7º seminário integrado de software e hardware

21 a 25 de julho de 1980
FEC/UNICAMP

Promoção
-SBC/Sociedade Brasileira de Computação

Patrocinio
SEI/Secretaria Especial de Informática
FEC/Faculdade de Engenharia de Campinas
- UNICAMP/Universidade Estadual de Campinas

IMPLEMENTAÇÃO DE UM SISTEMA DE COMUNICAÇÃO ENTRE PROCESSOS ATRAVÉS DE UMA MÁQUINA DE ARQUITETURA DISTRIBUÍDA

AUTORES: Wilson Vicente Ruggiero (*)
Selma S.S. Melnikoff (*)
Carlos J. Lucena (+)

APRESENTADORA: Selma S.S. Melnikoff

ENTIDADES: EPUSP (*)/PUC-RJ (+)

Este trabalho teve patrocínio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

RESUMO

É possível conceituar com rigor a noção de sistema de computação distribuído. Tal conceito traz consequências importantes tanto para a arquitetura de computadores quanto para os métodos de programação paralela que são induzidos por essas arquiteturas. Um sistema distribuído deve ser programado através de processos e mensagens.

O presente trabalho descreve a implementação de um sistema de comunicação entre processos realizada através de uma arquitetura especialmente projetada em conformidade com os princípios básicos da computação distribuída. Esta implementação foi desenvolvida visando a solução de um problema de aquisição de dados para controle de processos. O hardware e software deste sistema estão operacionais e ilustram uma experiência na utilização prática de conceitos da área de sistemas concorrentes que vem sendo discutidos apenas no plano teórico.

1. INTRODUÇÃO

1.1 - Máquina de Arquitetura Distribuída

É comum encontrar-se na literatura usos diversos e, às vezes conflitantes, para o conceito de sistema distribuído. Vários autores usam a mesma expressão com diferentes sentidos [HEWI77], [LAMP76], [LISK79]. Desta forma se faz necessário definir precisamente o que vem a ser um sistema de arquitetura distribuída no contexto deste trabalho.

Um sistema distribuído é composto de um número ilimitado mas finito de elementos de processamento-armazenamento, capazes de operar concorrentemente. Num sistema com tal configuração, a noção de estado global é totalmente substituída pela de estado local.

A propriedade de execução concorrente coloca este tipo de sistema na classe de máquinas MIMD (FLYN 72). A troca de atributos globais por atributos locais é responsável pela distribuição de controle e de dados, característica esta fundamental de tais arquiteturas.

Tradicionalmente, as máquinas de arquitetura centralizada são baseadas no modelo de Von Neumann [NEUM46], [NEUM47]. Diversas tentativas foram feitas para aplicar esse mesmo modelo a sistemas concorrentes, mas os resultados mostram que o modelo se torna um fator excessivamente limitante [WULF72], [BARN68], [GLUS74]. A grande limitação do modelo de Von Neumann está no caráter sequencial e centralizado do controle e na conotação global dada ao espaço de memória.

O projeto de máquinas com arquitetura distribuída deve ser baseado em modelos concorrentes de computação [RUGG78], [GOST77], [GLUS74].

Da mesma forma que as máquinas de controle centralizado foram guiadas pelo modelo de Von Neumann, as máquinas paralelas devem ser dirigidas pelos modelos concorrentes. Assim, tentam-se evitar as características restritivas dos modelos sequenciais, procurando obter um entrosamento perfeito entre as técnicas de programação e interpretação paralela.

Os conceitos de programação paralela e computação distribuída levaram à exploração de arquiteturas para máquinas que forneçam infraestrutura para a realização destes propósitos.

Para que o sistema seja largamente distribuído, estas máquinas — as máquinas de arquitetura distribuída — são constituídas a partir de módulos com capacidade de processamento e armazenamento primário. Estes módulos, que são denominados de nós, trabalham independentemente entre si, porém de uma forma cooperativa. Isto significa que, de tempos em tempos, os módulos se comunicam entre si, a fim de trocar informações necessárias para o prosseguimento das tarefas.

A comunicação entre os nós é feita através de um sistema de comunicação que, por meio de um

esquema de endereçamento conveniente, transmite e recebe as informações, fornecendo ainda, meios de detetar os erros introduzidos no decorrer destas operações. Este sistema é constituído por um sub-sistema de interconexão e um protocolo de comunicações. O subsistema de interconexão é o meio físico através do qual a informação é transmitida e o protocolo de comunicações é o conjunto de procedimentos que coordenam a troca de informações. No presente trabalho, o que chamamos de sistema de comunicação é basicamente a implementação do protocolo de comunicação.

A estrutura de uma máquina de arquitetura distribuída pode ser visualizada na Fig. 1.

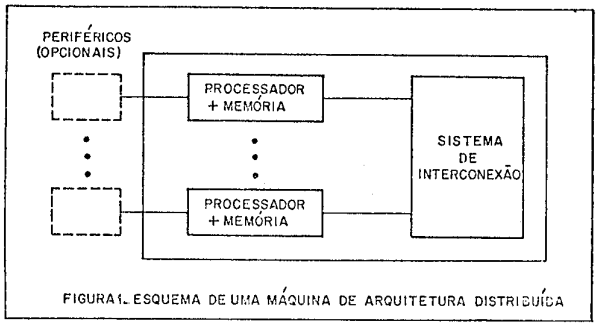

FIGURA 1 - ESQUEMA DE UMA MÁQUINA DE ARQUITETURA DISTRIBUÍDA

As implementações das funções lógicas de cada nó são realizadas através de um conjunto de processos que são as unidades básicas capazes de realizar funções concorrentes.

