Análise de Complexidade e Desempenho de Algoritmos para Multiplicação de Matrizes

Clarissa Marquezan, Diego Contessa, Rodrigo Alves, Tiarajú Diverio
Universidade Federal do Rio Grande do Sul - Instituto de Informática
{clarissa, contessa, sanger, diverio}@inf.ufrgs.br

Introdução

O conhecimento de técnicas de paralelização de algoritmos e de programação concorrente é importante para muitas áreas do conhecimento, como a química, a física e a engenharia. Grande parte destes problemas podem ser modelados, ou podem ser reduzidos a problemas da Álgebra Linear, onde a operação de multiplicação de matrizes é uma das operações básicas. Assim, é necessário que se tenha o cuidado de projetar algoritmos que efetuem essa operação com o melhor desempenho possível.

O presente trabalho tem por objetivo apresentar algoritmos de multiplicação de matrizes, visando avaliações de complexidade e desempenho. Os algoritmos implementados tratam exclusivamente de matrizes densas constituídas por elementos em ponto flutuante. Cabe ressaltar que serão utilizadas apenas matrizes quadradas cuja ordem seja n, sendo n uma potência de 2.

Ambiente de programação e execução

Para a implementação dos algoritmos utilizou-se o compilador *gcc* da linguagem *C*. Também usou-se a biblioteca *MPICH 1.2.2* (padrão *MPI*), para as versões paralelas.

O ambiente de execução dos programas foi o *cluster* Myrinet do Instituto de Informática da UFRGS, sendo utilizadas quatro máquinas *Dual Pentium Pro* de 200 MHz, com 128 MB de memória *RAM*.

Descrição dos algoritmos estudados

A seguir são introduzidos os algoritmos convencional, de *Strassen* e de *Winograd.* Um dos objetivos desse trabalho e da descrição dos algoritmos é analisar os efeitos causados pela variação na ordem dos laços que acessam os dados das matrizes.

Algoritmo convencional

Esse é o algoritmo tradicionalmente empregado. Nele, é calculado um elemento de cada vez, através de multiplicações e somas entre elementos das linhas da primeira matriz e elementos das colunas da segunda matriz (produtos escalares). Informalmente, pode-se dizer que opera num sistema de "*linha-por-coluna*". O cálculo segue a fórmula:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$, onde i, j=1 .. n, sendo n a ordem das matrizes A e B.

O esquema básico se constitui de três laços aninhados, associados, respectivamente, às variáveis i, j e k, que fazem com que seja possível a varredura das matrizes para o cálculo do produto [ROO 00]. No caso desse algoritmo, o laço mais externo das iterações incrementa a variável i, o segundo incrementa a variável j e o laço mais interno trata da variável k. Dessa forma, diz-se que ele trabalha na ordem *ijk*.

Também foram testadas outras variações, usando formas diferentes no acesso às matrizes, como por exemplo, "*coluna-por-linha*". Essa forma é chamada, na literatura, de ordem *kij* [COS 95], e consiste numa inversão nos índices dos laços. Desse modo, o laço mais externo incrementa a variável k, o segundo incrementa a variável i e o mais

interno incrementa a variável *j*. Essa variante produz, a cada iteração, resultados parciais em todas as posições da matriz resultado, diferentemente da primeira implementação, que a cada iteração calcula uma posição da matriz resultado.

A implementação de uma variante do método causou a implementação de mais quatro formas, diferindo entre si pela ordem que os índices de acesso às matrizes são incrementados [KRO 85]. Analisando os resultados, se decidiu usar nas comparações de desempenho as formas *ijk* (mais tradicional) e *kij* (mais eficiente em primeira análise).

Algoritmo de *Strassen*

Em 1969, *Strassen* desenvolveu um novo método para multiplicar matrizes. Esse método mostra-se mais eficiente que o algoritmo convencional para matrizes de ordem suficientemente alta [LAK 90]. Sejam A e B duas matrizes quadradas de ordem 2. Para calcular o produto $C=AB$, são realizadas as operações mostradas na Fig. 1 (a). Esse algoritmo realiza sete multiplicações e dezoito adições. A redução no número de multiplicações, muito mais onerosas que as somas, determina seu ganho de eficiência.

O método de *Strassen* não requer comutatividade nas multiplicações e, por esse motivo, é possível considerar os elementos A_{ij} e B_{ij} como sendo matrizes, resolvendo o produto de forma recursiva. (Essa visão foge do escopo desse trabalho).

O algoritmo desenvolvido trabalha de forma semelhante ao convencional, tendo sido também implementado nas variantes *ijk* e *kij*. Porém, cada operando é uma matriz de ordem 2. As multiplicações entre esses operandos são feitas usando as equações da Fig. 1(*a*). Vale notar que essas alterações fazem com que se multiplique duas linhas por duas colunas a cada vez (ou o inverso, no caso da variante *kij*).

