VALIDAÇAO FUNCIONAL DE UM ESQUEMA PARA TRATAMENTO DE OPERAÇOES RECURSIVAS NUM GRAFO DE FLUXO DE DADOS

Francisco Enêas Lemos
SCOPUS Tecnologia S/A
FEI - Faculdade de Engenharia Industrial

Wilson Vicente Ruggiero
SCOPUS Tecnologia S/A
Escola Politécnica da USP

1. SUMARIO

Este artigo apresenta um programa concebido e implementado com a finalidade de realizar a simulaçao funcional de grafos de fluxo de dados. Estes grafos permitem descrever programas computacionais explicitando e explorando o paralelismo contido em seus algorítmos e também incorporar os conceitos de "controle por fluxo de dados". A primeira parte do artigo introduz os conceitos básicos relacionados com o controle por fluxo de dados, e grafos de fluxo de dados. A seguir, é apresentada a estrutura interna do programa simulador destes grafos, e suas formas de utilizaçao.

2. INTRODUÇAO

Computaçao dirigida por fluxo de dados tem sido objeto de estudo de vários grupos de pesquisadores [Dennis-73] [Plas et al-76] [Arvind et al-78] [Catto-81], como uma forma de encontrar soluçoes para realizar processamento em alta velocidade. Ela está baseada na idéia de que uma instruçao é executável assim que ela possui todos os operandos necessários disponíveis. A consequência desta regra básica é que a

dependência entre os dados deve ser dedutível dos programas. Esta dependência pode ser expressa, por exemplo, através de grafos orientados tornando o fluxo de dados um modelo natural de execução a ser considerado para explorar o paralelismo contido nos programas.

Nesse contexto, grafos de fluxo de dados [Dennis-73] se apresentam como uma forma adequada para representação gráfica de programas, e também como um método formal para especificar programas para máquinas de computação dirigidas pelo fluxo de dados (data-driven machines).

O programa descrito neste artigo permite a simulação funcional destes grafos, desde que descritos de acordo com o modelo gráfico definido em [Lemos-86]. Este programa foi denominado SIGRAFO, escrito em PASCAL, e implantado num microcomputador NEXUS-2600.

3. MODELO DE "FLUXO DE DADOS"

A compreensão das idéias contidas no "modelo de fluxo de dados" implica na discussão prévia dos dois mecanismos básicos de organização de programas: o "Mecanismo de Controle", segundo o qual uma instrução provoca (causa) a execução de outras instruções e o "Mecanismo de Passagem de Parâmetros", segundo o qual uma instrução obtém os parâmetros que ela necessita, ou transfere parâmetros às demais instruções.

O mecanismo de controle utilizado pelo modelo do fluxo de dados denomina-se "controle por disponibilidade", no qual a disponibilidade de operandos é quem determina o início da execução da operação que deve ser realizada sobre estes operandos. O mecanismo de passagem de parâmetros utilizado por este modelo é denominado "passagem por valor", ou seja: os dados são comunicados diretamente entre operadores, sem que existam células de memória intermediárias entre produtores e consumidores destes dados.

No modelo do fluxo de dados (ver figura 1), uma instrução é ativada quando ela receber marcas de dados em todos os seus arcos de entrada. Na figura 1, os pontos de interrogação (?) nas instruções indicam que a marca é necessária como parâmetro de entrada para a intrução correspondente, e os fluxos de dados e controle são explicitados através dos arcos que interligam estas instruções.

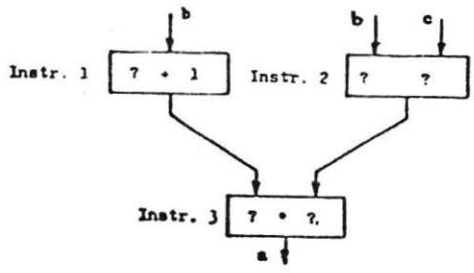

FIGURA 1: Modelo do fluxo de dados para a expressão
a = (b + 1) * (b - c)

Uma instrução, ao ser ativada, consome um conjunto de marcas de seus arcos de entrada, realiza a operação correspondente e produz um novo conjunto de marcas que é colocado em seus arcos de saída. Quando uma intrução produz um resultado, ele é transferido diretamente para a instrução que deve consumí-lo. Se existir mais de um consumidor para o ítem de dado produzido, ele será copiado tantas vezes quantas forem necessárias, o que caracteriza o mecanismo de passagem de parâmetros "por valor" mencionado anteriormente.