Estes processos se comunicam entre si através do sistema de comunicação, cujos protocolos se constituem no núcleo do software de uma máquina de arquitetura distribuída. O sistema de comunicação se encontra residente em cada nó e é um componente básico e indispensável para um sistema deste tipo.

Como foi dito anteriormente, a principal característica de uma máquina de arquitetura distribuída é a substituição do estado global do sistema pelo conjunto de estados locais de cada nó. Isto significa que, em um dado instante, cada elemento constituinte só conhece integralmente o seu estado local, o que lhe é suficiente para o seu processamento.

O conhecimento com absoluta certeza do estado global não pode ser essencial em nenhum instante, para o prosseguimento da computação, pois a sua determinação em tais sistemas possui um caráter probabilístico (RUGG80).

1.2 - Filosofia de Comunicação entre Processos

Na programação paralela, os processos constituem os módulos que realizam as ações concorrentes do sistema, como as subrotinas no caso sequencial. Para que estas ações sejam cooperativas, é necessário que haja comunicação e sincronização entre processos.

Comunicação e sincronização entre processos tem sido obtidas na programação paralela por meio do uso de espaços comuns de memória (memória partilhada) e da comunicação através de mensagens. Para o primeiro caso existem mecanismos de programação bem definidos embutidos em linguagens específicas e o segundo caso é um estilo de programação ainda pouco explorado, que será ilustrado neste trabalho através de um exemplo (seção 3).

A técnica de memória partilhada tem apresentado bons resultados na programação paralela, quando os sistemas não apresentam um alto grau de distribuição. Semáforos (DIJK68), regiões críticas (HOAR71) e monitores (HOAR74) são os principais mecanismos desenvolvidos para esta finalidade e foram fortemente influenciados pela conotação global dada ao espaço de memória no modelo de Von Neumann.

A comunicação através de memória partilhada não deve ser utilizada em sistemas largamente distribuídos, pois exigem um espaço de endereçamento comum entre os processos. O seu uso provocaria a centralização do sistema, reproduzindo todos os fatores limitantes do modelo de Von Neumann, quando este é aplicado a máquinas paralelas.

Pode-se notar que a comunicação através de mensagens é um mecanismo mais apropriado à programação em máquinas de arquitetura distribuída pelo fato de conservar a localidade do estado de cada nó e possibilitar a cooperação efetiva entre os processos.

As mensagens fluem do processo origem e atingem o processo destino depois de um intervalo arbitrário de tempo. Cada processo possui a capacidade de armazenar as mensagens a ele dirigidas, garantindo a ordem de chegada. Estas mensagens são entregues somente quando o processo requisitá-las. Obtem-se com isto, o caráter assíncrono de comunicação entre os processos.

Os métodos de sincronização e comunicação baseados em memória partilhada que, como foi dito, são extensões do caso sequencial inspirado no mo

delo de Von Neumann, encontram-se extensivamente estudados na literatura. É possível, por exemplo, analisar a correção de programas paralelos que utilizem os chamados mecanismos de sincronização estruturados (regiões críticas e monitores) e desenvolver sistematicamente programas paralelos corretos por construção. Noções recentes sobre estruturação e verificação de dados foram também incorporados à programação paralela com espaço compartilhado de memória, como é o caso da adaptação do conceito de tipo abstrato de dados através do mecanismo de monitores.

Uma sistematização equivalente está tendo lugar na área de programação paralela através de processos e mensagens (CUN80B). Presentemente, está-se investigando métodos formais de especificação de programas baseados em processos e mensagens, de forma a permitir a análise de suas principais propriedades (correção, terminação, etc) (CUN80A). A noção de tipos abstratos de dados também pode ser incorporada à programação através de mensagens, permitindo a formulação de tipos de dados para comunicação. Pretende-se reunir essas idéias em uma linguagem de programação que permita a programação sistemática com mensagens e processos (MELN80).

No presente trabalho, utilizou-se um tipo de dados para comunicação definido através de quatro operações: Envia Mensagem, Espera Mensagem, Espera Mensagem de Tipo Específico, Procura Mensagem de Tipo Específico. Essas operações estão definidas na seção 4.3. Os programas de aplicação que utilizam o sistema de comunicação operam, em última análise, sobre este tipo de dados. A análise dos programas de aplicação (por exemplo, existência ou não de situação de "deadlock") pode ser feita por métodos já desenvolvidos, se se utilizar uma notação adequada para definir as operações do tipo (CUN80A).

2. DESCRIÇÃO FUNCIONAL DO SISTEMA DE COMUNICAÇÃO

O Sistema de Comunicação (SC) é um conjunto de programas, cujas cópias residem em cada nó da máquina de arquitetura distribuída. Estes programas, juntamente com o hardware, constituem a infraestrutura básica do sistema.

A principal função do SC é promover a comunicação entre os seus processos usuários (contidos nos programas de aplicação), através do sistema de interconexão, de um modo confiável e transparente.

Além disso cabe ao SC coordenar o partilhamento do do processador do nó através de mecanismos que permitam ativar e suspender processos, chavear o uso do processador, gerenciar os processos conforme a sua prioridade, etc.

A comunicação entre os processos é feita através de mensagens. A forma pela qual os processos usuários se comunicam entre si não é influenciada pela alocação dos processos aos nós. Em outras palavras, o SC faz com que o usuário do tipo de dados de comunicação não precise se preocupar com a implementação do tipo. Desta forma, o mapeamento dos processos deixa de ser um vínculo do sistema, uma vez que suas interfaces com o SC são uniformes.

