LINGUAGENS FORTRAN E C PARA PROCESSAMENTO CONCORRENTE/PARALELO

Calicchio S., Rodrigues M.
CPqD - Telebrás - Rodovia Campinas/Mogi Mirim Km 118,5
Caixa Postal: 1179
13085 - Campinas - SP - Brasil

Monticelli A., Garcia A.
Universidade Estadual de Campinas - UNICAMP
Faculdade de Engenharia Elétrica
13081 - Campinas - SP - Brasil.

Sumário

Este artigo apresenta as extensões para as linguagens Fortran e C para suporte a programação concorrente e paralela. Os mecanismos de concorrência e de suporte ao processamento paralelo utilizados para definição das extensões são os mecanismos implementados no sistema operacional PP-SO/P (SOP) [1]. As linguagens estendidas poderão então expressar explicitamente o paralelismo de um algoritmo e a concorrência entre processos. A comunicação entre processos (processadores) adota um modelo baseado em troca de mensagens proporcionando a sua utilização mesmo em hardware que possui o esquema de memória partilhada.

Abstract

This paper presents extensions to the languages Fortran and C aimed at supporting parallel and concurrent programming. The operating system PP-SO/P (or SOP, for short) [1], offers the adequate support for concurrent/paralell processing. Through the extended language the programmer will be able to explicitly express parallel algorithms and concurrent processes. Communication among processes (processors) is based on message passing model - regardless of the actual hardware, which may be a machine with shared memory, for example.

1 INTRODUÇÃO

Neste trabalho tratamos de linguagem de alto nível e sistemas operacionais para multicomputadores e multiprocessadores. As aplicações visadas cobrem um amplo espectro que inclui programação concorrente e programação paralela. O hardware de interesse é o Processador Preferencial PP [2] desenvolvido pelo CPqD da Telebrás, além de arquiteturas dele derivadas como é o caso das máquinas paralelas descritas em [9] e [7] - isto não significa, entretanto, que as idéias e técnicas aqui expostas não possam ser utilizadas em outros sistemas. Assim sendo, apesar deste artigo fazer parte de um conjunto de quatro artigos que relatam trabalhos relacionados [9], [10], [11], seu escopo é um pouco mais amplo que os demais, no sentido de que as técnicas aqui discutidas não se limitam ao sistema de processamento paralelo introduzido em [11].

O sistema operacional que é a base de todo o trabalho aqui relatado é o SOP, descrito em [1], que é um sistema operacional que comporta processamento distribuído, processamento em tempo-real e sistema multiusuário. Além disso o SOP oferece mecanismo de manipulação de processos e de troca de mensagens que, como veremos mais adiante, podem ser utilizados em sistemas de processamento paralelo, como é o caso, por exemplo, de aplicações que envolvem a utilização maciça de cálculo numérico, e nas quais um problema de grande porte é decomposto em partes menores que por sua vez são mapeadas em uma arquitetura hardware paralela.

As linguagens nas quais estamos interessados são Fortran e C. Mais especificamente, mostraremos como essas linguagens podem ser estendidas para poderem expressar explicitamente o paralelismo do algoritmo (problemas de decomposição matemática, por exemplo) e a concorrência de processos (programação em tempo real, por exemplo). Quando falamos em representação explícita de paralelismo estamos pensando em técnicas que permitam ao programador expressar o paralelismo de seu algoritmo através dos recursos de linguagens Fortran e C (estendidas), em contraposição a técnicas implícitas de extração de paralelismo que partem de um programa sequencial (que executa em uma máquina convencional) e, em nível de compilação, reescrevem o programa em forma paralela [3].

O trabalho está organizado em quatro seções, além desta introdução e de uma seção de conclusões. Na seção 2 nós resumimos alguns conceitos de software e hardware em sistemas de computação paralela e concorrente. Na seção 3 nós destacamos as características mais importantes do sistema operacional SOP no que se refere a programação em tempo real e programação paralela – nessa seção nós introduzimos, por exemplo, os mecanismo que o SOP oferece para a criação de processos para a troca de mensagens entre eles. Na seção 4 nós discutimos as extensões que estamos introduzindo nas linguagens Fortran e C visando dotar essas linguagens de mecanismos que permitam a sua utilização em programação em tempo real e programação paralela. Finalmente na seção 5 nós discutimos alguns tópicos pertinentes que não estamos abordando nesta fase do projeto, mas que deverão ser objetos de trabalhos futuros.

