COMPILAÇÃO DE PROGRAMAS SEQUENCIAIS PARA MULTIPROCESSAMENTO.

Autor: Fábio Carneiro Mokarzel
Afiliação: Instituto Tecnológico de Aeronáutica
CTA - ITA - IED - Tel: (0123) 21-9530
CEP : 12225 - São José dos Campos - SP

SUMÁRIO

Este trabalho apresenta uma metodologia de compilação de programas escritos em linguagens seqüenciais para um ambiente de multiprocessamento, resultante de um estudo sobre o paralelismo não explícito desses programas.

Tal metodologia se divide em dez etapas e consiste basicamente em construir um grafo das dependências entre os comandos do programa a ser compilado, aplicar técnicas para eliminar algumas dessas dependências, fazer uma decomposição dos comandos repetitivos em comandos de menor escopo e aplicar técnicas para a resolução paralela dos comandos resultantes.

INTRODUÇÃO

Um dos principais fatores responsáveis pela alta produtividade dos computadores dos últimos anos é a sua capacidade de multiprocessamento. Arquiteturas de computadores têm sido desenvolvidas e muitas já estão em plena utilização, podendo suportar centenas ou até milhares de processadores e sua finalidade é a de explorar ao extremo o paralelismo dos programas.

As linguagens mais conhecidas na atualidade, FORTRAN, COBOL, ALGOL, PASCAL, BASIC, PL1, nas quais está escrita a grande maioria dos programas hoje em utilização nos mais variados tipos de entidades da nossa sociedade, possuem um caráter seqüencial e o paralelismo existente em seus programas não é explícito. Isto dificulta a geração de um código eficiente com um bom grau de aproveitamento desse paralelismo nesses potentes multiprocessadores.

Este trabalho apresenta uma metodologia para a construção de compiladores que analisam programas seqüenciais, descobrindo boa parte de seu paralelismo não explícito, orientando a geração de um

código baseado nesse paralelismo detectado. Ela é uma coletânea de idéias, técnicas e diretrizes sobre o assunto, baseadas, a maioria, em trabalhos feitos pelo prof. David J. Kuck e sua equipe, na Universidade de Illinois, EUA. A linguagem na qual os programas deste trabalho estão escritos é um sub-conjunto de PASCAL contendo comandos de atribuição, de entrada e saída, condicionais (IF-THEN e IF-THEN-ELSE) e de repetição (WHILE e FOR). Ela não possui comandos do tipo GO TO nem subprogramação.

ETAPAS DA COMPILAÇÃO DE PARALELISMO

O método de compilação de programas seqüenciais para um ambiente de multiprocessamento, aqui desenvolvido, se divide em 10 etapas a saber:

- Análise sintática e armazenamento do programa fonte;
- Construção do grafo de dependências do programa;
- Redução do número de dependências do grafo;
- Decomposição dos comandos repetitivos;
- Quebra de ciclos;
- Substituição de variáveis por espressões;
- Repetições sucessivas das 2 etapas anteriores;
- Tratamento dos comandos repetitivos decompostos;
- Tratamento das expressões;
- Geração de código.

A referência Mokarzel, 1984 apresenta detalhes minuciosos das 4 primeiras etapas e idéias práticas das 6 última , podendo ser consultada para maiores esclarecimentos. Este trabalho limita-se a um rápido resumo destas etapas.

ANÁLISE SINTÁTICA E ARMAZENAMENTO DO PROGRAMA FONTE

Na compilação visando a execução em paralelo de um programa, é exigido que o programa fonte seja analisado inúmeras vezes, de modo que é necessário manter sua estrutura para armazenar o programa. Assim, durante a análise sintática do programa fonte, este é armazenado para análises posteriores. Durante este armazenamento, o programa fonte deve sofrer algumas modificações, cuja finalidade é simplificar o trabalho das próximas etapas da compilação. Assim, comandos de entrada e saída e comandos condicionais são transformados em comandos

especiais de atribuição; comandos de atribuição são criados para os limites dos comandos FOR e para as condições dos comandos WHILE; os nomes dos índices dos comandos FOR são mudados; índices de repetição são dados aos comandos WHILE para torná-los semelhantes aos comandos FOR; comandos de atribuição são criados para o cálculo dos limites finais destes índices e alguns comandos WHILE são efetivamente transformados em FOR.

CONSTRUÇÃO DO GRAFO DE DEPENDÊNCIAS DO PROGRAMA.

Definimos "componente" de um programa, qualquer de seus comandos de atribuição ou qualquer cabeçalho de comando repetitivo desse programa.

