Maria Filomena dos Santos Sustelo

SIMULADOR EM MATLAB DE ALGORITMOS PARALELOS EM ARQUITECTURAS HOMOGÉNEAS E HETEROGÉNEAS

Universidade do Algarve, 1998

SIMULADOR EM MATLAB DE ALGORITMOS PARALELOS EM ARQUITECTURAS HOMOGÉNEAS E HETEROGÉNEAS

Unidade de Ciências Exactas e Humanas

da

Universidade do Algarve

Campus de Gambelas - Faro - Portugal

Maria Filomena dos Santos Sustelo

Dezembro de 1998

SIMULADOR EM MATLAB
DE ALGORITMOS PARALELOS EM
ARQUITECTURAS HOMOGÉNEAS E HETEROGÉNEAS

Tese de dissertação de Mestrado realizada sob a orientação do Professor Doutor António Eduardo Barros Ruano, para obtenção do grau de Mestre em Engenharia de Sistemas e Computação – Àrea de Computação, requerida pela Mestranda Maria Filomena dos Santos Sustelo

Campus de Gambelas, Dezembro de 1998

À minha filha Sara

DECLARAÇÃO DE ORIGINALIDADE

Em conformidade com o disposto na alínea b), nº 2 do artigo 5º do Decreto de Lei nº 216/92 de 13 de Outubro, Maria Filomena dos Santos Sustelo declara que todo o trabalho desenvolvido nesta tese de dissertação de mestrado, é de sua autoria.

A Mestranda

(Maria Filomena dos Santos Sustelo)

O Orientador

(Professor Doutor António Eduardo Barros Ruano)

AGRADECIMENTOS

Queria começar por agradecer ao Professor António Ruano o interesse e incentivo que sempre dispensou à realização deste trabalho. Para além das suas sugestões a nível da implementação algorítmica, conteúdo e organização desta tese de mestrado em engenharia de sistemas e computação.

Agradeço também ao Helder Daniel por toda a sua colaboração e apoio técnico dispensado. Este trabalho não seria possível sem a utilização do computador do laboratório de controlo automático do sector de electrónica e computação desta universidade, onde se encontram instaladas plataformas paralelas reais que, desde o início, foi colocado à minha inteira disposição.

Gostava ainda de agradecer aos colegas do sector, particularmente à Professora Doutora Graça Ruano, Margarida Madeira, Isabel Leiria pela amizade e apoio em todos os momentos.

Este trabalho não poderia ter sido realizado sem a infinita paciência dos meus pais, António Martins Sustelo em particular da minha mãe, Maria Antónia dos Santos Sustelo, e do meu marido Jorge.

RESUMO

As exigências crescentes no desempenho de algoritmos para satisfação de requisitos em tempo real, nas áreas de controlo e processamento de sinal, levou à necessidade de desenvolvimento acelerado dos processadores de diferentes tipos, tais como: *digital signal processors*, *vector processors*, *CISC* e *RISC*, com características cada vez mais sofisticadas no que respeita a novas funções e possibilidades, bem como à utilização simultânea de vários processadores, quer de um tipo único, ou de diferentes tipos, em plataformas de processamento paralelo.

No desenvolvimento de aplicações em plataformas paralelas, os engenheiros de áreas de controlo e processamento de sinal, têm por hábito iniciar a implementação de algoritmos sequenciais em *Matlab*, só mais tarde surgindo o problema da implementação paralela do algoritmo. Esta última fase é a que normalmente consome mais tempo.

Ao longo desta tese de dissertação de mestrado em Engenharia de Sistemas e Computação, será apresentado o desenvolvimento de uma ferramenta de *software*, concebida para auxiliar o programador na fase da paralelização do algoritmo em arquitecturas homogéneas e heterogéneas.

Poderá constatar-se que o conjunto de funções existentes na toolbox, proporciona ao seu utilizador de uma forma amigável, a possibilidade de comparar, sem esforço, o desempenho de soluções paralelas alternativas, além de possibilitar a monitorização da execução dos vários processos em cada elemento de processamento. Todas estas facilidades são fornecidas não saindo do ambiente Matlab.

Este simulador servirá para futuros trabalhos de investigação na área, podendo a sua aplicabilidade ser ampliada para outros tipos de processadores existentes ou que possam surgir com novas características e potencialidades.

ABSTRACT

The growing demands in the acting of algorithms for satisfaction of requirements in real time, in the areas of control and signal processing, took to the need of accelerated development of the processors of different types, such as: digital signal processors, vector processors, CISC and RISC, with characteristics more and more sophisticated in what it respects to new functions and possibilities, as well as to the simultaneous use of several processors, of an only type, or of different types, in platforms of parallel processing.

In the development of applications in parallel platforms, the engineers of areas of control and signal processing, they have for habit to begin the implementation of sequential algorithms in Matlab, only later appearing the problem of the parallel implementation of the algorithm. This last phase is the one that it usually consumes more time.

Along this thesis of mestrado dissertation in Engineering of Systems and Computation, the development of a software tool will be presented, conceived to aid the programmer in the phase of parallelization of the algorithm in homogeneous and heterogeneous architectures.

It can be verified that the group of existent functions in the toolbox, provides to the user in a friendly way, the possibility to compare, effortlessly, the acting of alternative parallel solutions, besides facilitating the monitoring of the execution of the several processes in each processing element. All these means are supplied not leaving the Matlab environment.

This simulator will be good for future investigation works in the area, being able to not its applicability to be enlarged for another types of existent processors or that can appear with new characteristics and potentialities.

ÍNDICE

1 INTRODUÇÃO

1.1 Enquadramento do Tema

Nas áreas de controlo e de processamento de sinal, as especificações de desempenho das modernas aplicações de tempo real são cada vez mais exigentes, requerendo normalmente a execução de algoritmos computacionalmente mais pesados em períodos de amostragem mais reduzidos. A satisfação destas especificações traduz-se assim na necessidade de executar um maior número de operações por unidade de tempo. Tal necessidade conduziu ao desenvolvimento acelerado de processadores de diferentes tipos, tais como: processadores digitais de sinal, processadores vectoriais, *CISC* e *RISC*, com características cada vez mais sofisticadas, nomeadamente no que diz respeito a:

- velocidade de processamento;
- capacidade de processamento;
- capacidade de comunicação;
- capacidade de controlo.

Independentemente das crescentes capacidades dos processadores actuais, por várias razões, nomeadamente pela relação custo/desempenho apresentada, aplicabilidade ao problema específico, propriedade de tolerância a faltas, etc., o interesse na utilização de plataformas homogéneas de processamento paralelo para aplicações de tempo real é uma realidade indiscutível. O maior ênfase da investigação actual reside, todavia, na utilização de plataformas heterogéneas de processamento paralelo, dado possibilitarem uma melhor adequação entre as necessidades de hardware e os recursos disponíveis, conduzindo assim a uma melhor eficiência dos recursos computacionais existentes.

1.2 Motivação para a Concepção do Simulador

Apesar de existirem na literatura especializada vários trabalhos no âmbito do escalonamento e mapeamento de algoritmos em arquitecturas homogéneas, não existe, ainda, uma metodologia disponível adequada ao programador. A complexidade dos sistemas heterogéneos torna este problema ainda mais interessante [1], e, infelizmente, bastante mais complicado. A determinação da melhor topologia de rede para um algoritmo particular é outro tópico de difícil resolução. A inexistência de ferramentas de software que possam resolver eficazmente os problemas acima apresentados conduz obviamente o programador para uma experimentação, nunca exaustiva, das possíveis soluções. Obviamente esta estratégia (a experimentação possível) é limitada, primeiramente, à existencia física e disponibilidade do hardware a testar; em segundo lugar, o número de soluções paralelas possíveis de serem testadas é sempre pequeno dado que, na prática, todo os passos envolvidos na paralelização de um algoritmo sequencial tornam o processo moroso.

Assim sendo, torna-se necessário desenvolver novas ferramentas para resolver, ou ajudar a resolver, os problemas acima referidos [2], e esta é uma necessidade de índole geral. Concentrando-nos agora nas áreas de aplicação em causa, sistemas de controlo e processamento de sinal, seria de todo o interesse que as ferramentas acima descritas se integrassem facilmente com os ambientes de programação normalmente utilizados nestas áreas, com especial destaque para o *Matlab*.

1.2.1 Utilização do Ambiente *Matlab*

Nas áreas de controlo e de processamento de sinal o *Matlab* tem vindo a tornar-se de facto um standard para análise e projecto de algoritmos. A designação *MATLAB* [3] advém do termo inglês *MATrix LABoratory*. Trata-se de um ambiente de computação matricial com elevado desempenho. O seu elemento básico de informação é a matriz, que se define, sem necessidade

do recurso a informação sobre o seu dimensionamento. Esta característica, permite a qualquer utilizador a resolução rápida das suas aplicações, sem que seja necessário recorrer a código de baixo nível. O *Matlab* dispõe de uma família ou colecção de funções designadas por *Toolboxes,* para apoio a áreas específicas, tais como, processamento de sinal, análise e projecto de sistemas de controlo, além de outras. Estas podem ser facilmente aplicadas e estendidas através da criação de ficheiros de texto, constituídos por programas em linguagem *Matlab*.

Nas áreas de controlo e processamento de sinal, o engenheiro, ao desenvolver aplicações paralelas tem por hábito iniciar a sua implementação com o *Matlab* [4] e as suas *toolboxes* para desenvolver e testar os seus algoritmos sequenciais. É apenas depois, quando o algoritmo sequencial está finalizado, que surge o problema da paralelização.

1.2.2 O Problema da Paralelização

É habitual passar pelas seguintes etapas no desenvolvimento de uma aplicação paralela:

- considerar a arquitectura de destino, quer se trate de um sistema homogéneo ou heterogéneo;
- particionar o algoritmo;
- determinar a melhor topologia de rede;
- codificar o algoritmo na respectiva linguagem paralela da arquitectura de destino;
- e finalmente, para avaliar o desempenho de execução do algoritmo paralelo, medir o tempo de execução paralelo, ou outros factores.

Na maior parte dos casos, o problema da paralelização é um processo iterativo, com vários ciclos a ocorrerem entre estas cinco fases. Todo este processo consome um tempo considerável.

Por outro lado, para um significativo número de casos, soluções paralelas que parecem prometer um bom desempenho, atingem fracos resultados em determinadas plataformas, quer em termos de eficiência, tempo de execução, ou ambas. São várias as razões que podem originar o fraco desempenho obtido, diferindo obviamente de problema para problema. Existe no entanto um factor comum a todas elas: é difícil, num programa paralelo, identificar as razões da ineficiência, as quais estão claramente relacionadas com a dificuldade de monitorar a execução paralela do programa.

1.3 Objectivos Propostos

Tendo em conta o acima referido, o trabalho que irá ser apresentado nesta tese de Mestrado visa a implementação de um simulador em *Matlab*, de algoritmos paralelos em arquitecturas homogéneas e heterogéneas.

Mais especificamente, tendo como base um simulador existente de arquitecturas homogéneas paralelas em *transputers*, neste trabalho pretende-se:

- melhorar a geração de código automático de funções em *Matlab* que implementam os processos simulados;
- estender a utilização do simulador para sistemas homogéneos, com vários processadores do tipo *C40s Texas DSP*;
- alargar a utilização do simulador a sistemas heterogéneos, compostos por *Inmos Transputers* e *C40s Texas DSP*;
- Validar o simulador resultante dos pontos anteriores em problemas teste.

1.4 Organização da Tese

Esta tese está organizada numa estrutura de 6 capítulos.

No presente capítulo (Capítulo 1) é feita uma abordagem sobre os objectivos propostos para este trabalho de mestrado, o enquadramento do tema na actualidade, apresentadas as razões e motivações da concepção do simulador, e por fim um resumo dos vários assuntos tratados em cada capítulo.

Com a disponibilização de uma *Toolbox* [4] concebida em *Matlab* houve necessidade de realizar todo um estudo prévio antes da concepção de novas funções e potencialidades para a nova versão do simulador. Neste contexto surge o Capítulo 2, que visa o levantamento a referências bibliográficas de assuntos relacionados com o tema e o Capítulo 3 onde consta a recolha de informação resultante da análise ao *software* disponibilizado.

Como resultado da análise realizada nos Capítulos 2 e 3 e do estudo dos objectivos inicialmente propostos incluídos no Capítulo 1, é apresentada no Capítulo 4 uma abordagem detalhada sobre a nova versão do simulador, designado por *Secsim,* nos seus diferentes módulos, seu encadeamento e funcionamento, e onde se descrevem as novas funções e potencialidades adicionadas.

Para validação do simulador, e para apresentação das suas potencialidades são apresentados, no Capítulo 5, casos de aplicações simuladas na nova versão do simulador de arquitecturas paralelas homogéneas e heterogéneas.

Esta tese termina no capítulo 6, onde se apresentam as conclusões e comentários finais, bem como perspectivas de trabalho futuro.

2 REVISÃO BIBLIOGRÁFICA

2.1 Introdução

Tendo por base uma *Toolbox* [4] existente desenvolvida em *Matlab*, e antes da concepção de uma nova versão do simulador aí incluído, houve necessidade de realizar leituras a referências bibliográficas para actualização do conhecimento na área de processamento paralelo.

Este Capítulo visa sumariar os conceitos mais importantes necessários para o desenvolvimento do trabalho proposto. Assim, a informação contida nesta capítulo, é o resultado de um conjunto de elementos recolhidos de referencias bibliográficas com o fim de expor matérias relacionadas com o tema desta tese e resolver cada um dos objectivos propostos.

Os vários assuntos encontram-se organizados na sequência que a seguir se apresenta:

- relação entre as entidades lógicas (os processos) e físicas (os processadores) presentes no sistema a simular, e noções para concepção de aplicações paralelas;
- formulação de medidas de desempenho para aplicações sequenciais e paralelas;
- descrição do *hardware* e *software* de processadores tais como, *Inmos Transputers*, *C40s Texas DSP;*
- arquitecturas e detalhes de topologias homogéneas e heterogéneas.

2.2 Conceitos importantes

2.2.1 Processador, processo, mecanismos de comunicação e sincronização de processos

Os conceitos expostos neste parágrafo, relacionam-se com a existência de um ou mais processadores no sistema, em que é manifesta a necessidade de controlar a atribuição de processos aos processadores existentes.

Processador, também chamado de elemento de processamento, é um dispositivo de *hardware*, enquanto que **processo** é uma unidade de *software*, por vezes referenciado em alguma bibliografia por tarefa. Um processo pode também ser definido como uma instância de um programa em execução ou uma entidade activa no sistema que necessita de um elemento de processamento para executar. Contudo não se pode colocar um processo em vários processadores, sendo apenas possível replicar um processo por vários processadores.

Existem características sobre os processos, as quais foram recordadas e se apresentam a seguir:

- de granularidade grossa (*coarse granularity*), onde o tempo de execução é longo comparado com o tempo de arranque (*setup*) e os tempos de comunicação; e de granularidade fina (*fine granularity*) no qual o tempo de execução é pequeno comparado com o tempo de arranque e os tempos de comunicação.
- leves, quando o arranque e o tempo de execução se processam de forma rápida; e pesados quando o arranque e o tempo de execução se processam de forma lenta.
- de alta prioridade, apenas quando executam essencialmente funções de comunicação; e de baixa prioridade quando se processam cálculos.

Dependendo da situação ou do estado em que os processos se encontram num determinado momento, existem termos que os referenciam:

- pronto, quando o processo está pronto para executar, a aguardar em fila de espera pelo acesso ao processador durante a sua porção de tempo (*time_slice*);
- a executar, significa que o processo está actualmente em execução no processador;
- bloqueado, quando o processo está a aguardar que seja completada qualquer comunicação, sincronização ou acesso a um recurso do sistema.

A figura seguinte ilustra um diagrama de transição de estados de processos:

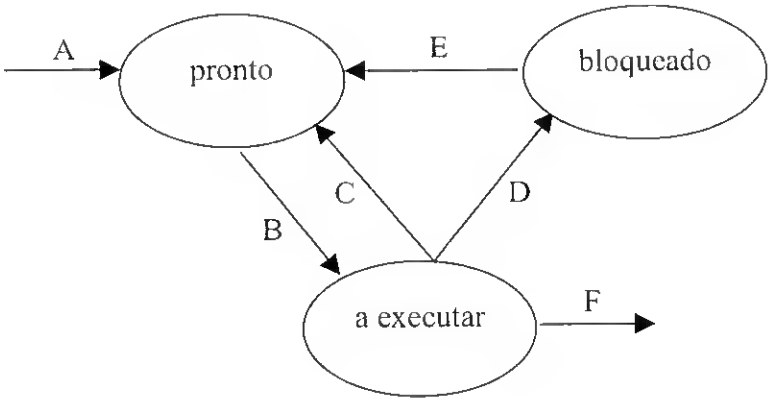


Fig. 1 - Diagrama de Transição de Estados dos Processos

A) encaminhamento de um novo processo para memória;
B) processo seleccionado para execução, por se tratar do mais prioritário relativamente a outros;
C) processo retirado do processador pela existência de outro com maior prioridade (preempção);
D) passagem do processo para o estado de bloqueado para que seja completada qualquer comunicação, sincronização ou acesso a um recurso do sistema;
E) comutação do processo para o estado de pronto a executar, após ter sido completada qualquer comunicação, sincronização ou acesso a um recurso do sistema;
F) fim de execução do processo.

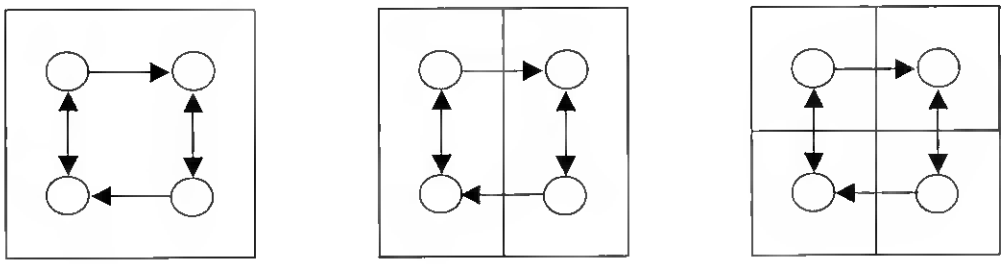
A programação concorrente, e, nomeadamente, o processamento paralelo, introduz dois problemas, que se relacionam com necessidades de comunicação entre processos e de sincronização. Um exemplo concreto desta situação, é quando dois processos têm necessidade num determinado instante, de utilizarem a mesma zona de memória.

No modelo de programação adoptado, assume-se que na **comunicação e sincronização de processos** a troca de dados entre dois processos só deva ocorrer quando ambos estão prontos. No entanto há a lembrar que, os processos podem estar em execução assíncrona, em diferentes processadores.

Decisões sobre a atribuição dos processos aos processadores disponíveis (*mapping*), são influenciadas por vários factores como sejam, a carga do processador, o volume de dados na comunicação entre processos e o tamanho da memória do processador.

Para a implementação de uma solução paralela, deve-se modularizar o problema de modo a criar mais processos lógicos do que processadores físicos. Os processos lógicos podem assim, ser alocados a processadores físicos.

As figuras a seguir exemplificam o mapeamento de processos, se forem considerados 4 processos e 1, 2, ou 4 processadores:



a) **b)** **c)**

Fig. 2 – Mapeamento de processos

Na figura 2a), é apresentado um sistema com 1 único processador e 4 processos em execução paralela. A comunicação entre os processos estabelece-se a nível interno do processador, através de canais lógicos. Existem neste caso 2 canais lógicos unidireccionais e 2 canais lógicos bidireccionais.

Na figura 2b), é apresentado um sistema de 2 processadores com 2 processos cada, em execução paralela. A comunicação entre os processos estabelece-se a dois níveis:

- interno, através da existência de 1 canal lógico bidireccional em cada processador;
- externo, através da existência de 2 canais físicos unidireccionais ou dependendo do tipo de processador, de apenas 1 canal físico bidireccional entre os processadores.

Na figura 2c), é apresentado um sistema de 4 processadores com apenas 1 processo cada, em execução paralela. A comunicação entre os processos estabelece-se somente a nível externo, através da existência de 4 canais físicos, sendo 2 bidireccionais e 2 unidireccionais.

2.2.2 Concepção paralela de algoritmos

São várias as áreas de investigação que aplicam assuntos relacionados com a concepção paralela de algoritmos.

Nenhum problema ou programa possui uma única correspondência de estruturas paralelas. Idealmente, a ideia da concepção paralela de algoritmos consiste na divisão do problema em processos paralelos, que incluam quantidades semelhantes de trabalho a ser executado por cada processador (*load_balancing*). É também desejado evitar custos de comunicação excessivos entre processadores.

Antes de se iniciar a implementação de qualquer algoritmo paralelo, convém identificar qual o tipo de paralelismo incorporado dentro do programa. Existem algumas aproximações gerais para a concepção de algoritmos paralelos, as quais se descrevem a seguir:

- Tarefas independentes, em que cada processo está a executar o mesmo programa isolado de outros processos. O exemplo genérico desta aproximação é o *process_farm*, que consiste basicamente na existência de um processo que controla a distribuição das tarefas, o *farmer* e um número de processos escravos, os *workers*. O *farmer* distribui as tarefas aos *workers*, que são os que na realidade realizam o trabalho, e no final recolhe os resultados. Cada *worker* é logicamente independente de todos os seus vizinhos.

Desta forma, a estrutura lógica de processos é uma estrela, conforme se vê na figura que se segue:

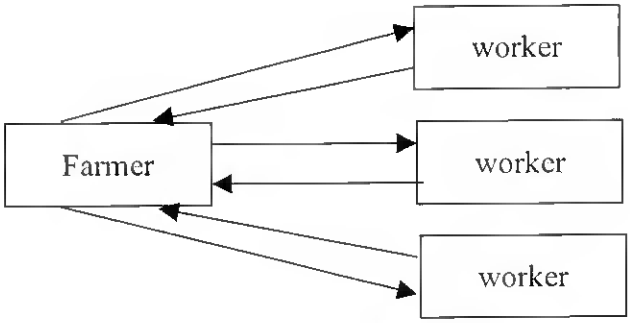


Fig. 3 - *Process Farm* em estrela

Nesta configuração, o *farmer* requere n pares de canais de entrada e saída, dependentes do número de *workers*.