Programas computacionais dirigidos por dado, podem ser descritos através de grafos orientados, nos quais os NOS representam sub-componentes do programa (processos assíncronos ou instruções), e os arcos orientados representam o fluxo dos dados entre os NOS. O controle de execução desses programas pode ser caracterizado por dois pontos básicos:

1) Uma instrução é executada e produz um resultado quando, e somente quando, todos os operandos necessários á sua execução estão disponíveis. Portanto, se duas ou mais instruções tem seus operandos disponíveis simultaneamente, elas podem ser executadas de forma concorrente;

2) Uma instrução, qualquer que seja o seu nível, é puramente funcional, ou seja: os suportes de seus operandos são disjuntos dos suportes dos operandos de todas as demais instruções do programa.

4. GRAFOS DE FLUXO DE DADOS

São grafos bipartites F.D. = (N,A), capazes de representar algorítmos computacionais cuja execução é dirigida pelo fluxo de dados ! Lemos-86 !, onde:

- N é um conjunto finito de NÓS (ou operadores) que realizam funções primitivas (indivisíveis a nível de operadores), capazes de efetuar a computação necessária. UM NÓ primitivo pode possuir uma ou duas entradas (onde ele recebe seus argumentos), e uma ou duas saídas (onde ele produz seus resultados).

- A é um conjunto finito de arcos orientados capazes de veicular informações entre os NÓS, permitindo que resultados provenientes de um NÓ se tornem disponíveis como argumentos para outros NÓs.

A execução de um grafo de fluxo de dados é descrita como sendo uma sequência de ativações dos NÓS que o compõem. Ativar um NÓ qualquer do grafo significa aplicar a função primitiva realizada por este NÓ ao conjunto de argumentos por ele recebidos.

Para qualquer NÓ do grafo, as ativações se processam de acordo com as seguintes regras:

1. Um NÓ é dito habilitado se, e somente se, ele possuir em suas entradas todas as marcas necessárias para realizar a operação que lhe cabe, ou seja: possuir todos os argumentos necessários á execução de sua operação nas posições corretas das entradas.

2. Os NÓS habilitados são ativados.

3. Ativar um NÓ significa consumir as marcas existentes em suas entradas, utilizar os valores associados a estas marcas para determinar resultados correspondentes á função aplicada pelo NÓ, e associar estes resultados a novas marcas que são colocadas nos arcos de saída do NÓ.

O exemplo da figura 2 indica as fases de execução de um grafo elementar que realiza a operação z = (X + Y) * (X - Y).

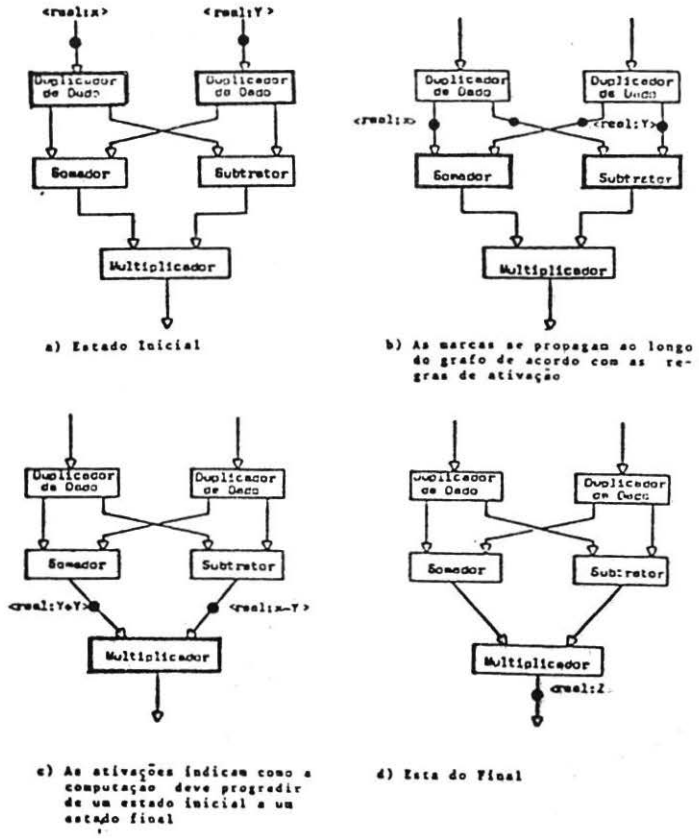

FIGURA 2: Execução de um grafo de fluxo de dados

5. CONSTRUÇÕES CONDICIONAIS

Nós primitivos, da forma como foram apresentados até aqui, não são suficientes para exprimir operações condicionais, onde a computação depende de algum predicado. São expressões do tipo:

IF ⟨condição B⟩ THEN ⟨Expressão V⟩ ELSE ⟨Expressão F⟩

Se a condição B é verdadeira, a marca será argumento para a expressão V. Se a condição B é falsa a marca será argumento para a expressão F.