A seguir, é apresentada a idéia de "dependência" entre componentes de um programa. Deve-se consultar as referências Banerjee, 1979, Kuck, 1981 e Mokarzel, 1984 para uma definição mais rigorosa. Seja o bloco de comandos de atribuição da figura 1.

```
C1 :  A  :=  B + C ;
C2 :  D  :=  A + E ;
C3 :  E  :=  F + G ;
C4 :  H  :=  E + I ;
C5 :  E  :=  J + K ;
C6 :  L  :=  E + H ;
```

Figura 1 - Programa com várias dependências entre seus componentes.

Observa-se três tipos distintos de dependências neste programa:

1º) Dependência direta: C2 deve ser executado depois de C1, pois precisa do valor de A, calculado em C1. Simbolicamente pode-se escrever (C1 dd C2);

2º) Antidependência: C3 deve ser executado depois de C2, pois não pode modificar o valor de E antes dele ser usado por C2. Simbolicamente escreve-se (C2 da C3);

3º) Dependência de saída: C5 deve ser executado depois de C3, caso contrário, C6 poderia usar o valor de E calculado por C3 e não por C5. Simbolicamente escreve-se (C3 ds C5).

Acrescenta-se também que, se F é o cabeçalho de um comando repetitivo e C, um componente pertencente ao escopo de F, existe uma "dependência em repetição" de F para C (simbolicamente, (F dr C)).

Quando os componentes estão em escopos de comandos repetitivos e apresentam variáveis indexadas cujos índices são expressões contendo as variáveis de controle desses comandos, é necessária uma observação mais cuidadosa para a identificação das dependências existentes. Seja o trecho de programa da figura 2.

```
F  :  FOR   I := 1  TO N DO
          BEGIN
C1 :      A :=  A + 1 ;
C2 :      Y :=  2 * A ;
C3 :      Z[I] := Y + V[I] ;
C4 :      A := X[I+1] + X[I-1]
C5 :      X[I] := W[I] + 1;
C6 :      W[I+1] := X[I] + 1;
          END;
```

Figura 2 - Programa com vários componentes no escopo de um comando de repetição.

A dependência (C5 dd C6) existe pois C5(I) calcula X[I] usado depois por C6(I); (C6 dd C5) existe pois C6(I) calcula W[I+1] usado depois por C5(I+1); (C4 da C5) existe pois C4(I-1) usa X[I], modificado depois por C5(I); (C2 ds C2) existe pois C2(I) calcula Y, que é recalculado depois por C2(I+1); etc...

"Grafo de dependências" de um programa é um grafo que expõe as dependências entre os seus componentes. Ele deve ser a nível de componentes e a nível atômico. A figura 3 mostra o grafo de dependências a nível de componentes do programa da figura 2 e a figura 4 o mostra a nível atômico para o programa da figura 1.

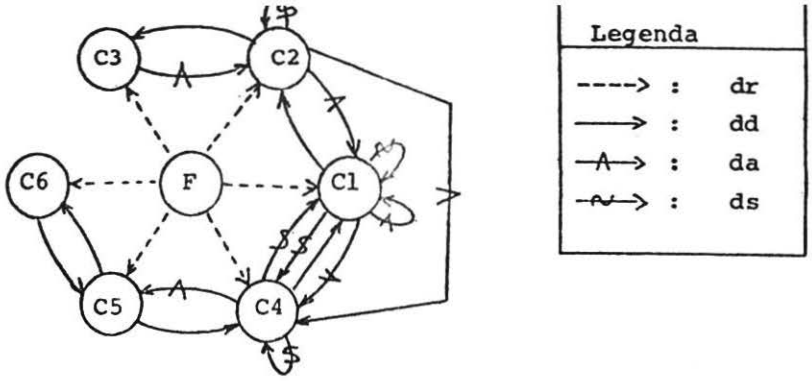

Figura 3 - Grafo de dependências a nível de componentes.