2 MODELOS DE PARALELISMO E CONCORRÊNCIA

Nesta seção discutiremos alguns modelos básicos relativos a sistemas computacionais (sistemas distribuídos ou computadores paralelos) formados por um número relativamente grande de processadores – digamos, dezenas ou centenas de processadores. A idéia aqui não é fazer uma apresentação completa sobre o assunto, mas tão somente introduzir a terminologia que deverá ser adotada no artigo. Em particular pretendemos introduzir os modelos de programação paralela e concorrente nos quais estamos interessados, bem como discutir as suas relações com as arquiteturas hardware visadas na fase atual do projeto [9], [10], [11].

De uma maneira informal poderíamos dizer que multicomputadores são sistemas computacionais distribuídos do tipo dos utilizados, por exemplo, em controles em tempo real ou em redes de estações CAD / CAM, com vários computadores independentes que se comunicam através de uma rede. Na mesma linha informal, multiprocessadores seriam computadores paralelos, nos quais os processadores estariam reunidos para formar uma única máquina – digamos, agrupadas em um mesmo bastidor. Apesar do atrativo da simplicidade, essas definições informais nem mesmo funcionam como esperamos. Em particular, para definir com clareza o modelo computacional (software) que utilizaremos, bem como as relações com as máquinas utilizadas, precisaremos de algo um pouco mais elaborado.

Uma maneira um pouco mais formal de se classificar multicomputadores e multiprocessadores utiliza como divisor de águas o nível em que se dá a interação entre membros do sistema computacional [5]. Por nível de interação entenderemos a maneira como se dá a partilha de dados entre os membros do sistema computacional: por acesso direto à memória principal (em nível de palavra) ou através de mecanismo de nível mais elevado como, por exemplo, arquivos completos – no primeiro caso um processador teria acesso à qualquer módulo de memória do sistema, enquanto que no segundo, de nível mais elevado, o acesso se limita a um arquivo ou bloco de dados. De acordo com esta ótica o termo memória partilhada se refere a sistemas computacionais que utilizam o nível de interação mais baixo.

As duas arquiteturas básicas estão ilustradas nas Figuras 1 e 2. No sistema multiprocessador os processadores partilham a memória principal Fig. 1, enquanto que em um multicomputador cada processador tem sua própria memória local Fig. 2 – neste esquema um processador não tem acesso direto a memória de um outro processador, o que não significa que eles não possam partilhar dados. No multiprocessador Fig. 1 a interligação pode ser, por exemplo, uma rede de comutação com vários estágios [5], enquanto que para o multicomputador Fig. 2 a topologia de hipercubo poderia ser dada como exemplo de interligação. Barramentos comuns (simples ou múltiplos) podem ser usados como interligação em ambas arquiteturas.

Por outro lado, os modelos de computação paralela adotados (no sentido software), podem ser por memória partilhada ou por troca de mensagens. O modelo que utiliza troca de mensagens se casa naturalmente com sistemas multicomputadores, da mesmo forma que os modelos de memória partilhada têm afinidade com os multiprocessadores. Estas afinidades entretanto não são excludentes: assim, poderíamos, com um custo ex-

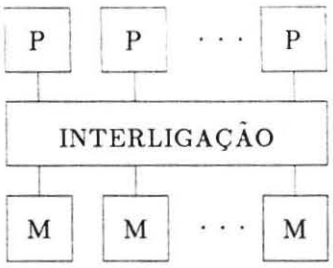


Figura 1: *Multiprocessador*

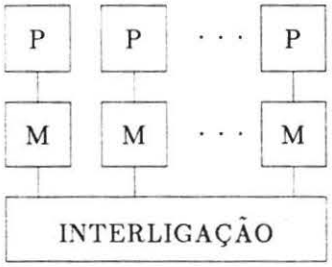


Figura 2: *Multicomputador*

tra de comunicações, implementar um modelo de computação por memória comum em um multicomputador, da mesma forma que podemos utilizar a memória comum existente em um hardware multiprocessador para implementarmos um esquema de troca de mensagens. Estas distinções e nuances serão importantes na escolha do modelo de computação paralela que adotaremos, como veremos a seguir.