Note-se que esta flexibilidade é fundamental para dotar a máquina com a característica de reconfiguração. Dependendo de conveniência ou necessidade, os processos podem ser reagrupados dinamicamente, isto é, podem ser transferidos de um nó a outro durante a execução.

Deve-se levar em consideração, no entanto, que o custo de comunicação entre processos residentes no mesmo nó é menor do que o custo de comunicação que envolve processos em nós distantes.

O aspecto assíncrono da comunicação através de mensagens se deve à capacidade de armazenamento de mensagens nos processos.

Cada processo tem acesso a uma área, onde as mensagens são armazenadas na ordem de chegada ao processo e são somente retiradas quando o processo está apto a recebê-las e tratá-las.

Tem-se na Fig. 2 a representação esquemática de dois processos P1 e P2 que trocam mensagens. A1 e A2 são, respectivamente, as áreas de mensagens dos processos P1 e P2.

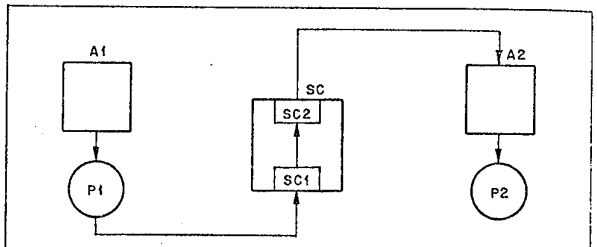

FIGURA 2_ESQUEMA DE TROCA DE MENSAGENS ENTRE P1 E P2

A transferência de mensagens é feita através dos SC's dos nós envolvidos, os quais inserem a mensagem enviada na área de mensagem do processo destino (que no exemplo é o P2) e avisa-o da chegada da mensagem. O processo origem pode ficar bloqueado após o envio, dependendo do número admis

sível de mensagens pendentes nessa comunicação. Uma mensagem pendente é aquela que foi aceita pelo SC origem e ainda não foi entregue ao SC destino.

Um processo só deve requisitar uma mensagem quando o seu estado interno for adequado à recepção. Ao entrar neste estado, o processo informa o SC da necessidade de uma mensagem. O SC analisa a área de mensagens e entrega um dos elementos nela contidos. Se a área estiver vazia, o processo fica bloqueado até a chegada de uma mensagem.

Além das primitivas do envio e recepção anteriormente descritas, podem existir outras operações de comunicação, próprias de uma aplicação específica. Um exemplo típico é a operação de recepção seletiva, onde o processo especifica ao SC, os tipos de mensagens que ele está capacitado a tratar no estado em que se encontra.

O número de operações e a sua flexibilidade dependem das características dos processos usuários. A seleção do conjunto deve ser feita considerando-se a sua conveniência na programação dos processos, a complexidade da implementação e o custo da execução. Cada área de aplicação requer, portanto, um tipo de dados de comunicação distinto.

A comunicação entre os processos pode envolver a comunicação entre os nós, a qual é feita pelas funções do SC que realizam recepção e transmissão através do sistema de interconexão.

A função de recepção tem a finalidade de retirar, do sistema de interconexão, as mensagens dirigidas a este nó e distribuí-las entre os processos destinos ali residentes. Por outro lado, a função de transmissão encaminha as mensagens enviadas aos processos externos e realiza a inserção destas no sistema de interconexão. Cabe ainda a estas funções assegurar a recepção correta de mensagens transmitidas, através de retransmissões e testes de consistências das mensagens.

Além das funções anteriormente descritas, o SC possui ainda um mecanismo para limitar o tempo de bloqueio dos processos, baseado num esquema de contagem de tempo (temporização). Os processos dispõem de primitivas para ligar e desligar convenientemente as temporizações, para controlar a ocorrência de eventos esperados.

3. EXEMPLOS DE PROGRAMAÇÃO ATRAVÉS DE PROCESSOS E MENSAGENS

Uma vez descritas as funções do SC, será apresentado nesta seção um exemplo que ilustra o princípio da programação por processos utilizando mensagens.

Escolheu-se, para isto, o problema clássico de produtor e consumidor, cuja especificação está descrita a seguir:

- Existem duas entidades: o produtor que fornece os elementos e o consumidor que utiliza estes elementos.
- Existe uma área de armazenamento ("buffer") de capacidade limitada, entre o produtor e o consumidor.

A solução ao problema é apresentada utilizando-se três processos, a saber:

- processo produtor (PP): produz um elemento e pede ao processo gerenciador para inseri-lo na sua área de armazenamento;
- processo consumidor (PC): pede ao processo gerenciador um elemento da área de armazenamento e consome-o;
- processo gerenciador (PG): gerencia a utilização da área de armazenamento empregada por PP e PC, a fim de evitar conflitos e informa o sucesso ou falha da operação requisitada.

A descrição do fluxo de dados é mostrada na Fig. 3, utilizando-se a notação do Graph Model of Behavior [RUGG78].

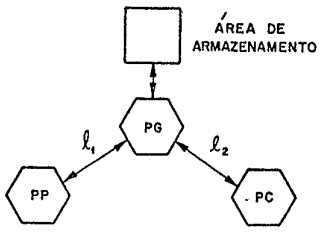

FIGURA 3 - FLUXO DE DADOS ENTRE PROCESSOS

Os diálogos existentes nos enlaces l1 e l2 estão apresentados na Fig. 4, onde estão especificados a sequência e os tipos de mensagens que são trocadas entre os processos.