Para obter este tipo de representação, pode-se incorporar novos NÓS ao modelo, denominados respectivamente: UNIÃO, DISTRIBUIDOR e TESTE.

Construções condicionais são obtidas, a partir da utilização destes NÓS, configurando-os conforme indicado na figura 3. Nesta figura, um argumento genérico X é selecionado entre os NÓs que aplicam as expressões (expressão V) ou (expressão F) de acordo com o valor do teste realizado com os argumentos X e Y.

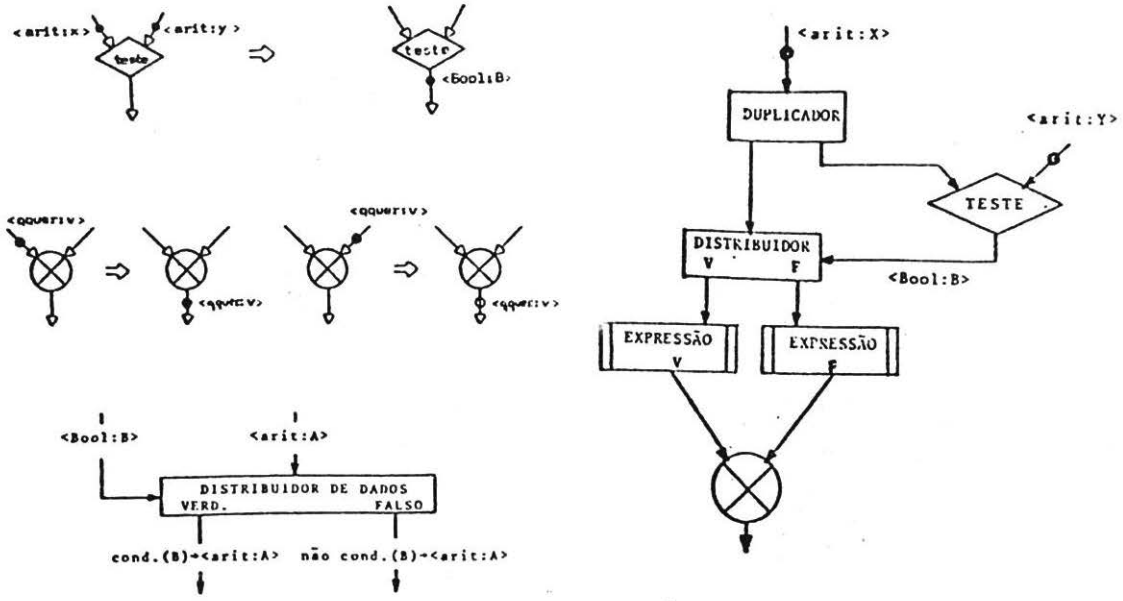

FIGURA 3: Grafo de fluxo de dados para computações condicionais

6. CONSTRUÇÕES ITERATIVAS E RECURSIVAS

O problema da descrição de operações iterativas ou recursivas explorando paralelismo, pode ser solucionado permitindo que os arcos transportem mais de um argumento durante a execução, o que corresponde a tornar os NÓS reentrantes. A reentrância não cria problemas para NÓS de uma única entrada, desde que estes NÓS possam consumir seus argumentos á medida em que eles são produzidos, mas certamente apresenta problemas em NÓS de múltiplas entradas. Neste caso, não seria mais possível declarar um NÓ ativável pela simples presença de marcas (argumentos) em suas entradas, uma vez que estas marcas poderiam pertencer a partes totalmente distintas do processo de computação.

Existem pelo menos três possíveis abordagens para solucionar este tipo de problema:

A. Proibir o uso de NOS reentrantes, fazendo com que estes NOS sejam multiplicados tantas vezes quantas forem necessárias para garant r a unicidade das marcas nos arcos. Numa operação iterativa, por exemplo, cada passo de iteração deveria ser descrito por um grafo separado dos demais ¦ Miranker-77 ¦.

B. Impor regras restritivas á ativação dos NOS, de modo a forçar uma separação física de objetos não relacionados. Por exemplo, permitir que um NO seja ativado quando ele possuir, não apenas as marcas necessárias em suas entradas (argumentos), mas também nenhuma marca (resultados) ainda não consumida em suas saídas ¦ Dennis-73 ¦. Solução semelhante pode ser obtida sincronizando as operações reentrantes, liberando o passo seguinte somente quando o passo anterior fosse completado ¦ Kamram-81 ¦.

C. Uma terceira solução, permite obter uma separação lógica de objetos não relacionados, e consiste em considerar que as marcas transportam, além das informações de dados ou controle já mencionadas, também uma "COR" (denominada nome de ativação), que permite distinguir as marcas não relacionadas. Neste caso, um NO é ativável somente quando ele possuir um conjunto completo de marcas identicamente rotuladas necessárias á sua ativação ¦ Catto-81 ¦ ¦ Arvind-Gostlow-82¦, ¦ Lemos-86 ¦.