O processador paralelo **P3** é um híbrido de multiprocessador e multicomputador. Em cada módulo (topologia intramódulo) as UCPs partilham a memória global do módulo e portanto se comportam como um multiprocessador - devemos notar que apesar da memória ser composta de oito partes ela é encarada, do ponto de vista hardware, como uma memória global. Em um nível mais elevado (topologia intermódulos) temos os módulos conectados de maneira completa (cada módulo se liga a todos os demais) através dos transputers - o que seria um esquema do tipo multicomputador, se encarássemos cada módulo como um computador (multi) com memória comum. Não é difícil notar que o carater híbrido do hardware coloca o problema da escolha do modelo de computação paralela: troca de mensagens, memória partilhada ou híbrido?

Nossa opção atual é por um modelo baseado em troca de mensagens, mesmo em máquinas cujo hardware, total ou parcialmente [9], utiliza um esquema tipo memória comum. Assim o programador terá um único paradigma de programação paralela (troca de mensagens) abstraindo-se das características do hardware. Caberá ao sistema operacional traduzir, quando necessário, o esquema baseado na trocas de mensagem em termos do hardware específico - como é o caso do que ocorre dentro de um módulo no sistema do **P3** , por exemplo. Nessa máquina encararemos cada unidade **U-32** como um nó de uma rede de computadores individuais que se comunicam através de troca de mensagens - esse modelo vale tanto para a comunicação intermódulos como para a comunicação intramódulo. Uma vantagem dessa opção é que poderemos utilizar o mesmo modelo tanto para computadores paralelos como para redes de computadores. Outra vantagem é que o sistema operacional SOP, no seu estágio atual, já oferece as primitivas para implementação desse modelo em linguagens de alto nível como é o caso das linguagens Fortran e C.

3 SISTEMA OPERACIONAL SOP

A extensão das linguagens Fortran e C para processamento concorrente / paralelo está baseada no ambiente de execução do sistema operacional SOP [1]. Este sistema foi desenvolvido visando atender às necessidades de projetos para controle em tempo real e processamento distribuído. Está implementado no hardware do Processador Preferencial (PP) [2]. As atuais funções deste sistema, permitem com pequenas modificações, a sua utilização em processamento paralelo de baixo acoplamento. Dentre estas funções destacam-se:

- Iniciação e terminação de processos, implementado através das primitivas START e STOP.
- Comunicação e sincronização de processos através do mecanismo de sinais, estando os processos no mesmo processador ou em processadores distintos. Implementada através das primitivas SEND, RECEIVE.
- E/S remota que permitirá a carga de programas em processadores não equipados com dispositivos de memória de massa.
- Carregador residente em cada processador, responsável pela carga de programas solicitadas através do mecanismo de sinal, possibilitando o carregamento a partir de outro processador.

Convém salientar também a importância das funções de rastreamento, depuração e falha implementadas no SOP para o desenvolvimento de um sistema de teste eficiente, fundamental para este tipo de aplicação.

4 EXTENSÕES DAS LINGUAGENS FORTRAN E C

As extensões de linguagem que pretendemos introduzir visam uma ampla variedade de arquiteturas e aplicações. Entre as arquiteturas podemos ter desde conjuntos de processadores (PPs ligados através de uma rede), até computadores paralelos como é o caso do **P3** . Entre as aplicações visadas temos desde o controle em tempo real (distribuído ou não) até a resolução de sistemas de equações utilizando algoritmos paralelos. Dada a abrangência dos objetivos visados, procuramos sempre ter em mente duas preocupações básicas: a primeira foi a de se utilizar um procedimento unificado que permitisse a utilização das linguagens estendidas (C e Fortran) tanto em aplicações que envolvem concorrência como as que utilizam paralelismo de processos; a segunda preocupação foi a de utilizarmos de maneira eficiente as ferramentas de comunicação / sincronização de processos embutidas no sistema operacional SOP.