Pode-se no entanto também realizar uma estrutura prática, organizando os *workers* numa matriz linear:

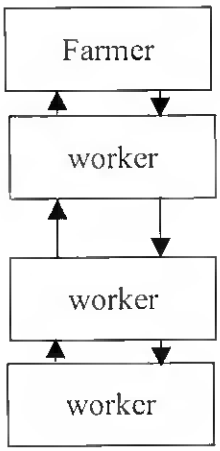


Fig. 4 - *Process Farm* em *Pipeline*

Nesta figura, cada *worker* inclui um ou mais processos responsáveis pela comunicação. Estes processos emitem instruções e dados ao longo da matriz, a partir do *farmer*, por fim devolvem o resultado ao *farmer*.

- Paralelismo geométrico, no qual cada processo executa o mesmo programa em dados correspondentes a uma sub-região do sistema, sendo a comunicação estabelecida entre vizinhanças. Este tipo de paralelismo, é mais utilizado em outras áreas, que não são objecto de estudo para esta abordagem. É o caso, por exemplo, de processamento de imagem.

- Paralelismo algorítmico, onde cada processo é responsável por parte do algoritmo, e de todos os dados que passam através de cada processo. O exemplo genérico desta aproximação é o *pipeline* ou, em dimensões superiores, o *systolic array.*

Um *systolic array* é uma construção em rede de processos, cada um com funções especiais, através da qual todos os dados fluem. Existe essencialmente um processo *master* que actua como controlador do resto da matriz de processos.

É apresentado na figura que se segue, um esquema em *systolic array,* que permite fluxos multi-dimensionais e multi-direccionais incluindo retorno de resposta:

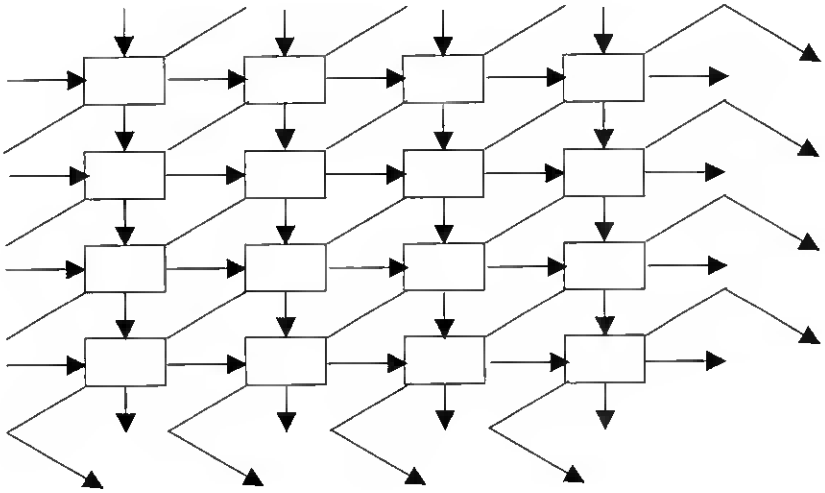


Fig. 5 - *Systolic Array*

Caracteriza-se essencialmente por ser uma construção que combina outros sistemas, desde SIMD (*Single Instructions Multiple Data*), MIMD (*Multiple Instructions Multiple Data*), e *pipeline* em todas as direcções, sendo os cálculos conduzidos por meio de um relógio central.

Um *pipeline* é um *systolic array* mas apenas a uma dimensão, com um processo *master* que controla a execução, e uma cadeia de processos, onde cada um transfere como entrada a saída do seu antecessor. No inicio da cadeia, um processo, o *pipe head* alimenta dados para o *pipeline* com instruções do processo de controlo. No outro extremo da cadeia, outro processo,

o *pipe tail* recebe a saída do *pipeline*, e executa processamento auxiliar. O processo da cauda é o que envia o resultado final ao processo de controlo.

2.3 Medidas de desempenho

Considerações matemáticas sobre medidas de desempenho de sistemas paralelos foram desenvolvidas por Amdahl [8] em meados de 1967, tendo-se tornado conhecidas por lei de *Amdahl*. Duas dessas grandezas referidas em [9] são o *speedup* e a eficiência. O *speedup* identifica quão mais rápido um algoritmo pode ser executado em N processadores, em vez de em apenas um processador; a eficiência determina a utilização média dos N processadores. Os resultados obtidos dos cálculos provenientes de medidas de desempenho, dependem de uma aplicação paralela específica, não sendo generalizados para outros algoritmos, mesmo que semelhantes.

Os conceitos formulados sobre o *speedup*, eficiência e suas variantes são ainda hoje bastante aplicados em áreas de processamento de sinal e de controlo, entre outras, para análises comparativas da execução dos diferentes algoritmos paralelos em plataformas paralelas homogéneas, relativamente aos sequenciais. No entanto tem-se demonstrado em investigações mais recentes que a aplicabilidade dessas medidas de desempenho (na sua forma original) em arquitecturas heterogéneas apresenta algumas limitações.

Actualmente já se utilizam outras formas para o cálculo do desempenho de arquitecturas paralelas heterogéneas, por ter sido comprovado em estudos (ver em [10]) que devido á diversidade de capacidades de computação dos sistemas heterogéneos as tradicionais medidas não são as mais adequadas.

2.3.1 *Speedup*

O *speedup* de uma aplicação paralela incorporada num arquitectura homogénea relaciona-se apenas com o desempenho de um processador específico do sistema. Esta grandeza define-se como a relação entre o tempo de execução do(s) processo(s) atribuído(s) a um determinado conjunto de processadores e o tempo de execução num processador de referência. O *speedup* assim descrito designa-se por *speedup* de carga fixa e depende essencialmente do tamanho do(s) processo(s). Por outro lado também se pode calcular esta medida de desempenho com base num tempo fixo, ou na capacidade da memória. O *speedup* de tempo fixo, no qual o tempo de execução se mantém, destaca quantos mais processos podem ser executados em paralelo. Para o cálculo do *speedup* baseado na capacidade de memória considera-se como limitação a capacidade de memória ligada a processos com grande granularidade, permitindo o aumento linear da memória com base no numero de processadores.

A forma simplificada do *Speedup* proposta pela lei de Amdahl permite apenas graus de paralelização relativos a partes do programa em 1 ou N processadores, não sendo aplicada a valores intermédios.

São apresentadas nos pontos a seguir, definições para programas de tamanho fixo [9]:

- O número máximo de processadores que podem ser usados em paralelo durante o tempo de execução de um determinado algoritmo é dado pelo identificador Pc, referente ao máximo grau de paralelização;
- O tempo de execução do algoritmo, com o máximo grau de paralelização, $P_C \geq k$, num sistema com k processadores é dado pelo identificador T_k;
- O número de processadores do sistema paralelo é dado pelo identificador N;
- Por fim, a parte sequencial de um programa que é dada pelo identificador f, refere-se à percentagem de operações do programa que não pode ser executada em paralelo nos processadores, e que portanto necessita de ser executada sequencialmente.

A equação que corresponde ao factor speedup em N processadores, proposta por Amdahl, é definida da seguinte forma:

$$S_N = \frac{T_1}{T_N} = \frac{N}{1 + f*(N-1)}, \quad (1)$$

sendo T_N o tempo de execução em paralelo num sistema com N processadores:

$$T_N = f*T_1 + (1-f)*\frac{T_1}{T_N} \quad (2)$$

Dado que $0 \leq f \leq 1$, $1 \leq S_N \leq N$, isto é, o *speedup* nunca pode ser superior ao número de processadores no sistema.

Atendendo a que numa arquitectura paralela homogénea se considera um único tipo de processador como nó de referência, esse processador de referência numa arquitectura paralela heterogénea representaria as características de todos os processadores da rede, o que é uma irrealidade. Numa tentativa de ultrapassar esta limitação é argumentado em [1] que uma arquitectura homogénea pode ser considerada como uma subclasse de arquitecturas heterogéneas. Deste modo definiu-se o *speedup* de uma arquitectura heterogénea como uma relação do tempo de execução sequencial mínimo entre os processadores e o tempo de execução paralelo da arquitectura, sendo mais uma vez considerado o melhor processador da arquitectura como nó de referência. A eficiência da arquitectura é também definida em conformidade.

Num artigo publicado recentemente [10] tal nó de referência é identificado por um proposto conceito de processador virtual, o qual assegura que as capacidades dos processadores são exploradas pela maximização da eficiência da arquitectura. São formulados conceitos de *speedup* e eficiência para processamento sequencial e paralelo, de algoritmos em plataformas heterogéneas e testada a aplicabilidade dos mesmos através da atribuição de processos aos processadores para atingir a máxima eficiência.

2.3.2 Graus de eficiência

Outra medida útil de avaliação do desempenho de um sistema paralelo também em parte relacionada com a granularidade do mesmo é a eficiência, expressa em percentagem. Esta medida proporciona a indicação da utilização média dos N processadores existentes, além de permitir uma comparação dos vários *speedups* (com diferentes processadores) obtidos.

Amdahl [8] definiu eficiência como uma relação entre o *speedup* atingido e o *speedup* máximo:

$$E_N = \frac{S_N}{N} \tag{3}$$

Consequentemente, a eficiência está limitada à gama $0 \leq E_N \leq 1$.

2.4 Tipo de Processadores a Simular

2.4.1 Descrição do *hardware* do *Transputer* e do Processador Digital de Sinal *TMS230C40*

A designação *TRANSPUTER* é o acrónimo de *TRANSistor comPUTER*. Trata-se de um elemento de processamento tipo VLSI (*very large scale integration*), concebido para facilitar a paralelização de algoritmos com grande desempenho [6].

A família de *transputers* consiste em vários tipos de dispositivos VLSI, incluindo os de 16 bits: T212, T225; os de 32 bits: T414, T425; e os de 32 bits com virgula flutuante: T800, T805 além dos processadores T9000.

São apresentadas a seguir apenas algumas das características do *transputer* T805, importantes para o desenvolvimento do simulador.

O *transputer* T805 pertence à categoria de processadores RISC de 32 bits, com 25 MHz de velocidade de relógio. Para a comunicação entre pares de dispositivos da família incorpora habitualmente, quatro pares de portas de ligação série, bidireccionais e full_duplex,

permitindo taxas de transmissão síncronas de 20 Mbits/seg entre outros elementos de processamento. Desta forma, este processador supera os engarrafamentos da maquina clássica de *Von Neumann,* os quais, são encontrados em sistemas baseados em barramento permitindo igualmente acrescentar sem quaisquer problemas outros *transputers* ao sistema.

É apresentada na figura a seguir possíveis ligações físicas entre 4 *transputers* e respectiva identificação dos canais existentes:

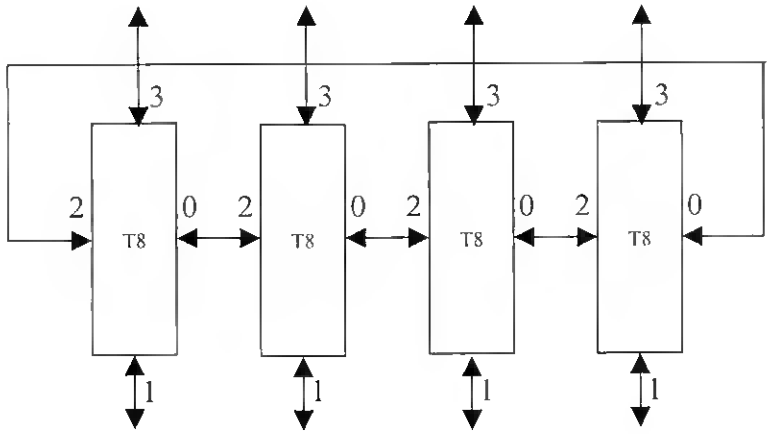


Fig. 6 - Ligação de 4 T8 através de uma topologia em anel.

De notar, para este caso a existência de 8 canais disponíveis para possíveis ligações a um máximo de outros 8 *transputers.*

A arquitectura real onde se implementaram algoritmos sequenciais e paralelos, objectos de estudo, consiste numa placa TMB16 instalada numa slot de expansão ISA, com possibilidade de ligação de 10 transputers. O primeiro processador da rede de *transputers*, o *root*, é aquele que está em contacto directo com o sistema de ficheiros existentes no PC, e que transmite informação proveniente aos restantes processadores da rede.

O *TMS320C40*, é um processador digital de sinal (*DSP*) de 32 bits, com 40 Mhz de velocidade de relógio. Para a comunicação com outros dispositivos do mesmo tipo, incorpora 3 pares de portas de ligação paralelas (sendo ao todo 6). Cada uma dessas portas é

unidireccional e half_duplex, permitindo taxas de transmissão assíncronas de 20 Mbytes/seg. entre outros elementos de processamento, com pico de desempenho na ordem de 50 Mflops.

Na figura que se segue é visível o aspecto de possíveis ligações físicas entre quatro *TMS320C40*:

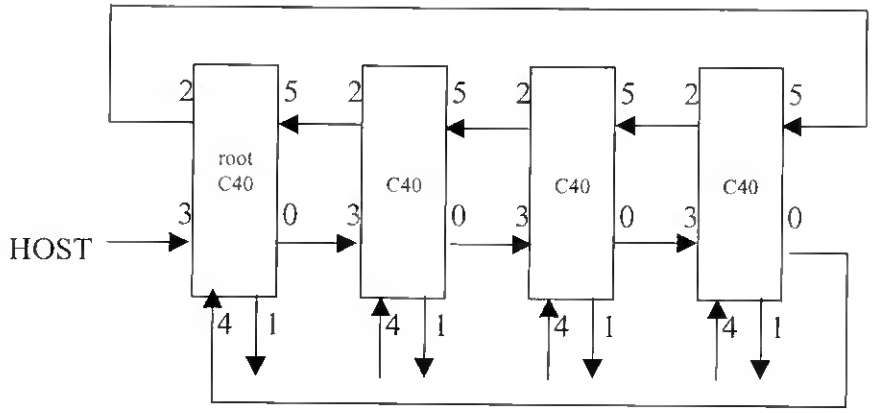


Fig. 7 - Ligação de 4 *C40s* através de uma topologia em anel

De notar a existência em cada *C40* de 6 canais físicos unidireccionais, e a possibilidade de serem acrescentados à topologia apresentada outros processadores.

Nesta configuração o processador *root* é o único que se poderá ligar ao *host*. Toda a comunicação com o sistema de ficheiros do *host* passa através do *root*.

2.4.2 Linguagens associadas

A linguagem original dos *Transputers* é o *Occam* [11], que ainda hoje é a adoptada por muitos utilizadores. No entanto com a evolução deste tipo de processadores, surgiram e podem ser utilizadas outras linguagens, com outras funções e potencialidades. Assim, o *transputer T805* pode ser programado em *Occam* e em C, utilizando o compilador *3L Parallel C V2.2.4* [12], que tem por base o *ANSI C*, acrescido de funções específicas, ou o *INMOS ANSI C* [13].

O *Occam*, como linguagem proprietária dos *transputers*, possui características que lhe são exclusivas; isto é; não podem ser aplicadas a dispositivos de outros tipos.

A unidade básica do *Occam* é o "processo" que, após realizar um conjunto de instruções, termina. Pode existir mais do que um processo a ser executado em qualquer altura no sistema.

Os processos em *Occam* são criados a partir das três primitivas que a seguir se apresentam e respectiva sintaxe:

- atribuição (nome_var:=expressão);
- entrada (canal_entrada ? dados_entrada);
- e saída (canal_saída ! dados_saída).

Estas primitivas são combinadas para formar as seguintes estruturas:

- SEQ, significa sequencial. A base do seu funcionamento é apresentado na figura abaixo:

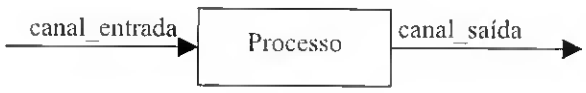


Fig. 8 – Estrutura Sequencial

O canal de entrada recebe dados de entrada os quais são processados pelo processo e o resultado emitido pelo canal de saída. Esta situação pode ser generalizada para n processos, resultando num sistema *pipeline*;

- PAR, significa paralela. É representada, na figura que se segue:

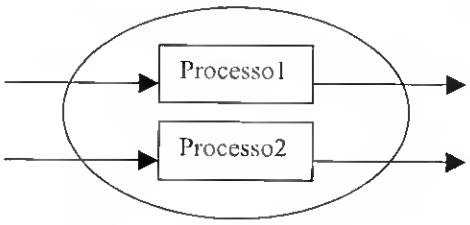


Fig. 9 – Estrutura Paralela

Os dois processos sequenciais são executados em simultâneo no sistema apresentado;

- ALT, significa alternativa. O seu funcionamento é ilustrado na figura que se segue:

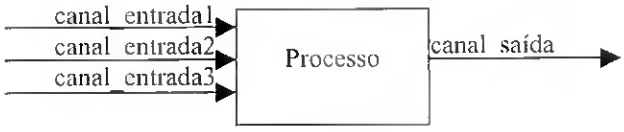


Fig. 10 – Estrutura Alternativa

É executado apenas o processo associado ao canal correspondente ao primeiro dado de entrada *ready*, e o resultado emitido pelo canal de saída;

- IF, significa condicional. Funciona na base de um desmultiplexador e consiste na existência de um canal com dados de entrada que por meio de uma estrutura condicional if, direcciona os dados para o canal de saída apropriado, conforme é mostrado na figura que se segue:

Fig. 11 – Estrutura Condicional

- WHILE, significa iteração. Funciona na base de um multiplexador, que consiste de um ciclo infinito, o qual recebe dados de entrada do primeiro canal de origem disponível, processa esses dados e emite o resultado pelo canal de saída correspondente. O esquema relacionado com esta estrutura é idêntico ao apresentado na estrutura ALT da figura 10;
- além de outras.

Com as estruturas apresentadas, é possível estabelecer uma decomposição hierárquica de processos para simplificação de problemas complexos.

Outros processadores mais recentes que o *transputer*, como é o caso do *TMS320C40 DSP*, utilizam normalmente a linguagem C para programação. Para o caso concreto foi utilizado o compilador 3L Parallel C V2.0.2 [14], baseado também no ANSI C, acrescido de funções que lhe são especificas.

O Parallel C das versões apresentadas, é baseado no conceito da comunicação de processos sequenciais a executarem em sistemas de *transputers* e ou *DSPs*. Contempla todas as facilidades do ANSI C, e dependendo da versão para *transputers* ou *DSPs*, é complementado com bibliotecas especificas para suporte de facilidades de processamento paralelo, tais como: canais de comunicação, controlo de processos, controlo de hardware, além de outras.

Evidentemente, todas estas funções não são tratadas da mesma forma, dependendo pelo que já foi referido dos processadores em causa.

São apresentadas a seguir algumas das bibliotecas do Parallel C, as quais incluem funções (ver em [12] e [14] a sintaxe característica das funções em C) que se podem confrontar com as primitivas e estruturas do Occam referidas acima:

- <alt.h>, é uma biblioteca que inclui dois conjuntos de funções que produzem resultados comparáveis à estrutura ALT do Occam, além de admitir outras possibilidades. É o caso das funções alt_nowait, que devolvem um determinado valor se não existir pelo menos um canal pronto para comunicar. Funções como alt_wait, aguardam até que um dos canais se torne pronto.
- <chan.h>, é uma biblioteca com funções relacionadas com a comunicação entre *transputers* ou *DSPs* através de canais. Existem essencialmente dois grupos de funções. As que se iniciam com o prefixo *chan_in*, são utilizadas para entrada de dados através canais. Todas as outras que começam com *chan_out*, referem-se à correspondente saída de dados através de canais. Os dados que fluem por meio dos canais especificados, podem ser uma palavra de 32 bits, ou mensagens de qualquer tipo e tamanho.

2.4.3 Representação dos dados

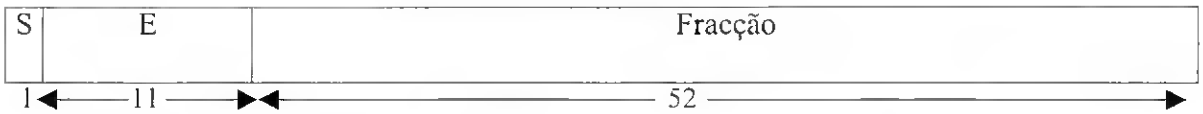
Relativamente a *transputers*, uma palavra (*word*) no *transputer T805* são 32 bits ; ou seja; 4 bytes. Existem no entanto *transputers* como é o caso dos T2 em que uma palavra é formada por 16 bits.

O tipo de dados inteiro ou caracter para *transputers* T805 são representados por defeito como na tabela que se segue [12]:

Tipos	Bits	Bytes	Mínimo	Máximo
Char	8	1	0	255
Signed char	8	1	-128	127
Short int	16	2	-32768	32767
Unsigned short integer	16	2	0	65535
Integer	32	4	-2147483648	2147483647
Unsigned integer	32	4	0	4294967295
Long integer	32	4	-2147483648	2147483647

Tabela 1: Tipos de dados inteiro e caracter para T805

Qualquer apontador é representado por uma única palavra cujo valor é o endereço do objecto apontado. O *transputer* T805 segue as normas standard *IEEE floating-point* para a representação de valores reais. Estes formatos standard são representados na memória do *transputer* como é mostrado na figura a seguir:



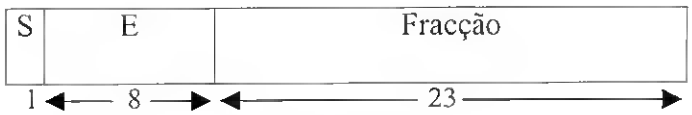


Fig. 12 – Representação de *floating-point* em *Transputers*

Notar que um real ocupa 4 bytes (32 bits) e um double ocupa 8 bytes (64 bits).

