AUMENTANDO A CONECTIVIDADE DE PROCESSOS EM REDES DE TRANSPUTERS

SÉRGIO V. CAVALCANTE† E MÁRCIA DE BARROS CORREIA‡

RESUMO

As restrições impostas pela limitação do número de canais inter_processadores das linguagens disponíveis para os transputers [2, 3, 4, 5] dificultam bastante sua utilização. Nossa proposta é eliminar esta limitação. Em particular, vamos abordar a linguagem Occam 2 [1, 6, 10, 11, 13], verificando sua implementação atual para depois apresentar nossa proposta.

ABSTRACT

The limited inter-processors channel number of the available languages for transputers causes general order restrictions. Our proposal is to eliminate this limitation. In particular, we will have a view about the actual implementation of Occam 2 and then we will make our proposal.

---

† Engenheiro Eletricista (UFPE, 1986), Mestre em Informática (UFPE, 1990), Engenheiro Eletricista do Depto. de Física, UFPE. Áreas de interesse: arquitetura de computadores, sistemas distribuídos e linguagens concorrentes. (081) 2710111 - Ramal 215.

‡ Engenheira Eletricista (UFPE, 1971), M. Sc. em Engenharia Elétrica (PUC/RJ, 1976), Dr (INPT-Toulouse, 1983), Profa. Depto. de Informática - UFPE, Recife - PE. Área de interesse: Arquitetura de computadores. (081) 2713052.

1. INTRODUÇÃO

Utilizando memória local e protocolos de comunicação simples, os transputers aumentam consideravelmente o desempenho na comunicação entre processadores, viabilizando sistemas com alto grau de paralelismo. Por outro lado, o número reduzido de links seriais por transputer e as restrições impostas pelo protocolo de comunicação tornam as redes de transputers sistemas pouco flexíveis.

Na alocação de processos em redes de transputers é comum, devido a estas restrições, haver a necessidade de adicionar processos auxiliares de multi e demultiplexação, bem como de roteamento de mensagens. A modificação proposta neste artigo melhora o método de comunicação entre processadores tornando-o mais próximo do modelo de Occam. Os processos auxiliares são eliminados tornando prescindível o conhecimento da topologia da rede pelos programadores. O roteamento ficaria a cargo dos compiladores. Desta forma usuários não especialistas poderão usar redes de transputer com facilidade e naturalidade.

2. COMUNICAÇÃO ENTRE PROCESSOS EM OCCAM

A comunicação de Occam é baseada no modelo de CSP [9] e ocorre através de passagem de mensagem por canais unidirecionais, ponto-a-ponto e de maneira síncrona (*Rendez-vous*). Os processos podem ser executados paralelamente em uma rede de processadores ou num único processador por *time-sharing*.

Em adição aos registradores usados no processamento sequencial (*workspace*, *operand* e *next inst*) e da pilha de avaliação (formada pelos registradores *A*, *B* e *C*), dois outros servem para controlar a lista de escalonamento de processos: o *front* e o *back*. Um processo está **ativo** se está em execução ou na lista de escalonamento. Em contrapartida, os processos esperando pelo rendez-vous ou por um tempo especificado pelo *timer* estão **inativos**. O chaveamento de processos ocorre quando o processo em

execução se torna inativo [4].

Os canais lógicos entre processos alocados no mesmo processador são implementados na memória local. Cada canal utiliza uma palavra de memória. Exceto pelo tamanho da memória, não há limitação quanto ao número de canais.

O primeiro canal que se tornar apto ao rendez-vous terá um apontador para a próxima instrução armazenado no canal. Quando o segundo processo envolvido se torna apto, a mensagem é copiada e o primeiro canal volta a ficar ativo.

A comunicação interprocessadores é feita através dos links seriais que suportam até dois canais lógicos com sentidos de comunicação opostos. Um link é composto por duas interfaces de enlace interligadas por um par de fios. Estas interfaces são dispositivos existentes nos transputers para promover e controlar a comunicação.