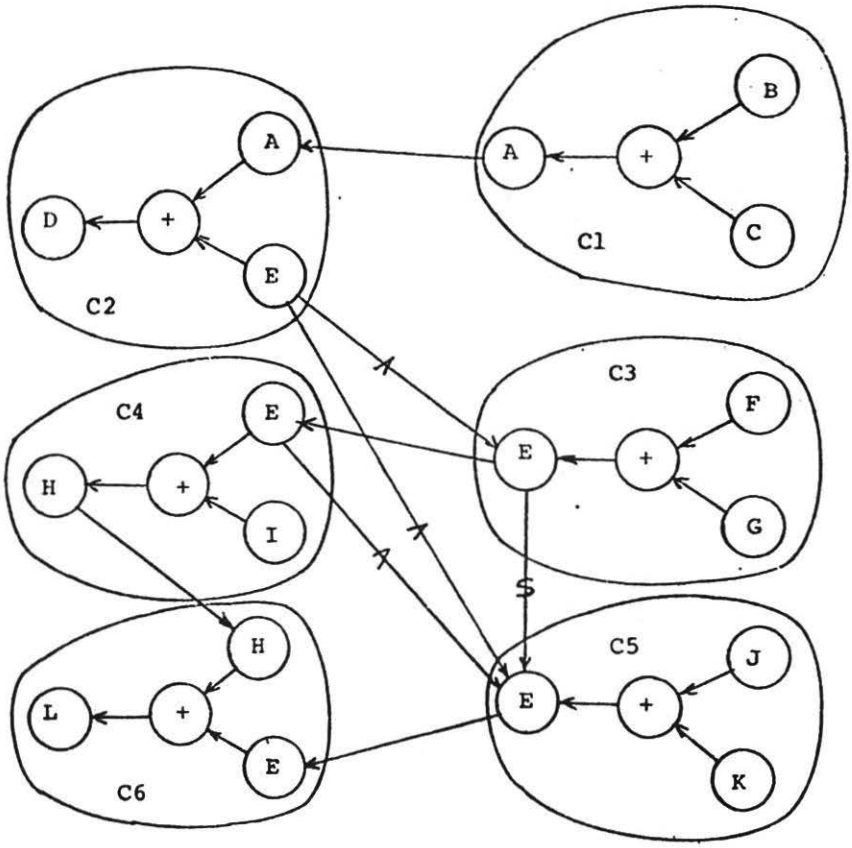

Figura 4 - Grafo de dependências a nível atômico.

Observa-se que o grafo de dependências a nível atômico deve conter todas as dependências do grafo a nível de componentes e também as dependências internas dos componentes.

REDUÇÃO DO NÚMERO DE DEPENDÊNCIAS DO GRAFO.

Ao se reduzir, por processos razoáveis, o número de dependências do grafo de um programa, aumenta-se seu potencial em paralelismo. Algumas técnicas existem para fazê-lo. Pode-se efetuar mudança no nome de algumas ocorrências de variáveis escalares dentro do programa e também expandí-las para variáveis indexadas, reduzindo assim o número de dependências do grafo. A figura 5 mostra o programa da figura 2 e seu grafo de dependências alterados pela devida aplicação dessas técnicas.

```
CØ :  A2[Ø]   :=  A2[-5Ø] ;
F  :  FOR  I:=1  TO N DO
          BEGIN
C1 :      A1[I-1]   := A2[I-1] + 1 ;
C2 :      Y[I-1]    := 2 * A1[I-1] ;
C3 :      Z[I]      := Y[I-1] + V[I];
C4 :      A2[I]     := X[I+1] + X[I-1];
C5 :      X[I]      := W[I] + 1 ;
C6 :      W[I+1]    := X[I] + 1 ;
          END;
```

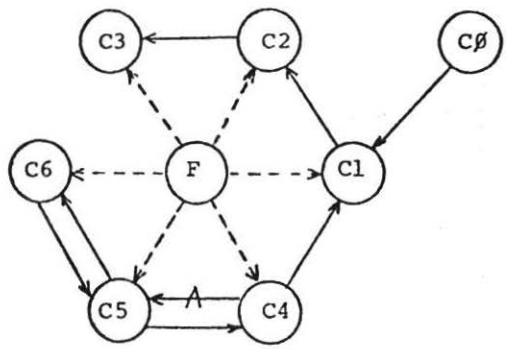

Figura 5 - Redução das dependências do grafo do programa da figura 2.

A mudança do nome e a expansão de variáveis indexadas ainda estão estudo.

DECOMPOSIÇÃO DOS COMANDOS REPETITIVOS.

O grafo da figura 5 apresenta alguns ciclos e alguns componentes fora de ciclos. Definimos "ciclo máximo", um ciclo que não seja subconjunto próprio de outro ciclo, e "ponto isolado", um nó do grafo não pertencente a nenhum ciclo. Definimos ainda "pi-bloco", um ciclo máximo ou o conjunto formado por um ponto isolado.