A representação dc dados quer em *Transputers* quer em *C40s* consiste nos 3 campos a seguir:

- expoente (E);
- sinal (S);
- e fracção.

Ao contrário dos *transputers*, os *C40s* seguem as normas da *Texas Instruments* [5] para representação dos dados. Neste tipo de processadores a unidade básica dc informação são valores caracter (*word*) sendo utilizados 32 bits para a sua representação. Uma word possui também o tamanho de um *integer*, de um *float* ou de um *double*.

Os *C40s* suportam 3 possíveis representações para valores reais: pequeno (*short)* , de precisão única (*single-precision*) e de precisão estendida (*extended-precision*). Por estes e outros motivos os *C40* realizam cálculos de uma forma rápida e precisa.

Estes formatos standard são representados na memória do *C40* como se pode ver na figura que se segue:

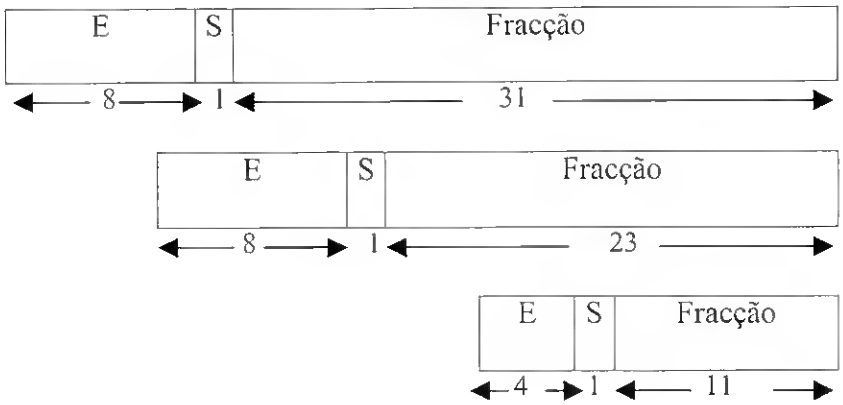


Fig. 13 – Representação de *floating-point* em *C40s*

De notar que o *short* ocupa 2 bytes (16 bits), o *single-precision* 4 bytes (32 bits) e o *extended-precision* 5 bytes (40 bits).

2.4.4 Conversão de dados

Existem funções na biblioteca <ieee.h> do *3L Parallel C V2.0.2* [14] que permitem converter números de vírgula flutuante entre o formato 32-bits da *Texas Instruments* e os formatos 32 e 64 bits do padrão *IEEE* usados por *transputers*, Suns, PCs e outros processadores. Estas funções são particularmente úteis para a implementação de aplicações em arquitecturas mistas que incluam *C40s* e *transputers*.

São dois os tipos de funções existentes [12]: as que se iniciam com *ieee_single* referentes a uma palavra e com *ieee_double* para duas palavras.

No Capítulo 5 desta tese é apresentado um exemplo de paralelização algorítmica numa arquitectura de *transputers* e *C40s*, onde se aplicam estas funções:

- *ieee_single_from_float* e *ieee_single_from_float_vec* para conversão de um real de 32 bits ou um vector de reais no formato da *Texas Instruments* para formatos de 32 bits do padrão IEEE.
- *ieee_single_to_float* e *ieee_single_to_float_vec* para conversão de um real de 32 bits ou um vector de reais do padrão IEEE para formatos de 32 bits da *Texas Instruments*.

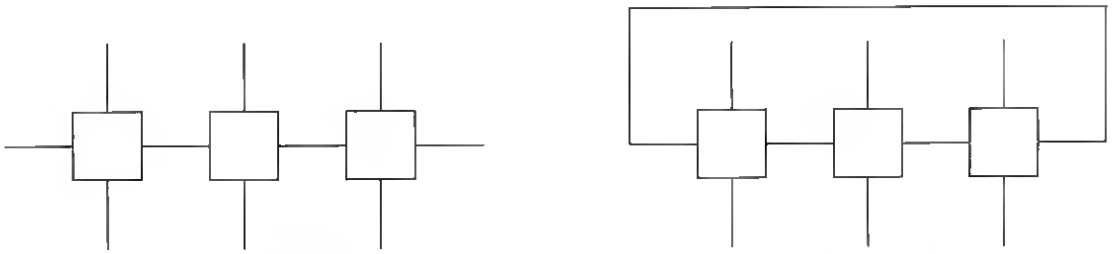
2.5 Topologias Homogéneas e Heterogéneas

Com um sistema de *transputers* e/ou processadores digitais de sinal podem ser criadas diferentes topologias de rede. Se a arquitectura apresentar o mesmo tipo de processadores diz-se homogénea. No caso de ser constituída por diferentes tipos de processadores diz-se heterogénea. Diversas topologias podem ser criadas para avaliação do desempenho de

algoritmos sequenciais relativamente a versões paralelas dos mesmos. Essas topologias vão desde *pipelines*, anéis, estrelas ou, com maior número de ligações, hipercubos, entre outras.

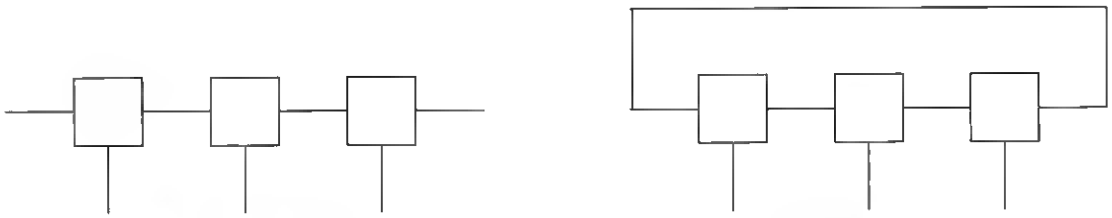
2.5.1 Exemplos de arquitecturas com *Transputers*, DSPs e mistas

Para exemplificar possíveis topologias de rede, considera-se a existência de 3 *Transputers* e 3 *C40s*. Deste modo, através de ligações físicas entre processadores, organiza-se a arquitectura pretendida como se pode ver nas figuras que se seguem:



a) - *Pipeline* **b) - Anel**

Fig. 14 – Topologias de *Transputers*



a) - *Pipeline* **b) - Anel**

Fig. 15 – Topologias de *C40s*

Relativamente a arquitecturas mistas com o número de elementos de processamento apresentados, a situação não é tão linear como as anteriores. É possível através do 3L Parallel C e do interface *HET4030tl* da *Hunt Engineering* [15] programar arquitecturas heterogéneas de *Transputers* e *C40s*. Utilizam-se apenas 3 dos 4 canais físicos INMOS bidireccionais existentes em cada *Transputer*, que por sua vez através do *HET4030tl* são divididos em 2 canais físicos *C40*, um em cada direcção, como se pode constatar na figura da página a seguir:

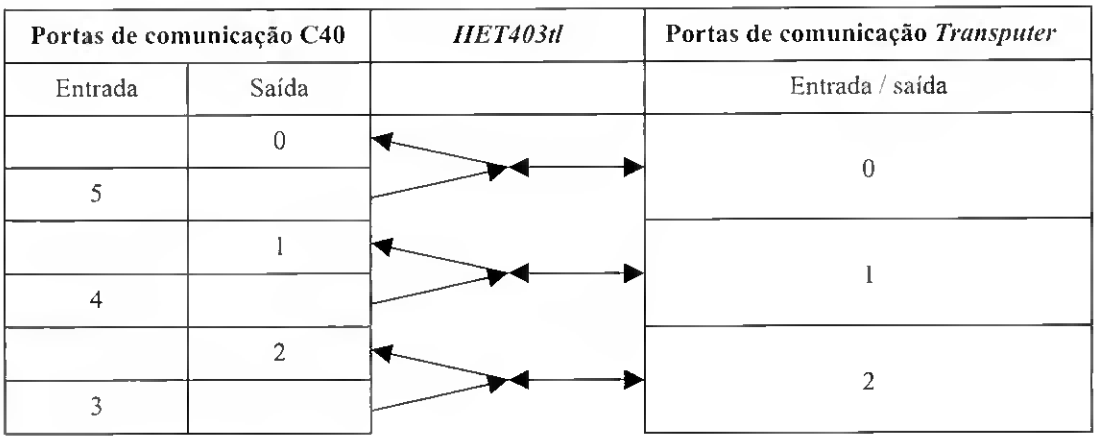


Fig. 16 – Interface lógica para *HET403tl*

Os *C40* podem assim ser ligados aos *transputers* utilizando o *HET403tl*, através das ligações apresentadas na figura que se segue:

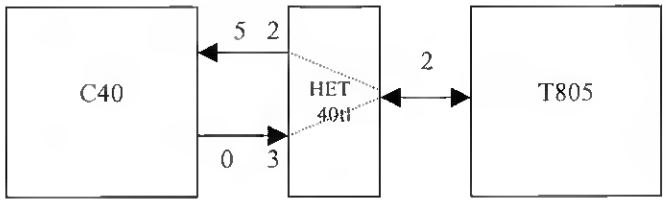


Fig. 17 - Ligação entre *C40* e *T805*

2.5.2 Comunicação e sincronização de processos

O tratamento de aplicações de processamento paralelo em sistemas com *transputers* e/ou *C40s* é baseado no conceito da comunicação sequencial de processos, onde cada processo tem a sua própria região de memória para código e dados [12], [14]. A comunicação entre processos é estabelecida através de ligações ponto a ponto, conhecidas por canais. Uma topologia de rede é assim constituída pela colecção de um ou mais processos sequenciais concorrentes, que comunicam entre si por meio de canais. Cada canal, pode apenas transportar mensagens numa direcção e os processos podem possuir qualquer número de canais lógicos de entrada e saída, desde que o seu número seja fixo ao longo do sistema. Além

da unidireccionalidade, os canais de comunicação também proporcionam a sincronização, de modo a que a comunicação apenas tenha lugar quando tanto o processo emissor como o processo receptor estejam no estado *ready* (prontos) tanto para emitir como para receber a mensagem; isto significa que se um dos processos transitar em primeiro lugar para o estado *ready* então através de um mecanismo de controlo, esse processo é forçado a aguardar até que o outro comute igualmente para o mesmo estado. A única responsabilidade do programador em todo este processo é a de evitar que ocorram bloqueios; ou seja, situações de *deadlock*.

Já foi referido que cada *transputer* possui quatro portas de ligação *INMOS* de canais bidireccionais, para permitir ligação em ambas as direcções a um máximo de quatro processadores da mesma família. No caso do *C40,* este inclui três pares de portas de ligação conhecidas por *links C40* com dois canais unidireccionais cada, para ligação a um máximo de outros três *C40s*. Os canais físicos baseiam-se no mesmo conceito dos canais lógicos, proporcionando a sincronização e comunicação unidireccional ou bidireccional.

Dois ou mais processos que pretendam comunicar não necessitam de estar no mesmo processador, podendo ser atribuídos a outros. É possível executar qualquer número de processos concorrentes num único *transputer* ou *C40,* limitados apenas pela memória disponível, sendo os canais existentes designados por canais internos. Se dois ou mais processos não se encontram no mesmo processador existe *software* que permite o envio de mensagens ao processador correcto através de ligações físicas entre processadores.

Do ponto de vista da programação canais internos ou lógicos e canais externos ou físicos, são tratados da mesma forma e usados para o envio e recepção de mensagens através do mesmo grupo de instruções. Os canais físicos são identificados por endereços fixos, enquanto que os canais internos têm endereços atribuídos por *software*.

A cada processo executado num *Transputer* ou *C40* está associado um nível de prioridade. A única diferença é que o primeiro apenas admite dois níveis de prioridade, enquanto que o segundo pode suportar até oito níveis de prioridade.

Para uma distribuição equitativa dos tempos de utilização do processador pelos processos que lhe foram atribuídos, é normalmente reservado um tempo de execução limitado para cada processo, designado por *Time_Slice*. Um processo de alta prioridade apenas pode ser retirado do processador ou preemptado após ter finalizado a execução de uma operação, como o envio de uma mensagem a outro processo. Por esta razão estes devem ser implementados de forma a não realizarem tempos de computação excessivos. Um processo com prioridade baixa pode ser retirado do processador em qualquer altura, no caso da sua fatia de tempo (*time slice*) se ter esgotado ou quando um processo de alta prioridade se encontra no estado *ready*. Toda esta situação está relacionada com a gestão do processador ou estratégia de escalonamento.

O *Parallel C* da versão para *C40* [14] proporciona a existência de canais virtuais que automaticamente transportam mensagens entre processadores através de nós de rede intermédios. Múltiplos canais virtuais podem partilhar o mesmo canal físico. A maior limitação dos canais virtuais consiste na lentidão relativamente aos canais físicos.

2.5.3 Ficheiros de configuração de topologias

Quando se pretende programar uma aplicação paralela, é óbvia a necessidade de se estabelecerem ligações para comunicação dos processos. A atribuição dos processos aos processadores disponíveis é também necessária.

Para este tipo de aplicações, o Parallel C das versões para *Transputer* [12] e *C40* [14] obriga à existência de um ficheiro executável onde deverá constar informação sobre:

- identificação dos processadores onde a aplicação irá correr e as ligações físicas existentes;
- ficheiros que contém o código de cada processo da aplicação;

- atribuição dos processos aos processadores;
- e ligações lógicas existentes entre os vários processos.

Este ficheiro de configuração (com extensão cfg) é apenas executado em plataformas paralelas reais e obrigatoriamente obedece a um conjunto de regras de sintaxe.

```
!      DESCRIÇÃO DO HARDWARE (TRANSPUTERS)

! Identificação dos processadores através da declaração processor
processor host
processor root
processor processador3
! Outros processadores

! Ligações físicas entre processadores através da declaração wire
wire ? root[0] host[0]
wire ? root[1] processador3[0]
! Restantes ligações físicas se existirem mais processadores

!      DESCRIÇÃO DO SOFTWARE

! Identificação dos processos através da declaração task
task afserver ins=1 outs=1
task filter ins=1 outs=1
task processo3 ins=1 outs=1
! Restantes processos, nº de canais de entrada e saída , declarações para optimização de memória  e se
! necessário o nome do ficheiro
! Alocação de processos aos processadores existentes através da declaração place
place afserver host
place filter root
place processo3 processador3
! Restante mapeamento

! Ligações entre os processos
connect ? filter[0] afserver[0]
connect ? afserver[0] filter[0]
connect ? filter[1] processo3[1]
! Outras ligações entre processos caso existam
```

```
!      DESCRIÇÃO DO HARDWARE (C40S)

! Identificação dos processadores através da declaração processor
processor root
processor processador2
! Outros processadores

! Ligações físicas entre processadores através da declaração wire
wire ? root[0] processador2[3]
! Restantes ligações físicas caso existam mais processadores
! (continua)
```

```
! (continuação)
!       DESCRIÇÃO DO SOFTWARE

! Identificação dos processos através da declaração task
task processo1  ins= nº canais entrada  outs= nº canais saída
task processo2  ins= nº canais entrada  outs= nº canais saída
! Restantes processos, nº de canais de entrada e saída, declarações para optimização de memória e se
 ! necessário nome do ficheiro

! Alocação de processos aos processadores existentes através da declaração place
place processo1  root
place processo2 processador2
! Restante mapeamento

! Ligações entre os processos
connect ? processo1[0] processo2[0]
! Outras ligações entre processos caso existam
```

Os ficheiros apresentados referem-se a topologias de *Transputers* e *C40s*.

2.6 Conclusões

Neste capítulo, foi sumariada a pesquisa a diversa bibliografia relacionada com o tema desta tese, tendo como fim, resolver cada um dos objectivos mencionados no Capítulo 1.

Incluíram-se matérias necessárias não só à compreensão do trabalho anteriormente desenvolvido, o qual se detalha no próximo capítulo, bem como de outros assuntos necessários à organização e clarificação do trabalho proposto para esta tese.

Optou-se inicialmente pela definição de conceitos sobre processos, processadores, algoritmos paralelos, medidas de desempenho, além de outras. Seguiu-se para a descrição de detalhes do *hardware* e *software* e de possíveis topologias entre processadores que são objecto de estudo.

3 ANÁLISE AO SOFTWARE DISPONIBILIZADO

3.1 Introdução

No Capítulo anterior foram apresentados conceitos relacionados com o tema proposto para este trabalho, necessários para o desenvolvimento dos objectivos propostos.

Este capítulo visa a recolha de informação resultante da análise ao código em *Matlab* dos programas e funções incluídos no *software* disponibilizado [4], de modo a poder desenvolve-lo de acordo com o pretendido.

3.2 Informação Recolhida da Análise ao *Software* Disponibilizado

Nesta secção são indicados pontos importantes da análise realizada aos programas e funções existentes na *Toolbox* disponibilizada.

Os vários assuntos encontram-se organizados na sequência que a seguir se apresenta:

- limitações do *Matlab* para desenvolvimento de todas as potencialidades características de plataformas paralelas reais, concluídas na versão disponibilizada e também para a nova versão a apresentar no próximo capítulo;
- análise ao código das funções *Matlab* desenvolvidas pelo utilizador;
- utilização do programa para inicialização do simulador de arquitecturas paralelas homogéneas, designado por *Presim*;
- verificação nos ficheiros de inicialização criados pelo programa anterior da necessidade de geração automática de código com o fim de possibilitar uma menor intervenção do utilizador no que se refere à edição de dados, estruturas e instruções necessárias à execução completa da simulação;
- análise aos resultados obtidos pelo simulador, *Sim*.

3.2.1 Restrições do *Matlab* para o desenvolvimento do simulador

Este simulador foi inicialmente concebido para permitir a simulação de aplicações de controlo em plataformas paralelas homogéneas, contituídas por *Inmos Transputers*.

O utilizador define inicialmente através de uma interface amigável e em ambiente *Matlab*, a topologia a usar e o código das funções que irão ser acedidas pelo(s) processo(s) do sistema. Posteriormente, poderá simular e analisar a execução paralela, permitindo comparações entre as diferentes aproximações obtidas, e também investigar possíveis engarrafamentos de processos.

O objectivo principal do simulador não é atingir tempos de execução exactamente iguais aos obtidos numa plataforma paralela real, mas sim, comparar sem esforço, diferentes soluções de algoritmos paralelos.

Face ao objectivo apresentado, o simulador possui algumas limitações:

1. Como os processos são escritos em funções *Matlab*, há necessidade de executá-los a partir do início do código até ao final da função. Isto significa que forçar qualquer processo a libertar o processador, ou seja preemptar o processo, não é possível ser implementado no *Matlab*. Esta limitação tem duas consequências:

1.1. Os Processadores Digitais de Sinal podem suportar até oito níveis de prioridade enquanto que os *Transputers* apenas dois níveis. Para este trabalho foi considerado que cada processo apenas possui dois níveis de prioridade, alta ou baixa. Relativamente aos processos de alta prioridade espera-se que sejam executados num curto espaço de tempo. Se um ou mais processos de alta prioridade estão em condições de serem executados num determinado processador, então apenas um deles é seleccionado e executa até ter de aguardar por uma função de comunicação, ou termino do processamento. Se não existirem processos de alta prioridade em situação

de prosseguir, mas um ou mais processos de baixa prioridade estiver num estado de pronto, então um deles é executado.

Como é pretendida a possibilidade de simulação de processos de alta e de baixa prioridade, e como não existe a possibilidade de implementar a preempção com o ambiente *Matlab*, houve necessidade de considerar que os tempos de execução no CPU de processos de alta prioridade são de desprezar em comparação com os tempos de execução no CPU para processos de baixa prioridade. Isto significa que, a um processo de alta prioridade é atribuída uma menor porção de tempo de execução entre sucessivas comunicações, sendo este de desprezar em comparação com o mais pequeno período de tempo de execução de processos de baixa prioridade entre comunicações de processos situados no mesmo processador. Esta não é, contudo, uma grande limitação, por os processos de alta prioridade serem habitualmente usados para encaminhamento de dados, e não para finalidades de cálculo;

1.2. A impossibilidade de preempção de um processo, também implica que o simulador não possa oferecer a possibilidade da estratégia de escalonamento de tarefas usada em processos de baixa prioridade como num processador real. Por exemplo, o *Transputer* não preempta processos de alta prioridade, mas utiliza a atribuição de fatias de tempo (*time-slicing*) a processos de baixa prioridade [6]. Pelas restrições acima apresentadas assume-se que um processo de baixa prioridade continua a sua execução até necessitar de uma função de comunicação.

2. O tempo actual de execução do processador também não é fácil de simular pois depende principalmente, não só do tipo do processador e sua frequência de relógio, como também das instruções actuais do CPU geradas pelo compilador, de serem executadas na memória interna ou externa, entre outros factores. Em termos de *Matlab*, o tempo de execução computacional pode apenas ser medido utilizando a função *flops*, a qual

calcula sensivelmente o número genérico de operações em vírgula flutuante necessárias para executar um código específico em *Matlab*. Assume-se assim neste simulador que o tempo de execução é apenas função do número de operações em vírgula flutuante executadas pelo algoritmo, onde o tempo necessário para o cálculo de uma operação genérica em vírgula flutuante, é obtido pela média de tempos de execução, no processador considerado, de diferentes códigos de teste. Para a nova versão do simulador consideraram-se medidas obtidas em sistemas de *Transputers*. A nova versão inclui também medidas obtidas em sistemas de *DSPs*.

3. As diferenças na comunicação entre processadores dependem principalmente do tipo de portas de ligação disponíveis. Os *Transputers* comunicam através de portas de entrada/saída série bidireccionais, a comunicação é sincronizada e não existe *buffer*. As mensagens são transmitidas como sequências de bytes cada uma das quais é reconhecida antes de a próxima ser transmitida. A velocidade de transmissão em Kbytes/seg. depende da velocidade das portas de ligação utilizadas, que pode ser 5, 10 ou 20 Mbits/seg., e na existência de dados na memória interna ou externa [6]. Os *DSPs* [5] comunicam através de portas de entrada/saída paralelas uni-direccionais sendo a comunicação assíncrona. Cada porta de ligação de um processador *DSP* opera a 20 Mbytes/sec. No simulador o produto de bytes enviados e o parâmetro denotado como o tempo necessário para comunicar um byte, resulta no tempo considerado para a comunicação externa, acrescido do tempo necessário para iniciar a ligação (*setup time*). Estes parâmetros são obtidos para as diferentes velocidades das portas de comunicação, pela média dos tempos de comunicação, medidos no sistema. É assumido que a comunicação interna é instantânea.