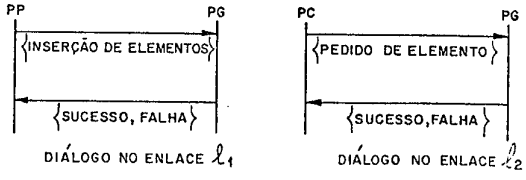

FIGURA 4: DIÁLOGOS ENTRE PROCESSOS

A interpretação dos processos encontra-se sob a forma de programa não executável, onde se utilizaram os comandos de controle e tipos de variáveis segundo a notação da linguagem PL/M. As variáveis do tipo abstrato mensagem são declaradas utilizando-se a declaração de STRUCTURE de PL/M.

O conjunto de regras em BNF apresentado no próximo parágrafo descreve as operações primitivas sobre o tipo de dado paracomunicação utilizadas nesta seção. Esta notação não faz parte da linguagem PL/M.

<operação primitiva> :: = [<tipo de operação>,
(<lista de identificador de processos>),
<identificador de mensagem>]

<lista de identificador de processos>:: = <identificador de processo>|
<lista de identificador de processos>,<identificador de processo>

<identificador de processo> :: = <variável>

<identificador de mensagem> :: = <variável>

<tipo de operação> :: = ENVIA | ESPERA

A solução do problema não considera a possibilidade de que os processos "morram" no decorrer da execução. Para tanto seria necessário ligar as temporizações, quando se espera resposta à uma mensagem enviada.

Tem-se, então, a interpretação dos processos PP, PC e PG, empregando-se as convenções anteriormente descritas.

```
PP:DO;
   .
   .
   .
   L1: CALL PRODUZ$ELEMENTO;
       CALL MONTA$PEDIDO$DE$INSERÇÃO (M);
   L2: (ENVIA,  (PG), M);
       (ESPERA, (PG), R); /* recebe mensagem R */
       IF R.TIPO = SUCESSO
          THEN GO TO L1;
          ELSE GO TO L2;
   END;
PC:DO;
   .
   .
   .
   CALL MONTA$PEDIDO$DE$ELEMENTO (P);
   L1: (ENVIA, (PG), P);
       (ESPERA, (PG), R); /* recebe mensagem R */
       IF R.TIPO = SUCESSO
          THEN CALL CONSOME$ELEMENTO;
       GO TO L1;
   END;
PG:DO;
   .
   .
   .
   L1: (ESPERA, (PP, PC), P); /* recebe mensagem P */
       DO CASE P.TIPO;
          IF EXISTE$POSICAO  /* pedido de inserção */
             THEN DO; CALL INSERE (P.DADO);
                      (ENVIA, (PP), SUCESSO);
                  END;
             ELSE (ENVIA, (PP), FALHA);
          IF EXISTE$ELEMENTO /* pedido de elemento */
             THEN DO; CALL RETIRA (SUCESSO.DADO);
                      (ENVIA, (PS), SUCESSO);
                  END;
             ELSE (ENVIA, (PS), FALHA),
       END;
END;
```

4. IMPLEMENTAÇÃO DO SC

4.1 - Considerações Gerais

A máquina de arquitetura distribuída projetada e implementada no Laboratório de Sistemas Digitais da Escola Politécnica da Universidade de São Paulo tem como unidade básica o nó, com capacidade de armazenamento-processamento e cujo elemento central é o microprocessador INTEL 8085(SHIM80).

O sistema de interconexão é constituído por dois anéis onde cada nó está ligado através de duas interfaces de comunicação, conforme mostra a Fig. 5.

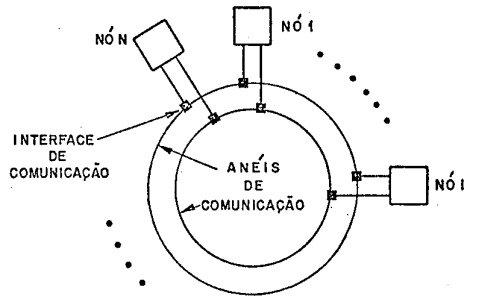

FIGURA 5 - ESTRUTURA DA MÁQUINA DE ARQUITETURA DISTRIBUÍDA

A transferência de dados entre a memória de um nó e o sistema de interconexão é feita através de acesso direto à memória.

Os programas que constituem o SC foram escritos em "assembler" de forma a serem compatíveis com as chamadas em PL/M, que é a linguagem de alto nível disponível aos usuários.

A implementação do SC visou principalmente a eficiência na execução dos seus programas. Alguns testes de consistência que seriam normalmente feitos foram propositalmente eliminados do SC, a favor de maior eficiência. Fica a cargo de processo de aplicação utilizar adequadamente as primitivas disponíveis.

Estas considerações, no entanto, não devem comprometer o bom funcionamento do sistema. Isto porque, no presente caso, a máquina de arquitetura distribuída foi utilizada para uma aplicação dedicada, onde os programadores dos processos usuários estão conscientes destas particularidades.

Nesta implementação o SC foi especializado para um sistema de aquisição de dados. Da análise das características deste sistema concluiu-se que não há necessidade de reagrupamento dinâmico de processos no decorrer da execução das tarefas. Isto significa que os processos são agrupados na fase de iniciação e nunca são desmembrados.

Os mecanismos explícitos de criação e destruição de processos podem ser introduzidos ao SC atualmente implementado, acrescentando-se a ele algumas funções adicionais, tais como o gerenciamento das áreas de dados e programas, relocação de códigos na memória e controle de ativações múltiplas.

