ANÁLISE DA COMPLEXIDADE DE ALGORITMO PARA ARQUITETURAS PARALELAS: ESTUDO DA TÉCNICA DA DIVISÃO E CONQUISTA

Laira Vieira Toscani*
Celso Carneiro Ribeiro**

SUMÁRIO

Este trabalho analisa a complexidade de uma implementação do método de desenvolvimento de algoritmos Divisão e Conquista numa arquitetura paralela com estrutura de árvore, comparando-a à complexidade de uma implementação seqüencial.

1. INTRODUÇÃO

Os sistemas multiprocessadores que vem recebendo muita atenção nos últimos anos só poderão ser utilizados em toda sua potencialidade quando se dispor de algoritmos paralelos eficazes /PRE 82/.

Alguns métodos de desenvolvimento de algoritmos seqüenciais já foram bastante estudados e difundidos. Extensões dos mesmos para máquinas paralelas estão sendo desenvolvidas. Torna-se então importante avaliar o ganho em eficiência resultante da utilização de máquinas e algoritmos paralelos.

A Divisão e Conquista é um dos métodos mais simples e de maior utilização em diversos problemas, tais como classificação,

* Mestre em Informática (PUC/RJ, 73), profa. adjunta da UFRGS; área de interesse: desenvolvimento e complexidade de algoritmos. Caixa Postal 1501, 90.001 - Porto Alegre, RS.

** Doutor Engenheiro pela ENST (especialidade em informática) e professor assistente da PUC/RJ; área de interesse: paralelismo, otimização combinatória e suas aplicações. Caixa Postal 38.063, CEP 22.452 - Rio de Janeiro, RJ.

busca binária e multiplicação de matrizes. Este trabalho analisa a complexidade de algoritmos que utilizam a Divisão e Conquista em máquinas paralelas com estrutura de árvore.

2. ALGORITMO PARALELO PARA A DIVISÃO E CONQUISTA

A Divisão e Conquista pode ser definida brevemente como segue: dada uma instância de um problema, ela é decomposta em subinstâncias menores (que são resolvidas separadamente) cujas soluções parciais são então combinadas para se obter a solução da instância original. Se o tamanho das subinstâncias é relativamente grande, com soluções não imediatas, o processo é aplicado novamente, para se obter subinstâncias ainda menores, até que se obtenham subinstâncias tão pequenas que possam ser resolvidas facilmente, de modo quase direto.

2.1 Máquina paralela com estrutura lógica de árvore

Uma máquina paralela com estrutura de árvore consiste de um conjunto de processadores (cada um com memória própria) conectados de maneira a formar uma árvore. Cada processador executa seu programa independentemente e se comunica com outro processador através das linhas de comunicação.

A notação que será usada a seguir para especificar comunicação entre dois processadores foi criada por Hoare e utilizada em /PET 81/, sendo sua sintaxe descrita através de uma BNF.

<comando-de-entrada>::=<fonte>?<lista-de-variáveis>
<comando-de-saída>::=<destino>!<lista-de-expressoes>
<fonte> ::=<nome de processador>
<destino> ::=<nome de processador>

A comunicação entre dois processadores se efetua quando: (a) o comando de entrada para um processador especifica como fonte o nome do outro processador; (b) o comando de saída do outro processador especifica como destino o nome do primeiro processador; e (c) a lista de variáveis do comando de entrada é coerente com a lista de expressões do comando de saída.

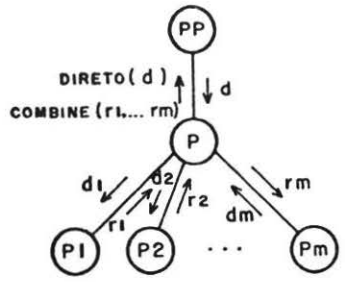
Será usada a notação { } para identificar um bloco de comandos.

2.2 Programa abstrato de Divisão e Conquista para uma máquina paralela com estrutura de árvore.

Suponha que um problema d é decomposto em **m** subproblemas $d_1, d_2, \ldots, d_m$, através da função **decompõe**; a função **direto** resolve os problemas simples, o predicado **simples** é verdadeiro quando é aplicado a um problema resolvível pela função **direto**. A função **combine**, combina as m soluções $r_1, r_2, \ldots, r_m$ dos problemas $d_1, d_2, \ldots, d_m$ a fim de obter o resultado do problema d. O seguinte programa abstrato define a Divisão e Conquista.

```
1. programa Divisão-e-Conquista
2.          var d, d1,d2,...dm,r1,r2,....,rm;
3.          processadores PP,P1,P2, ..., Pm;
4. {PP? d;
5.  se simples (d)
6.     então PP! direto (d);
7.     senão{d1,d2, ..., dm←decompõe(d);
8.           P1!d1; P2! d2; ...; Pm! dm;
9.           P1?r1; P2?r2; ..., Pm? rm;
10.          PP!combine (r1,r2, ..., rm)
11.         }
12. }
```

