}$	$A_{1,3}$ $B_{3,1}$	$A_{1,0}$ $B_{0,2}$	$A_{1,1}$ $B_{1,3}$
$A_{2,3}$ $B_{3,0}$	$A_{2,0}$ $B_{0,1}$	$A_{2,1}$ $B_{1,2}$	$A_{2,2}$ $B_{2,3}$
$A_{3,0}$ $B_{0,0}$	$A_{3,1}$ $B_{1,1}$	$A_{3,2}$ $B_{2,2}$	$A_{3,3}$ $B_{3,3}$

Figura 8: Primer “desplazamiento”

$A_{0,2}$ $B_{2,0}$	$A_{0,3}$ $B_{3,1}$	$A_{0,0}$ $B_{0,2}$	$A_{0,1}$ $B_{1,3}$
$A_{1,3}$ $B_{3,0}$	$A_{1,0}$ $B_{0,1}$	$A_{1,1}$ $B_{1,2}$	$A_{1,2}$ $B_{2,3}$
$A_{2,0}$ $B_{0,0}$	$A_{2,1}$ $B_{1,1}$	$A_{2,2}$ $B_{2,2}$	$A_{2,3}$ $B_{3,3}$
$A_{3,1}$ $B_{1,0}$	$A_{3,2}$ $B_{2,1}$	$A_{3,3}$ $B_{3,2}$	$A_{3,0}$ $B_{0,3}$

Figura 9: Segundo “desplazamiento”

$A_{0,3}$ $B_{3,0}$	$A_{0,0}$ $B_{0,1}$	$A_{0,1}$ $B_{1,2}$	$A_{0,2}$ $B_{2,3}$
$A_{1,0}$ $B_{0,0}$	$A_{1,1}$ $B_{1,1}$	$A_{1,2}$ $B_{2,2}$	$A_{1,3}$ $B_{3,3}$
$A_{2,1}$ $B_{1,0}$	$A_{2,2}$ $B_{2,1}$	$A_{2,3}$ $B_{3,2}$	$A_{2,0}$ $B_{0,3}$
$A_{3,2}$ $B_{2,0}$	$A_{3,3}$ $B_{3,1}$	$A_{3,0}$ $B_{0,2}$	$A_{3,1}$ $B_{1,3}$

Figura 10: Tercer "desplazamiento"

Desplazamiento II

Desplazamiento III

Tiempo de ejecución para el algoritmo de Cannon

1. La "alineación" inicial supone un desplazamiento circular de todas las filas y de todas las columnas, con una distancia máxima de $q-1$ nodos: esto lleva simplemente $2\left(t_0+\frac{n^2}{pr_\infty}\right)$.
2. Cada paso del algoritmo involucra de nuevo el envío de la pareja de bloques de A y de B al vecino, con lo que el total de las comunicaciones supone $2q\left(t_0+\frac{n^2}{pr_\infty}\right)$.
3. El tiempo de computación es el mismo de antes, n^3/p.
4. Por tanto, el tiempo paralelo es:

$$T_P = \frac{n^3}{p} + 2qt_0 + 2\frac{n^2}{qr_\infty}.$$

Coste y optimalidad

- Observemos que el coste del algoritmo de Cannon es

$$pT_P = n^3 + 2q^3t_0 + 2\frac{n^2q}{r_\infty}.$$

- Por tanto, el coste es óptimo (al igual que en el algoritmo simple) si $q = O(n)$, es decir, $p = O(n^2)$.