Fortran e C podem ser utilizadas para programar algoritmos paralelos, sem a necessidade de introduzir extensões de linguagem (em nível de sintaxe), desde que seja montado um sistema adequado de suporte para troca de mensagens - alterações no sistema operacional e criação de biblioteca de rotinas de comunicação a ser utilizada pelos nós do sistema de computação paralela. Um exemplo importante dessa abordagem é o projeto de processamento paralelo do Caltech [6]. As aplicações típicas de paralelismo exploradas em [6] envolvem a resolução de sistemas de equações (algébricas / diferenciais) nas quais o problema original é decomposto em uma série de subproblemas (programas) iguais que são alocados em processadores de uma máquina paralela - no caso um hipercubo.
O mesmo não ocorre quando visamos aplicações que envolvem concorrência - neste caso é essencial que as linguagens sejam estendidas para possibitarem a manipulação de processos. Apesar de não serem indispensáveis, essas mesmas extensões podem ser aproveitadas no tratamento de programas paralelos - além de facilitar a implementação e o entendimento, este tratamento unificado poderá se mostrar valioso em futuros desenvolvimentos quando visarmos aplicações envolvendo concorrência e paralelismo, simultaneamente.
Vejamos inicialmente o conceito de concorrência no contexto de um sistema monoprocessador. Toda a programação concorrente está baseada na definição de processo - processo é um conjunto de ações executadas sequencialmente, constituindo-se na unidade básica de processamento concorrente. Assim, os processos concorrentes que convivem em uma mesma UCP, disputam o mesmo recurso computacional - poderíamos dizer que os processos concorrem pela utilização do mesmo recurso, no caso a UCP. Um programa concorrente é um conjunto de um ou mais processos executados em um mesmo processador.
Em sistemas com mais que um processador, vários programas concorrentes podem ser executados simultaneamente, um em cada processador. Neste caso dizemos que os processos que em um dado instante estão sendo executados pelas UCPs, são processos paralelos (aqui a palavra paralelo é usada no sentido de simultaneidade). Dessa forma, em um sistema multiprocessador / multicomputador não se pode falar de um programa que está sendo executado pelo sistema - o que temos são vários programas, independentes ou não, cada um sendo executado por uma UCP.
Um programa executado em um processador de gerência, conforme será formalizado mais adiante, nada mais é que um programa concorrente do tipo monoprocessador, com a diferença fundamental de que ele é capaz de carregar outros programas nos demais elementos do conjunto de processadores. O formalismo da linguagem paralela trabalha com uma máquina virtual na qual é executado um programa paralelo global - os códigos que serão executados nos outros processadores aparecem explicitamente neste programa global como processos paralelos. Isto no entanto é apenas um formalismo; em nível de precompilação as partes de código explicitadas como paralelas são interpretadas como programas a serem carregados em outros processadores do sistema, e assim recaímos na situação descrita anteriormente.
Um programa que incorpore estes novos conceitos será submetido a um pré-compilador cujas funções básicas são as de análise das extensões, tradução das extensões para a sintaxe original e iniciação de informações necessárias para montagem e carga dos programas.
Dadas as limitações de espaço, discutiremos apenas os aspectos principais dos novos conceitos e das correspondentes extensões sintáticas.

Neste item introduziremos as definições de programa e processo. Essas definições serão introduzidas em um nível de abstração bastante elevado. Uma idéia mais concreta pode ser obtida da leitura dos dois exemplos apresentados mais adiante.
O termo " Declarações' representa apenas as declarações disponíveis nas linguagens Fortran ou C. Os termos " Definições' e " ações' representam as definições e ações disponíveis nessas linguagens mais as extensões nescessárias.