Algoritmo de *Winograd*

Em 1987, *Winograd* [LAK 90] criou uma variante do algoritmo de *Strassen*, que realiza três somas a menos que o método de *Strassen*. Isso trouxe uma melhoria no tempo de execução. A Fig.1(*b*) apresenta as equações usadas no algoritmo de *Winograd*.

$X_1=(A_{11}+A_{22})+(B_{11}+B_{22})$
$X_2=(A_{21}+A_{22})B_{11}$
$X_3=A_{11}(B_{12}-B_{22})$
$X_4=A_{22}(B_{21}-B_{11})$
$X_5=(A_{11}+A_{12})B_{22}$
$X_6=(A_{21}-A_{11})(B_{11}+B_{12})$
$X_7=(A_{12}-A_{22})(B_{21}+B_{22})$

$C_{11}=X_1+X_4-X_5+X_7$
$C_{12}=X_3+X_5$
$C_{21}=X_2+X_4$
$C_{22}=X_1+X_3-X_2+X_6$

(*a*)

$x_1=A_{21}+A_{22}$
$x_2=x_1-A_{11}$
$x_3=A_{11}-A_{21}$
$x_4=A_{12}-x_2$
$x_5=B_{12}-B_{11}$
$x_6=B_{22}-x_5$
$x_7=B_{22}-B_{12}$
$x_8=x_6-B_{21}$

$y_1=x_2x_6$
$y_2=A_{11}B_{11}$
$y_3=A_{12}B_{21}$
$y_4=x_3x_7$
$y_5=x_1x_5$
$y_6=x_4B_{22}$
$y_7=A_{22}x_8$

$z_1=y_1+y_2$
$z_2=z_1+y_4$

$C_{11}=y_2+y_3$
$C_{12}=z_1+y_5+y_6$
$C_{21}=z_2-y_7$
$C_{22}=z_2+y_5$

(*b*)

Figura 1: Equações usadas no algoritmo de *Strassen* (*a*) e de *Winograd* (*b*)

O algoritmo foi implementado da mesma forma que o anterior, com alterações apenas na forma como se dá o produto entre as matrizes de ordem 2.

Paralelização dos algoritmos

Os programas paralelos utilizam o paradigma *mestre-escravo*, tendo-se um mestre e sete escravos. Garante-se que cada processador terá apenas um processo.

É realizado um particionamento estático no qual, no início da execução, os dados são distribuídos de forma mais igualitária possível entre os escravos.

Na implementação da variação *ijk* dos algoritmos, cada escravo recebe apenas as linhas da matriz A as quais lhe foram atribuídas para o cálculo de sua parte da matriz C, além de toda matriz B. Na implementação da variante *kij*, cada escravo recebe uma parte da matriz A, e um pedaço de igual tamanho da matriz B. Então, calcula um resultado parcial com todas as posições da matriz C. Ao final de suas computações, cada escravo envia seu resultado ao mestre que calcula o resultado final da multiplicação.

Complexidade dos algoritmos

A complexidade de tempo dos algoritmos implementados foi determinada em termos do número de multiplicações e adições necessárias a cada método, pois essas são as operações fundamentais do produto de matrizes.

A primeira linha da Tab. 1 mostra uma comparação entre o número de operações em função da ordem das matrizes, para os diferentes métodos. O restante da tabela traz o número de operações executadas para matrizes de ordem 128 e 2048.

	Convencional		*Strassen*		*Winograd*	
	Multiplicações	Adições	Multiplicações	Adições	Multiplicações	Adições
N	N^3	$N^3 - N^2$	$7(N/2)^3$	$(11N-4)(N^2/4)$	$7(N/2)^3$	$(19N^3/8)-N^2$
128	2.097.152	2.080.768	1.835.008	5.750.784	1.835.008	4.964.352
2048	8.589.934.592	8.585.740.288	7.516.192.768	23.618.125.824	7.516.192.768	20.396.900.352

Tabela 1: Quantidade de operações de cada método

Observando os dados da Tab. 1, pode-se notar que os métodos de *Strassen* e de *Winograd* realizam um número muito maior de operações que o convencional. Apesar desse fato, esses algoritmos se mostram mais eficientes (ver seção Avaliação de Desempenho). Isso se deve ao fato dos algoritmos realizarem muito mais somas, e um número reduzido de multiplicações, que são as operações mais custosas. Nota-se também que o algoritmo de *Winograd* executa o mesmo número de multiplicações que o de *Strassen*, mas seu número de somas é menor, o que causa um ganho de eficiência.

Avaliação de desempenho

Os algoritmos foram implementados nas variações *ijk* e *kij*, de forma seqüencial e paralela (utilizando oito processadores). Na Tab. 2 são apresentados os tempos de execução para duas ordens de matrizes (256 e 1024) utilizando todas as variações implementadas. Estes tempos foram obtidos através de uma média aritmética simples entre, no mínimo, dez execuções de cada versão. Observando-se a tabela, é visível o ganho obtido, mesmo no paradigma seqüencial, apenas com a alteração do método utilizado e com a variação na ordem dos laços. Cabe ressaltar que o estudo da complexidade dos métodos foi comprovado pelo ganho obtido na mudança de método.