As transmissões são respondidas com um sinal de reconhecimento enviado automaticamente pelo receptor, indicando apenas que outro dado já pode ser transmitido. O pacote de comunicação é composto por dois bits iguais a 1, oito bits de dado e mais um bit 0 de parada. O reconhecimento dispõe de dois bits 0 e 1.

Cada interface de enlace dispõe de três registradores: um apontador, um contador dos bytes das mensagens e um apontador para o apontador da próxima instrução do processo que estiver usando o link. Esses registradores são utilizados de forma similar a dos canais intra-processadores [4].

O rendez-vous é garantido pelas interfaces de enlace que só promovem a comunicação se ambas estiverem prontas, ou seja se os processos associados a estas estiverem aptos. O sincronismo na comunicação entre transputers é necessário ao modelo de concorrência assumindo que não dispõe de armazenamento temporário de mensagem.

A questão decorrente desse modo de comunicação é que como cada link só pode ter dois canais lógicos e o número de links por transputers é muito pequeno, há uma grande limitação no número de

canais inter-processadores. Em alguns casos, para solucionar este problema, é necessário adicionar processos auxiliares como multiplexadores ou retransmissores de mensagem. A adição desses processos auxiliares toma tempo do processador, obriga ao usuário ter conhecimento da topologia da rede e implica na perda de sincronismo entre processos, mesmo que alocados em processadores interligados, pois eles funcionam como buffers de mensagem [8].

O objetivo da proposta que passaremos a descrever é aumentar o número de canais lógicos por link . Com isso serão evitados os processos auxiliares. Programas que funcionavam em ambiente uniprocessador passarão a funcionar em redes de transputer sem que haja necessidade de adaptação por parte do usuário.

3 MODIFICAÇÃO PROPOSTA

Primeiramente veremos as modificações na implementação dos canais inter-processadores, visando a multiplexação dos links seriais. A partir daí, começaremos a abordar os problemas e a adaptação da nossa proposta com vistas ao roteamento de mensagens.

Cada canal lógico **inter-processador** é implementado por meio de canais existentes em ambos os processadores através de uma combinação dos métodos utilizados em canais intra-processadores e inter-processadores [4, 7]. Os canais estão associados a identificadores e são compostos de duas palavras de memória.

A primeira palavra implementa o canal propriamente dito sendo iniciada com o valor especial *empty*. Quando um processo se tornar apto para o *rendez-vous*, terá um apontador para a próxima instrução armazenado nesta palavra. A segunda palavra armazena um apontador que é usado para implementar a fila de espera de transmissão ou a de recepção (do tipo FIFO) de cada link. A fila de transmissão (recepção) usa os registradores *prox trans* (*prox rec*) e *ult trans* (*ult rec*) para apontar para o primeiro e para o último canal da fila, respectivamente.

A seguir veremos os passos envolvidos no processo de comunicação examinando um exemplo em que um processo *P* deseja

transmitir para um processo *Q* em outro transputer. Chamemos de *Interface 1* a interface de enlace que irá transmitir e de *Interface 2* a que irá receber (Fig. 1).