3.3 Descrição Sumária das Funcionalidades incluídas na *Toolbox* em *Matlab*

3.3.1 Funções *Matlab* desenvolvidas pelo utilizador

Os processos são implementados em funções *Matlab*, escritas e identificadas pelo utilizador. No código de cada função são incluídas funções especiais disponíveis pela Toolbox ao utilizador, as quais se detalham no próximo capítulo.

Testes realizados nesta versão apresentam algoritmos paralelos que abrangem funções de alta e de baixa prioridade. Funções de alta prioridade implementam algoritmos de encaminhamento (*routing)* de dados de entrada e de resultados. Quanto às funções de baixa prioridade, realizam as instruções requeridas para processamento de dados provenientes de outras funções e se necessário enviam resultados. O número de canais existentes é convencionado pelo utilizador.

O código resultante de cada processo que utilize determinada função de uma forma geral é implementado de acordo com a seguinte sequência de instruções:

- acesso às variáveis globais necessárias;
- reposição do espaço de trabalho do processo onde se encontra gravada informação relacionada com número do processo, canais existentes, variável de controlo, recursos do sistema usados, além das variáveis locais ao processo;
- localização no espaço de trabalho do processo da posição onde se encontra a variável local a ser encaminhada e ou calculada pelo processo;
- início de ciclo
    - utilização de funções desenvolvidas em *Matlab* que implementam as primitivas do *Occam* ou funções do *Parallel C* para recepção de dados;
    - utilização de funções desenvolvidas em *Matlab* que implementam primitivas do *Occam* ou funções do *Parallel C* para emissão de dados;
    - outras instruções do processo;

- fim de ciclo
- fim de execução do processo

3.3.2 Inicialização do simulador

Antes de iniciar a simulação com o programa *Sim,* o utilizador tem de definir a configuração, isto é, a topologia que será usada na simulação, através da inserção de valores nas várias variáveis globais existentes. A informação relaciona-se essencialmente com: funções existentes, processos que utilizam os códigos das várias funções, prioridades e número de canais lógicos dos processos, alocação de processos aos processadores disponíveis. Todo este processo é realizado com a execução do programa *Matlab PreSim,* de inicialização do simulador.

Foi possibilitada a explicitação de algoritmos SIMD (acrónimo de *Single Instruction, Multiple Data*), pelo uso de réplicas ao código de cada função cujo número é definido pelo utilizador. Os canais lógicos entre processos são especificados em termos de processos de origem e destino sendo os correspondentes identificadores automaticamente gerados.

3.3.3 Ficheiros de inicialização

Como resultado da execução do programa *PreSim,* são gerados automaticamente ficheiros *Matlab* de inicialização e de dados. O utilizador tem de subsequentemente pesquisar no ficheiro de inicialização espaços vazios, editar manualmente as variáveis locais dos processos e respectivas inicializações, bem como outras funções especiais necessárias. Ele terá também de editar o conteúdo das funções *Matlab* as quais contêm o código de cada processo usando as convenções dadas no ficheiro criado automaticamente em termos de identificadores de processos e canais.

3.3.4 Resultados do Simulador

A simulação é então executada utilizando o programa *Matlab* disponível na *Toolbox* designado por *Sim*. Este programa utiliza os ficheiros de configuração previamente criados e executa o programa paralelo simulado. Após finalizada a simulação toda a informação resultante pode ser visualizada em gráficos *Matlab*, para análise do utilizador. A informação disponibilizada preocupa-se principalmente com as seguintes medidas de desempenho de execução:

- tempos de execução paralelos;
- eficiência do processador;
- tempo de execução sequencial;
- speedup;
- e eficiência paralela.

As últimas duas medidas apenas serão disponíveis se existir uma função *Matlab* a qual implemente o código sequencial.

Além da informação apresentada também se poderá visualizar:

- gráficos de tempos inactivos e de computação para processadores seleccionados pelo utilizador, ao longo dos quais o utilizador poderá seleccionar períodos de execução;
- gráficos de eventos para processos especificados pelo utilizador, ao longo dos quais o utilizador escolhe os períodos de execução. Os eventos considerados são associados a canais de comunicação, nomeadamente:
    - o processo está à espera de receber dados;
    - o processo está actualmente a receber dados;
    - o processo está em execução;
    - o processo está à espera de enviar dados;
    - o processo está actualmente a enviar dados.

3.4 Conclusões

Toda a informação recolhida neste capítulo serviu de base para a concepção da nova versão do simulador apresentada no Capítulo 4.

Foram inicialmente apresentadas limitações do simulador provenientes da utilização do ambiente de computação *Matlab*, especificamente no que se refere à impossibilidade de preemptar processos, e respectivas consequências, além das limitações encontradas no cálculo dos tempos de computação e de comunicação entre processos.

Posteriormente foi apresentada uma abordagem sobre a versão anterior do simulador onde se incluíram os passos necessários para utilização do grupo de programas existentes na *toolbox* e uma análise com vista à concepção da nova versão do simulador.

4 VERSÃO MELHORADA DO SIMULADOR

4.1 Introdução

Pretende-se neste capítulo realizar uma abordagem detalhada, de natureza específica, sobre as as funções e potencialidades incluídas na versão actual do simulador, seus diferentes módulos, encadeamento e funcionamento. Aqui se descrevem o conjunto de variáveis globais geradas não só no código da versão anterior do simulador, como também as concebidas nesta versão, além de serem apresentadas as funções externas e internas do simulador.

Os passos a seguir pelo utilizador para a utilização do grupo de programas existentes e incluídos na *toolbox* são igualmente apresentados, através de um exemplo didáctico.

Na versão actual da *Toolbox* em *Matlab*, não houve necessidade de geração de novas funções externas ao simulador, dado as existentes se adaptarem perfeitamente às exigências do utilizador na concepção de algoritmos paralelos.

No simulador disponibilizado foram criadas as variáveis globais necessárias à simulação da comunicação entre processos situados num mesmo processador ou em processadores distintos. Os canais existentes relacionam-se apenas com as ligações lógicas dos processos.

Na concepção da nova versão do simulador houve necessidade de implementar automaticamente, ligações físicas entre sistemas constituídos por *Transputers* e ou *C40s*.

4.2 Caracterização do Conjunto de Variáveis Globais existentes na *Toolbox*

Os identificadores usados para referenciar as variáveis globais do sistema são apresentados a seguir, segundo a ordem pela qual foram gerados no ficheiro de inicialização do simulador:

- ***NUM_FUN***, é uma variável que contém a informação relacionada com o número total de funções existentes no sistema. Cada código de função pode assim ser utilizado por diferentes processos do sistema, de 1 a *NUM_PROC*.
- ***NUM_PROC***, é uma variável que contém a informação sobre o número total de processos existentes no sistema.
- ***NUM_CHANNELS***, é uma variável onde consta a informação sobre o número total de canais lógicos existentes no sistema.
- ***FUN_ID***, é uma matriz com os vários identificadores, do tipo cadeia de caracteres, das funções existentes no sistema, de 1 a *NUM_FUN*.
- ***FUNCTIONS***, é uma matriz que contém a informação relacionada com as várias funções do sistema, de 1 a *NUM_FUN*. Esta matriz possui tantas linhas como o número de funções existentes, e para cada função, em colunas, existem campos identificados com os códigos que a seguir se apresentam:
  - *First*: é um identificador , com o número do primeiro processo que utiliza o código da respectiva função, podendo ser qualquer um, de 1 a *NUM_PROC*;
  - *How_many*: refere-se ao número total de processos, que executam o código da respectiva função;
  - *Num_channels*: refere-se ao número total de canais lógicos, existentes no código da respectiva função;
  - *Prio*: assume apenas os valores 0 e 1. Estes valores estão relacionados com o tipo de prioridade da respectiva função. Se for 0 a função é de baixa prioridade, se for 1 a função é de alta prioridade;
  - *Length*: contém o tamanho da cadeia de caracteres do identificador da respectiva função;

- *Num_local_var*: indica o número total de variáveis locais existentes no código da respectiva função;
- *Num_local_var_ini*: indica o número total de variáveis locais inicializadas, existentes no código da respectiva função.

- ***PROCESSES***, é uma matriz que contém a informação relacionada com os vários processos existentes no sistema, de 1 a *NUM_PROC*. Esta matriz possui tantas linhas quanto o número de processos existentes, e para cada processo existem, em colunas, campos identificados com os códigos indicados a seguir:
  - *Processor*: pode assumir valores entre 1 e *NUM_PROCESSORS* e identifica o processador onde o respectivo processo é executado;
  - *Prio*: assume apenas os valores 0 e 1. Estes estão relacionados com o tipo de prioridade do respectivo processo. Se for 0 o processo é de baixa prioridade , se for 1 o processo é de alta prioridade;
  - *Proc_fun*: pode assumir valores entre 1 até *NUM_FUN* e identifica a função associada com o processo.
- ***CHAN_ID***, é uma matriz que contém a informação relacionada com o número e identificadores dos canais associados aos vários processos do sistema de 1 a *NUM_PROC*. Esta matriz possui tantas linhas quanto o número de processos existentes, e para cada processo, em colunas, existem os campos identificados com os códigos que a seguir se apresentam:
  - *Number*: indica o número total de canais associados a cada processo do sistema;
  - *Id*: tantas colunas, quantas as existentes no campo *number*, contendo os respectivos identificadores dos canais associados a cada processo do sistema.
- ***NUM_PROCESSORS***, é uma variável que contém a informação sobre o número total de processadores existentes no sistema.

- ***ID_TOPOLOGY,*** é uma variável que assume apenas os valores 1, 2 ou 3. Estes estão relacionados com o tipo de processadores contidos na topologia em rede. Se for 1, trata-se de uma arquitectura homogénea com *transputers*; se for 2, identifica uma arquitectura homogénea com *C40s*; se for 3, trata-se de uma arquitectura heterogénea constituída por *transputers* e *C40s*. Expansões futuras utilizarão outros valores.
- ***ID_PROCESSOR,*** é uma matriz com os vários identificadores, do tipo cadeia de caracteres, dos processadores existentes no sistema, de 1 a *NUM_PROCESSORS*.
- ***PROCESSOR,*** é uma matriz que contém informação relacionada com o número de canais físicos e identificadores associados aos vários processadores do sistema, de 1 a *NUM_PROCESSORS*. Esta matriz possui tantas linhas quanto o número de processadores existentes, e para cada processador existem, em colunas, campos identificados com os códigos indicados a seguir:
  - *Id_top:* indica o tipo de topologia, assumindo os valores 1 para homogénea de *transputers*, 2 para homogénea de *C40's* ou 3 para heterogénea;
  - *type_processor:* identifica o tipo de processador, incluído na topologia. É 1 se *transputer* ou 2, se *C40;*
  - *Number_links*: a terceira coluna da matriz contém o número total possível de ligações físicas ocupadas associadas a cada processador do sistema (de 1 a *NUM_PROCESSORS*), isto é, aquelas que se encontram ligadas a outros processadores;
  - *Id_link*: existem tantas colunas, quantas as existentes no campo *Number_links*. Identifica cada canal de ligação, através da qual o respectivo processador se encontra ligado a outro, de 1 até *NUM_PROCESSORS*.
- ***MAX_P_PRIO,*** é uma variável que contém a informação sobre o maior número de processos prioritários existentes.

- ***PRIORITARIO***, é uma matriz que possui tantas linhas quanto o número máximo de processos prioritários existentes em um ou mais processadores, de *1 a MAX_P_PRIO*, e tantas colunas quanto o número de processadores existentes, de 1 a *NUM_PROCESSORS*.
- ***CHANNEL***, é uma matriz que contém a informação sobre processos de origem e destino dos vários canais existentes, de 1 até *NUM_CHANNELS*, bem como sobre o estado do canal (inactivo ou aguardar comunicação) e sobre os dados a comunicar. Existem tantas linhas quanto o número de canais simulados. A cada canal está associado um registo com os campos, em colunas, que a seguir se apresentam:
  - *Source*: identifica o processo de origem, que poderá ser qualquer um de 1 a *NUM_PROC*;
  - *Dest* : identifica o processo de destino, que poderá ser qualquer um de 1 a *NUM_PROC*;
  - *Waiting_to_send*: indica se por esse canal se está a aguardar o envio de dados. Pode tomar o valor 0 se não se está à espera ou 1 se está a aguardar comunicação;
  - *Waiting_to_receive*: indica se por esse canal se está a aguardar recepção de dados. Pode tomar o valor 0 se não está se está à espera ou 1 se está a aguardar uma recepção de dados;
  - *Num_var*: indica a posição da variável no espaço de trabalho (*WORKSPACE* do processo de destino) pela qual a comunicação se estabelecerá. Poderá ser qualquer uma variável de 1 a *Maxvar*;
  - *m*: informa sobre o número de linhas de dados a comunicar dentro do limite entre 1 a *Maxdim*;
  - *n*: informa sobre o número de colunas de dados a comunicar situadas dentro do limite entre 1 a *Maxdim*;
  - *data*: dados para comunicar no formato vector de 1 a *m*n*.

- ***NUM_LOCAL_VAR,*** é uma variável, que contém a informação sobre o número total de variáveis locais, existentes nas várias funções do sistema.
- ***NUM_LOCAL_VAR_INI,*** é uma variável, que contém a informação sobre o número total de variáveis locais inicializadas, existentes nas várias funções do sistema.
- ***VAR_ID,*** é uma matriz que contém a informação relacionada com os vários identificadores, do tipo cadeia de caracteres, das variáveis locais, separadas por vírgulas, existentes em cada código de função, de 1 até *NUM_FUN*. Existem tantas linhas quanto o número de funções existentes e tantas colunas quanto o número de caracteres, incluindo as virgulas, que identificam as várias variáveis locais da função.
- ***LOCAL_VAR,*** é uma matriz que contém informação relacionada com as variáveis locais existentes no sistema, de 1 a *NUM_LOCAL_VAR*. Esta matriz possui tantas linhas quanto o número de variáveis locais existentes, e para cada variável local existem em colunas campos identificados com os códigos que a seguir se apresentam:
  - *Len*: retorna o tamanho da cadeia de caracteres do identificador de cada uma das variáveis locais incluído a virgula se existir;
  - *Le*: retorna o tamanho de *Len* acrescentado ao tamanho do identificador da próxima variável local, incluindo a virgula se existir;
  - *Pos*: identifica a posição onde se encontra a variável local situada no comentário delimitado por: %- e -% incluído em cada função do sistema;
  - *Proc_fun*: pode assumir valores entre 1 até *NUM_FUN* e identifica a função associada com a variável local.
- ***VAR_ID_INI,*** é uma matriz que contém a informação relacionada com os vários identificadores, do tipo cadeia de caracteres, das variáveis locais inicializadas, separadas por vírgulas, existentes em cada código de função, de 1 até *NUM_FUN*. Existem tantas linhas quanto o número de funções existentes e tantas colunas quanto o número de

caracteres, incluindo as virgulas, que identificam as várias variáveis locais inicializadas da função.

- ***LOCAL_VAR_INI,*** é uma matriz que contém a informação relacionada com as variáveis locais inicializadas existentes no sistema. Esta matriz possui tantas linhas quanto o número de variáveis locais inicializadas existentes, e para cada variável local inicializada existem, em colunas, os campos identificados com os códigos que a seguir se apresentam:
  - *Len1*: retorna o tamanho da cadeia de caracteres do identificador das variáveis locais inicializadas, incluído a virgula se existir;
  - *Le1*: retorna o tamanho de Len1 acrescentado ao tamanho do identificador da próxima variável local inicializada , incluindo a virgula se existir;
  - *Pos1*: refere-se à posição onde se encontra a variável local situada no comentário delimitado por: %+ e +% incluído em cada função do sistema;
  - *Proc_fun*: pode assumir valores entre 1 até NUM_FUN e identifica a função associada com a variável local inicializada.
- ***HAS_REC_FROM,*** é um vector que identifica qual dos processos emissores enviou a mensagem recebida no último *ALT_WAIT* do processo respectivo.
- ***EVENTS***, contém a informação sobre a ocorrência temporal dos eventos de cada processo. Os eventos associados a cada processo encontram-se descritos por linhas, em colunas sendo representados os campos a seguir:
  - *Num_events*: indica o número de eventos do respectivo processo;
  - *Event*: refere-se aos possíveis eventos (descritos seguidamente): *EVENT_START_SEND, EVENT_END_SEND, EVENT_START_REC, EVENT_END_REC*;
  - *Time*:identifica o tempo no qual o evento ocorreu.

Os códigos dos respectivos eventos, são os a seguir apresentados:

- ***EVENT_START_SEND*** , código que indica um inicio de envio de dados;
- ***EVENT_END_SEND***, código que indica a finalização de envio de dados;
- ***EVENT_START_REC***, código que indica um inicio de recepção de dados;
- ***EVENT_END_REC*** , código que indica a finalização de recepção de dados;
- ***EVENT_WANTS_TO_SEND***, código que indica que um processo pretende enviar dados;
- ***EVENT_WANTS_TO_REC***, código que indica que um processo pretende receber dados;
- ***EVENT_END_WORKING***, código que indica que um processo finalizou a sua execução;
- ***EVENT_START_EXE*** , código que indica que um processo inicializou a sua execução;

- ***PROCESSOR_TIME***, é um vector com o tempo actual de funcionamento de cada processador da topologia, de 1 a *NUM_PROCESSORS*.
- ***READY_HPP***, é uma matriz que contém a informação relacionada com os processos de alta prioridade que se encontram no estado *ready* (prontos), associados a um processador específico, de 1 a *NUM_PROCESSORS*. Esta matriz possui tantas linhas quanto o número de processadores existentes, e para cada processador existem, em colunas, os campos identificados com os códigos que a seguir se apresentam:
  - *Numb_r_hpp*: indica o número de processos de alta prioridade que se encontram no estado *ready* no processador correspondente;
  - Para cada um dos processo, estão associados dois campos: *Time_ready*, identificando o instante de tempo no qual o processo passou ao estado *ready*, e *Proc_id*, o respectivo identificador do processo;
- ***READY_LPP*** , é uma matriz que contém a informação relacionada com os processos de baixa prioridade que se encontram no estado *ready* (prontos), associados a um processador específico, de 1 a NUM_PROCESSORS. Esta matriz possui tantas linhas quanto o número

de processadores existentes, e para cada processador existem em colunas campos identificados com os códigos que a seguir se apresentam:

- *Numb_r_lpp*: indica o número de processos de baixa prioridade que se encontram no estado *ready* no processador correspondente.
- Para cada um dos processo, estão associados dois campos: *Time_ready*, identificando o instante de tempo no qual o processo passou ao estado *ready*, e *Proc_id*, o respectivo identificador do processo;

- ***TIME,*** é um vector que representa os tempos actuais de cada processo do sistema , de 1 até *NUM_PROC.*
- ***NUM_ALT_WAIT,*** indica o número total de *alt_wait* existentes no momento.
- ***ALT_WAIT*** é uma matriz, com um número de linhas correspondentes a ***NUM_ALT_WAIT,*** cujos campos, situados em colunas, são os a seguir apresentados:
  - *Dest*: processo de destino, que poderá ser qualquer um de 1 a *NUM_PROC*;
  - *Numinp*: número de canais associados (aceita no mínimo duas e no máximo 3 entradas de dados em simultâneo);
  - Para cada canal, a informação adicional: *Channel*: identificador do canal; *NumVar*: a posição da variável no espaço de trabalho (*WORKSPACE*) de cada processo.
- ***NUM_ALT_SEND,*** indica o número total de *alt_send* existentes no momento.
- ***ALT_SEND,*** é uma matriz , com um número de linhas correspondentes a ***NUM_ALT_SEND***, cujos campos, situados em colunas, são os a seguir apresentados:
  - *Source*: processo de origem, que poderá ser qualquer um de 1 a *NUM_PROC*;
  - *Numout*: número de canais associados (aceita no mínimo duas e no máximo 3 saídas de dados em simultâneo);
  - *Max_time*: especifica um tempo máximo para realização da transferencia de dados;
  - Para cada canal, o identificador do canal.

- ***WORKSPACE***, é uma matriz onde em cada linha se encontram as variáveis locais a cada processo do sistema , de 1 até *NUM_PROC*. Em colunas temos os seguintes campos:
  - *Nargs*: número de variáveis no espaço de trabalho do processo;
  - Para cada uma das variáveis, a seguinte informação: *m* que corresponde ao número de linhas da variável; *n* ao número de colunas da variável e *var*: a variável no formato vector de 1 até m*n.
- ***FLOP_TIME***, é um vector com os tempos levados para realizar uma operação de virgula flutuante em *transputers* ou em *C40s*.
- ***INTERNAL_FIXED_COM_TIME***, é um vector (com 2 elementos) com informação sobre o tempo fixo de *overhead* associado com uma comunicação interna.
- ***INTERNAL_VAR_COM_TIME***, é um vector (com 2 elementos) com o tempo levado para transmitir uma variável real numa comunicação interna.
- ***EXTERNAL_FIXED_COM_TIME*** é um vector com informação sobre o tempo fixo de *overhead* associado com uma comunicação externa, entre *transputers*, entre *DSPs* e entre estruturas heterogéneas.
- ***EXTERNAL_VAR_COM_TIME***, é um vector com os tempos levados para transmitir um real numa comunicação externa entre *transputers, C40s* e entre *transputer* e *C40s*.