A abordagem adotada para a solução do problema será a terceira (item C), considerando que no modelo de fluxo de dados, as marcas são usadas para transferir resultados parciais diretamente dos produtores para os consumidores destes dados. Quando um NO é executado, seus resultados são transportados por marcas contendo referências aos NOS que irão consumí-los, e suas posições exatas no conjunto de argumentos desses NOS. Estas referências devem identificar de forma unívoca, cada marca gerada pelo programa em cada instante. Nos casos em que mais de uma cópia de um NO do grafo possa estar sendo executado de forma concorrente, será associado a cada ativação de um determinado grafo, um nome da ativação único, que acrescido ás informações já transportadas pelas MARCAS, identificará um contexto único ao qual a marca pertence.

A nível do modelo ¦ Lemos-86 ¦, uma marca no grafo será representada como se segue:

⟨T:V⟩na:Ci DST:Dj

onde cada campo possui o seguinte significado:

(1) T e V são respectivamente o tipo e o valor do argumento transportado pela marca.

(2) O nome de ativação (Ci) permite distinguir ativações distintas de um mesmo NÓ, que podem ser executadas de forma concorrente.

(3) O campo de destinação (Dj) identifica o NÓ e o arco do grafo de fluxo de dados que irá consumir a marca produzida numa determinada ativação deste grafo.

Neste ponto é interessante generalizar o conceito de procedimento utilizado em linguagens convencionais, estendendo-o para um ambiente dirigido pelo fluxo de dados.

Um programa dirigido pelo fluxo de dados (isento de marcas) pode ser considerado como sendo um conjunto de subprogramas (Módulos) convenientemente formados, denominados procedimentos. Cada nível de execução de um procedimento será denominado ATIVAÇÃO, à qual corresponderá univocamente um NOME DE ATIVAÇÃO conforme definido. Procedimentos podem ser ativados para execução e estarão ativos num determinado contexto, se e somente se, pelo menos um de seus NÓS neste contexto encontra-se ativado. Num programa dirigido por dados estruturados por procedimentos, podem ocorrer várias ativações simultâneas destes procedimentos, que devem ser executadas de forma concorrente, limitadas apenas pela dependência entre os dados, e pelo hardware disponível em cada implementação específica.

O modelo considera que cada primitivo é ativado e passa seus argumentos à medida em que estes argumentos vão sendo recebidos em suas entradas. Neste caso, uma cópia do procedimento é ativada assim que o primeiro argumento é recebido, e a passagem de parâmetros se faz pela absorção das marcas nos seus arcos de entrada, e a colocação respectiva de uma cópia destas marcas nos arcos de entrada do procedimento correspondente. O retorno dos resultados para o procedimento que chamou também é feito a medida em que eles ficam disponíveis.

A figura 4 ilustra a utilização destes primitivos, indicando o ciclo de ativação, passagem de parâmetros e encerramento de um procedimento P2 ativado durante a execução de um procedimento P1. Existem basicamente cinco nós primitivos envolvidos na manipulação das ativações em diferentes contextos:

(A) Como a ativação de um novo procedimento Ni implica numa mudança de contexto, existirá um NÓ GNA (Gerador de Nomes de Ativação), que ao receber o identificador do procedimento, obtém um novo (nome de ativação), e inicia o processo de passagem de parâmetros.

(B) O primitivo "INCR dst" permite estabelecer uma correspondência bi-unívoca entre as m entradas do

procedimento invocador e do procedimento invocado. Para tanto, este NÓ incrementa o campo de destinaçao da marca, mantendo inalterados todos os demais campos.

(C) Um Nó denominado IPR (Identificador de Procedimento de Retorno) permite, ao procedimento invocador, identificar para o procedimento invocado seu ⟨ nome de ativaçao, e a destinaçao ⟩ a partir da qual devem ser retornados os resultados (correspondência bi-unívoca também com relaçao às saídas).

(D) Um outro elemento primitivo denominado RP (Rotulador de Parâmetros) terá como funçao executar uma operaçao de mudança de ⟨ nome de ativaçao ⟩ de uma parâmetro qualquer, permitindo com isto a comunicaçao entre processos para passagem de parâmetros ou para retorno de resultados.

(E) Quando a execuçao completa for encerrada, o ⟨ nome de ativaçao ⟩ deve ser retornado ao sistema para uma posterior utilizaçao. Com esta finalidade foi introduzido o NÓ primitivo LNA (Liberador de Nomes de Ativaçao).