4.2 - O Ambiente de um Nó

O SC encontra-se sempre residente em todos os nós da máquina de arquitetura distribuída. Os nós que possuem processos de aplicação a eles alocados são considerados nós ocupados.
Os nós que possuem somente o SC são chamados de nós livres.

A multiprogramação do nó é realizada através do elemento do SC, denominado de escalador de processos. Sua função é fazer o chaveamento dos processos para a utilização da unidade central de processamento, salvando e restaurando os respectivos estados.

O mecanismo de sincronização entre os elementos do SC, internos a um nó, foi implementado através dos semáforos de Dijkstra (DIJK68). Estes semáforos sinalizam ocorrência de eventos, ocupação e liberação de recursos.

A manipulação das filas de recursos são feitas obedecendo às prioridades dos processos. Quando os processos tem a mesma prioridade, o conflito é resolvido através da sua ordem de chegada na fila.

A troca de mensagens entre os processos é realizada através de uma área de comunicação do SC ("buffer pool"), onde se encontram os blocos disponíveis para serem usados na transferência de mensagens. Cada bloco comporta 128 "bytes" que é a dimensão da maior mensagem que circula no sistema.

A manipulação destes blocos pelos processos é feita através das primitivas do SC que são o pedido e a liberação de um bloco. Os blocos da área de comunicação foram feitas acessíveis aos processos usuários, para que estes manipulem diretamente as mensagens nesta área, em vez de trabalhar nas suas áreas locais. Esta alternativa foi escolhida pois, se as mensagens fossem transferidas por valor, isto é, fossem deslocadas da área local do processo para área do SC e vice versa, a eficiência do sistema na execução das operações primitivas de mensagens seria baixa, devido ao ciclo do processador ("overhead" alto).

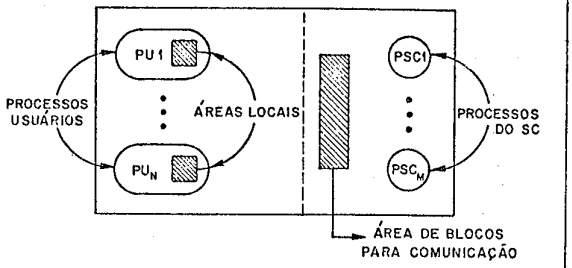

FIGURA 6 - ORGANIZAÇÃO DE UM NÓ

Tem-se na Fig. 6 a organização de um nó da máquina de arquitetura distribuída implementada, sob o ponto de vista dos processos que o compõem.

4.3. Mecanismo de Comunicação entre Processos

Das considerações feitas anteriormente tem-se que a comunicação entre os processos é realizada através de mensagens, independentemente da alocação destes processos.

Uma mensagem é constituída de 128 "bytes" contíguos, no máximo, e a sua estrutura se encontra na Fig. 7

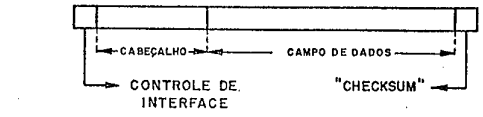

FIGURA 7 - ESTRUTURA DE UMA MENSAGEM

O campo de dados contém as informações que são efetivamente relevantes aos processos que se comunicam entre si, isto é, são os dados que um processo envia a um outro.

O cabeçalho é uma área onde são armazenadas as informações necessárias para a manipulação de mensagens, tais como, a identificação da origem e do destino, o comprimento e o tipo da mensagem, a identificação da mensagem, etc.

O controle de interface e o "checksum" são campos utilizados somente na comunicação que envolve o sistema de interconexão. Estes dois campos são manipulados pela interface de comunicação para deteção de erros de transmissão (SHIM80).

A seleção de operações primitivas com mensagens foi feita levando-se em consideração a complexidade da estrutura dos processos de aplicação, bem como, a conveniência para a depuração e modificação futura dos programas que constituem o sistema.

As operações primitivas implementadas são as seguintes: envia mensagem, espera mensagem (de qualquer tipo), espera mensagem de tipo específico e procura mensagem de tipo específico.

As duas últimas operações foram introduzidas no sistema pois observou-se, durante o projeto dos processos de aplicação, que a sua existência facilitava bastante a estruturação e a legibilidade dos programas.

A descrição das operações primitivas foram feitas utilizando-se a estrutura de controle de PL/M e encontra-se nos parágrafos a seguir:

Envia mensagem:

Entrega a mensagem ao SC origem para que este encaminhe ao SC destino;

Espera Mensagem (de qualquer tipo)

```
if existe mensagem na fila de mensagem
   do processo corrente
   then retira e devolve o primeiro ele
        mento da fila;
   else do; bloqueia o processo;
            aguarda a chegada de uma men
               sagem a este processo;
            retira e devolve o primeiro
               elemento da fila;
   end;
```

Espera mensagem de tipo específico

```
L: Procura a mensagem do tipo especificado, na
     fila de mensagens;
   if achou a mensagem
      then retira e devolve o elemento encon
           trado;
      else do;
              bloqueia o processo;
              aguarda a chegada de uma mensa
                  gem a este processo;
              go to L;
      end;
```

Procura mensagem do tipo específico

```
Procura a mensagem do tipo especificado na fi
     la de mensagens;
if  achou a mensagem
    then retira e devolve o elemento encontra
         do;
    else sinaliza a não existência da mensagem;
```

Como foi observado na seção 4.2, a manipulação de mensagens deve ser feita diretamente na área de comunicação do SC, sempre que possível, para evitar a sobrecarga devido a transferência de dados.