4.3 Funções do Simulador incluídas na *Toolbox*

As funções de Matlab integradas na *Toolbox* podem dividir-se em funções externas, isto é, destinadas a serem utilizadas nos processos escritos pelo utilizador, e em funções internas, isto é, destinadas a implementar o funcionamento dos processadores a simular. Embora o acesso às funções internas não possa ser impossibilitado ao utilizador, o seu uso não é aconselhado.

4.3.1 Funções externas (disponíveis ao utilizador)

Na *toolbox* estão disponíveis ao utilizador funções, específicas para comunicação de processos. Nesta secção são descritas as respectivas funções e o código em *Matlab* de cada uma delas é apresentado no Anexo C.

Para todas as funções, a variável *flops* denota o número de operações de vírgula flutuante realizadas pela invocação do processo desde a comunicação anterior. Este argumento não será pois referenciado na descrição que se segue.

- ***wait (chan, flops, number_of_variable)***, é uma função em *Matlab* para simular a primitiva *OCCAM*, **?** (*input*), ou a função em *C*, *chan_in*. Cada um dos argumentos significa respectivamente: o identificador do canal e o número da variável ou a ordem em que se encontra posicionada no espaço de trabalho do processo.
- ***send (variable, chan, flops)***, é uma função em *Matlab* para simular a primitiva *OCCAM*, **!** (*output*) ou a função em C, *chan_out*. Cada um dos argumentos significa respectivamente: a variável a ser enviada e o identificador do canal.
- ***alt_wait(chan_1, var_1, chan_2, var_2, ..., chan_I, var_I, flops)***, é uma função em *Matlab* para simular a primitiva de *OCCAM*, *ALT* ou a função em C, *alt_wait*. Esta função, permite seleccionar apenas um dos canais especificados (*chan_1, chan_2, ..., chan_n*), para entrada de dados (aquele que tiver recebido primeiramente os dados), dados estes que serão colocados na respectiva variável do espaço de trabalho do processo.
- ***[dest_ident, source_ident, number_of_variable]=processes_id (chan_i)*** é uma função em *Matlab* que retorna os identificadores dos processos de origem e destino, além da posição da variável no espaço de trabalho do processo de destino, associados ao canal chan_i.
- ***HAS_REC_FROM***, é uma variável global, a qual retorna o identificador do processo emissor correspondente ao último alt_wait do respectivo processo.

Para salvaguardar e restaurar o espaço de trabalho do processo existem as seguintes funções, também disponíveis ao utilizador:

- ***save_workspace (proc_number, i1, i2, ..., i21)***, é uma função em *Matlab*, a qual tem como parâmetros: o número do processo, e as suas variáveis locais. A seguinte ordem deve ser seguida para as variáveis: primeiramente a variável de controlo, em seguida os vários identificadores dos canais associados ao processo, e finalmente as respectivas variáveis locais do processo.
- ***[o1, o2, ..., o21] = restore_workspace (proc_number)***, é uma função em *Matlab*, que tem como parâmetro de entrada o número do processo e retorna as variáveis locais associadas com o respectivo processo. A ordem das variáveis locais deve ser idêntica à utilizada na função anterior.

Finalmente:

- ***finish (proc_number, flops)***, é uma função em *Matlab*, usada para informar o simulador de que o processo terminou a sua execução.

Esquema

4.3.2 Funções internas da *Toolbox*

- ***change_workspace(var, proc_number, number_of_variable)***, é uma função *Matlab*, usada para alterar o conteúdo da variável, cuja posição no espaço de trabalho relacionado com o processo *proc_number* é dado por *number_of_variable*, com a variável *var*.
- ***choose_plot(how_many, i)***, é uma função *Matlab* para selecção de gráficos.
- ***[t_on, t_off, x, y]=cpu_time(p, op, ini_time, end_time)***, é uma função *Matlab*, usada para retornar tempos em que os processadores se encontram activos e inactivos assim como os

processos que neles se encontram. A utilização de p permite determinar processos alocados a processadores.

- ***[x, y]=get_events(proc, ini, fim)***, é uma função *Matlab* para criação do eixo dos tempos e o eixo dos eventos para o processo proc, durante o intervalo de *ini* a *fim* segundos.
- ***time_lapse=getTime(quant, flo_com, chan, proc)***, é uma função *Matlab* que retorna o tempo gasto para efectuar operações em vírgula flutuante (se *flo_com*=0) ou comunicar um vector com elementos reais, através do canal *chan*. A utilização de *chan* permite determinar se é um canal interno ou externo , enquanto que *proc* determina o processo de origem associado ao canal para assim se saber o tipo de processador e o *FLOP_TIME* respectivo.
- ***insert_e(event, proc_numb, time).*** Esta função coloca o evento *event* numa lista associada ao processo *proc*, permitindo guardar a história do processo ao longo do tempo .
- ***insert_syst_events(process, time)***. Esta função é responsável pela inserção de processos *ready* na respectiva fila de espera. As filas são ordenadas por tempo, e se existe na mesma fila de espera um processo com o mesmo tempo de reactivação, o novo processo será inserido após esse processo.
- ***op=operation(proc_numb)***. Esta função é utilizada para retornar o identificador da função a executar.
- ***len=put_u(chan,nv);*** Esta função coloca a variável *nv* no espaço de trabalho do processo de destino.
- ***rendez_vous(chan),*** esta função indica de que a comunicação associada com o canal *chan* foi executada.
- ***[op, proc_numb]=retr_syst_events***, esta função é responsável por retirar o primeiro processo das filas ready, e retomar a sua execução no ponto onde foi interrompido.

Também insere na lista de eventos do processo uma confirmação do tempo da nova execução.

- ***sig_w_re(chan, number_of_variable)*** esta função coloca a indicação que o processo de destino relacionado com o canal *chan* pretende receber dados, através da variável do processo de destino que possui a posição *number_of_variable* no seu espaço de trabalho.
- ***sig_w_se(chan),*** esta função coloca a indicação que o processo de origem relacionado com o canal *chan* pretende enviar dados.
- ***store_u(chan, u)***; esta função coloca a variável *u* num espaço de armazenamento temporário associado com o canal *chan*.
- ***tes_alt(chan),*** esta função testa se este canal *chan* está a ser usado por uma instrução de alt_wait.
- ***has_got=tes_alts(chan, time),*** esta função testa se existe um *par_send* associado com o canal *chan*. Se existe, *has_got* será verdadeiro Se não houver *par_send*, retorna o valor de falso.
- ***flag=wait_rec(chan)***, esta função retorna um valor booleano, o qual indica se o canal *chan* está a ser utilizado para recepção de uma mensagem.
- ***flag=wait_sen(chan),*** esta função retorna um valor booleano o qual indica se o canal está *chan* está a ser utilizado para envio de uma mensagem.
- ***[x]=strcat(a,b),*** esta função permite concatenar duas *strings* a e b e retornar o resultado em x.
- ***[x]=strcount(a),*** esta função conta o número de ocorrências de a, e devolve x.
- ***[x,y]=strcut(a,b),*** esta função permite eliminar as ocorrências de *b* na *string a*, devolvendo em *x* a *string a* sem a *string b*, e em *y* o restante.

A última função aqui apresentada, foi especialmente útil para eliminar vírgulas no conjunto de variáveis locais comentadas, incluídas no código de cada função *Matlab,* editada pelo utilizador e deste modo isolá-las.

4.3.3 Interligação de funções

Neste parágrafo são apresentadas árvores lógicas que permitem ilustrar o relacionamento existente entre as funções internas e externas bem como a interligação dos vários componentes da simulação. O método seguido utiliza estruturas de controlo iterativas e condicionais conforme se pode ver nas duas figuras a seguir:

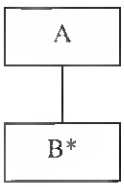


Fig. 18 – Estrutura Repetitiva

Esta representação indica que a entidade **A** é constituída por várias ocorrências de **B**.

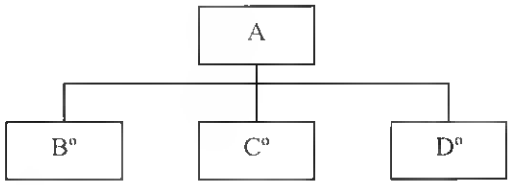


Fig. 19 – Estrutura Condicional

Esta representação indica que a entidade **A** é constituída por **B**, **C** ou **D** se e só se uma das condições for verdadeira. Para o caso de todas as condições serem verdadeiras **A** será constituído por **B**, **C** e **D**.

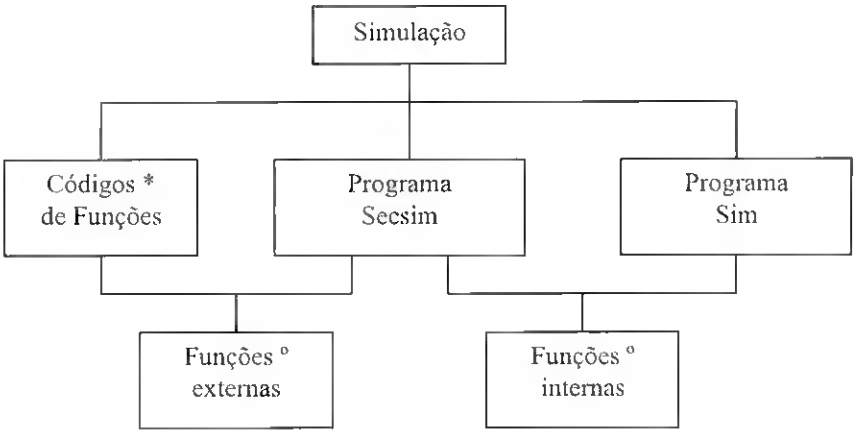


Fig. 20 –Árvore lógica da simulação em geral

Pode-se constatar da figura acima que quer os códigos de funções quer o programa *Secsim* de inicialização da simulação acedem a funções externas. As funções internas são acedidas pelo programa de execução da simulação *Sim* e também pelo programa *Secsim.*

A seguir é apresentada uma figura que ilustra a sequência lógica de acesso a funções externas pelo código de cada função:

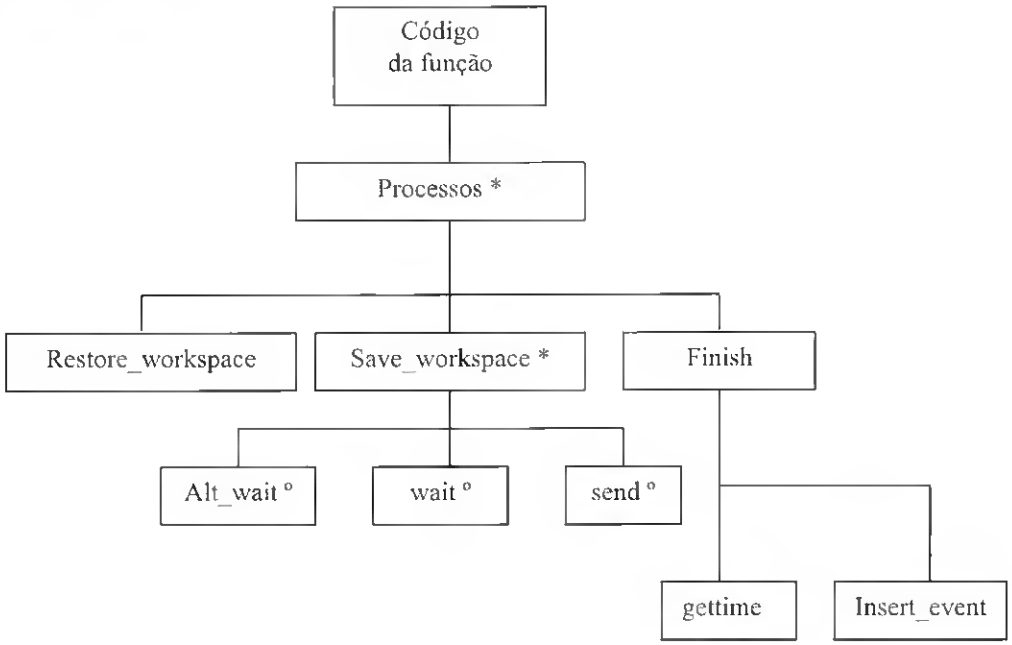


Fig. 21 –Árvore lógica de ligações entre funções externas

A figura 21 apresentada na página anterior indica que cada código de função pode ser acedido por vários processos. Um processo obrigatoriamente inclui um *restore_workspace*, um *finish* e várias ocorrências de *save_workspace*. Cada *save_workspace* situa-se antes de *alt_wait*, *wait* ou *send*. Estas funções de comunicação de processos são apresentadas nas 3 figuras que se seguem, e o respectivo código em *Matlab* encontra-se no Anexo C desta tese.

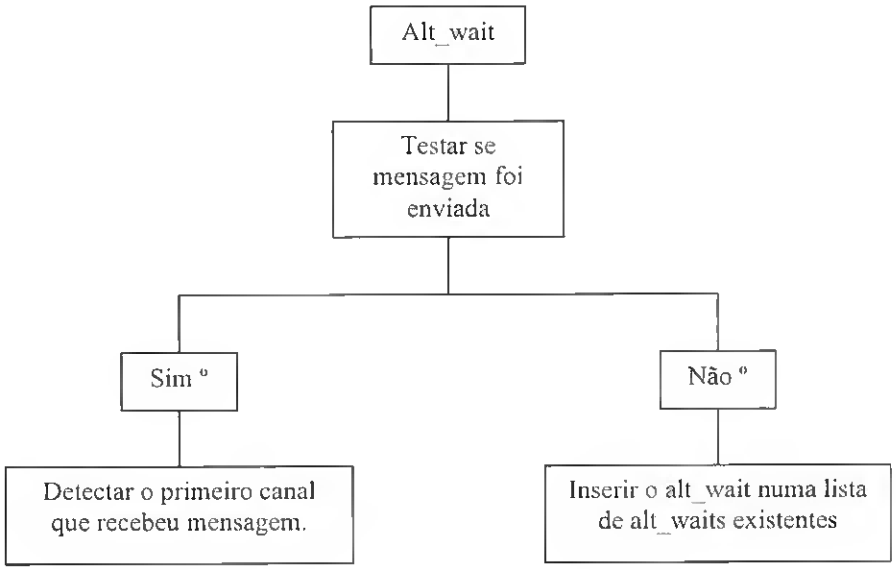


Fig. 22 - Árvore lógica da função *alt_wait*

Testar se mensagem foi enviada, acede para o efeito à função interna *wait_sen*;

Detectar o primeiro canal que recebeu a mensagem, acede posteriormente à função *wait*;

Inserir o alt_wait numa lista de alt_waits existentes, utiliza para o efeito as funções *processes_id* e posteriormente *wait*;

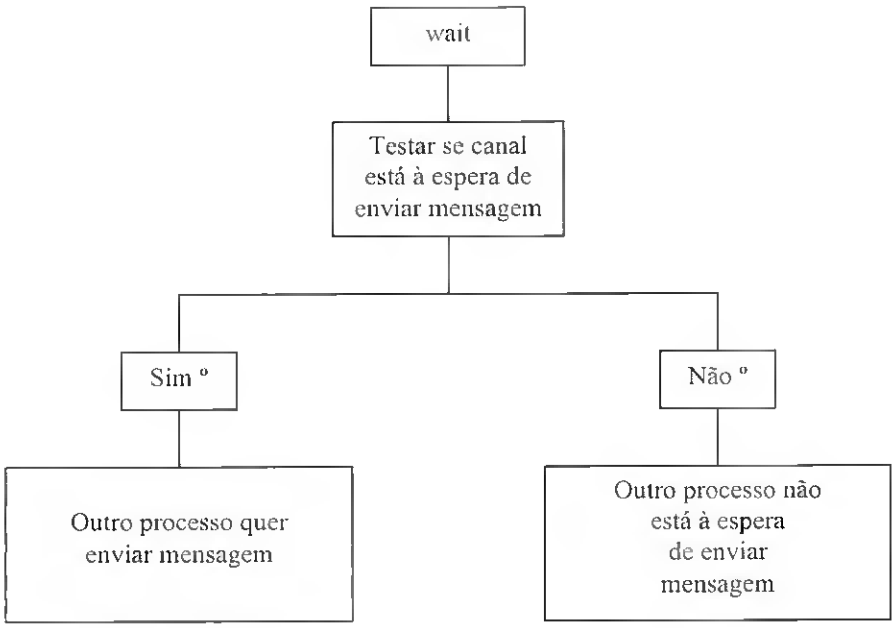


Fig. 23 - Árvore lógica da função wait

Testar se canal está à espera de enviar mensagem, acede para o efeito à função interna *wait_sen*;

Outro processo quer enviar mensagem , utiliza a sequência de funções *rendez_vous, processes_id, put_u, gettime, insert_event, insert_syst_events;*

Outro processo não está à espera de enviar mensagem, utiliza a sequência de funções: *sig_w_re, processes_id, gettime, insert_event.*

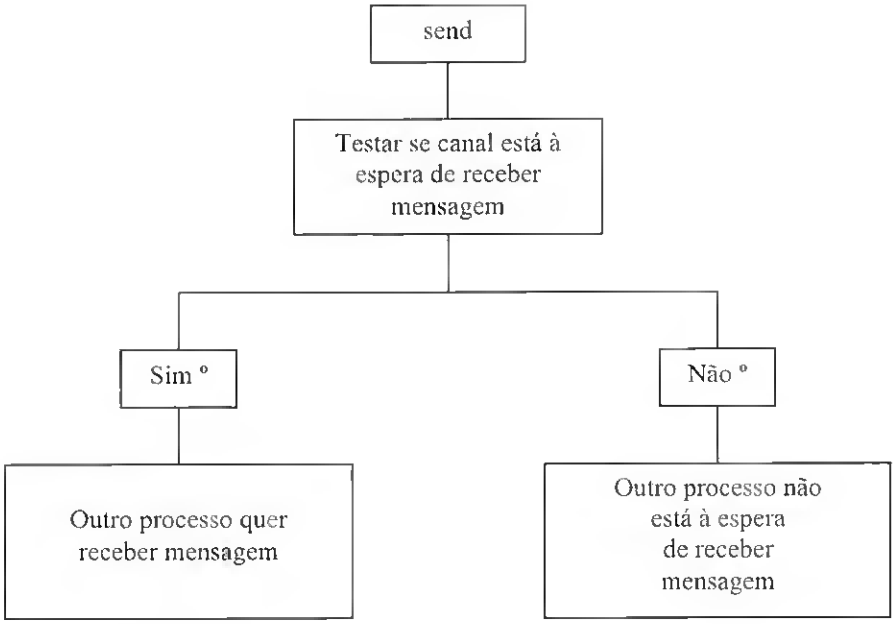


Fig. 24 - Árvore lógica da função send

O **Testar se canal está à espera de receber mensagem** è feito pela utilização da função *wait_rec;*

Outro processo quer receber mensagem , utiliza a sequência de funções *rendez_vous, processes_id, change_workspace, gettime, insert_event, insert_syst_events, tes_alt e tes_alts*;

Outro processo não está à espera para enviar mensagem, utiliza a sequência de funções *sig_w_re, store_u, processes_id, gettime, insert_event.*

A figura que se segue refere-se à ligação dos componentes do programa *Secsim*:

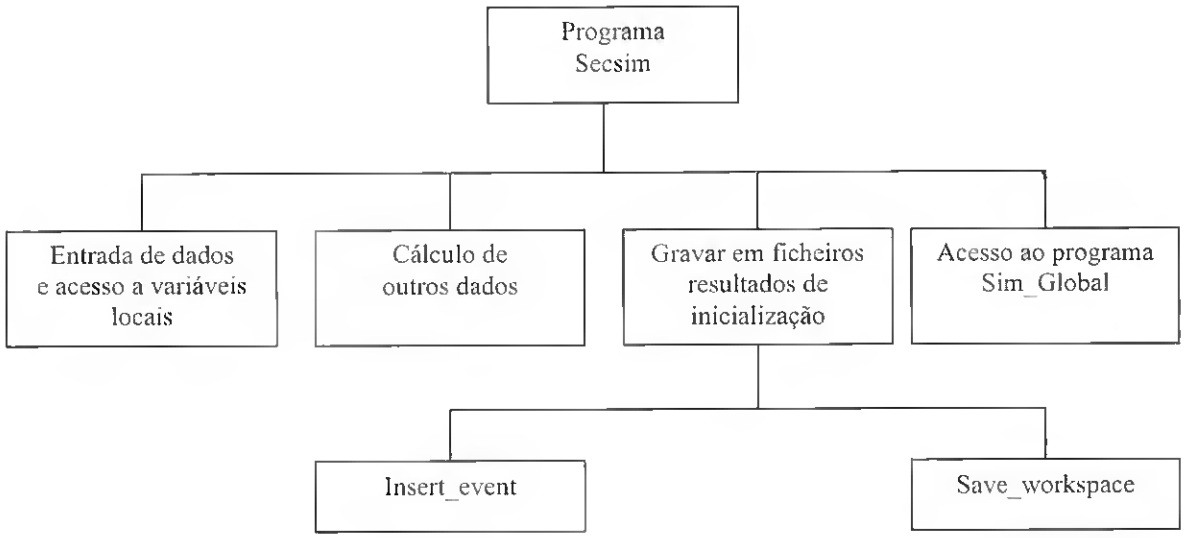


Fig. 25 –Árvore lógica de ligação de componentes do programa *Secsim*

Entrada de dados e acesso a variáveis locais, corresponde aos dados introduzidos pelo utilizador relativos a variáveis globais como: NUM_FUN, FUN_ID, FUNCTIONS, NUM_PROCESSORS, ID-TOPOLOGY, ID_PROCESSOR, além de outras. O número e respectivos identificadores das variáveis locais inicializadas e não inicializadas são lidos de cada um dos códigos de funções existentes e colocados em NUM_LOCAL_VAR, NUM_LOCAL_VAR_INI, VAR_ID, VAR_ID_INI.

Cálculo de outros dados, permite identificar ligações entre processos e processadores. Estes resultados são depois incluídos em variáveis globais como: CHANNEL, CHAN_ID, MAX_P_PRIO, PRIORITARIO, além de outras.