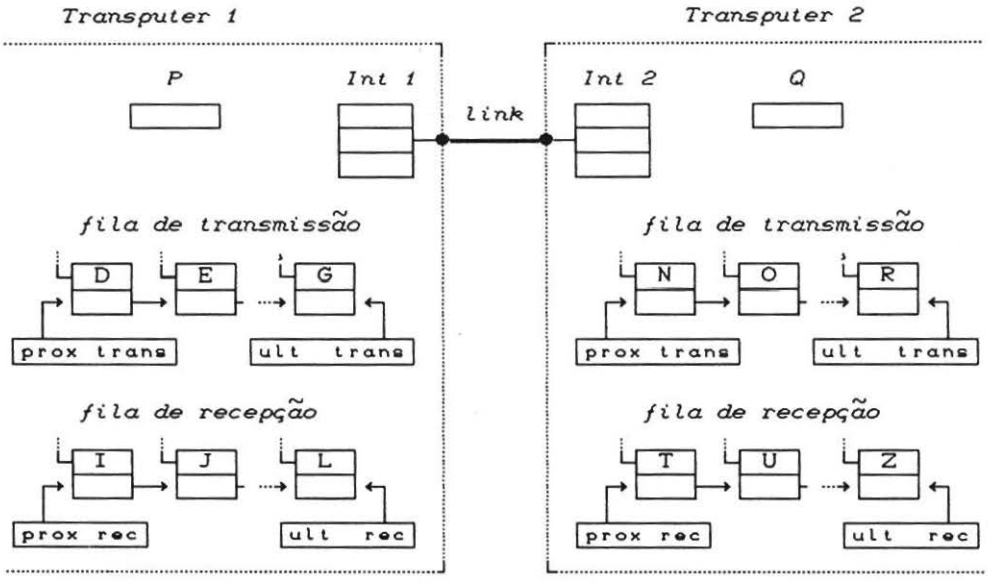

Figura.1. Estado global do sistema na comunicação

1. No início do processo de transmissão, a pilha de avaliação contém os valores necessários à comunicação e aponta para o canal correspondente (Fig 2).

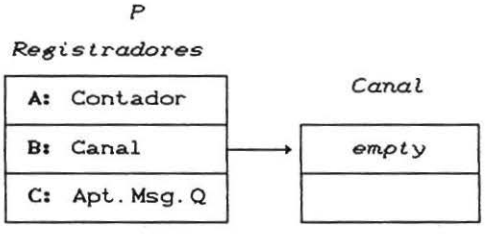

Figura 2. Início do procedimento de transmissão

2. O canal do processo *P*, que continha o valor *empty*, passa a ter um apontador para a próxima instrução do processo. O canal entra na *fila de transmissão* e o processo torna-se inativo (Fig 3).

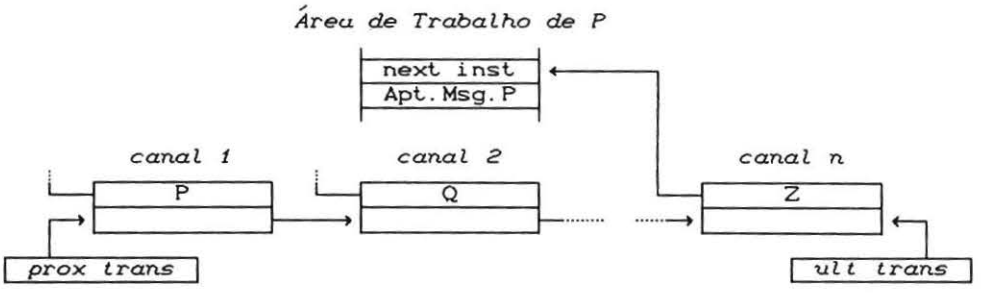

Figura 3

3. A Interface 1 transmite o identificador do processo, informando à Interface 2 que o processo transmissor está apto para o *rendez-vous*.

4. A Interface 2 verifica se o processo receptor também está apto, testando o valor contido no canal correspondente. Se for diferente de *empty*, significa que está apto, e o procedimento continua no passo 5. Se for *empty*, o receptor não está apto e o procedimento continua no passo 6.

5. A Interface 2 envia uma sequência de reconhecimento com valor 100 (binário), informando que o receptor também está apto. A Interface 1 passa a transmitir a mensagem efetivando o *rendez-vous*. O valor do canal em cada transputer volta a ser *empty* e os processos são ativados. Fim do algoritmo.

6. O canal receptor passa a conter um valor *non-empty* (qualquer valor diferente de *empty*), e uma sequência de reconhecimento com valor 101 é enviada, informando que a transmissão da mensagem não pode ser realizada no momento.