Para maior facilidade de deteção de paralelismo e para o início da confecção de um cronograma de execução do programa, pode-se decompor um programa em seus pi-blocos, obtendo-se o grafo dos pi-blocos, que é um cronograma preliminar da sua execução. A figura 6 mostra o programa da figura 5 decomposto em seus pi-blocos.

```
CØ :   A2[Ø] := A2[-5Ø];
       FOR  I1 :=  1  TO  N  DO
            BEGIN
C4 :        A2[I1] :=  X[I1+1] + X[I1-1] ;
C5 :        X[I1]  :=  W[I1] + 1;
C6 :        W[I1+1]:=  X[I1] + 1;
            END;
       FOR  I2 :=  1  TO  N  DO
C1 :        A1[I2-1] :=  A2[I2-1] + 1;
       FOR  I3 :=  1  TO  N  DO
C2 :        Y[I3-1]  :=  2 * A1[I3-1];
       FOR  I4 := 1   TO  N  DO
C3 :        Z[I4]  :=  Y[I4-1] +  V[I4];
```

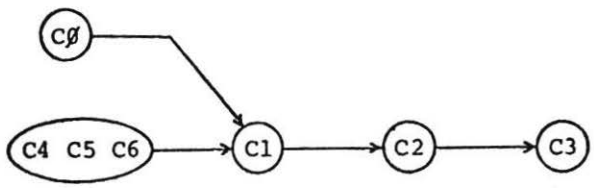

Figura 6 - Programa submetido à decomposição de comandos repetitivos e grafo de seus pi-blocos.

QUEBRA DE CICLOS.

A existência de ciclos nos grafos de dependências pode levar a sistemas de recorrência, que são entidades de difícil solução paralela. É sempre vantajoso quebrar ciclos dos grafos, quando possível, reduzindo-se o tamanho dos sistemas de recorrências e as dificuldades de seu tratamento. Podemos citar o caso em que o ciclo existe apenas no grafo a nível de componentes, desaparecendo no grafo a nível atômico. Este tipo de ciclo pode geralmente ser quebrado com a introdução de comandos auxiliares. Há também os casos de ciclos formados não só de dependências diretas, que às vezes são resolvidos com a partição do campo de variação das variáveis de controle dos comandos repetitivos envolvidos. No entanto, nem todos os ciclos de um grafo podem ser quebrados.

SUBSTITUIÇÃO DE VARIÁVEIS POR EXPRESSÕES.

A figura 7 mostra um trecho de programa e, ao lado, a substituição de variáveis por expressões correspondentes. No programa original, tem-se uma cadeia de comandos dependentes cujo tempo de execução é de 3 unidades. No programa transformado, tem-se 3 comandos independentes, e o tempo de execução é o tempo da maior expressão, que é de 2 unidades.

```
A := B + C;
D := A + E;        equivale a        A := B + C ;
F := D + G;                          D := B + C + E ;
                                     F := B + C + E + G;
```

Figura 7 - Ilustração da etapa de substituição de variáveis por expressões.

REPETIÇÕES SUCESSIVAS DAS DUAS ETAPAS ANTERIORES.

A necessidade da aplicação desta etapa existe para casos não freqüentes na prática. Acontece que, com a aplicação da substituição de variáveis por expressões, alguns ciclos do programa podem se tornar "quebráveis". O programa é novamente submetido à quebra de ciclos, apresentando novas possibilidades de aplicação da substituição de variáveis. Este processo pode ser repetido várias vezes, até que o

grafo de dependências se estabilize.

TRATAMENTO DOS COMANDOS REPETITIVOS DECOMPOSTOS.

O resultado das 7 etapas anteriores é um programa composto de comandos repetitivos cíclicos, acíclicos e comandos de atribuição fora do escopo de qualquer comando repetitivo. O tratamento dos comandos repetitivos acíclicos é trivial. Todas as suas iterações podem ser executadas em paralelo. Já o tratamento dos cíclicos varia de acordo com a complexidade de seus ciclos máximos. Dentre os métodos para seu tratamento podemos citar aqueles para resolução de recorrências e o método da "frente de onda".

As recorrências lineares são as mais simples. Por exemplo, seja o seguinte sistema:

$$x_i = C_i + \sum_{j=1}^{i-1} a_{ij} * x_j \qquad (1 \leq i \leq 4)$$

Nele estão implícitas as atribuições da figura 8 , cujo tempo de execução seqüencial é de 12 unidades, considerando igual os tempos de adição e multiplicação, e cujo tempo de execução paralela é de 6 unidades.