Gravar em ficheiros resultados de inicialização, significa que o programa *Secsim* salvaguarda em ficheiros resultados sobre as variáveis locais previamente lidas das funções, inicializa cada um dos processos e seus canais lógicos e inclui instruções relacionadas com algumas funções externas e internas da *toolbox*, tais como: *save_workspace* e *insert_event*.

Acesso ao programa Sim_Global, acede a variáveis globais incluídas no programa Sim_Global.

A figura a seguir refere-se à ligação dos componentes do programa *Sim*:

Fig. 26 –Árvore lógica de ligação de componentes do programa *Sim*

Entrada de identificadores de ficheiros, refere-se à introdução pelo utilizador de identificadores de ficheiros gerados pelo programa anterior *Secsim*.

Entrada de dados, tem a haver essencialmente com dados que o utilizador deverá introduzir relativos a: tempos de inicio e fim da simulação corrente, além de outros para que no cálculo de resultados e gráficos estes valores sejam considerados.

No cálculo de **Resultados e gráficos,** são acedidas a maior parte das funções internas da *Toolbox* para o cálculo de medidas de desempenho, apresentação de gráficos, etc.. Algumas dessas funções são *retr_syst_event*, *show_t*, que por outro lado acedem a *cpu_time*, *choose-plot*, *spet_plot* e *get events*.

Gravar em ficheiro dados da simulação, é opcional dado que é o utilizador quem decide, salvaguardar ou não dados para posterior utilização.

4.4 Passos do Utilizador para Execução dos Programas de Simulação

O utilizador antes de iniciar a execução dos programas *Secsim e Sim,* deve passar pelas etapas indicadas a seguir para a concepção de uma aplicação paralela:

1. projectar inicialmente, de preferência em papel a arquitectura de destino, quer se trate de um sistema homogéneo ou heterogéneo;
2. particionar o algoritmo sequencial em módulos;
3. definir a topologia a utilizar na paralelização do algoritmo;
4. codificar o algoritmo particionado em funções *Matlab.*

Seguidamente, o utilizador deverá:

1. executar o programa *Secsim* de inicialização do simulador, para definir a topologia e a alocação dos processos à arquitectura simulada. Deste modo, cada processo, de alta ou baixa prioridade, a ser executado num processador específico, irá aceder ao código de determinada função;
2. e finalmente para avaliar o desempenho do algoritmo paralelo, executar o programa de simulação *Sim.*

4.4.1 Desenho da topologia lógica e física

Nesta etapa, o projectista deverá ter em atenção a atribuição dos processos aos diferentes processadores do sistema, além do número de ligações lógicas e físicas entre processos executados em processadores da topologia. A convenção quanto ao número dos canais existentes na comunicação de processos é definida pelo utilizador.

É apresentado um exemplo, supondo que as designações para processos de alta e baixa prioridade e processadores do sistema são as seguintes:

- *HP(1)*, processo 1 de alta prioridade e *HP(2)*, processo 2 de alta prioridade;
- *LP(1)*, processo 1 de baixa prioridade e *LP(2)*, processo 2 de baixa prioridade;

- *Trans*, processador 1 e *C40*, processador 2.

Na figura a seguir é apresentado um esquema demonstrativo:

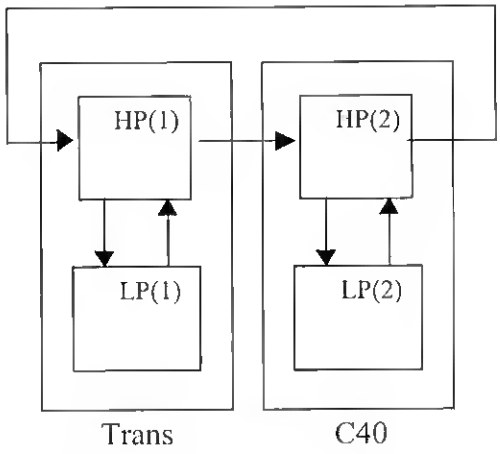


Fig. 27 – Topologia heterogénea com 2 processadores

4.4.2 Instruções a incluir no código de cada função *Matlab*

Suponhamos que queremos simular a topologia apresentada na figura 27. A sintaxe para o código de cada função *Matlab* editada pelo utilizador deverá obrigatoriamente incluir, pela ordem indicada, o seguinte (supondo a existência de um número de variáveis locais necessárias, **i** entre as quais algumas se encontram inicializadas, **j**, onde i >j):

1. Identificação da função, através da sintaxe:

function identificador_função(número_processo)

2. Variáveis locais inicializadas, comentadas, utilizando a sintaxe:

%+ id_var_local_1=valor, id_var_local_3=valor, …, id_var_local_j=valor+%

3. Variáveis locais existentes na função, comentadas, utilizando a sintaxe:

%- id_var_ local_1, id_var_local_2, id_var_local_3, id_var_local_i-%

4. Variáveis globais necessárias;

5. Código da função;

6. Dependendo do tipo de prioridade da função, alta ou baixa, devem ser realizadas operações específicas para cada uma delas. No caso de processos de alta prioridade, dever-se-ão limitar ao mínimo instruções que não as necessárias ao encaminhamento de dados, utilizando funções externas ao simulador, tais como: *alt_wait* ou *wait* e *send.* Quanto às funções de baixa prioridade, realizam todas as operações de cálculo necessárias e utilizam de igual modo algumas das funções referidas para recepção de dados e envio de resultados;

4.5 Programa de Inicialização do Simulador

Realizou-se um melhoramento à versão anterior, no sentido em que, onde o utilizador tinha de inserir manualmente nos ficheiros que implementam os processos as funções externas disponíveis no simulador, estas são agora automaticamente actualizadas. As actualizações da variável de controlo para a correcta execução do processos são também inseridas automaticamente. Foi ainda facilitada a inicialização das variáveis locais de cada processo. Considerou-se a necessidade de existirem ligações físicas apenas entre processos prioritários atribuídos em diferentes processadores.

Todo o processo de execução do programa *Secsim* relaciona-se essencialmente com: funções existentes, processos que utilizam os códigos das várias funções, prioridades, número de canais lógicos dos processos e a alocação de processos aos processadores disponíveis.

É apresentado a seguir um exemplo, da execução do ficheiro de inicialização baseado na topologia da figura 27, onde se pretende simular uma arquitectura heterogénea constituída por 2 processadores, um *Transputer* e um *C40*, com 2 processos cada, em que o utilizador digitaria o que se encontra a *bold*:

*» **Secsim***

*how many different functions=**2***

FUNCTION 1

String identifier = ***'HP'***

*How many processes uses this function=**2***

*How many channels for this function=**4***

*High(1) or Low(0) priority? **1***

FUNCTION 2

String identifier = ***'LP'***

*How many processes uses this function=**2***

*How many channels for this function=**2***

*High(1) or Low(0) priority? **0***

*How many processors=**2***

*Transputers(1), C40s(2), Transputers and C40s(3), other processors(4)=**3***

*Processor identifier (trans(1) or C40(2) or other(3))= **1***

*Processor identifier (trans(1) or C40(2) or other(3))= **2***

Processes which execute the code HP

*In which processor is HP(1) executing ? **1***

Channels associated with HP(1)

*Is HP(1) the source(1) or dest. (0) of its channel numb. 1 ? **1***

*To which code ? **1***

*To which of HP codes ? **2***

*Is HP(1) the source(1) or dest. (0) of its channel numb. 2 ? **1***

*To which code ? **2***

*To which of LP codes ? **1***

*Is HP(1) the source(1) or dest. (0) of its channel numb. 3 ? **0***

*From which code ? **2***

*From which of the LP codes ? **1***

*Is HP(1) the source(1) or dest. (0) of its channel numb. 4 ? **0***

From which code ? ***1***

From which of the HP codes ? ***2***

In which processor is HP(2) executing ? ***2***

Channels associated with HP(2)

HP(2) is already sending to HP(1)

HP(2) is already receiving from HP(1)

Is HP(2) the source(1) or dest. (0) of its channel numb. 3 ? ***1***

To which code ? ***2***

To which of LP codes ? ***2***

Is HP(2) the source(1) or dest. (0) of its channel numb. 4 ? ***0***

From which code ? ***2***

From which of the LP codes ? ***2***

Processes which execute the code LP

In which processor is LP(1) executing ? ***1***

Channels associated with LP(1)

LP(1) is already sending to HP(1)

LP(1) is already receiving from HP(1)

In which processor is LP(2) executing ? ***2***

Channels associated with LP(2)

LP(2) is already sending to HP(2)

LP(2) is already receiving from HP(2)

Filename for the topology of system – ***'aaa'***

Filename for the initialization – ***'aaa'***

*How many lines=**30***

*How many columns=**20***

»

4.6 Ficheiros Gerados Automaticamente

4.6.1 Ficheiro de inicialização

Como resultado da execução do programa *Secsim, é* gerado automaticamente um ficheiro *Matlab* de inicialização. Na versão aqui apresentada, o utilizador não necessita de se preocupar com a edição no conteúdo deste ficheiro de variáveis locais dos processos e respectivas inicializações, nem com a inclusão de outras funções especiais, pois todo este processo é gerado automaticamente. Ele apenas terá de editar o conteúdo das funções *Matlab,* obrigatoriamente com a sintaxe apresentada em 4.4.2.

É apresentado abaixo, o código do ficheiro de inicialização *aaa_ini* gerado automaticamente pelo programa Secsim, do exemplo da figura 27 apresentado neste capítulo bem como a seguir uma breve explicação ao mesmo:

```
%  This file initializes the simulation of the transputer,C40 or mixed systems.
%
sim_glob;
load aaa.top -mat;
num_working =2;
num_lines =30;
num_col =20;
num_col_proc =10;
i=1;
u=0;
sweep=1;
next_column_in_sweep=1;
first_row=1;
sig=0;
sigg=0;
u1=0;
den=0;
R=rand(30,20);
%More initializations for process HP(1)
% chan 1 has destination HP(2)
chan1=1;
% chan 2 has destination LP(1)
chan2=2;
% chan 3 comes from HP(2)
chan3=4;
% chan 4 comes from LP(1)
chan4=3;
after=0;
% (continue)
```

```
% (continuation)
save_workspace(1,chan1,chan2,chan3,chan4,after,u,i,num_col,num_working);
insert_event(EVENT_START_EXE,1,0);
HP(1);
% More initializations for process HP(2)
% chan 1 has destination HP(1)
chan1=4;
% chan 2 has destination LP(2)
chan2=5;
% chan 3 comes from HP(1)
chan3=1;
% chan 4 comes from LP(2)
chan4=6;
after=0;
save_workspace(2,chan1,chan2,chan3,chan4,after,u,i,num_col,num_working);
insert_event(EVENT_START_EXE,2,0);
HP(2);
% More initializations for process LP(1)
%
% chan 1 has destination HP(1)
chan1=3;
% chan 2 comes from HP(1)
chan2=2;
after=0;
save_workspace(3,chan1,chan2,after,sweep,num_col_proc,next_column_in_sweep,num_working,first_row,nu
m_lines,R,u,sig,sigg,u1,den);
insert_event(EVENT_START_EXE,3,0);
LP(3);
% More initializations for process LP(2)
% chan 1 has destination HP(2)
chan1=6;
% chan 2 comes from HP(2)
chan2=5;
%
after=0;
%
save_workspace(4,chan1,chan2,after,sweep,num_col_proc,next_column_in_sweep,num_working,first_row,nu
m_lines,R,u,sig,sigg,u1,den);
insert_event(EVENT_START_EXE,4,0);
LP(4);
%
```

O código do ficheiro acima é assim explicado nos pontos apresentados a seguir:

1. Invoca o programa *sim_glob*, onde se encontram inicialializadas variáveis globais necessárias à simulação;
2. Carrega o ficheiro *aaa.top* da topologia do sistema em memória;

3. Inicializa variáveis locais de cada processo entre as quais consta a declaração do número de processadores existentes, além de outras;
4. Inicializa cada um dos processos *HP(1)*, *HP(2)*, *LP(1)* e *LP(2)*, da seguinte forma:

   4.1. gera os identificadores e respectivos valores de cada um dos canais de origem e destino;

   4.2 grava no espaço de trabalho do processo através da função *save_workspace*, o número do processo, os identificadores dos canais, a variável de controlo *after*, o número de processadores, além das restantes variáveis locais;

   4.3 invoca a função *insert_event* para colocar o evento EVENT_START_EXE associado ao processo na lista do seu historial;

   4.4 por fim invoca o processo.

4.6.2 Ficheiro de dados

A execução do programa Secsim gera também os dados resultantes da inserção de valores nas variáveis globais existentes.

A título de exemplo são apresentados a seguir apenas alguns desses dados:

```
» load aaa.top -mat
» FUNCTIONS=
   1   2   4   1   2   4   2
   3   2   2   0   2  12   8

» PROCESSES =
   1   1   1
   2   1   1
   1   0   2

» FUN_ID =
HP
LP

» PROCESSOR=
   3   1   1   2   0   0   0
   3   2   1   1   0   0   0
» PRIORITARIO
```

2 0 2 1 2

» ID_TOPOLOGY=

3

» CHANNEL=

1 2 0 0 0

1 3 0 0 0

3 1 0 0 0

2 1 0 0 0

2 4 0 0 0

4 2 0 0 0

4.6.3 Ficheiro de configuração

A execução do programa *Secsim,* gera de igual modo automaticamente um ficheiro de configuração, para que o utilizador possa posteriormente testar numa arquitectura paralela real a simulação realizada.

É apresentado a seguir o resultado do ficheiro de configuração config.cfg tendo como base o exemplo da figura 27 apresentado neste capítulo.

```
!       Ficheiro de configuração para topologias mistas =3
!       Hardware configuration
processor host type=pc
processor root type=t800
processor wrk type=C40 kernel="slot?.krn" ! Plese edit the number of slot in ?
wire ? root[0] host[0]
wire ? root[2]wrk[0]
!       Software configuration
task afserver ins=1 outs=1
task filter ins=2 outs=2 ! if necessary edit data=
! its necessary create two files for rooter processes and igually two files for calc processes
! one for send the data for the others.
!for exemple you can try this code and modify it if necessary:
task HP1 ins=3 outs=3 priority =0! if necessary edit stack, heap, opt
task HP2 ins=2 outs=2 priority =0! if necessary edit stack, heap, opt
task LP1 ins=1 outs=1 priority=1 ! if necessary edit data
task LP2 ins=1 outs=1 priority=1 if necessary edit stack, heap, opt
! (continue)
```

```
(continuation)
place afserver host
place filter root
place HP1 root
place HP2 wrk
place LP1 root
place LP2 wrk
default connect physical
connect ? afserver[0] filter [0]
connect ? filter[0] afserver [0]
! the rest of connections between the other processes is not made in current version
! but we try to make them in the next version of simulator
```

4.6.4 Resultados gerados pelo programa de simulação, *Sim*

Após finalizada a simulação toda a informação resultante pode ser visualizada em gráficos *Matlab*, para análise do utilizador. Esses gráficos são apresentados no próximo capítulo tendo como base exemplos de arquitecturas homogéneas de *transputers* ou C40s e heterogéneas com o mesmo tipo de processadores.

4.7 Conclusões

Neste capítulo descrevem-se as actuais funções e potencialidades do simulador *Secsim*.

Primeiramente optou-se pela caracterização de todas as variáveis globais, seguindo-se a apresentação de funções externas e internas, bem como o seu encadeamento.

Foram igualmente indicados cada um dos passos a seguir pelo utilizador para utilização do grupo de programas existentes e incluídos na *toolbox* em *Matlab*, através de um exemplo elucidativo.

No próximo capítulo são apresentados em detalhe alguns exemplos, bem como a análise dos resultados obtidos pelo programa de simulação *Sim*.

5 EXEMPLOS

5.1 Introdução

Com este capítulo, pretende-se ilustrar o funcionamento da *package* através de exemplos. Numa primeira fase apresenta-se um exemplo didáctico, com a finalidade de ilustrar os ficheiros gerados automaticamente por *SecSim*. Em seguida, utilizando um caso de teste, a triangularização de uma matriz, apresentam-se os resultados que podem ser obtidos através da simulação de algoritmos paralelos em arquitecturas homogéneas e heterogéneas de *Transputers* e *C40s*.

Conforme já previamente referido, para simular os tempos de comunicação é necessário fornecer ao simulador o tempo de comunicação de uma variável real, entre os diferentes processadores. Para isso foram executados testes em arquitecturas paralelas reais, que serão apresentados neste capítulo nas secções relevantes.

5.2 Exemplo Didáctico

Vamos admitir a simulação de dois processos, identificados por P1 e P2, que comunicam através de canais C1_para_C2 e C2_para_C1:

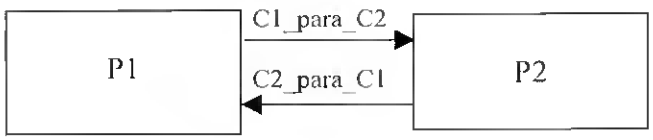


Fig. 28 - Topologia para o exemplo didáctico

Assume-se que P1 e P2 teriam o código *OCCAM* apresentado na página seguinte:

```
PROC P1(C1_para_C2,C2_para_C1)
  --- variáveis u1,v1;
  SEQ
    --- primeiras instruções
    C1_para_C2 ! u1
    --- instruções medianas
    C2_to_1 ? v1
    --- últimas instruções
```

```
PROC P2(C1_para_C2,C2_para_1)
  --- variáveis u2,v2;
  SEQ
    --- primeiras instruções
    C1_to_2 ? u2
    --- instruções medianas
    C2_to_1 ! v2
    --- últimas instruções
```

Após se ter definido a topologia executando *SecSim*, o ficheiro de inicialização gerado automaticamente por este programa apresentaria as seguintes linhas:

```
% Inicializações para o processo P1(1)
% chan 1 tem como destino Proc_2(1)
chan1=1;
% chan 2 vem de Proc_2(1)
chan2=2;
after=0;
% Aqui outras inicializações
save_workspace(1,chan1,chan2,after,u1,v1);
insert_event(EVENT_START_EXE,1,0);
P1(1);
% Inicializações para o processo P2(1)
% chan 1 has destination Proc_1(1)
chan1=2;
% chan 2 comes from Proc_2(1)
chan2=1;
after=0;
% Aqui outras inicializações
save_workspace(2,chan1,chan2,after,...);
insert_event(EVENT_START_EXE,2,0);
P2(2);
```

Os códigos *Matlab* para P1 e P2 correspondentes seriam semelhantes aos apresentados abaixo, onde as instruções em itálico representariam o que o utilizador digitou, e o resto das instruções seriam inseridas automaticamente como resultado da execução do ***SecSim***:

```
function P1(proc_numb)
%+ u1=0 +%
%- u1, v1 -%
[chan1,chan2,after,u1,v1]=restore_workspace(proc_numb)
ff=flops;
if after==0
  primeiras instruções
 after=1;
 save_workspace(proc_numb ,chan1,chan2,after,u1, v1)
 send(chan1, u1,flops-ff)
 return
end
if after==1
 outras instruções
 after=2;
 save_workspace(proc_numb ,chan1,chan2,after,u1, v1)
 wait(chan2,u2, flops-ff,5)
 return
end
 últimas instruções
finish(proc_numb,flops-ff)
```

```
function P2(proc_numb)
%+ u2=0 +%
%- u2, v2 -%
[chan1,chan2,after,u2,v2]=restore_workspace(proc_numb)
ff=flops;
if after==0
  primeiras instruções
 after=1;
 save_workspace(proc_numb ,chan1,chan2,after,u2, v2)
 wait(chan2,u2, flops-ff,4)
 return
end
if after==1
 outras instruções
 after=2;
 save_workspace(proc_numb , chan1,chan2,after,u2, v2)
 send(chan2, v2,flops-ff)
 return
end
 últimas instruções
finish(proc_numb,flops-ff)
```

Através da utilização do programa *Sim*, a execuçao do programa paralelo poderia seguidamente ser simulada.

5.3 Caso de Teste: Triangularização de uma Matriz

Para ilustrar a execução do simulador, são apresentadas duas paralelizações alternativas da fase de triângularização do algoritmo de Householder QR, descrito em detalhe em [7].

O problema aqui tratado pode formular-se do seguinte modo: Dada uma matriz **A**, de dimensões m*n, onde m≥n, tal factorização decompõe essa matriz numa matriz ortonormal **Q**, e numa matriz triangular superior $\mathbf{R}_1$, tais que:

$$\mathbf{A} = \mathbf{QR} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} \tag{4}$$

A fase de triangularização que envolve o cálculo da matriz $\mathbf{R}_1$ é traduzida pelo algoritmo 1, que se descreve na página seguinte:

Alg.1 – Fase de triangularização

1. for k=1 to n
   1.1 Cálculo do vector h de Householder
   1.2 $\mathbf{R}_{k\cdots m,i} = -\sigma \mathbf{e}_1$
   1.3 $\mathbf{r} = \mathbf{h}^T \mathbf{R}_{k\cdots m,k+1\cdots n}$
   1.4 $\mathbf{R}_{k\cdots m,k+1\cdots n} = \mathbf{R}_{k\cdots m,k+1\cdots n} - \mathbf{hr}$
end

Este algoritmo foi paralelizado em diferentes topologias, nomeadamente numa rede de *transputers*, numa rede de *DSPs*, e numa arquitectura mista.

5.3.1 Rede de *Transputers*

Foi utilizada uma rede de cinco *transputers*, numa topologia em anel. Em cada *transputer* existe um processo *router* de alta prioridade, responsável pelas comunicações, e um processo de baixa prioridade, *worker*, responsável pelos cálculos como se pode constatar na figura da página a seguir:

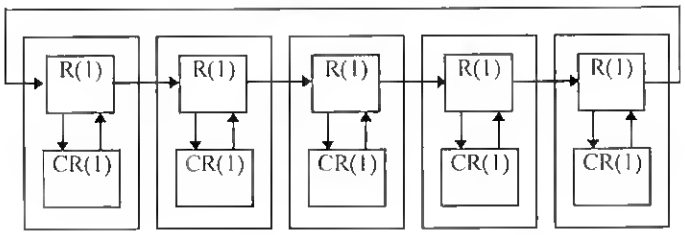


Fig. 29 – Topologia com 5 *Transputers*

A matriz **A** foi particionada em 5 partições iguais, onde a ordenação das colunas foi tal que a 1ª coluna de **A** coincide com a 1ª coluna da partição afecta ao processador 1, a 2ª coluna de **A** é a 1ª coluna da partição afecta ao processador 2, etc.