6.1 Quando o processo receptor tornar-se apto verificará que o transmissor correspondente já está apto, através do valor *non-empty* contido em seu canal. O canal passa a conter um apontador para a próxima instrução do processo receptor, o

identificador do canal do processo receptor entra na *fila de recepção* e o processo fica inativo. O identificador é transmitido informando à Interface 1 que o receptor também está apto. A Interface 1 transmite a mensagem. Os canais voltam a conter o valor *empty* e os processos são ativados.

4. ASPECTOS DE ARQUITETURA

A nossa proposta implica num aumento dos bits associados ao protocolo de comunicação devido à necessidade de transmitir identificadores. Por isso, optamos por buscar um método, que tornasse o tamanho do identificador adaptável ao número de canais lógicos por link necessários a cada programa. Assim, sendo K o número de canais inter-processadores de um programa, o número n de bits de cada identificador será dado por:

$$2^{n-1} < K \leq 2^n$$

Quando um identificador for transmitido ou recebido, a comunicação será de n bits e na comunicação de dados, será de oito bits. Isto torna-se possível utilizando em cada interface um registrador de deslocamento de tamanho variável.

Devido ao fato do estado da comunicação ser conhecido por ambas as interfaces a qualquer tempo, estas sempre saberão o que estarão recebendo uma da outra (se um identificador ou um dado). Porém, como a comunicação pelo link pode ser uni ou bidirecional [4], a única variável desconhecida por uma interface neste processo, é o fato da outra interface do link transmitir ou não num determinado instante.

Como um canal que não tem mensagem a transmitir, envia sempre o valor 0 durante a comunicação unidirecional, e sabendo que uma transmissão de mensagem começa com dois bits 11 (*start bits*) e o sianl de reconhecimento começa por 10, a interface pode adaptar o registrador de deslocamento **durante** a comunicação. Veja a figura 4. Com a chegada dos bits 11, a porta *nand* mudará o estado do

Flip-Flop (inicialmente Q = 1), habilitando o registrador *num.de.bits.alternativo*, que determina o tamanho do registrador de deslocamento no caso de haver uma recepção de mensagem. Como o tipo de mensagem a receber, dado ou identificador, já é conhecido, o número de bits a receber, contido neste registrador, também o é. *Num.de.bits.transmissão* controla o registrador de deslocamento durante a transmissão de mensagens.