Cada processador da máquina executará o mesmo programa (**Divisão-e-Conquista**). PP representa o processador predecessor de P, exceto quando P é o processador raiz (neste caso, PP será o ambiente). Seja **d** a instância a ser resolvida pelo processador P, cujos sucessores são $P_1, P_2, \ldots$ e P_m. Esta instância é decomposta nas subinstâncias $d_1, d_2, \ldots, d_m$, que são tratadas em paralelo pelos processadores $P_1, P_2, \ldots, P_m$ que obtém respectivamente as soluções $r_1, r_2, \ldots x_m$.



3. ANÁLISE DA COMPLEXIDADE

A complexidade de pior caso de um algoritmo é, em geral, avaliada em função do **tamanho da entrada** (espaço de memória necessário para codificar e armazenar todos os dados do problema) dos dados do problema que ele se propõe a resolver. Por exemplo, se os dados correspondem a uma matriz, o tamanho da entrada do problema associado é o número de elementos desta matriz. Caso correspondam a uma lista o tamanho da entrada do problema será o número de elementos da lista.

Para exprimir a complexidade é conveniente utilizar a notação 0. Seja f uma função real não negativa da variável inteira positiva n. Diz-se que f é 0(g), ou f = 0(g), quando existirem constantes a > 0 e $n_0 \in N$ tais que f(n) $\geq$ a g(n), para qualquer $n \geq n_0$.

A partir de um programa abstrato é possível estudar-se a complexidade de uma classe de algoritmos. Esta classe é proveniente do programa abstrato pela substituição das definições abstratas dos predicados e funções (no programa da seção anterior: **simples**, **direto**, **decompõe** e **combine**) por predicados e funções executáveis. Esse estudo será desenvolvido sobre o programa abstrato **Divisão-e-Conquista**.

A função **decompõe** decompõe um problema de tamanho n em m problemas de tamanho $\frac{n}{c}$, onde m e c são constantes inteiras maiores que 1. A complexidade da solução de um problema de **tamanho** n é então expressa em termos de n, m, c e da complexidade das funções **direto**, **decompõe** e **combine**.

Chame genericamente $T_f\ n$ a complexidade do cálculo da função f para um dado de tamanho n: $T_{par}(n)$ a complexidade de tempo do programa abstrato **Divisão-e-Conquista** executado pelo nodo raiz da árvore de processadores da máquina em questão. Para fins de uma análise comparativa será chamada de $T_{seq}(n)$ a complexidade de tempo do programa abstrato seqüencial equivalente a **Divisão-e-Conquista** cujo estudo é apresentado em /TOS 86/.

Sem perda de generalidade supõe-se n uma potência de c (pode-se provar, que os resultados não se alteram se for retirada esta suposição).

A cada nível da árvore de processadores a complexidade é igual a complexidade do programa **Divisão-e-Conquista**, sem considerar as esperas referentes à $P_1?r_1, \ldots P_m? r_m$ e é

$$\begin{cases} T_{\textbf{direto}}(n) & \text{se simples(p)} \\ T_{\textbf{decompõe}}(n)+T_{\textbf{combine}}(\frac{n}{c}, \frac{n}{c}, \ldots, \frac{n}{c}) & \text{caso contrário} \end{cases}$$

onde n é o tamanho do problema atual d.

A complexidade total é igual a complexidade do programa abstrato **Divisão-e-Conquista** na raiz da árvore, que é dada por

$$T_{par}(n) = \begin{cases} T_{\textbf{direto}}(n) & \text{se simples (p)} \\ T_{\textbf{decompõe}}(n) + T_{par}(\frac{n}{c}) + T_{\textbf{combine}}(\frac{n}{c}, \ldots, \frac{n}{c}) & \text{caso contrário} \end{cases}$$

Este resultado diz que se d é suficientemente simples (**simples**(d)=verdade), d é resolvido pela função direto (em tempo $T_{\textbf{direto}}(n)$). Senão d é decomposto (em tempo $T_{\textbf{decompõe}}(n)$), em m subproblemas de tamanho $\frac{n}{c}$, que uma vez resolvidos em paralelo (em tempo $T_{par}(\frac{n}{c})$) tem seus resultados combinados (em tempo ($T_{\textbf{combine}}(\frac{n}{c}, \ldots, \frac{n}{c})$).

Entretanto $T_{par}(n)$ está expressa de forma muito genérica, sendo necessário para sua análise fazer-se algumas suposições com respeito a $T_{\textbf{direto}}(n)$, $T_{\textbf{decompõe}}(n)$ e $T_{\textbf{combine}}(\frac{n}{c}, \frac{n}{c}, \ldots, \frac{n}{c})$. Assim, foram selecionados alguns casos de estudo descritos abaixo.