Definição de Programa:

```
Label: MODULE [Processor=] [,  Attribute=]
              [,  Copies=],([Parameter])

         [Declaracoes]
         [Definicoes]
         [Acoes]

END Label
```

Definição de Processo:

```
Label:  PROCESS [Mode=] [,  Processor=]
                [,  Attribute=]
                [,  Copies=] ([Parameter])

         [Declaracoes]
         [Definicoes]
         [Acoes]

END Label
```

As informações associadas a MODULE e PROCESS têm os seguintes significados:

Mode	Indicação se o processo é paralelo ou concorrente Parallel: Processo paralelo. Concurrent: Processo concorrente.
Processor	Número do processador onde será carregado o programa. Se não for especificado o processador o programa será carregado no processador do módulo de gerência [9].
Attribute	Informações para carga do processo (se paralelo) E/S: programa utiliza entrada e saída EX: só este programa pode ser carregado no processador. Informações sobre o processo (se concorrente) Por exemplo prioridade, tamanho da pilha, etc.
Copies	Número de cópias do programa a ser carregado.
Parameter	Parâmetros para o programa ou processo.

OBS:

- Os campos Copies e Processor não têm significado para processo concorrente.
- Uma definição de programa não pode conter outra definição de programa.
- Uma definição de processo não pode conter outra definição de processo.

No exemplo 1 abaixo mostramos a estrutura de um programa com três partes concorrentes: Proc1, Proc2 e o que chamamos de processo principal, formado pelas Declarações, Definições e ações não pertencentes às definições dos processos Proc1 e Proc2. O processo principal é responsável pela ativação dos demais e todos serão executados em um mesmo processador.

Exemplo 1:

```
Prog: MODULE ()

          Declaracoes
```

```
        Proc1: PROCESS Mode=Concurrent ()

           Declaracoes
           Acoes

        END Proc1

        Proc2: PROCESS Mode=Concurrent ()

           Declaracoes
           Definicoes
           Acoes

        END Proc2

        Acoes

END Prog
```

No exemplo 2 abaixo apresentamos um programa com três processos paralelos: Par1, Par2 e o que chamamos de processo paralelo principal, formado pelas Declarações, Definições e ações não pertencentes às definições dos processos Par1 e Par2. Estes processos paralelos serão carregados e executados nos processadores 0, 1 e 2 respectivamente.

Exemplo 2:

```
Prog: MODULE Processor = 0 ()

        Declaracoes

        Par1 : PROCESS Mode=Parallel,
                       Processor = 1 ()

          Declaracoes
          Acoes

        END Par1

        Proc1 : PROCESS Mode=Concurrent ()

          Declaracoes
          Definicoes
          Acoes

        END Proc1


        Par2 : PROCESS Mode=Parallel,
                       Processor = 2 ()

          Declaracoes
          Acoes

        END Par2

        Acoes

END Prog
```

4.2 Mecanismo para ativação de processos concorrentes e paralelos

A ativação da parte concorrente (processo) de um programa é a colocação do processo no estado de pronto para execução enquanto que a ativação de um processo paralelo é tratada como a carga de um programa isolado, visto que a mínima entidade passível a carregamento no SOP é um programa.
Esta ativação é feita pela ação START cujo formato é apresentado a seguir:

```
[Id=]START Process= [,Mode=] [, Processor=]
[,Attribute=] ([Parameter])
```

onde:

- Process: identifica o processo a ser ativado.
- Mode: identifica se o processo é para execução paralela ou concorrente.
- Processor: Número do processador onde o processo será carregado. Não significativo para processos concorrentes.
- Attribute: informações sobre o processo.
- Parameter: parâmetros para o processo.
- Id: variável que receberá a identificação do processo ativado.

O exemplo 3 abaixo mostra a ativação dos processos paralelos Par1, Par2 e a ativação do processo concorrente Proc1. Note que o número do processador, no caso dos processos paralelos, pode ser indicado também na sua definição, isto permite a ativação do processo por qualquer programa externo ao módulo.