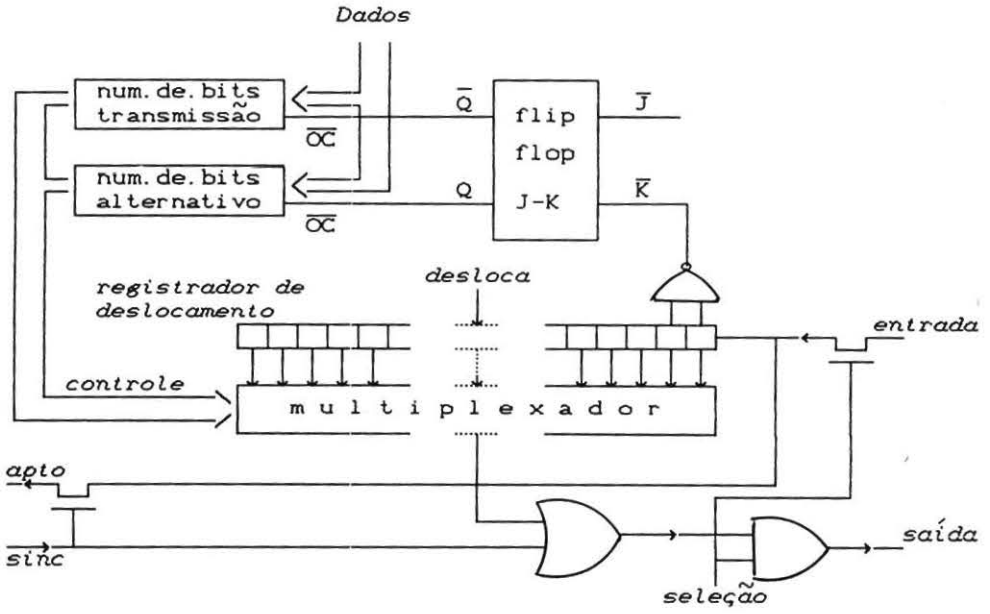

Figura 4. Modificação da interface de enlace

5. RELAÇÃO DO IDENTIFICADOR COM O ENDEREÇAMENTO DO CANAL

Uma relação direta do identificador do canal com o endereço de memória em que o canal está alocado, foi o método encontrado para tornar mais eficiente a comunicação. No momento em que um identificador é recebido, a interface de enlace deve testar se o

canal correspondente está vazio, ou se aponta para a área de trabalho do processo correlacionado. Para isto, a interface deve saber o endereço do canal a partir de seu identificador.

Adaptamos para isto o método de alocação de vetores do compilador de Occam visto que este é simples e rápido no que se refere ao endereçamento dos elementos dos vetores [4]. No nosso casop, os canais relativos a cada link são agrupados em vetores onde o identificador de cada canal é o índice do vetor e cada interface de enlace conhece o endereço base de seu vetor. Assim relacionamos o endereço do canal com seu identificador, como desejado.

6. ELIMINANDO OS PROCESSOS DE ROTEAMENTO

Desejamos que num determinado transputer, que está intermediando a comunicação entre processos em transputers distintos, a mensagem que for recebida por um certo link seja retransmitida por outro link, sem interferência do processador, sendo o controle efetuado pelas interfaces de enlace que são microprogramadas [4].

Daqui por diante, para facilitar a explicação, chamaremos as interfaces envolvidas de *interface receptora* ou *i.r*, e *interface transmissora* ou *i.t* e de *c*, um canal lógico auxiliar, todos contidos no transputer intermediário.

Normalmente, a interface de transmissão/recepção conhece o tamanho da mensagem que será transmitida/recebida por meio de informações fornecidas pelo processo que vai se comunicar [4]. Como no nosso caso o tamanho da mensagem não é conhecido por *i.r*, pois não há um processo receptor associado a esta interface, o pacote de comunicação foi modificado. Um bit foi acrescentado de modo que quando seu valor é 1 o pacote ao qual este pertence é o último da mensagem.

Figura 5. Pacote de comunicação

Para que as interfaces possam controlar esta comunicação, fizemos com que *i.r* receba e guarde a mensagem do canal *c* na memória (no endereço seguinte ao do canal auxiliar) e ponha este canal na fila de transmissão de *i.t*. Assim, é preciso que ao receber a mensagem, *i.r* saiba qual interface deverá retransmití-la, que no nosso caso é *i.t*. De modo a aumentar a eficiência, a identidade da interface transmissora será conhecida por informações contidas no canal lógico auxiliar e fornecidas em tempo de compilação, evitando sobrecarregar a comunicação. Verificamos que acrescentando informação ao valor *non-empty* do algoritmo de comunicação poderíamos passar informações sobre o estado e identidade das interfaces envolvidas, de uma para a outra. Além disso, estas modificações serviram também para informar às interfaces se estão servindo a um processo ou apenas fazendo o roteamento de mensagens, visto que os procedimentos nestes casos, mesmo parecidos, não são idênticos. Assim, necessitamos agora de valores especiais, além do valor *empty*, que identifiquem cada uma das interfaces. Estes valores, que no nosso caso estamos simbolizando por *i.r* e *i.t*, serão usados no lugar do valor *non-empty*, como veremos na seção seguinte.