Para simplificar os cálculos será suposto que **simples**(d) = **verdade** sss **tamanho** (d) = 1; isto é, o problema será decomposto até atingir o tamanho 1, quando então será resolvido pela função **direto**. Assim, $T_{direto}(n) = 0(1)$. Esta suposição é bem razoável e não altera a ordem de complexidade de **Divisão-e-Conquista**, se **simples**(d) = **verdade** sss **tamanho**(d) = k, onde k é uma constante qualquer.

Caso 1

O caso mais simples ocorre quando $T_{\textbf{decompõe}}$ e $T_{\textbf{combine}}$ são funções constantes, isto é $T_{\textbf{decompõe}}(n) = 0(1)$ e $T_{\textbf{combine}}(\frac{n}{c}, \ldots, \frac{n}{c}) = 0(1)$. Este caso ocorre no algoritmo de busca binária e algoritmo **MaxMin** de /AHO 74/.

$$T_{par}(n) = \begin{cases} 0(1) & \text{se } n = 1 \\ 0(1) + T_{par}(\frac{n}{c}) & \text{se } n > 1 \end{cases}$$

A solução dessa equação de recorrência e as das demais que surgirão nesta seção são discutidas em detalhes em /TOS 87/. Neste primeiro caso $T_{par}(n) = 0(\log_c n)$ e são necessários

$$\frac{1 - m^{1+\log_c n}}{1 - m}$$

processadores (pela definição de m-árvore completa de altura $\log_c n$). Na solução seqüencial $T_{seq}(n) = 0(m^{\log_c n})$.

Caso 2

Supõe-se que $T_{\textbf{decompõe}}(n) + T_{\textbf{combine}}(\frac{n}{c}, \frac{n}{c}, \ldots, \frac{n}{c}) = 0(n^k)$, $k \geq 1$; então

$$T_{par}(n) = \begin{cases} 0(1) & \text{se } n = 1 \\ 0(n^k) + T_{par}(\frac{n}{c}) & \text{se } n > 1 \end{cases}$$

Obtendo-se então $T_{par}(n) = b\, n^k \sum_{i=0}^{\log n - 1} \frac{i}{c^{ik}} + a$, onde a, b são constantes. Logo $T_{par}(n) = 0(n^k)$.

Sabendo-se que (conforme /TOS 86/)

$$T_{seq}(n) = \begin{cases} 0(n^k) & \text{se } m < c^k \\ 0(n^k \log_c n) & \text{se } m = c^k \\ 0(n^{\log_c m}) & \text{se } m > c^k \end{cases}$$

Verifica-se que embora quando $m < c^k$, não haja ganho em ordem de complexidade, nos outros casos o ganho é considerável. O algoritmo de classificação por intercalação **OrdInter** é um exemplo desse caso para $m = c$; os algoritmos **Mult** para multiplicação de inteiros e o de multiplicação de matrizes ilustram o caso para $m > c^k$. Estes três algoritmos são analisados em /TER 82/.

Nestes dois casos se considerou que há processadores suficientes para o problema ser decomposto até o tamanho 1, até atingir os processadores folhas e então serem resolvidos diretamente (isto é, a altura da árvore de processadores é maior ou igual a $\log_c n$). Entretanto, com a tecnologia atualmente disponível existe uma limitação grande no número de processadores que constituem a árvore. Nesse caso, ao atingir-se um processador folha o problema tem tamanho $n' > 1$ e tem então que ser resolvido seqüencialmente a partir deste ponto. O programa abstrato correspondente é igual ao anterior, incluindo a linha

6.5 senão se "P é folha" então PP! **solução**(d);

entre as linhas 6 e 7, sendo **solução** (d) o programa seqüencial que resolve o problema.

Caso 3

Supondo que se tem disponível uma máquina paralela, com estrutura lógica de uma m-árvore completa com K processadores, a complexidade do programa abstrato modificado é dado pela fórmula abaixo, que é aplicada uma só vez

$$T(n) = \begin{cases} T_{par}(n) & \text{se } n \leq n'' \\ T'(n) & \text{se } n > n'' \end{cases}$$

onde n" é o tamanho da maior instância original que pode ser resolvida totalmente em paralelo (isto é tamanho da maior instância que

após as decomposições sucessivas contenha nos processadores-folha subproblemas que podem ser resolvidos por direto). Caso $n \leq n''$ o tempo total é dado por $T_{par}(n)$. Caso contrário ($n > n''$), a complexidade do programa abstrato é definido por $T'(n)$. E $n''=c^h$