Exemplo 3:

```
Prog: MODULE Processor = 1 ()

  Declaracoes
```

```
Par1: PROCESS Mode = Parallel ()

  Declaracoes
  Definicoes
  Acoes

END Par1

Par2: PROCESS Mode = Parallel ()

  Declaracoes
  Acoes

END Par2

Proc1: PROCESS Mode = Concurrent ()

  Declaracoes
  Acoes

END Proc1


i = START Process = Proc1,
          Mode = Concurrent ()
j = START Process = Par1,  Mode = Parallel,
          Processor = 2 (' MATRIZ_A')
k = START Process = Par2,  Mode = Parallel,
          Processor = 3 (' MATRIZ_B')

END Prog
```

4.3 Mecanismos de comunicação e sincronização entre processos

A comunicação entre processos através de sinal permitirá que processos no mesmo processador ou em processadores distintos troquem informações. A ação SEND fará com que o sinal indicado seja entregue ao processo destino, sem causar a suspensão do processo que a executa - ou seja, trata-se de um SEND não bloqueado. A ação RECEIVE é responsável pela recepção de sinais; o processo que a executa será suspenso caso nenhum dos sinais esperados tenha sido entrege e a alternativa ELSE não tenha sido especificada; quando um dos sinais indicados já está a disposição para ser consumido, a ação (procedimento) associada a este sinal será executada. A definição de um sinal para comunicação é dada por:

```
SIGNAL   Signal_name,  Sid = (Data_types)
```

onde:

- Signal_name - nome do sinal
- Sid - número que identifica o sinal
- Data_types - tipos dos dados do sinal separados por vírgula.

As ações para envio e recepção de sinal são SEND e RECEIVE, definidas como:

```
SEND Signal_name (Data) TO Id
```

onde:

- Signal_name - nome do sinal
- Data - dados a serem enviados.
- Id - identificação do processo que irá receber o sinal.

```
[Id =] RECEIVE (
        {Signal_name [IN Var] : Procedure,} +
         [ELSE : Acoes] )
```

onde:

- Entre {}+ indica repetição de comando.
- Id - é uma variável para recepção da identificação de quem enviou o sinal
- Signal_name - nome do sinal para recepção
- Var - variáveis separados por vírgulas para recepção dos dados do sinal
- Procedure - procedimento a ser executado se determinado sinal for recebido.
- ELSE - opção para continuar se nenhum dos sinais chegou.

A sincronização entre processos de programas distintos é feita via sinal (em geral sem dados), enquanto que a sincronização entre processos do mesmo programa, pode ser feita via sinal ou através de evento. As ações para sincronização através de evento são DELAY e CONTINUE definidas por:

```
DELAY e
CONTINUE e
```

onde: e é um evento. O processo que executa a ação DELAY é suspenso até que outro processo envolvido

na sincronização executa a ação CONTINUE. Para a declaração de um evento foi criado o tipo EVENT. O exemplo 4 abaixo ilustra a utilização destes mecanismos, onde o processo paralelo Prog envia o sinal x para o processo paralelo Par1. Os processos Proc1 e Proc2 utilizam de mecanismo de Evento para sincronização.

Exemplo 4:

```
Prog: MODULE Processor = 0 ()

  SIGNAL x,  SID = 10 (INTEGER,  INTEGER)
  e EVENT

  Par1: PROCESS Mode = Parallel,
                Processor = 1 ()

    Declaracoes
    Id = RECEIVE (
           x (IN k,  m): f(k,  m),
           y (IN l,  n): t(l,  n)
                 )

  END Par1

  Proc1: PROCESS Mode = Concurrent ()

    Declaracoes
    DELAY e

  END Proc1

  Proc2: PROCESS Mode = Concurrent ()

    Declaracoes
    CONTINUE e

  END Proc2

  i = START Process=Par1,  Mode=Parallel,
            Processor=1 ()

  SEND x (3,  2) TO i

END Prog
```