Introduziremos o novo algoritmo também com um exemplo de comunicação. Como a comunicação, do ponto de vista dos transputers que contém os processos transmissor e receptor não sofreu maiores modificações, iremos analisar a comunicação apenas do ponto de vista do transputer intermediário, ignorando desta forma se a comunicação está sendo realizada com aqueles transputers ou com outros intermediários, uma vez que não há diferença. Identificaremos ambos apenas como *transputer transmissor* e

transputer receptor (fig. 6).

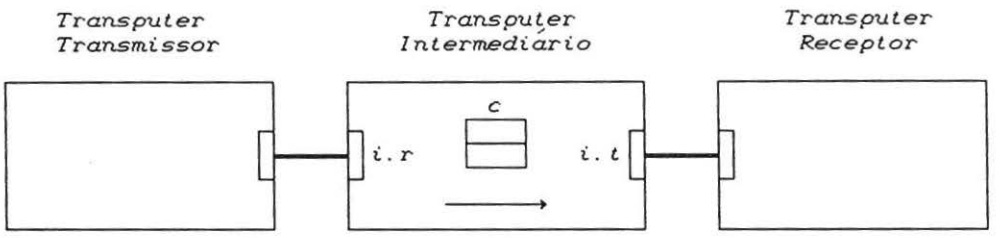

Figura 6

Geralmente os transputers têm quatro interfaces de enlace, cada uma delas com um nome. Como sabemos, durante a comunicação os canais contêm um apontador para a área de trabalho do processo associado. Nos transputers intermediários os canais auxiliares não têm processos associados. Assim, ao ser criado pelo compilador, o canal auxiliar contém o identificador da interface de transmissão (*i.t*, no nosso exemplo). Este valor servirá não só para indicar qual é a interface transmissora, como também para informar às interfaces que o canal em questão é apenas um auxiliar, cabendo a elas o controle do roteamento da mensagem (Fig 7). Lembramos que consideramos como *non-empty*, qualquer valor diferente do valor especial *empty*. Portanto, os identificadores de interface se enquadram neste caso. Por outro lado, é importante não confundir o identificador do *canal*, que é transmitido no início da comunicação com o identificador de *interface de enlace*, que introduzimos agora e não é transmitido.

Canal

Figura 7

7. O ALGORITMO DE ROTEAMENTO

1. Como o canal *c* está com o valor *i.t*, que é do tipo *non-empty*, a recepção do identificador deste canal acarretará imediatamente num *rendez-vous* com o transputer transmissor.

2. Como o canal *c* contém o valor *i.t*, a interface de recepção, põe o canal *c* na fila de transmissão de *i.t* e também põe o valor *empty* no canal *c*.

3. Se, enquanto o canal *c* tiver o valor *empty*, houver tentativa de transmissão de outra mensagem por parte do transputer transmissor, o *rendezvous* não ocorrerá. O procedimento será semelhante àquele do passo 4 e 6 (não continuando nos passos seguintes) do algoritmo da seção 3. Porém, não é qualquer valor *non-empty* que é posto no canal *c* e sim o valor do identificador da interface de recepção *i.r*.

4. Como o canal *c* está na fila de transmissão de *i.t*, seguimos os procedimentos normais de transmissão, vistos nos passo 3 do algoritmo da seção 3, e realizamos o *rendezvous* com o transputer receptor.

5. A interface *i.t* verifica o valor do canal *c*:

a) se for *empty*, repõe seu identificador *i.t* no canal e volta para **1**;

b) se for $i.r$, repõe seu identificador $i.t$ no canal c, põe este canal na fila de recepção da interface $i.r$, que recebe nova mensagem do transputer transmissor e volta para o passo 2.

