Capítulo 2

Conceptos básicos de paralelismo

2.1. Introducción

En este capítulo introduciremos algunos conceptos del paralelismo que se requeriran en la posterior discusión. Particularmente relevantes para ese propósito seran las medidas de eficiencia de los algoritmos paralelos, las cuales se aplicarán en la evaluación del algoritmo para cálculo de múltiplo de puntos sobre curvas elípticas.

2.2. Modelos de computación paralela

2.2.1. Computadoras paralelas

Con el avance en el desarrollo de los microprocesadores, así como de la tecnología de interconexión a altas velocidades, ha sido posible construir computadoras con cientos o incluso miles de procesadores que colaboran para resolver un problema. Recientemente han proliferado los *clusters* de computadoras en los cuales se interconectan varias computadoras personales utilizando dispositivos de red de alta velocidad que, mediante el software adecuado, se comportan como una sola computadora cuyo poder de computo resulta de la suma de la capacidad de todas las máquinas en el cluster [11].

Sin embargo, en la práctica el enorme incremento en el número de MIPS disponible en este tipo de máquinas no siempre se traduce en un incremento similar en *desempeño*. Para poder obtener el máximo desempeño, se deben mantener tantos procesadores *activos* como sea posible. Pero esto depende

particularmente del algoritmo utilizado. Puede ser que el problema en si mismo es tal que no permita una solución paralela "eficiente". También puede ocurrir que el tipo de paralelismo del algoritmo elegido no se adapte muy bien a la arquitectura de la máquina subyacente. Esto es, no todo algoritmo paralelo puede mapearse de forma eficiente a una computadora paralela real, o los procesadores están activos todo el tiempo pero realizan muchos cálculos redundantes. Para realizar un análisis de los factores que afectan el desempeño de los algoritmos paralelos se requieren modelos de paralelismo que sea independiente de las particularidades de cualquier arquitectura específica.

2.2.2. Modelos de paralelismo

Las computadoras paralelas pueden agruparse en dos grandes categorías:

1. Computadoras con memoria *compartida*
2. Computadoras con memoria *distribuida*

En las arquitecturas de memoria compartida, los procesadores se comunican haciendo *lecturas* y *escrituras* en la memoria compartida, mientras que en las arquitecturas de memoria distribuida, los procesadores se comunican a través de *envío y recepción de mensajes*. Las arquitecturas de memoria compartida requieren de mecanismos de sincronización global a diferencia de las arquitecturas de memoria distribuida en las cuales la sincronización está determinada por los datos.

Un modelo muy general de computadora paralela es la denominada *paracomputadora*. Una paracomputadora es un conjunto de procesadores de propósito general, P_i, $1 \leq i \leq p$, cada uno con su propia memoria local. Se supone que cada procesador conoce su propio *índice*. Los procesadores pueden correr de manera *síncrona* o *asíncrona*. En el modo síncrono, todos los procesadores corren programas idénticos pero operan sobre diferentes segmentos de datos dependiendo de sus índices. En un modo asíncrono, los procesadores pueden correr diferentes programas. En la discusión subsecuente, a menos que se indique lo contrario, se supone que los procesadores trabajan en modo síncrono. Los procesadores se comunican entre sí mediante lecturas y escrituras a una *memoria compartida* que consiste de un número *ilimitado* de celdas de memoria. Se supone que la entrada estará escrita en las primeras N celdas $M_1, M_2, \cdots, M_N$ de la memoria compartida y que el resto de las celdas se encuentran propiamente inicializadas (a ceros, por ejemplo). El número p de procesadores puede ser función del tamaño del problema,

es decir, dependiendo de la solución al problema, p puede ser función de N. En un paso de ejecución determinado, un procesador puede estar realizando algún cálculo o puede estar leyendo o escribiendo a la memoria compartida. Si limitamos la manera en la cual pueden efectuarse esas lecturas y escrituras a memoria compartida, obtenemos modelos más específicos.

1. *Modelos con capacidad de lectura y escritura concurrentes (CRCW)*
2. *Modelos con lectura concurrente y escritura exclusiva (CREW)*
3. *Modelos con capacidad de lectura y escritura exclusivas (EREW)*

Otra clase importante de modelos de computación paralela son los denominados *modelos de circuitos combinacionales*. Un circuito combinacional C es un grafo acíclico dirigido finito, con dos tipos de nodos: de *entradas* y de *operaciones*. Los nodos de entrada tienen *grado de entrada* cero y están etiquetados por los valores $x_1, x_2, \cdots, x_r$ y posiblemente las constantes 0 y 1. Los nodos de operación etiquetados con NOT tienen grado de entrada 1 y realizan la operación de *negación booleana*. Los nodos de operación etiquetados AND y OR tienen grado de entrada 2 y realizan la *conjunción* y *disyunción* de sus entradas, respectivamente. No existen, sin embargo, restricciones sobre el *grado de salida* de cualquier nodo. Algunos de los nodos de entrada y/o de operación pueden considerarse como nodos de *salida*.