4.4 Mecanismos de comunicação complementares

As mesmas rotinas de comunicação utilizadas em programação concorrente (por exemplo, SEND/RECEIVE) também podem ser usadas em programas paralelos e, é claro, em programas híbridos (paralelo / concorrente). Entretanto, aplicações em engenharia e ciências (que em geral envolvem uso maciço de cálculo numérico e matricial) muitas vezes apresentam regularidades que podem facilitar a programação. A referência [6] que descreve o projeto de computação paralela do Caltech, sugere algumas rotinas de comunicação / sincronismo para as linguagens Fortran e C que exploram essas regularidades. Particularmente estamos interessados nas rotinas do tipo complementar, isto é, rotinas que são chamadas por todos os processadores / processos que participam da troca de dados (dois ou mais). Uma discussão mais ampla sobre o comportamento e a implementação SOP dessas rotinas foge do escopo do presente artigo, mas, a título de exemplo, e para dar uma idéia do estilo de programação que pretendemos adotar, nos parágrafos seguintes descrevemos duas dessas rotinas.

Um primeiro exemplo de rotina de tipo complementar é a rotina SHIFT(), que faz a troca de pacotes de dados entre dois processadores / processos. O resultado final é equivalente a uma sinalização mais dois pares SEND / RECEIVE executados nos dois processadores envolvidos. A rotina em si é simples mas serve para ilustrar a idéia de complementaridade. O mecanismo de troca está ilustrado na Fig. 3, e o formato de chamada, nos processadores é dado por:

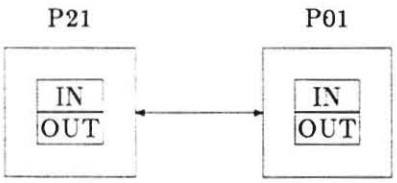
```
SHIFT (In='' ', Orig=21, Tam=15, Out='' ',
Dest=01, Tam=20)
e
SHIFT (In='' ', Orig=01, Tam=20, Out='' ',
Dest=21, Tam=15)
```

Outro exemplo ilustra uma rotina um pouco mais poderosa: BROADCAST(). Trata-se de uma operação de broadcast parcial (Fig. 3) onde, um processador despacha um pacote de dados para um conjunto especificado de processadores (um caso extremo seria aquele no qual o conjunto especificado de processadores incluisse todos os demais; teríamos então a operação broadcast propriamente dita, ou broadcast global). A chamada abaixo exemplifica este mecanismo:

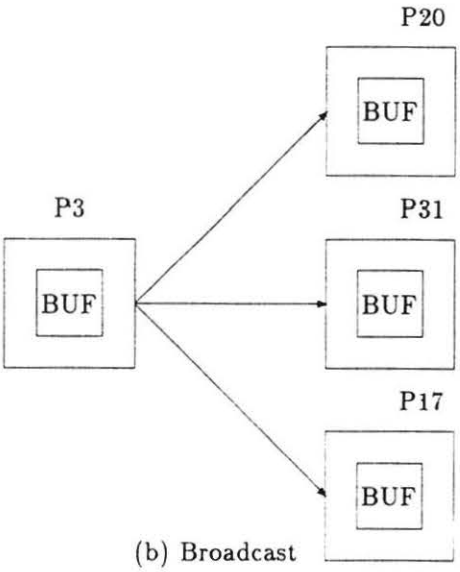
```
BROADCAST (Buff='' ', Orig=3, Nnos=4,
Nos='' Lista', Tam=10)
```

Onde ambas as rotinas descritas anteriormente exigem *rendezvous* [8], [4] entre os processos envolvidos na comunicação. Isto significa que elas introduzem pontos de sincronização nos programas paralelos: todos os processadores envolvidos devem chegar ao mesmo ponto para que a comunicação seja efetuada. Fora dos pontos onde essas rotinas são chamadas os processadores funcionam de maneira assíncrona. Diz-se então que os programas paralelos são fracamente sincronizados (sin-

cronismo forte seria aquele no qual todas as instruções seriam executadas passo-a- passo). A introdução de pontos de sincronização, apesar de um pouco restritiva, em geral facilita o desenvolvimento e a leitura de programas.