Seja $n' = \frac{n_0}{c^h}$, onde $h = \log_m \frac{K(m-1)+1}{m}$ e n_0 = tamanho da instância original. Então

$$T'(n) = \begin{cases} T_{seq}(n) & \text{se } n \leq n' \\ T_{\text{decompõe}}(n) + T'(\frac{n}{c}) + T_{\text{combine}}(\frac{n}{c}, \frac{n}{c}, \ldots \frac{n}{c}) & \text{se } n > n' \end{cases}$$

O desenvolvimento desta recorrência (conforme /TOS 87/) leva a $T(n) = O(T_{seq}(n))$, ou seja, no caso em que o número de processadores não permite obter um paralelismo total, não há ganho de complexidade teórica entre o programa paralelo e o seqüencial, o que confirma um resultado clássico em técnicas de paralelismo.

4. CONCLUSÕES

Foram estudados alguns casos típicos e usuais de algoritmos desenvolvidos pela técnica da Divisão e Conquista. Discutiu-se o número de processadores necessários e a ordem de complexidade destes algoritmos considerando-se máquinas paralelas com estrutura de árvore. Na maioria dos casos verificou-se um ganho considerável de eficiência, em relação a solução seqüencial. Quando não há processadores suficientes para decompor o problema até o mesmo poder ser resolvido diretamente, entretanto, não há ganho em ordem de complexidade, o que é um resultado muito importante no que diz respeito aos resultados práticos que podem ser obtidos com as atuais limitações existentes nas arquiteturas paralelas conhecidas.

Peters analisa em /PET 81/ o desempenho da técnica Divisão e Conquista numa máquina paralela com estrutura de árvore, para m = c = 2 e da Divisão e Conquista Multidimensional.

Em geral os problemas práticos quando implementados numa máquina paralela com estrutura de árvore requerem um grande número

de processadores, mas a cada instante apenas uma pequena fração está ativa, pois um único nível da árvore tem seus processadores ativos. Como uma alternativa /BUR 84/ propõe uma máquina virtual, cujos componentes fazem parte de uma rede de processadores interconectados e um algoritmo decide quando e onde cada processo será executado.

REFERÊNCIAS

/AHO 74/ AHO, A.V. HOPCROFT, J.E. & ULLMAN, J.D. The Design and Analysis of Computer Algorithms. Reading, Mass., Addison-Wesley, 1974.

/BRO 80/ BROWNING, S.A. Algorithms for Tree Machine. In: Introduction to VLSI Systems. (C. Mead & L. Conway, authors). Addison-Wesley, 1980. p.295-312.

/BUR 84/ BURTON, F.W. & HUNTBACH, M.M. Virtual Tree Machine. IEEE Transactions on Computers. Vol C-33. No.3 March 1984. p.278-80.

/HOR 83/ HOROWITZ, Z. & ZORAT, A. Divide and Conquer for Parallel Processing. IEEE Transactions on Computers. Vol C-32 No. 6, June 1983. p.582-5.

/KUN 79/ KUNG, H.T. The Structure of Parallel Algorithms. CMU-CS 79-143. Carnegie-Mellon University August, 1979.

/LIN 81/ LINT, B. & AGERWALA, T. Communication Issues in the Design and Analysis of Parallel Algorithms. IEEE Transactions on Software Engineering. Vol. SE-7 No. 2. March 1981. p.174-88.

/OPA 85/ OPATRNY, J. Parallel Programming Constructs for Divide and-Conquer, and Branch-and-Bound Paradigms. Information Systems and Operational Research INFOR. vol. 23 No.4, November 1985. p.403-16.

/PET 81/ PETERS, F. Tree Machines and Divide - and - Conquer Algorithms. Lecture Notes CS111, 1981. p.25-35.

/PRE 82/ PREPARATA, F. Algorithms Design and VLSI Architectures. IBM Note de Seminari e Conferenze. Elaboratori Paralleli e Calcolo Scientifico - Roma 3-5 Marzo 1982. 20p.

/RIB 83/ RIBEIRO, C.C. Parallel Computer Models and Combinatorial Algorithms. Annals of Discrete Mathematics 31. Elsevier. Science Publishers B.V. (North-Holland) 1987. p.325-64.

/TER 82/ TERADA, R. Desenvolvimento de Algoritmos e Complexidade de Computação. Depto. de Informática. PUC/RJ, 1982.

/TOS 86/ TOSCANI, L.V. & VELOSO, P.A.S. Divisão e Conquistes: Análise da Complexidade. Anais do VI Congresso da Sociedade Brasileira de Computação. Olinda-PE, 19-25, julho 1986. p.89-104.

/TOS 87/ TOSCANI, L.V. & RIBEIRO, C.C. Análise da Complexidade da Divisão e Conquista para Máquinas Paralelas com estrutura de árvore. A publicar.