Si existen r nodos de entrada y q nodos de salida, se dice que el circuito C calcula la función

$$f_C : \{0,1\}^r \rightarrow \{0,1\}^q.$$

De cualquier nodo de entrada debe existir *al menos una* trayectoria a algún nodo de operación. La *profundidad* d_ν de un nodo de operación ν se define como la longitud de la trayectoria máxima desde algún nodo de entrada a ν. La *profundidad d del circuito C* se define como el *máximo* de las profundidades de todos los nodos de salida. El número total s de nodos de operación en C es denominado el *tamaño* del circuito C. El número w_i de nodos de operación a profundidad i, es denominado *la anchura del circuito a la profundidad i*. El *máximo* de anchura en todos los niveles es denominado la *anchura, w, del circuito C*.

Sea $f =< f_N >$ una familia de funciones donde

$$f_N : \{0,1\}^{g(N)} \rightarrow \{0,1\}^{h(N)},$$

y $g(N)$ es una función creciente monótona sobre N. Sea $C =< C_N >$ una familia de circuitos combinatorios, donde C_N calcula f_N, esto es, C_N tiene $g(N)$ entradas y $h(N)$ salidas. Se dice que C es *uniforme* si para una N dada existe un algoritmo para generar al elemento C_N de la familia C.

2.2.3. Relaciones entre modelos de paralelismo

A continuación examinaremos la relación existente entre los diferentes modelos de paralelismo.

Teorema 2.1 *Sea $C =< C_N >$ una familia uniforme de circuitos combinatorios que calculan $f =< f_N >$. Sean s_N y d_N el tamaño y la profundidad de C_N, respectivamente. Entonces existe un algoritmo CREW para calcular f_N en $O(d_N)$ pasos utilizando*

$$p = \left\lceil \frac{s_N}{d_N} \right\rceil$$

procesadores.

Demostración
Supongamos que las N entradas al programa están almacenadas en las primeras N celdas de la memoria compartida.

Sea N_i el número de nodos en C_N a profundidad i, donde

$$\sum_{i=1}^{d_N} N_i = s_N.$$

Para simular el comportamiento de una compuerta con dos entradas, se requiere leer un par de datos de la memoria compartida (a lo sumo en dos pasos) y escribir el resultado en una celda distinta en la memoria compartida (esto toma un solo paso).

Sean $\mu_1, \mu_2, \cdots, \mu_k$ un conjunto de nodos a profundidad $i+1$ de cuyas entradas una es salida del nodo ν a profundidad i. Entonces, al simular los nodos a profundidad $i+1$, los procesadores correspondientes a los nodos $\mu_1, \mu_2, \cdots, \mu_k$ leen simultaneamente la celda de memoria que contiene el resultado del nodo ν. Por lo tanto, usando p procesadores, las compuertas a profundidad i pueden simularse en a lo sumo $k\lceil N_i/p \rceil$ pasos, donde $k \leq 4$.

Si T es el tiempo total para la simulación completa, entonces

$$T \leq \sum_{i=1}^{d_N} k \left\lceil \frac{N_i}{p} \right\rceil.$$

Y como $\lceil x \rceil < x + 1$ y $p = \lceil s_N/d_N \rceil$, tenemos que $T = O(d_N)$, lo cual completa la demostración. □

Al analizar la relación entre el modelo de circuitos y el modelo CRCW, resulta de mayor relevancia el número de aristas del circuito que el número

de nodos del mismo, por lo cual al llegar a este punto redefiniremos al *tamaño* del circuito como su *número de aristas*.

Teorema 2.2 *Sean s_N y d_N el tamaño (número de aristas) y profundidad, respectivamente, de un circuito combinatorio que calcula la función booleana f_N de N variables booleanas. Entonces existe un algoritmo CRCW que calcula f_N en d_N pasos utilizando s_N procesadores.*

Demostración
En primer lugar, asociemos un procesador con cada arista y una celda de memoria compartida por cada nodo en el circuito. Estas últimas son diferentes de las celdas que contienen las N entradas. Todas las celdas que corresponden a las compuertas AND son inicializadas en 1, mientras que las correspondientes a las compuertas OR son inicializadas a 0.