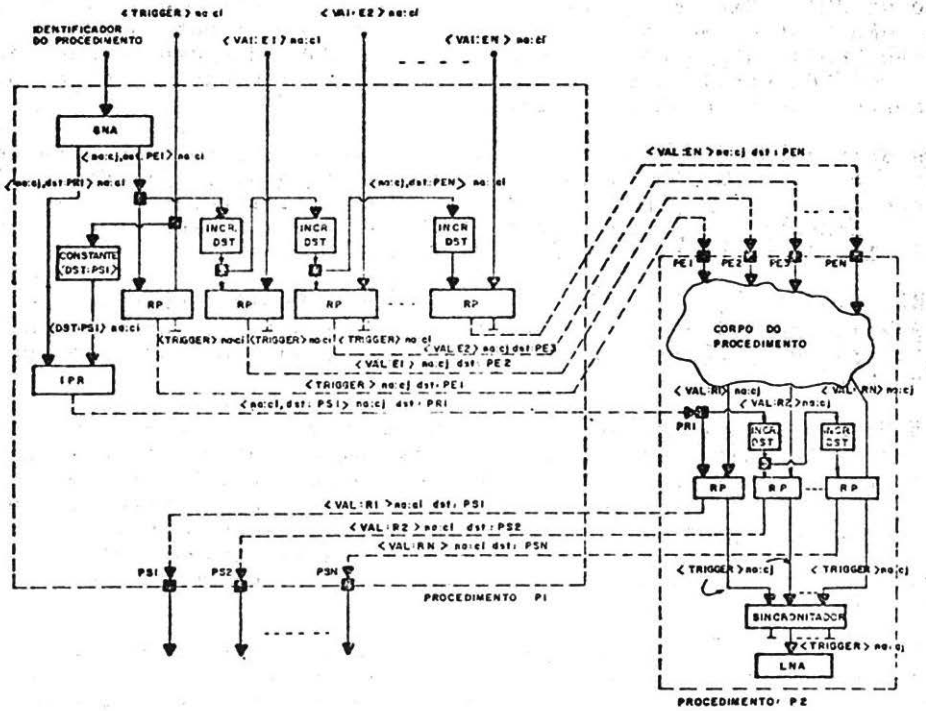

FIGURA 4: Nós primitivos para manipulaçao de ativaçoes

7. EXEMPLO DE CONSTRUÇAO RECURSIVA

O exemplo analisado correspondente ao cálculo da funçao FFAT (X,Y), descrita a seguir:

Funçao Fatorial:= FFAT (N,1)

onde:

```
FFAT (X,Y):=  IF   X>1  THEN FFAT (X -1,X*Y)
                        ELSE Y;
```

As figuras 5 e 6 que se seguem, detalham os grafos de fluxo de dados correspondentes à execuçao deste algoritmo.

O NÓ-GNA recebe como operandos um identificador da funçao (FFAT), que consiste na destinaçao (indicada por PE1) do NÓ da funçao que recebe o primeiro parâmetro de entrada, e também da destinaçao (indicada por PR1) do NÓ que recebe a informaçao para retornó do resultado.

O NÓ-GNA, uma vez ativado, produz o novo "nome de ativaçao", com uma destinaçao inicial PE1 (fornecida pelo identificador da funçao FFAT), que será incrementada pelos NÓS INCR.dst, e passada aos NÓS-RP (rotuladores de parâmetros), permitindo rotular os parâmetros de entrada da funçao.

A destinaçao do resultado, indicada na figura 6 por PS1 (arco pontilhado que sai da funçao FFAT e vai para o NÓ Duplicador), é fornecida ao NÓ-IPR através de uma constante, e será posteriormente utilizadas pelo NÓ-RP da figura 6 para devolver o resultado ao contexto invocador da funçao.

Na figura 6, um macro-nó denominado EXECUTA (indicado pelo retângulo pontilhado) é idêntico ao da figura 6, exceto pela destinaçao do parâmetro de saída da funçao, que foi indicado como sendo PS2.

O NÓ-RP (Figura 6) recebe como parâmetros, além do resultado propriamente dito (seu valor), também o nome de ativaçao (indicado no exemplo por Cj) e sua destinaçao (dst:PS1). Num caso mais geral, cada resultado possui sua própria instruçao de retorno (ver figura 5), e portanto, o procedimento chamado pode possuir um número variável de NÓS-RP que, quando ativados, produzem cada um deles, a informaçao de retorno correspondente a cada resultado.

Nas figuras 5 e 6, os arcos de saída dos primitivos IPR e RP foram desenhados em pontilhado, para indicar que sao arcos dinâmicos, ou seja: definidos durante o processo de execuçao do grafo.