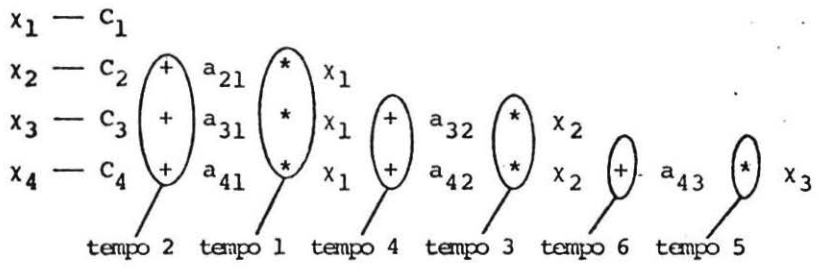

Figura 8 - Execução paralela de um sistema de recorrências lineares.

Para ilustrar o método da "frente de onda" consideremos o seguinte aninhamento de comandos repetitivos:

```
FOR  I := 1  TO    10    DO
     FOR   J := 1  TO    10    DO
          W[I,J] := W[I-1,J] + W[I,J-1] + W[I-2,J+1] + W[I+3,J-2] ;
```

As iterações desse aninhamento são ilustradas pelo espaço bi-dimen-

sional da figura 9.

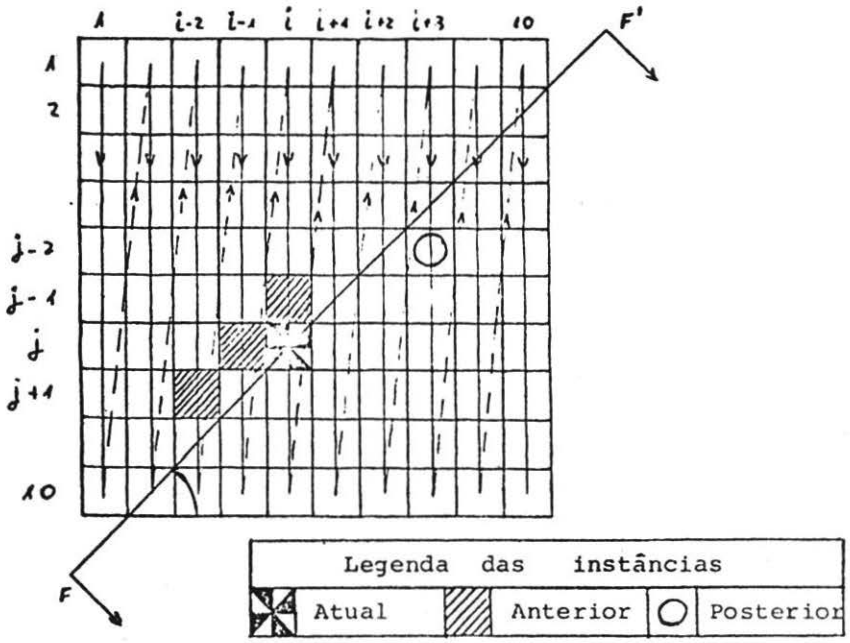

Figura 9 - Ilustração do método da "frente de onda"

A frente de onda FF' corta, num dado instante, vários elementos do array W, que nesse instante podem ser calculados simultaneamente. Este método exige critérios na escolha do ângulo da frente de onda e pode ser estendido para aninhamentos maiores. As referências Muraoka, 1971 e Kuck, 1972 apresentam um bom desenvolvimento teórico desse método.

TRATAMENTO DAS EXPRESSÕES.

A execução paralela de expressões pode ser orientada pelo algoritmo da árvore de tamanho mínimo, apresentado em Baer, 1969 e auxiliada pela propriedade da distributividade de operadores aritméticos. Para maior simplicidade, consideremos iguais os tempos de execução de uma adição e de uma multiplicação.

A figura 10 mostra uma expressão cuja execução seqüencial demanda unidades de tempo. Com a aplicação do algoritmo da árvore de tamanho mínimo, o tempo de execução é reduzido para 4 unidades