Desta forma, o procedimento de um processo para enviar uma mensagem consiste em:

a) pedir um bloco para SC
b) montar a mensagem no bloco obtido
c) enviar a mensagem

Ao efetuar a chamada da operação primitiva de envio de mensagem, o processo origem fornece ao SC o endereço do bloco onde ele montou a mensagem. A liberação deste bloco em relação ao processo origem é automática.

Da mesma forma, para receber uma mensagem, um processo deve proceder da seguinte maneira:

a) esperar uma mensagem
b) consultar ou armazenar os dados da mensagem
c) liberar o bloco.

O processo, ao sair do estado de espera de mensagem, recebe o endereço do bloco da área de comunicação, onde se encontra a mensagem a ele destinado, havendo portanto uma alocação implícita deste bloco.

4.4 - Mecanismo de Comunicação através do Sistema de Interconexão

A comunicação entre os nós é feita através

de processos de entrada (PE) e processos de saída (PS).

A entrada e a saída foram implementadas através de processos para se poder atribuir a eles prioridades mais altas do que as dos processos usuários. Esta medida foi tomada pois as suas funções envolvem operações de alta velocidade, tanto na recepção, quanto na transmissão.

Os processos de entrada tem a finalidade de receber mensagens e analisá-las quanto aos possíveis erros ocorridos na transmissão através do meio físico. Como o sistema tem dois anéis distintos, decidiu-se utilizar dois processos de entrada, cada um deles dedicado à recepção de um anel.

Um processo de entrada funciona, portanto, como distribuidor de mensagens entre os processos internos ao nó, pois ao receber uma mensagem sem erro, encaminha-a ao seu destino legítimo.

Os processos de saída, por outro lado, são as imagens dos processos destinos localizados em nós distantes. Em outras palavras, mensagens endereçadas aos processos externos são encaminhadas aos processos de saída convenientes e estes se encarregam de transmiti-las para o sistema de interconexão.

Para assegurar o recebimento correto das mensagens pelo processador destino, um processo de saída espera a chegada de uma mensagem de reconhecimento (ACK), referente à mensagem transmitida. Caso esta resposta não chegue dentro de um intervalo de tempo pré-estabelecido, o processo de saída providencia a retransmissão da mensagem.

Uma mensagem é transmitida no máximo quatro vezes, duas vezes em cada anel, até se obter a resposta do destino. Se, ao fim destas tentativas não houver a indicação do recebimento, o processo de saída providencia um tratamento de erro.

Um processo de saída sempre transmite uma mensagem por vez e fica bloqueado a cada transmissão, esperando o reconhecimento (ACK) da sua recepção. Somente depois da chegada deste, o processo de saída passa a transmitir a mensagem seguinte da sua fila de mensagem. Por esta razão, é interessante que exista mais de um processo de saída para haver uma utilização mais eficiente do sistema. Adotou-se o esquema de se ter um processo de saída para cada conjunto de processo alocado em um processador distante. Portanto, o número de processos de saída de cada nó depende das funções dos processos de aplicação nele residentes.

A implementação de entrada e saída ao sistema de interconexão através da filosofia adotada mostrou-se flexível o suficiente para a aplicação descrita, além de apresentar uma boa eficiência da utilização dos anéis de comunicação.

5. CONCLUSÕES

O hardware e o sistema de comunicação da máquina de arquitetura distribuída se encontram operacionais e os processos de aplicação estão sendo incorporados ao conjunto para que o sistema realize operações de aquisição de dados e controle de processos.

Para esta finalidade, a máquina necessita de um mínimo de 13 nós e a sua configuração pode ser expandida para comportar até 32 nós na sua versão mais completa.

A implementaçáó do SC resultou em um programaque ocupa aproximadamente 4K"bytes" de área de programa. A área de dados do SC pode ser configurada, dependendo do número de processo contidosem cada nó e da sua complexidade. A configuração mínima ocupa cerca de 4K"bytes" de dados.

O projeto de hardware e software da máquina de arquitetura distribuída descrita no presente trabalho foi desenvolvido conjuntamente, isto é, durante a concepção da sua arquitetura, já se previu a estrutura de software do sistema.

Uma das preocupações constantes com o projeto desta máquina foi a realização de um esquema eficiente de troca de mensagens, evitando memória partilhada. Com a construção de um protótipo de tal sistema se tornou viável a realização de experiências com o intuito de medir o desempenho do sistema proposto.

O projeto permitiu ainda a experimentação com métodos para programação paralela baseada em mensagens. Em particular, foi investigada a aplicação do conceito de tipos de dados no contexto desta modalidade de programação, o que originou uma implementação para a noção de tipos de dados para comunicação. Idéias originárias dessa experiência serão utilizadas no projeto de uma linguagem de programação de alto nível para programação por mensagens e no uso dessa linguagem para o desenvolvimento sistemático de software para aplicações concorrentes (MELN80).