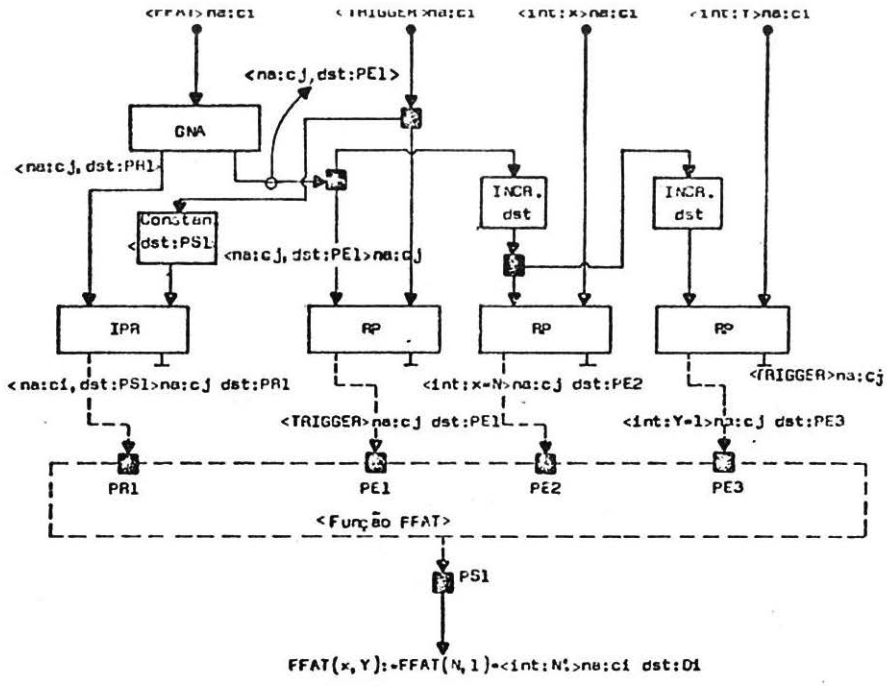

FIGURA 5: Operador EXECUTA aplicado à função FFAT (X,Y)

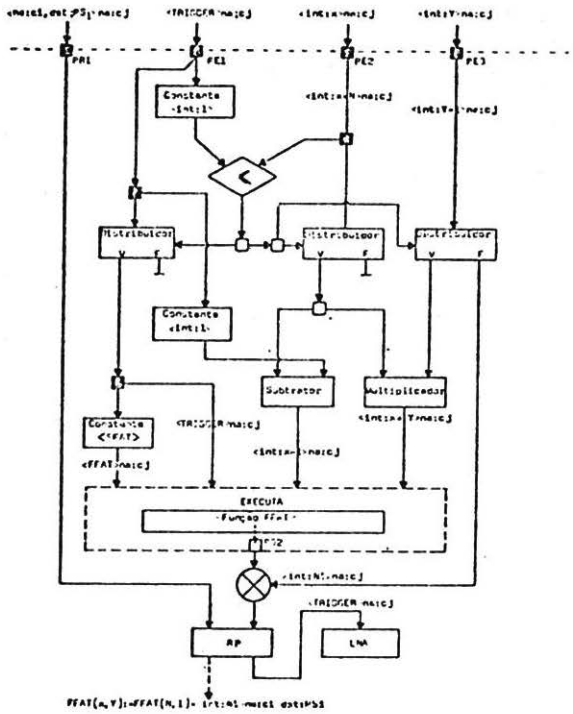

FIGURA 6:

Além de retornar os resultados, o NÓ-RP também envia uma informação de Trigger ao NÓ-LNA (figura 6), para que este NÓ possa liberar a utilização do nome de ativação para uma posterior utilização pelo sistema. Ainda na figura 5, cabe ressaltar que existe uma informação de Trigger que é enviada do contexto que invocou a função Fatorial, e se propaga através de um NÓ-RP, permitindo a ativação de eventuais NÓS internos ao corpo do procedimento invocado, que de outra forma não poderiam ser ativados (como é o caso das constantes).

8. ESTRUTURA BASICA DO PROGRAMA SIMULADOR (SIGRAFO)

A estrutura básica deste programa está centrada na manipulação de três "arrays" denominados:

- "array GRAFO"
- "array ATIVO"
- "array LISTA"

conforme indicado na figura 7.

O "array GRAFO" armazena as informações relativas à estrutura estática do grafo cuja execução será simulada, ou seja: à forma pela qual os NÓS estão interligados através dos arcos. Também armazena informações sobre o tipo do NÓ e seu tempo de execução.