5.3.1.1 Medidas de desempenho para a paralelização em *Transputers*

Numa primeira aproximação, uma abordagem convencional foi considerada. Na iteração *k*, o processo que detém a coluna *k* computa o vector de *Householder*, transmite-o aos outros *workers* através dos *routers*, e todos, em paralelo, executam as operações 1.3 e 1.4 do Alg.1 para a parte da sua partição ainda não triângularizada.

	Tempo exec. seq.	**Tempo exec. par.**	***Speedup***
Simulada	0.98	0.25	3.9
Medida	0.999	0.264	3.7

Considerando o caso de uma matriz com dimensões m=100 e n=50, esta primeira aproximação [7], usando um processo *worker* convencional, alcança, no simulador, um tempo de execução sequêncial de 0.98 segundos, um tempo de execução paralelo de 0.25 segundos, e um *speedup* de 3.9. Os valores actuais, medidos numa rede de *transputers*, programados em *Occam*, são 999 mseg., 264 mseg. e 3.7. A Figura 30 mostra alguns exemplos do tipo de gráficos disponíveis no simulador. A figura a) ilustra a eficiência de cada processador,

definida como a relação entre o tempo efectivo de cálculo de cada processador e o tempo total da simulação. A figura b) mostra, para o processador 1, os períodos de funcionamento (1) e inactivos (-1) entre 0 e 60 mseg.. Para entender a razão da existência de períodos inactivos, um gráfico de eventos é representado na figura c), para os processos *router* e *worker* associados ao processador 1, no mesmo intervalo de tempo. As convenções utilizadas são: à espera de receber mensagem (2), actualmente a receber (1), a calcular (0), actualmente a enviar uma mensagem (-1), e à espera para enviar (-2). Quando são mostrados dois gráficos na mesma figura, como é o caso, ao gráfico do topo é adicionado o valor 3, e ao gráfico de fundo é subtraído o valor 3. Podem agora ser entendidas as razões da existência de tempos inactivos, dado que, na primeira iteração, o *worker* 1 cálcula o seu vector de *Householder*, envia-o (A) aos outros processadores, e continua o actualizar as suas colunas ainda não triângularizadas (C). Nas próximas 4 iterações, ele tem de esperar para receber (B) os vectores de *Householder* que são calculados pelos *workers* seguintes, e continuar o seu encaminhamento para os *workers* seguintes (D). A figura d) mostra os eventos equivalentes para o processador 5.

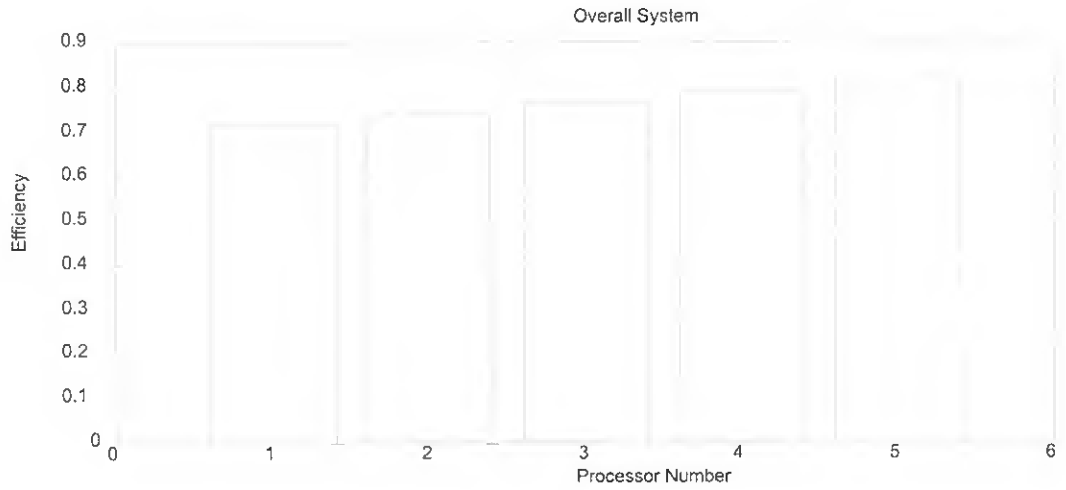


fig. a)

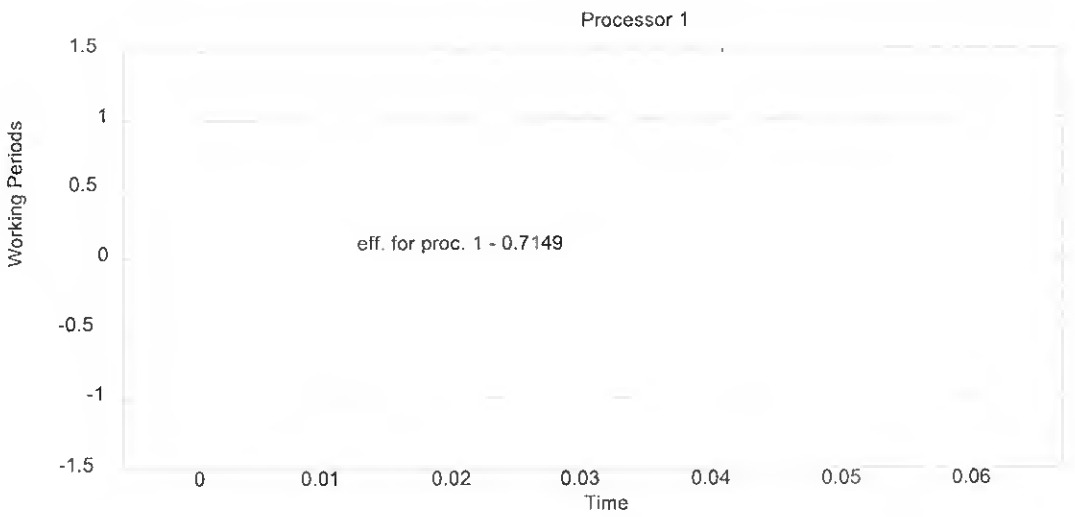


fig. b)

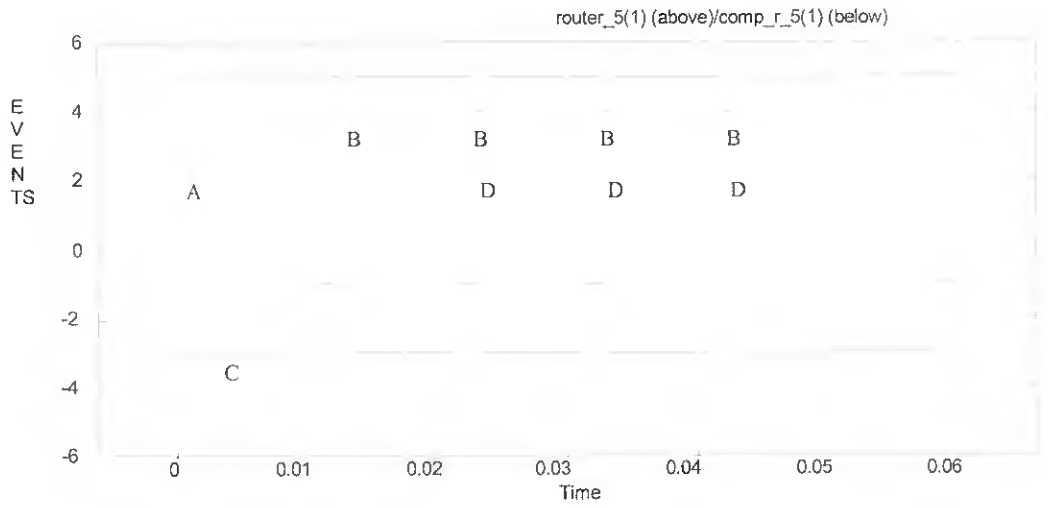


fig. c)

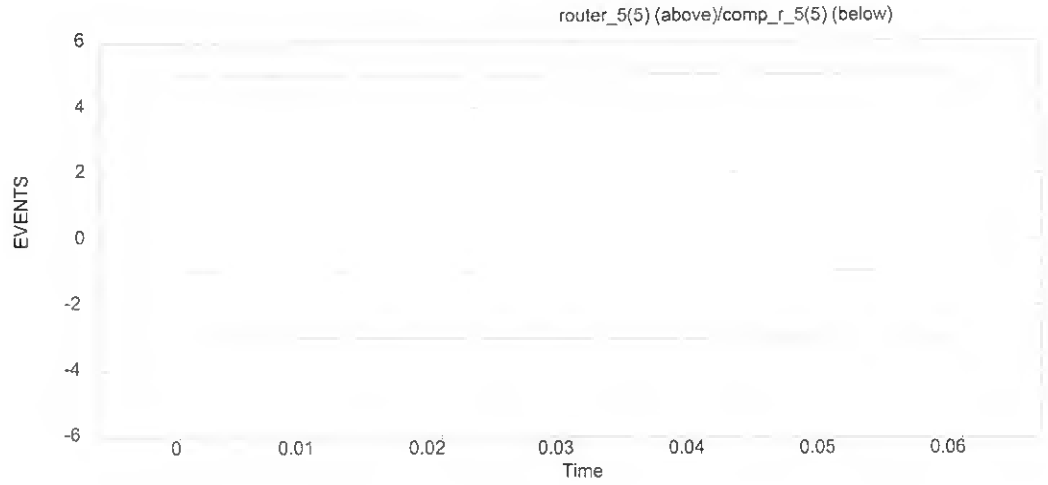


fig. d)

Fig. 30 - Gráficos da *toolbox Matlab* para a primeira aproximação (*Transputers*)

Utilizando agora um *worker* mais sofisticado [7], que calcula o seu vector *Householder* correspondente um passo à frente, outra simulação foi realizada, tendo-se obtido os seguintes tempos.

	Tempo Exe. Seq.	**Tempo Exe. Par.**	***Speedup***
Simulada	0.98	0.213	4.6
Medida	0.999	0.235	4.3

Agora o tempo de execução paralelo obtido é de 213 mseg. e resulta num *speedup* de 4.6. Os valores actuais, medidos na rede de *transputers*, são de 235 mseg. e 4.3. O mesmo tipo de gráficos da figura 30 são actualizados para esta segunda aproximação. Na figura 31 podemos observar que, devido a esta estratégia para o cálculo do vector de *Householde*r, o período de tempo que cada *worker* tem de esperar está bastante reduzido, ou até mesmo anulado, como nos casos da 5ª iteração, para o 1º processador, e nas iterações entre a 2ª e a 5ª, para o último processador.

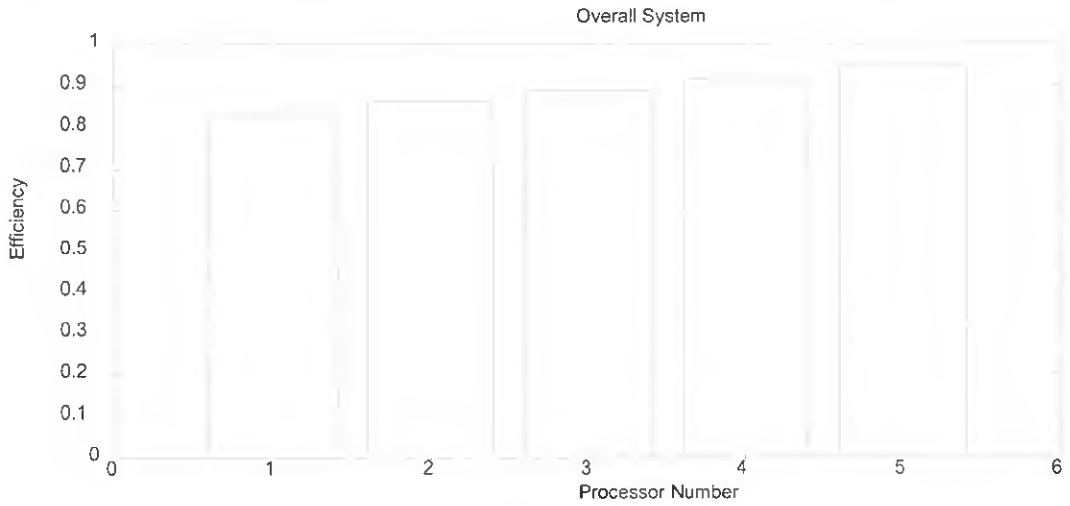


fig. a)

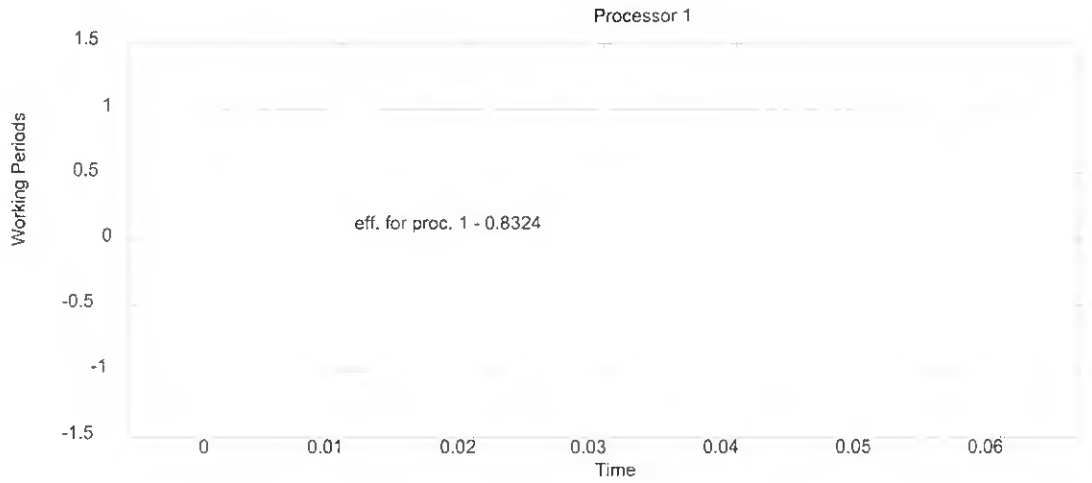


fig. b)

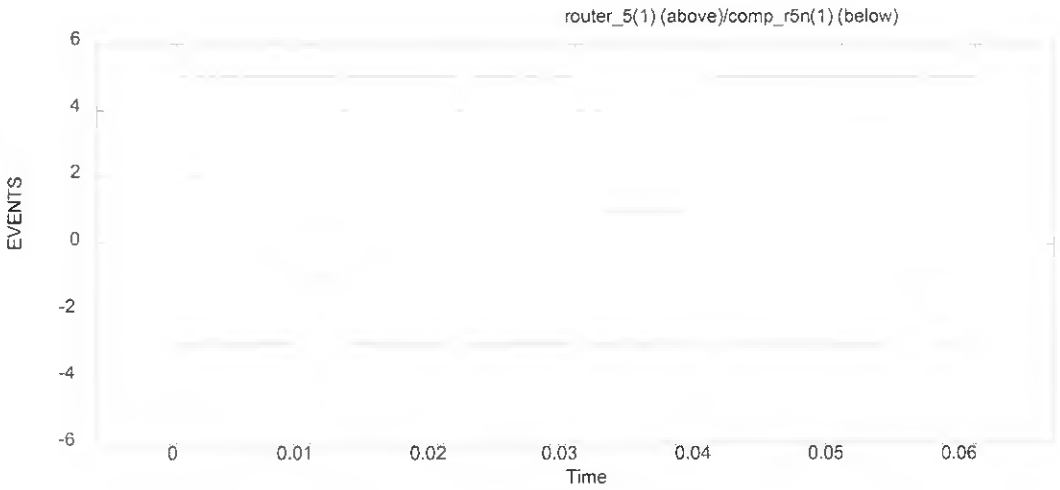


fig. c)

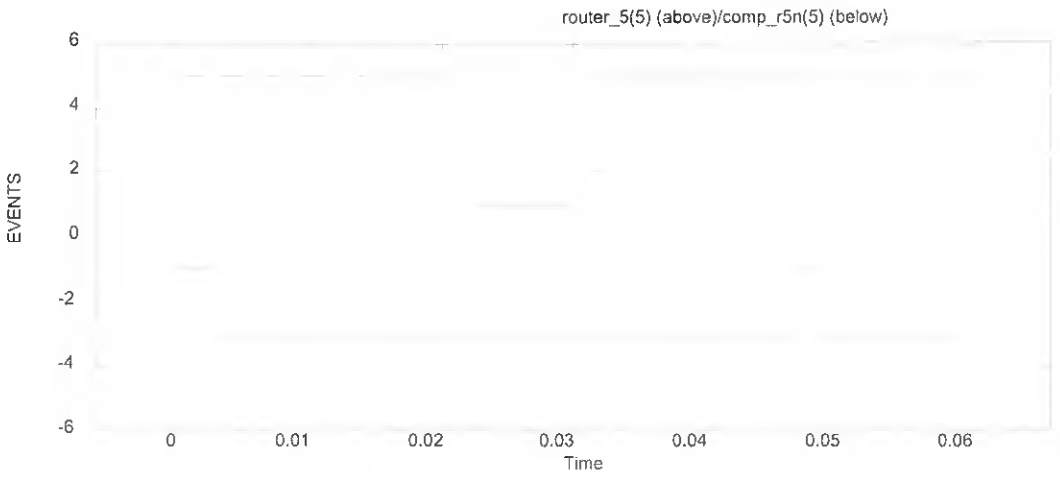


fig. d)

Fig. 31 - Gráficos da toolbox em *Matlab* para a segunda aproximação (*Transputers*)

5.3.1.2 Tempos obtidos na comunicação externa

Conforme já foi referido, é necessário determinar o tempo necessário para comunicar um real de *transputer* para *transputer*. Para isso foram medidos os tempos de comunicação de um vector de reais (variando o comprimento entre 10 e 100 elementos) entre 2 *transputers*. Os tempos (em μs) apresentados na figura 32 referem-se ao envio, e posterior leitura desses vectores. É pois evidente que o tempo para comunicar um real, entre um par de *transputers*, é da ordem de 2.3 μs, e o *setup time* é desprezável. O código utilizado apresenta-se no Anexo F.

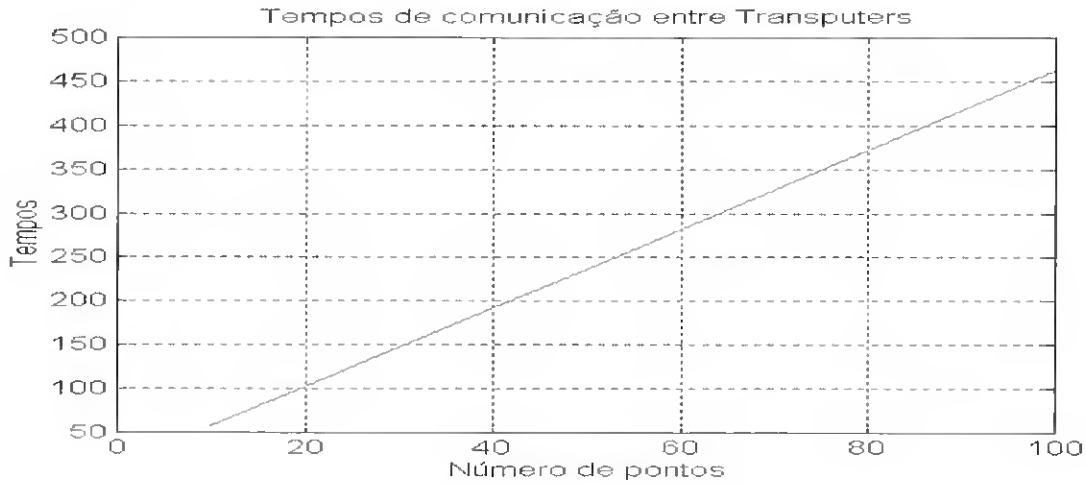


Fig. 32 –Gráfico de tempos obtidos na comunicação entre *Transputers*

Nos Anexos D e E são apresentados os programas sequencial e paralelo implementados neste tipo de processadores, mas apenas referentes a uma topologia com um máximo de 2 *transputers*. Considera-se para o caso um *flop time* de 1.8 μs, calculado com base na relação entre o tempo de execução do algoritmo sequencial num *transputer* e o número genérico de operações em virgula flutuante necessários para executar o código do respectivo algoritmo.

5.3.2 Rede de *C40s*

Foi neste caso utilizada uma rede de 2 *C40s*, numa topologia em anel. Em cada C40 existe um processo *router* de alta prioridade, responsável pelas comunicações, e um processo *worker* de baixa prioridade, responsável por cálculos como se pode constatar na figura abaixo:

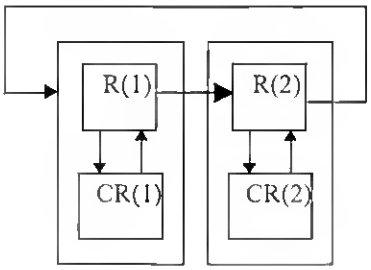


Fig. 33 – Topologia com 2 *C40s*

Utilizou-se para este caso o algoritmo paralelo convencional que consta no Anexo H. O algoritmo sequencial é o apresentado no Anexo G.

5.3.2.1 Medidas de desempenho para a paralelização

	Tempo Exe. Seq.	**Tempo Exe. Par.**	***Speedup***
Simulado	0.248	0.136	1.82
Medido	0.243	0.144	1.68

Considerou-se para o caso apresentado a triângularização de uma matriz com dimensões m=44 e n=44 em 2 *DSPs C40* com velocidades de 40MHz, usando a aproximação paralela convencional. A linguagem utilizada foi o *3L Parallel C*. Obteve-se um tempo de execução sequencial simulado de 0.248 segundos, um tempo de execução paralelo simulado de 0.136 segundos, e um *speedup* de 1.82. Os tempos reais medidos, na placa de *DSPs*, são respectivamente: 0.243 segundos, 0.144 segundos e um *speedup* de 1.68.