8. O IDENTIFICADOR DE CANAL: ADAPTAÇÃO AO ROTEAMENTO

Pudemos perceber na seção anterior, que duas interfaces de enlace diferentes no mesmo transputer acessam o mesmo canal. Cada interface dispõe de um endereço base de vetor que, juntamente com o identificador do canal, forma o endereço do mesmo na memória. Assim, para resolver o problema, basta que os endereços base de duas interfaces usadas na retransmissão de mensagens, sejam escolhidos de modo a que seus vetores de canais se sobreponham em algum trecho de memória. Em alguns casos, torna-se necessária a sobreposição total de um vetor sobre o outro. Por exemplo, se três interfaces ($i.1$, $i.2$ e $i.3$) têm, duas a duas, canais auxiliares, a sobreposição da base do vetor de uma delas sobre pelo menos uma das outras, é inevitável.

9. CONCLUSÃO

Ao promover a configuração dos vários processos de um programa, o usuário de transputers necessitava conhecer aspectos do sistema que o faziam chegar aos detalhes de conectividade do mesmo[1] [7, 8]. Em muitos casos, a inserção de processos auxiliares, como multi e demultiplexadores, e roteadores era inevitável. A legibilidade e entendimento dos programas diminuia bastante. Para multiplicar o número de processos de um programa, não bastava modificar parâmetros de configuração, mas também os programas auxiliares.

---

[1] Isto é valido para quaisquer das linguagens disponíveis para Transputers, já que a comunicacao entre processos e sempre feita atraves dos canais de Occam.

A perda na eficiência de comunicação, com a implementação em hardware desta modificação, é compensada pelos benefícios trazidos por ela. Ademais, é bom salientar que a simulação destes canais em processos auxiliares, além de não estar condizente com o modelo de Occam, tomava tempo do processador. Este tempo, provavelmente é maior que o tempo da comunicação realizada com a modificação, porque aqui utilizamos o hardware e a interface de enlace, que é independente, enquanto que no caso anterior usávamos software e o processador. Ainda neste ponto, em alguns casos os processos auxiliares ficavam inativos bloqueando toda a comunicação pelo link. Assim, acreditamos que a eficiência do conjunto não será muito afetada, podendo inclusive melhorar em certos casos.

O aumento do grau de abstração do usuário em relação à topologia da rede, também é um fator importante. Usuários menos especialistas podem passar a usar os sistemas de transputers, aumentando a popularidade dos mesmos.

10. BIBLIOGRAFIA

[1] INMOS Ltd.: *Occam 2 Reference Manual*; Prentice-Hall, 1988.

[2] INMOS Ltd.: *Transputer Reference Manual*; Prentice-Hall, 1988.

[3] INMOS Ltd.: *The Transputer*; IEEE Proceedings, 1985.

[4] INMOS Ltd.: *The Transputer Applications Notebook: Architecture and Software*; Redwood Burn Limited, 1989.

[5] INMOS Ltd.: *The Transputer Family*; Inmos Ltd., 1986.

[6] Burns, A.: *Programming in Occam 2*; Addison-Wesley Publishing Company, 1987, 189pp.

[7] Cavalcante, S. V.: *Aumentando a Conectividade de Processos em Redes de Transputers*; Dissertação de Mestrado, DI/UFPE, Recife 1990, 117pp.

[8] Demiralp, S. V.: *Parallel Processing with Occam and Transputer*; Parallel Processing Group, Computing Laboratory, University of Kent, Canterbury, Setembro 1988.

[9] Hoare, C. A. R.: *Communicating Sequential Processes*; Communicating of the ACM, Vol. 21, No. 8, Agosto 1978, (pp. 666-667).

[10] Kerridge, J.: *Occam Programming: a practical approach*; Blackwell Scientific Publications, 1987, 162pp.

[11] May,D. e Taylor,R.: *Occam*; Artigo submetido à International Conference on Parallel Processing, 1983.

[12] Welch,P.H.: *Emulating Digital Logic Using Transputer Networks*; Computing Laboratory, University of Kent, Canterbury.

[13] Welch,P.H.: *Occam 2 and Transputer Engineering*; impresso por UKC Computing Laboratory, 393pp.