REFERÊNCIAS BIBLIOGRÁFICAS

[BARN68] Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L., Stockes, R.A. "The ILLIAC-IV Computer", IEEE Trans. on Comp. C-47, 8 (August 1968)

[CUN80A] Cunha, P., Lucena, C.J., Maibaum, T. "A Methodology for Message Oriented Programming", Programmiersprachen und Programmentwicklung, Springer-Verlag, 1980

[CUN80B] Cunha, P., Lucena, C.J., Maibaum, T. "On the Design and Specification of Message Oriented Programming", International Journal of Computer and Information Science, aceito para publicação em 1980

[DIJK68] Dijkstra, E.W. "Cooperating Sequential Processes, "Programming Languages, F. Genuys (ed.), Academic Press, New York,1968

[FLYN72] Flynn, M. "Some Computer Organizations and their Efectiveness", IEEE Trans. on Comp. C-21, 9 (Sept. 1972)

[GLUS74] Glushkov, V.M., Ignatyev, M.B., Myasnikov, V.A., Torgashev, V.A. "Recursive Machines and Computing Technology", Information Processing 74, North-Holland Publishing Co., 1974

[GOST77] Gostelow, K., Arvind "A Computer Capable of Exchanging Processors for Time", IFIP Congress 1977, Toronto, Canada, August 1977

[HEWI77] Hewitt, C., Backer, H. "Laws for Communicating Parallel Processes", IFIP, North-Holland Publishing Company, 1977

[HOAR71] Hoare, C.A.R., "Towards a Theory of Parallel Programming", International Seminar on O.S. Techniques , Belfast, Northern Ireland August-September 1971

[HOAR74] Hoare, C.A.R. "Monitors, an Operating System Structuring Concept" CACM, October 1974

[LAMP76] Lampson, B., Sturgis, H. "Crash Recovery in a Distributed Data Storage System", Xerox Research Center, Palo Alto, Ca., 1976

[LISK79] Liskov, B. "Primitive for Distributed Computing", Proc. of the 7th Symposium on Operating Systems Principles, ACM-SIGOPS, December 1979

[MELN80] Melnikoff, S.S.S., Tese de Doutoramento em Andamento (EPUSP)

[NEUM46] Neumann, J.V., Burks, A.W., Goldstine, H. "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" John Von Neumann Colleted Works, Vol. V, Pergamon Press, 1963 (1946)

[NEUM47] Neumann, J.V., Goldstine, H. "Planning and Coding Problems", John Von Neumann Colleted Works, Vol. V, Pergamon Press, 1963 (1947)

[RUGG78] Ruggiero, W.V. "A Distributed Data and Control Driven Machine: Programming and Architecture", UCLA Technical Report UCLA-ENG-7878, November 1978

[RUGG80] Ruggiero, W.V. "Arquitetura de uma Máquina de Controle Distribuído", em preparação

[SHIM80] Shimizu, E.Y., Ruggiero, W.V., Moscato, L.A., "Componentes Básicos de uma Máquina de Arquitetura Distribuída: Identificação e Implementação", submetido ao VII Seminário Integrado de Software e Hardware (1980)

[WULF72] Wulf, W.A., Bell, C.G. "CMMP-A Multi-mini processor", AFIPS Proc. Fall Joint Comp. Conf., Vol. 41, AFIP Press, Montvale, N.J. 1972

Wilson V. Ruggiero: é formado pela Escola Politécnica da Universidade de São Paulo, em 1972.Recebeu o título de Mestre em Engenharia pela Escola Politécnica da Universidade de São Paulo, em 1975 e o título de Ph.D em Ciência de Computação por University of California at Los Angeles em 1978. É professor Assistente no Departamento de Engenharia Elétrica da EPUSP. Sua principal área de interesse é a de computação distribuída e redes de computadores.

Selma S.S. Melnikoff: é formada pela Escola Politécnica da Universidade de São Paulo, em 1971. Recebeu o título de Mestre em Engenharia pela Escola Politécnica da Universidade de São Paulo,em 1977. Está desenvolvendo sua tese de doutoramento na EPUSP. Sua principal área de interesse é a de computação distribuída e linguagens de programação.

Carlos J. Lucena: é formado pela Pontifícia Universidade Católica do Rio de Janeiro, em 1965. Recebeu o título de Master of Mathematics pela University of Waterloo, Canada, em 1969 e o título de Ph.D. em Ciência de Computação por University of California at Los Angeles, em 1974. É professor Associado do Departamento da Informática da PUC-RJ. Sua principal área de interesse inclui a metodologia de programação e projeto de linguagens de programação.

"Traçado de rotas sem falhas em placas de dimensões não pré-fixadas."

Autores: Hans Liesenberg
Nelson Castro Machado

Apresentador: Hans Liesenberg

Entidade: Unicamp-Imecc, 13100 Campinas/SP

Resumo: É descrito um traçador de rotas que utiliza como entrada, em lugar da localização específica de cada componente na placa, apenas uma descrição da posição relativa dos componentes, que chamaremos de vizinhaça orientada. A relação de vizinhança orientada entre componentes dispensa técnicas interativas projetista/máquina e garante uma resolução sem falhas em placas de tamanho não pré-fixado. O espaço entre dois componentes não é limitado de início; e depois de conhecido o número de conexões, que passam entre componentes vizinhos, uma localização definitiva para componentes e conexões é determinada, ou seja, a placa inicialmente "infinita" sofre uma "contração", assumindo dimensões finitas.

I - Introdução

O surgimento dos circuitos integrados provocou um aumento muito grande na densidade de conexões entre os componentes de uma placa de circuito impresso. O projeto manual de traçado de uma destas placas é uma tarefa que pode passar de trivial a impossível à medida que o número de conexões aumenta, e pode levar até diversas semanas para ser completada. O custo envolvido neste tempo justifica a necessidade da automação do projeto.