(a) Shift



(b) Broadcast

Figura 3: *Mecanismos SHIFT e BROADCAST*

5 DESENVOLVIMENTOS FUTUROS

Três problemas importantes não foram tratados nas seções precedentes, mas deverão ser objeto de trabalhos futuros: ambiente de depuração, simulação de programas paralalelos em máquinas sequenciais e balanceamento de carga entre processadores. O balanceamento de carga visa uma melhor utilização dos recursos hardware em tempo de execução - através de técnicas de balanceamento (estático e dinâmico) poderemos evitar estrangulamentos nas vias de comunicação bem como a ociosidade de processadores. As técnicas utilizadas nesta área são bastante complexas e estão sendo objeto de pesquisa de vários grupos interessados em computação paralela. Enquanto o balanceamento de carga visa a utilização eficiente da máquina em tempo de execução, o ambiente de depuração e a simulação de programas paralelos em máquinas sequenciais visam melhorar a eficiência do desenvolvimento de programas. O ambiente de depuração para programas concorrentes / paralelos é muito mais complexo que em sistemas com um processo / processador, e por isso mesmo deverá desempenhar um papel importante na aceitação e na utlização práticas das idéias aqui discutidas. Já o simulador de programas paralelos possibilitará o desenvolvimento e teste de programas paralelos em máquinas sequenciais - através do simulador podemos, entre outras coisas, testar a eficiência de algoritmos variando-se o número de processadores da máquina paralela simulada.

6 CONCLUSÕES

Este trabalho apresentou um modelo computacional baseado em troca de mensagens para sistemas que vão desde conjuntos de computadores sequenciais até máquinas paralelas. Procuramos apresentar o ambiente de programação em um elevado nível de abstração visando-se facilitar a portabilidade de programas. Procuramos também tratar das extensões de linguagem (Fortran e C) de maneira unificada o que, no futuro, assim esperamos, possa facilitar a utilização de programas nos quais processos paralelos (fracamente sincronizados) e processos concorrentes (executados de modo excludente em uma mesma máquina) possam conviver em um ambiente bastante flexível. O projeto que aqui relatamos em conjunto com outros artigos [9], [10], [11], tira proveito de desenvolvimento anteriores [1], [2], o que certamente facilitará a sua consecução. Muitas tarefas, entretanto, restam para serem formuladas e desenvolvidas como ficou claro na discussão apresentada na seção sobre Desenvolvimentos Futuros.

Referências

[1] E. Cavalli, M. C. Zabeu , " Sistema operacional para processamento paralelo', I SBAC-PP, anais, p. 101-114, Maio de 1987.

[2] E. Cavalli, M. C. Zabeu , " Sistema hardware para processamento paralelo', I SBAC-PP, anais. p. 91-100, Maio de 1987.

[3] D Gajski, D. Kuck, D. Lawrie e A. Sameh, " Cedar ', in Supercomputers: Design an Applications, Editado por Kai Hwang, p. 251-275, 1984.

[4] D. A. Mundie e D. A. Fisher, " Parallel processing in ADA', Computer, p. 20-25, Agosto de 1988.

[5] L. N. Bhuyan, " Interconection networks for parallel and distributed processing', Computer, p. 9-12, Junho de 1987.

[6] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon e D. W. Walker, " Solving problems in concurrent processors', Caltech, 1987.

[7] M. V. F. Pereira, M. J. Teixeira e L. A. Terry, CEPEL, " aplicações de processamento paralelo em sistemas elétricos de potência', I SBAC-PP, anais, p. 283-289, Maio de 1987.

[8] N. H. Gehani and W. D. Roome, " Concurrent C', Software Practice & Experience 16, 9, p. 821-844, Setembro 1986.

[9] E. Cavalli e A. Pestana, " Processador paralelo **P3** ', artigo submetido em conjunto ao II SBAC-PP.

[10] P. R. F. A. LUZ e L. G. R. ACACIO, " A Utilização do transputer T800 no sistema **P3** ', artigo submetido em conjunto ao II SBAC-PP.

[11] ZABEU M. C., " Análises e alternativas para contenção em multiprocessamento',artigo submetido em conjunto ao II SBAC-PP.