Sea un nodo μ en nivel i conectado a ν_{j_k}, un nodo a nivel j_k, por la arista e_k, donde $j_k < i$ para $k = 1, 2, \cdots, m$, esto es, las m entradas del nodo μ son las salidas de los m nodos ν_{j_k}, $k = 1, 2, \cdots, m$. Si μ es una compuerta AND, entonces el procesador correspondiente a la arista e_k escribe un 0 en la celda correspondiente a μ si y solo si lee un 0 en la celda ν_{j_k}, $k = 1, 2, \cdots, m$. Respectivamente, si es una compuerta OR escribe un 1 si y solo si lee un 1. Es obvio que la simulación completa toma d_N pasos. □

Para concluir esta sección mencionaremos que existe un algoritmo de $O(\log N)$ en tiempo y $O(N)$ espacio que simula un paso de lectura o escritura simultánea en el modelo CRCW de N procesadores sobre un modelo EREW de N procesadores.[1]

2.3. Medidas de desempeño

En esta sección caracterizaremos al desempeño de los algoritmos paralelos en el marco de referencia de los modelos introducidos en la sección anterior.

2.3.1. Aceleramiento, eficiencia y redundancia

La *razón de aceleramiento* S_p se define como el cociente entre el tiempo requerido para resolver un problema con el *mejor método serial conocido* y el tiempo que necesita para resolver el mismo problema un algoritmo paralelo que utiliza p procesadores. Por lo tanto, si $T(N)$ denota al tiempo requerido por el mejor algoritmo serial conocido para resolver un problema de tamaño

[1] Para mayores detalles, consultar [54], pp. 13-17.

N, y $T_p(N)$ denota el tiempo que necesita resolver el mismo problema un algoritmo paralelo utilizando p procesadores, entonces

$$S_p = \frac{T(N)}{T_p(N)}.$$

La *eficiencia por procesador*, E_p, es la razón entre el aceleramiento al número de procesadores, esto es,

$$E_p = \frac{S_p}{p}.$$

Dado que un algoritmo en paralelo puede implementarse trivialmente en forma serial, se puede comparar el desempeño del mismo algoritmo sobre una máquina serial y sobre un procesador en paralelo. Entonces tendremos la razón de aceleramiento

$$S_p^* = \frac{T_1(N)}{T_p(N)}.$$

De modo similar, la eficiencia por procesador en este caso es

$$E_p^* = \frac{S_p^*}{p}.$$

Puesto que un paso en un algoritmo paralelo corriendo en p procesadores toma a lo más p pasos cuando es implementado serialmente, tenemos que

$$T_1(N) \leq p \cdot T_p(N).$$

De lo anterior se sigue que

$$1 \leq S_p^* \leq p$$

y

$$0 \leq E_p^* \leq 1.$$

Se dice que un algoritmo tiene *aceleramiento óptimo* si tiene aceleramiento asintóticamente lineal, esto es, si

$$S_p(N) = \Omega(p).$$

2.4. Clase de complejidad paralela

La teoría de la complejidad actual se basa en dos clases de complejidad fundamentales: la clase P y la clase NP, en particular la clase de problemas NP-completos (NPC). Pertenecen a la clase P todos aquellos problemas que admiten una solución en tiempo *polinomial* sobre el tamaño de los datos de entrada en una máquina serial. Entre estos problemas se encuentran los de *ordenamiento, producto matricial, trayectoria más corta, cerradura transitiva, etc.*, por mencionar algunos. Por otra parte, los problemas en la clase NPC son tales que, en el peor de los casos, requieren una solución de tiempo *exponencial* en una máquina serial. Como ejemplos de problemas en la clase NPC podemos mencionar a los problemas de *satisfacibilidad, empaquetamiento y del agente viajero.*

2.4.1. La clase NC

Definiremos a la clase NC como

$$NC = \bigcup_{r \geq 1} NC^{(r)}$$

donde $NC^{(r)}$ es la clase de los problemas solubles por la familia de circuitos combinatorios $C =< C_N >$, donde el tamaño s_N y la profundidad d_N del circuito N-ésimo C_N son tales que para alguna $t \geq 1$,

$$s_N = O(N^t)$$

y

$$d_N = O((\log N)^r),$$

es decir, el tamaño de C_N está acotado por un polinomio en N y la profundidad de C_N está acotada por un polinomio en el logaritmo de N, donde N es el tamaño del problema.

Mediante el teorema (2.2) podemos traducir esta definición en términos del modelo de memoria compartida, como sigue: La clase NC consiste en todos los problemas que son solubles en tiempo polinomial en el logaritmo del tamaño del problema (*polilogarítmico*, para abreviar) usando un número polinomial de procesadores. Es obvio que la clase NC está contenida en la clase P, dado que cualquier problema en NC, cuando es resuelto serialmente, tomará tiempo polinomial. Uno podría pensar que todos los problemas en P, utilizando un número polinomial de procesadores, pueden ser resueltos en tiempo polilogarítmico. Sin embargo no ocurre así. Por ejemplo, el mejor

algoritmo paralelo que se conoce para el problema del flujo máximo toma $O(N^2 \log N)$ pasos usando $O(N)$ procesadores.[2]

[2]Ver [54], pag. 25