	Matriz de ordem 256				Matriz de ordem 1024			
	Seqüencial		Paralelo		Seqüencial		Paralelo	
	ijk	*kij*	*ijk*	*kij*	*ijk*	*kij*	*ijk*	*kij*
Convencional	6,1408	3,8621	1,9188	1,6376	503,2457	255,3755	95,6779	54,4270
Strassen	2,7819	2,5098	1,3401	1,4062	212,3698	172,2353	44,1091	39,8117
Winograd	2,4245	2,2165	1,2968	1,3684	192,1026	155,0460	41,0424	37,1859

Tabela 2: Tempo de execução dos algoritmos usando *ijk* e *kij* (em segundos)

A melhoria dos tempos obtida com a alteração da ordem de execução dos laços, deve-se à ocorrência de um número menor de *cache misses*, visto que essa ordem influencia diretamente a maneira como a memória é varrida. Ao implementar a variante *kij*, há uma maior probabilidade de que os dados a serem acessados estejam na *cache*.

É interessante notar, ainda, na Tab. 2, que houve uma perda de desempenho dos métodos de *Strassen* e *Winograd* no paradigma paralelo entre as variações *kij* e *ijk*, para matrizes de ordem 256, devido a comandos adicionais de controle que não fazem parte do algoritmo em si. Para matrizes de ordem maior, esses efeitos não são mais notados.

A Fig. 2 mostra as diferenças nos tempos de execução dos algoritmos. Percebeu-se que o algoritmo com melhor *Speedup* (5,72) e maior *eficiência* (0,715) é o de

Strassen na variante *ijk*. Entretanto, esse mesmo método apresenta o pior *Speedup* e a menor *eficiência* na variante *kij*. Estes valores são 4,11 e 0,514, respectivamente.

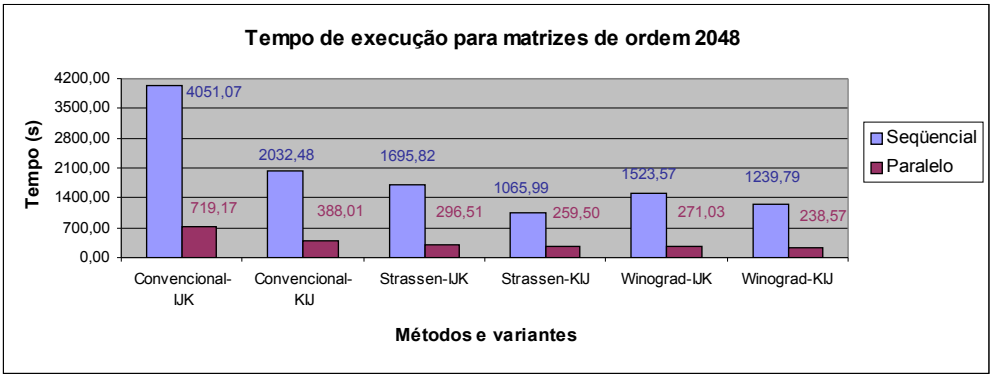

Figura 2: Comparação de desempenho entre métodos e variantes

Conclusões

A análise da complexidade de tempo de um algoritmo é importante mas não suficiente para compreensão total de um problema. Deve-se levar em conta a complexidade de comunicação, a forma como os dados são acessados, as estruturas de controle e os dados usados no desenvolvimento do algoritmo. Por exemplo, ao trocar a ordem dos laços no método de *Strassen,* obteve-se melhor desempenho que o método de *Winograd*, apesar do primeiro requerer um maior número de operações.

Deve-se notar que o estudo de paralelismo e de alto desempenho deve ser centrado na compreensão de algoritmos e técnicas gerais. A tecnologia determinará o quão rápido um algoritmo executará, mas o mais importante é que os algoritmos sejam descrições suficientemente genéricas de paralelismo, adaptadas para a arquitetura a ser utilizada.

Agradecimentos

Ao CNPq e ao Programa PIBIC CNPq/PROPESQ UFRGS pelas bolsas de Iniciação Científica.

Referências

[COS 95] COSNARD, M; TRYSTRAM, D. **Parallel Algorithms and Architectures**. London: International Thomson Computer Press, 1995. 550p.

[KRO 85] KRONSJÖ, L. **Computational complexity of sequential and parallel algorithms**. London: John Wiley & Sons, 1985. 224 p.

[LAK 90] LAKSHMIVARAHAN, S; DHALL, S. K. **Analysis and Design of Parallel Algorithms**: Arithmetic and Matrix Problems. New York: McGraw-Hill, 1990.

[ROO 00] ROOSTA, S. H. **Parallel Processing and Parallel Algorithms**: Theory and Computation. New York: Springer-verlag, 2000. 566p.