O traçado por computador não é uma atividade nova de pesquisa: investiga-se nesta área desde 1961 [1] e já se dispõe de uma vasta bibliografia sobre o assunto e sistemas industriais, extremamente sofisticados em operação. Há, entretanto, um considerável vazio entre métodos publicados e os sistemas industriais. A bibliografia normalmente descreve algoritmos de interesse teórico, mas cuja aplicação prática deixa muito a desejar em termos de eficiência ou devido a restrições ou generalizações, que não podem ser obedecidas em placas reais Por outro lado, os sistemas industiais, de grande valor econômico, quando não de uso privativo, estão disponíveis em forma de pacotes fechados sem nenhum detalhamento sobre a operação e estrutura dos métodos utilizados. Além disto, estes sistemas foram projetados para aplicações em indústrias desenvolvidas, altamente automatizadas e com grande volume de produção, o que até certo ponto conflita com as necessidades de indústrias e universidades brasileiras em que as condições são exatamente opostas.

Os algoritmos publicados de labirinto ou celulares [1,2,3,4] e heurísticos [5,6,7,8,9,10] utilizam-se de posições pré-fixadas dos componentes na placa e o traçado consiste em contornar ou evitar obstáculos decorrentes de pinos ou conexões já determinadas. Neste caso o aumento da densidade tende a tornar mais longas novas conexões. Algumas proibitivamente longas, ou impossíveis de serem traçadas, provocando falhas. Isto é consequência da localização fixa dos componentes já no início do traçado, pois quanto mais saturado o espaço entre dois componentes, maior a dificuldade em se achar um espaço para se traçar nova conexão.

A localização pré-fixada é indispensável quando os componentes são introduzidos por máquina durante a fabricação da placa. Mas isto só ocorre em linhas de produção altamente automatizadas, o que não é a realidade do país. Uma grande vantagem em não utilizar localizações pré-fixadas é que isto garante uma solução sem falhas em placas de duas ou mais faces. Torna-se desnecessária, portanto, a fase manual de correção do traçado obtido automaticamente.

II.1 - O algoritmo proposto

A principal característica do algoritmo proposto é que só existe uma relação cardeal de vizinhança entre componentes (por exemplo: o componente A é vizinho leste de B). Os canais comuns, que suportarão as conexões, são de dois tipos: os canais próprios, que interligam componentes vizinhos, e canais simples, que se encontram entre duas fileiras de componentes. Os canais comuns tem capacidade ilimitada, o que caracteriza a placa inicialmente "infinita". Esta situação é representada por um grafo (figura 1) onde os vértices são componentes ou interseções dos canais horizontais com os canais verticais; as arestas são segmentos de canais entre interseções. O grafo assim obtido é chamado de macro-reticulado. Os canais horizontais e verticais estão supostamente em faces distintas da placa.

O algoritmo de traçado é dividido em duas fases: a primeira determina, para cada sinal uma subárvore no macro-reticulado e na segunda fase, depois que foram determinadas todas as subárvores na primeira fase e, consequentemente, conhece-se o número de conexões que um canal deverá comportar, é determinado um espaço físico (canais individuais) para cada componente. Só depois da segunda fase as reais dimensões da placa são conhecidas. Obviamente, o sucesso do método depende do uso parcimonioso do espaço de maneira a expandir a placa o mínimo possível.

II.2 - Primeira fase, passo 1

A primeira fase baseia-se no algoritmo do Dijkstra [11] , que determina o comprimento do caminho mínimo entre dois vértices de um grafo a cujas arestas está associado um peso ou custo positivo. As arestas dos canais simples são rotuladas (peso associado) com valores maiores do que as arestas dos canais próprios a fim de que estes sejam ocupados sempre que possível. Este objetivo justifica-se, porque as arestas dos canais próprios sempre ocupam o espaço determinado pelas dimensões da borda do componente na qual incide a aresta. Portanto, é preferível ocupar este espaço com conexões do que ocupar arestas de canais simples que implicam no aumento das dimensões da placa.

Nomenclatura e notação:

G: grafo conexo com arestas rotuladas.
V(G): vértices de G.
A(G): arestas de G.
Seja $u,v \in V(G)$, então
(u,v): aresta com extremos u e v;
r(u,v): rótulo da aresta com extremos u e v;
l(u): rótulo associado ao vértice u;
g(u): grau do vértice u, ou seja, número de arestas incidentes em u.
subgrafo H de G $\Leftrightarrow V(H) \subseteq V(G)$ e $A(H) \subseteq A(G)$.
ciclo: sequência não vazia finita $v_0 a_1 v_1 \cdots a_k v_k$ onde $v_i \in V(G)$ $(0 \leq i \leq k)$, $a_i \in A(G)$ $(1 \leq i \leq k)$, os extremos de a_i são v_i e v_{i-1} $(1 \leq i \leq k)$, $a_i \neq a_j$ $(i \neq j)$ e $v_0 = v_k$.
caminho: sequência não vazia finita $v_0 a_1 v_1 \cdots a_k v_k$ tal que $v_i \in V(G)$ $(0 \leq i \leq k)$, $a_i \in A(G)$ $(1 \leq i \leq k)$, os extremos de a_i são v_i e v_{i-1} $(1 \leq i \leq k)$ e $v_i \neq v_j$ $(i \neq j)$.
caminho mínimo entre u e v: um caminho cuja soma dos rótulos das arestas, que fazem parte do caminho, é mínima em relação a todos os caminhos entre u e v.
d(u,v): distância do caminho mínimo entre u e v.
O caminho $v_0 a_1 v_1 \cdots a_k v_k$ pertence a um subgrafo H de G sse $\{v_i / 0 \leq i \leq k\} \subseteq V(H)$ e $\{a_i / 1 \leq i \leq k\} \subseteq A(h)$.
folhas: conjunto de vértices $\subseteq V(G)$.
subárvore T de G: grafo conexo tal que folhas $\subseteq V(T) \subseteq V(G)$, $A(T) \subseteq A(G)$, $g(u)=1$ u folhas e não existe