O "array GRAFO" possui um ponteiro denominado LISTA DE MARCAS, que permite interligá-lo com o "array LISTA" (ver figura 4) onde ficam armazenadas as marcas colocadas nos diversos arcos do grafo. Este "array" é uma lista ligada que permite tornar os NÓS reentrantes, distinguindo as marcas que se acumulam em seus arcos de entrada através de um campo de COR. Quando uma marca é colocada num arco de entrada de um NÓ genérico de dois operandos, o programa realiza uma varredura da lista de marcas deste NÓ, verificando se o par da marca produzida se encontra disponível. Caso não se encontre, a nova marca é armazenada na lista, caso contrário, a dupla de marcas é transferida para o "array ATIVO".

O "array ATIVO" é uma lista de eventos ordenados em função dos tempos de execução atribuídos aos diversos NÓS que compõem o grafo sob simulação. A variável TEMPO ("record deste array") é atualizada somando-se o tempo decorrido da simulação até o instante considerado, com o tempo gasto para execução da operação correspondente ao NÓ (previamente fixado pelo usuário). O "array

ATIVO" também armazena todas as informaçoes necessárias à execuçao dos NOS ATIVOS, ou seja:

- valor do operando esquerdo (Marca-E)
- valor do operando direito (Marca-D)
- Tempo estabelecido para ocorrência do disparo do NÓ (Tempo)
- Nome de ativaçao dos operandos (Cor)
- Indice do NÓ (descrito a seguir)

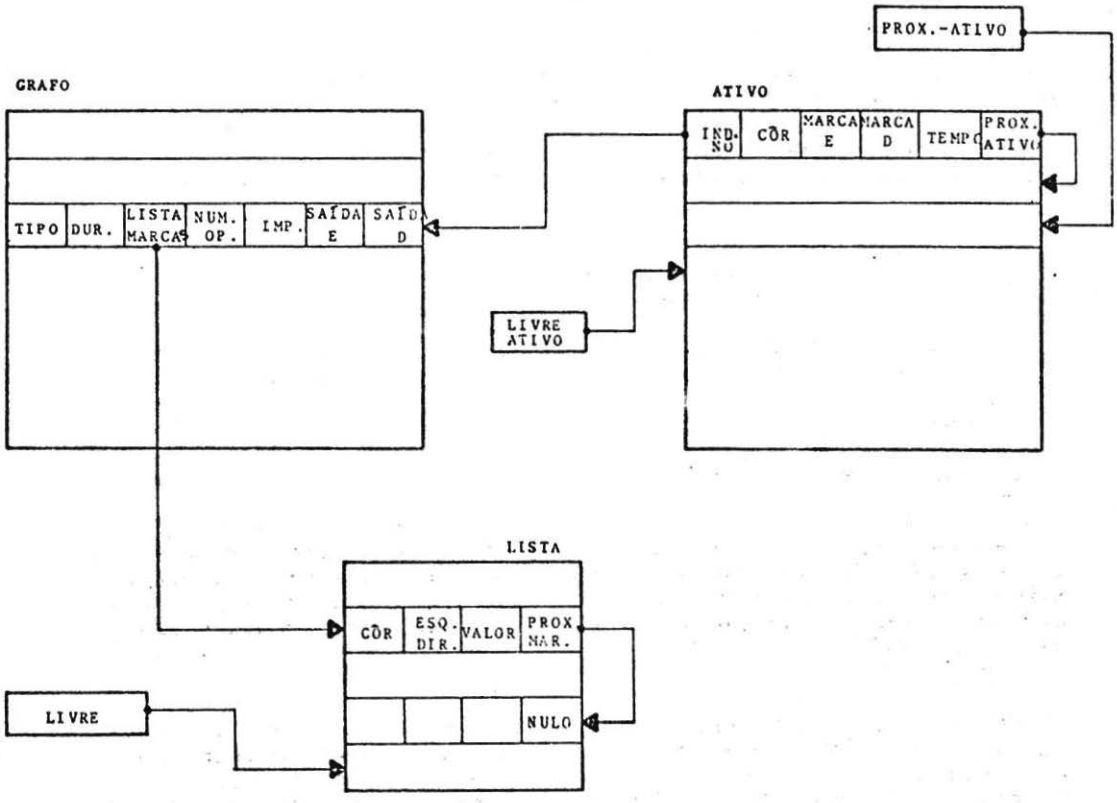

FIGURA 7: Interligaçao entre as estruturas GRAFO, ATIVO e LISTA.

O Indice do NÓ é um ponteiro que liga o "array ATIVO" ao "array GRAFO", de forma a permitir que um evento ativo, possa complementar as informaçoes que ele necessita para sua execuçao, ou seja: O NÓ cujos operandos necessários já foram recebidos tem acesso ao "array GRAFO" para buscar o tipo de operaçao que ele deve aplicar aos seus operandos, e a destinaçao dos resultados de sua execuçao (saída E, saída D).