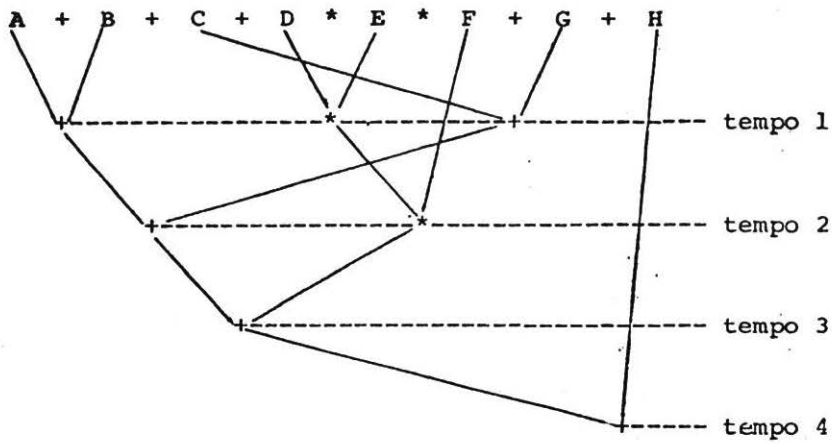

Figura 10 - Árvore de tamanho mínimo para o cálculo de uma expressão.

Algumas expressões têm seu tempo de execução reduzido quando são transformadas pela aplicação da propriedade de distributividade.Há casos, porém, em que esse tempo aumenta. Muraoka, 1971 e Kuck,1972 apresentam um algoritmo que verifica quando esta aplicação é útil para reduzir o tempo de execução de uma expressão.

GERAÇÃO DE CÓDIGO.

Esta etapa, que inclui a geração de uma estrutura intermediária para o programa, depende muito da arquitetura e do conjunto de instruções do multiprocessador para o qual o compilador é construído. No entanto, pode-se adiantar que, de alguma forma ela fará uso do grafo dos pi-blocos para a elaboração de um cronograma de execução paralela. Os detalhes da implementação desta etapa são deixados para pesquisas subseqüentes.

CONCLUSÕES.

O objetivo deste relatório é o de divulgar o trabalho por nós realizado a respeito do aproveitamento de programas seqüenciais nos "multiprocessadores do futuro", trabalho este minuciosamente descrito na referência Mokarzel, 1984. Podemos dizer que,do ponto de vista de execução de programas, o método proposto apresenta uma boa performance, apesar de terem sido feitas simulações sob hipóte

ses de um computador utópico.

O tempo de compilação dos programas testados foi muito grande em relação ao tempo de compilação para a execução seqüencial, apesar de que foi usado, nesta compilação, um computador com um só processador. É claro que o tempo de compilação de programas escritos em linguagens de paralelismo explícito deve ser bem menor que o tempo para as linguagens seqüenciais visando multiprocessamento, mesmo que esta compilação seja feita num avançado multiprocessador. Este método é então muito mais vantajoso para programas rotineiros que não precisam estar sendo compilados a todo o momento.

Para a continuidade de avanços neste campo de compiladores para multiprocessadores, são necessárias inúmeras pesquisas complementares, principalmente no que diz respeito a adaptação desta e de outras possíveis metodologias às arquiteturas reais. A referida referência apresenta várias sugestões para novas pesquisas. O assunto é vasto e há muito para ser descoberto e aperfeiçoado.

BIBLIOGRAFIA.

BAER, J.L.; BOVET, D.P. Compilation of aritmetic expressions for parallel computations. Information Processing 68, North-Holland Publishing Co., Amsterdam, 340-346, 1969.

BANERJEE, U. Speedup of Ordinary Programs. Ph.D. Thesis. Urbana, IL, University of Illinois, Oct. 1979.

CHEN, S.C.; KUCK, D.J. Time and parallel processor bounds for Linear recurrence systems. IEEE Transactions on Computers, 24 (7):701-717, July 1975.

KUCK, D.J. A survey of parallel machine organization and programming. Computing Surveys, 9(1):29-59, Mar. 1977.

KUCK, D.J.; KUHN, R.H.; PADUA, D.A.; LEASURE,B.; WOLFE, M. Dependence graphs and compiler optimizations. Publicação Reservada da ACM, 207-218, 1981.

KUCK, D.J.; MURAOKA, Y; CHEN, S.C. On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup. IEEE Transactions on Computers, 21(12):1293 1310, Dec. 1972.

MOKARZEL, F.C. Compilador de Programas Seqüenciais para Multiprocessamento: Análise e Metodologia para sua Implementação. Tese de Mestrado em Ciências. São José dos Campos, SP, ITA, Out. 1984.

MURAOKA, Y. Parallelism Exposure and Exploitation in Programs. Ph. D. Thesis. Urbana, IL, University of Illinois, 1971.

PADUA, D.A.; KUCK, D.J.; LAWRIE, D.H. High-speed multiprocessors and compilations techniques. IEEE Transactions on Computers 29(9):763-776, Sep. 1980.