O tempo de execução associado a cada um dos NOS primitivos do modelo pode ser alterado pelo usuário do programa. Para isto foi definido um "array" denominado OPERADOR onde estão especificados (em unidades de tempo) o tempo de duração associado a execução de cada operador, e o número de operandos que ele necessita. Com este tipo de abordagem, o usuário pode simular a execução de algoritmos sob diferentes condições de implementação física (tecnologias distintas de fabricação de componentes) obtendo conclusões preliminares de desempenho para uma posterior realização.

As cores (nomes de ativação) associadas às marcas são atribuídas pelo programa durante o processo de execução dos grafos, gerenciados pelos NOS primitivos apresentados no item 6. Para isto, o programa define um "array" booleano denominado "VET-COR" (contendo 1000 elementos), e consulta este "array" todas as vezes que uma cor é alocada ou liberada durante a execução.

9. MODOS DE EXECUÇÃO PERMITIDOS

O programa "SIGRAFO" permite dois modos de execução distintos:

1. Modo contínuo
2. Modo passo a passo

No modo contínuo, o grafo de fluxo de dados sob simulação é executado até o seu final, que pode ocorrer quando terminam os NOS ativos existentes no "array ATIVO", ou quando o tempo máximo de simulação, previamente estabelecido pelo usuário, é ultrapassado.

No segundo modo de execução, o "SIGRAFO" permite que o usuário processe seu algoritmo passo a passo, onde cada passo corresponde a executar todos os NOS ativos que podem ser executados em paralelo neste instante. Este modo de operação foi implantado com o objetivo de auxiliar a depuração de programas escritos com base na utilização de grafos de fluxo de dados.

Antes de iniciar a simulação propriamente dita, o programa solicita ao usuário que informe o modo de execução pretendido, quando então ele escolhe um dos dois modos permitidos.

10. CONCLUSÃO

O modelo gráfico apresentado neste artigo é parte de um estudo mais abrangente, que inclui a proposição de uma arquitetura de propósito geral dirigida por fluxo de dados, capaz de realizar sua implementação. Tanto o modelo quanto a arquitetura que o implementa foram detalhados em [Lemos-86], e permitiram concluir que a descrição de programas segundo os conceitos de controle por fluxo de dados é vantajosa na manipulação de estruturas iterativas e recursivas. Isto decorre da incorporação ao modelo de primitivos que gerenciam operações em diferentes contextos, tornando os nós reentrantes, e permitindo executar estas operações explorando alto grau de paralelismo.

Por outro lado, a realização do programa simulador permitiu analisar e validar este modelo, verificando o comportamento de todos os seus primitivos e, em particular, aqueles que tratam estruturas iterativas e recursivas. Além disso, o simulador realizado possui uma estrutura interna suficientemente flexível para permitir uma incorporação posterior de extensões e complementações do modelo, visando seu aproveitamento num processo de continuidade deste trabalho. Este simulador contribui para auxiliar o desenvolvimento de ferramentas de software para escrita e depuração de programas dirigidos por fluxo de dados, pois permite analisar o comportamento dinâmico e estimar tempos de execução de algoritmos que exploram alto grau de paralelismo.

11.REFERENCIAS BIBLIOGRAFICAS

01 - ARVIND, GOSTELOW, K. P., PLOUFFE, W. - "An Assynchronous Programming Language and Computing Machine" - Department of Information and Computer Science - University of California - Irvine - December, 1978.

02 - ARVIND, GOSTELOW, K. P. - "The U - Interpreter" - Computer - Vol. 15 - Number 2 - Pág. 42-50 - February, 1982.

03 - CATTO, A. J. - "Nondeterministic Programming in a Data Flow Environment", Ph. D. Thesis, Department of Computer Science, University of Manchester, June, 1981.

04 - DENNIS, J. B. - "First Version of a Data Flow Procedure Language" - Computation Structure - Group Memo # 93, November, 1973.

05 - KAMRAN, M. - "Contribution à La Compilation du Langage de Programmation d'une Machine Multiprocesseur à Controle dirige par les donnes" - These de Doctorat de 3eme. Cycle - Université Pierre et Marie Curie (Paris VI) - Mars, 1981.

06 - LEMOS, F. E. - "Arquitetura de Computadores Dirigidos pelo Fluxo de Dados" - Tese de Doutorado apresentada à Escola Politécnica da USP - Janeiro, 1986.

07 - MIRANKER, G. S. - "Implementation of Procedures on a Class of Data Flow Processors" - Proceedings of International Conference on Parallel Processing - 1977.

08 - PLAS, A., COMPTE, D., GELLY, O., SYRE, J. C. - "LAU System Architecture: A Parallel Data Driven Processor Based on Single Assignment" - Proceedings of the International Conference on Parallel Processing - 1976.