A figura 34 ilustra exemplos do tipo de gráficos disponíveis no simulador. O primeiro gráfico da figura a) mostra a eficiência paralela para cada processador, aproximadamente de 0.86 para o primeiro e 0.88 para o segundo, como se pode observar nas figuras b) e c).

No segundo gráfico da figura b) pode-se ver o funcionamento e tempos inactivos nos primeiros 30 mseg. do processador 1 e do processador 2. São apresentados no gráfico da figura d) os eventos para os processos *router* e *worker* associados ao processador 1. As convenções são as referidas em 5.3.1.1. Na figura e) podem ser visualizados os eventos equivalentes para o processador 2.

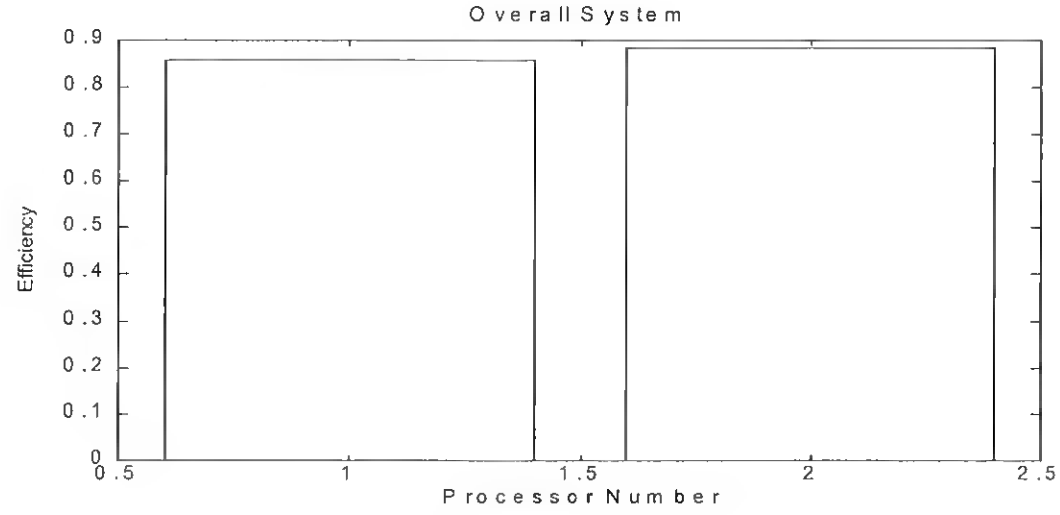


fig. a)

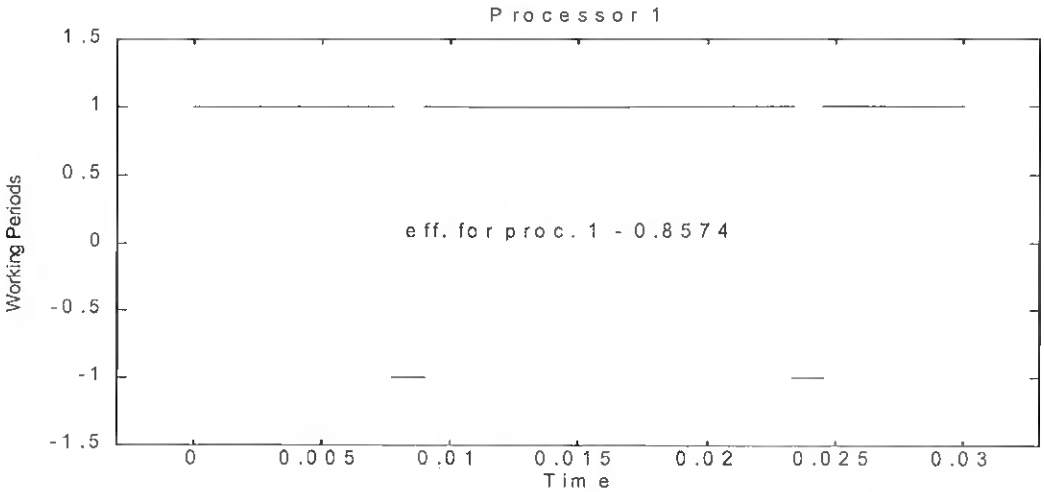


fig. b)

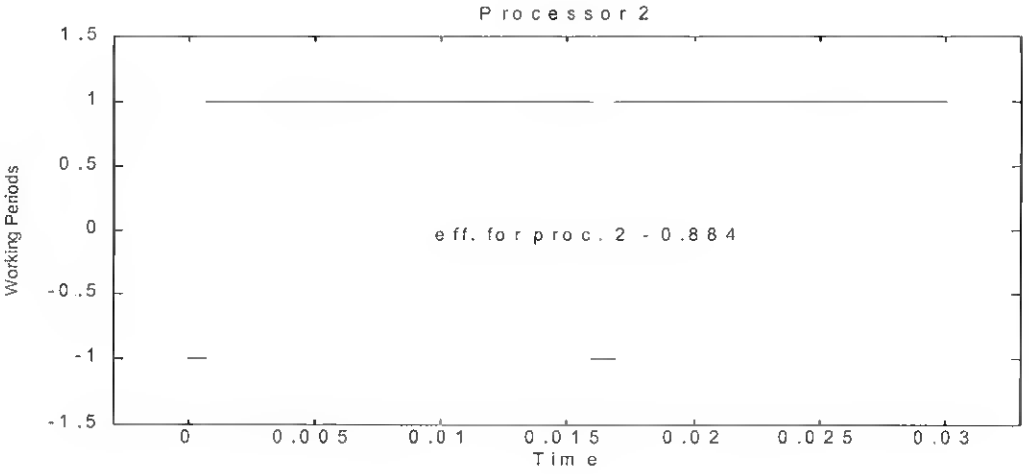


fig. c)

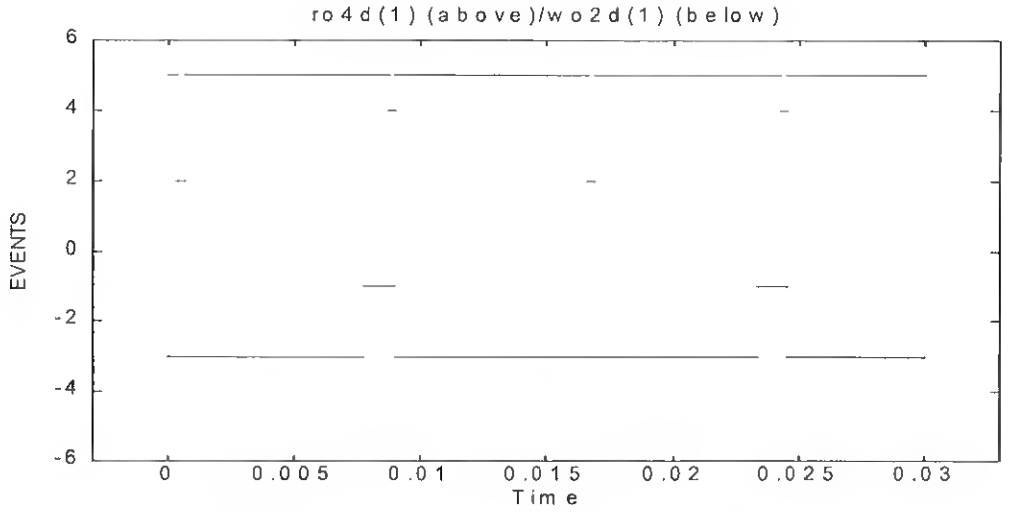


fig. d)

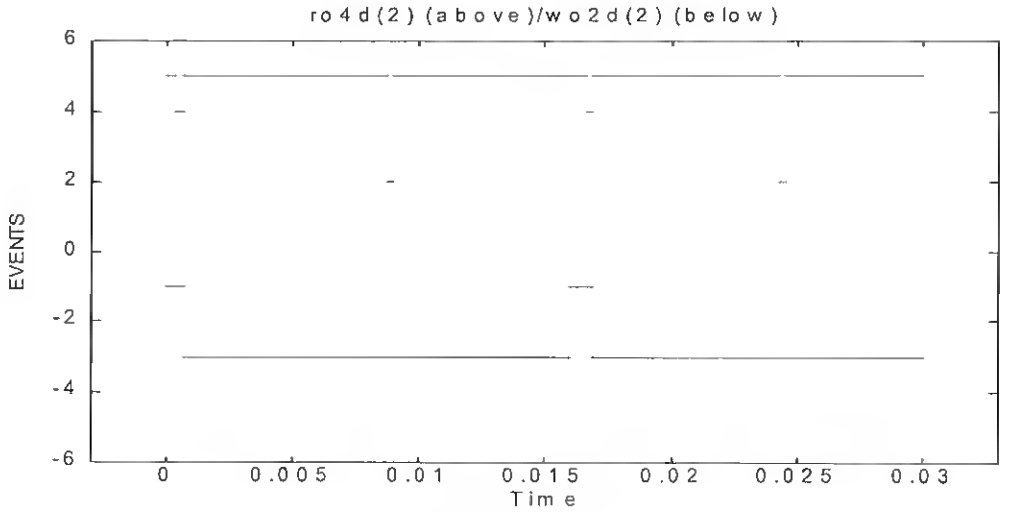


fig. e)

Fig. 34 - Gráficos da *toolbox Matlab* para a primeira aproximação (*DSPs*)

As medidas de desempenho apresentadas foram calculadas assumindo a comunicação entre *DSPs* implementada com canais virtuais, abaixo descrita.

5.3.2.2 Tempos obtidos na comunicação externa

No gráfico que a seguir se apresenta representam-se os tempos (em mseg.) gastos para comunicar vectores de reais (entre 10 e 100 pontos) entre 2 C40s, através de canais físicos. O código respectivo é apresentado no Anexo I.

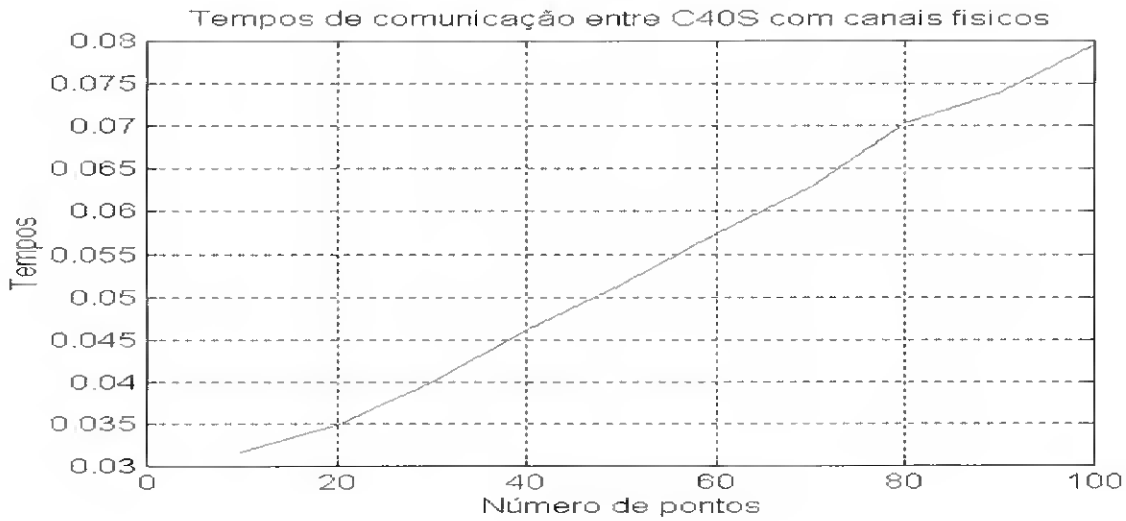


Fig. 35 –Gráfico de tempos obtidos na comunicação entre *C40s* com canais físicos

O gráfico da página a seguir, ilustra os tempos (em mseg.) gastos para comunicar vectores de reais (entre 10 e 100 pontos) entre 2 *C40s* (envio e recepção) através de canais virtuais. O código respectivo é apresentado no Anexo J.

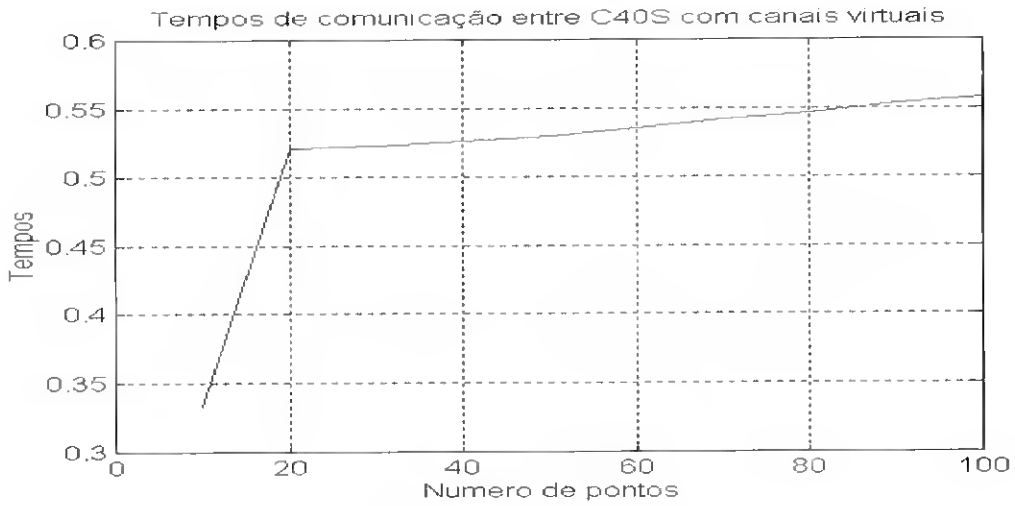


Fig. 36 – Gráfico de tempos obtidos na comunicação entre *C40s* com canais virtuais

Neste caso o *setup time* não é desprezável, tendo-se considerado para o simulador um valor de 0.26 mseg. (correspondente ao segmento de recta entre 20 e 100 pontos). O tempo considerado par transmitir um real foi de 0.235 μs e o *flop-time* correspondente de 1.6 μs, calculado com base na relação entre o tempo de execução do algoritmo sequêncial em um *C40* e o número genérico de operações em virgula flutuante necessárias para executar o código do respectivo algoritmo.

O *Parallel C* da versão para *C40* proporciona a existência de canais virtuais. Estes apresentam as seguintes características [14]:

- Transportam automaticamente mensagens entre processadores através de nós de rede intermédios;
- Um único *Wire* pode suportar qualquer número de canais virtuais. As ligações virtuais entre tarefas não estão limitadas aos canais físicos;
- Qualquer tarefa pode comunicar com outra, que esteja num dos processadores da rede física, independente da sua topologia.

Os canais virtuais apresentam contudo limitações que se relacionam sobretudo com a sua lentidão relativamente aos canais físicos.

Um canal virtual não vai além de 50% da velocidade de transmissão conseguida num canal físico para mensagens longas enviadas a um determinado nó da rede. O desempenho diminui para mensagens menores, até aproximadamente 25% de um canal físico para mensagens de 1000 palavras e abaixo de 10% para mensagens muito pequenas com menos de 100 palavras. O desempenho também diminui quando são enviadas mensagens através de nós intermédios. De notar que estes *overheads* são, na sua generalidade, inerentes a qualquer *software* de encaminhamento de mensagens. É no entanto possível optimizar o desempenho de canais virtuais pela inclusão no ficheiro de configuração da declaração: *UPR MAX= constante e UPR BUFFERS= constante,* onde o primeiro atributo *MAX* define o tamanho máximo em *bytes* dos pacotes *UPR* . Por defeito utiliza-se 2060 *bytes* [14]. Para o nosso exemplo, optou-se por diminuir este valor para 100 e assim reduzir a quantidade requerida de memória para pacotes *UPR.*

Em relação ao atributo *BUFFERS*, este permite o aumento do número de *buffers* em cada canal virtual, sendo particularmente útil para encaminhamento de mensagens longas através de nós intermédios. Por defeito é assumido o valor 2. No exemplo deste capítulo foi utilizado o valor 4 para o aumento de desempenho em comunicações realizadas sobre canais virtuais.

A figura seguinte compara as velocidades obtidas entre arquitecturas de C40s com canais físicos (recta) e lógicos (em escada), onde o eixo dos tempos está em mseg..

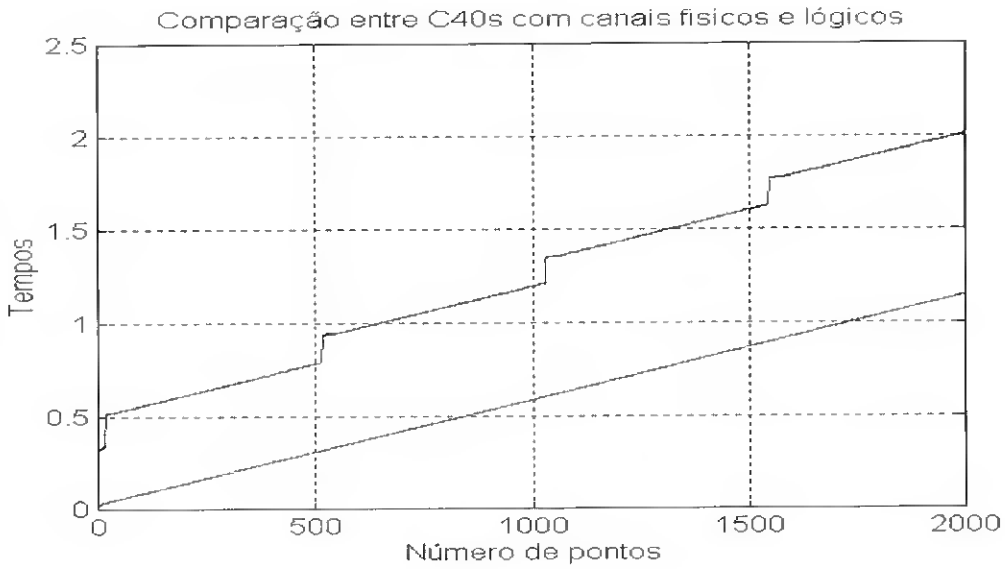


Fig. 37 - Gráfico de tempos de comunicação entre *DSPs* - canais virtuais e físicos

5.3.3 Rede de *Transputers* e *C40s*

Para o caso de arquitecturas mistas foram utilizados dois processadores numa topologia em anel, sendo o primeiro um *Transputer* e o segundo um *C40*. Do mesmo modo que nas arquitecturas anteriores, existe em cada um dos processadores um processo *router* de alta prioridade, responsável pelas comunicações, e um processo *worker* de baixa prioridade, responsável pelos cálculos como se pode verificar na figura abaixo:

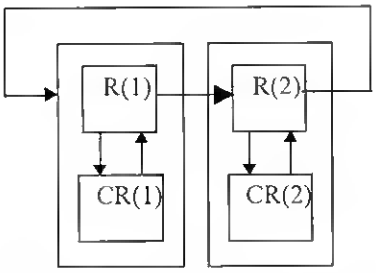


Fig. 38 - Topologia heterogénea com 1 *Transputer* e 1 C40

5.3.3.1 Medidas de desempenho obtidas na arquitectura heterogénea

Foi considerado neste exemplo o caso da triângularização de uma matriz com dimensões m=30 e n=20 em 1 *transputer* e 1 *DSP C40,* com velocidades de 25 e 40MHz respectivamente, usando a aproximação paralela convencional. Obteve-se um tempo de execução sequencial simulado de 199 mseg., um tempo de execução paralelo simulado de 108 mseg., e um *speedup* de 1.84. Os tempos actuais medidos, na plataforma, são respectivamente: 0.199 segundos (num *transputer*), 0.128 segundos e um *speedup* de 1.55.

	Tempo Exe. Seq.	**Tempo Exe. Par.**	***Speedup***
Simulado	0.199	0.108	1.84
Medido	0.199	0.128	1.55

Na figura 39 estão representados os gráficos habituais. O primeiro gráfico da figura a mostra a eficiência paralela para cada processador, cerca de 100% para o *transputer*, e de 20% para a *DSP*. Estes resultados explicam-se dado que a *DSP* é cerca de 5 vezes mais rápida (em termos de operações em vírgula flutuante) que um *transputer*, ficando assim a *DSP* á espera que o *transputer* acabe a parte dos seus cálculos.

Nos gráficos b) e c) pode-se comparar o funcionamento e tempos inactivos, nos primeiros 30 mseg., do *transputer* e da *DSP*, o que comprova o dito anteriormente. Assim, enquanto o *transputer* está permanentemente activo, a *DSP* tem apenas dois períodos de actividade, correspondentes a cerca de 20% do intervalo de tempo considerado.

Nos gráficos d) e e) são visualizados os eventos para os processos *router* e *worker* associados ao *transputer* e à *DSP*, respectivamente. Durante o intervalo de tempo A, o 1º vector de *Householder* é comunicado do *transputer* para a *DSP* através dos respectivos *routers*. Em seguida cada *worker* executa em paralelo as instruções 1.3 e 1.4 do Algoritmo 1, e, no instante

B, o 2º vector de *Householder*, computado pela *DSP*, é transmitido deste para o *transputer*. Enquanto a *DSP* finaliza a sua 2ª execução de 1.3 e 1.4, ficando, tanto o seu *worker* como o seu *router*, a partir do instante C, a aguardar a recepção do 3º vector de *Householder*, o *transputer*, pelo seu lado, armazena o 2º vector de *Householder* no seu *router*, até ao instante D, altura em que o respectivo *worker* finalizou a execução da 1ª iteração do Alg. 1. Só aí o *transputer* inicia a sua 2ª iteração, que finalizará, e, no instante E, calculará o 3º vector de *Householder*, enviando-o à *DSP*.

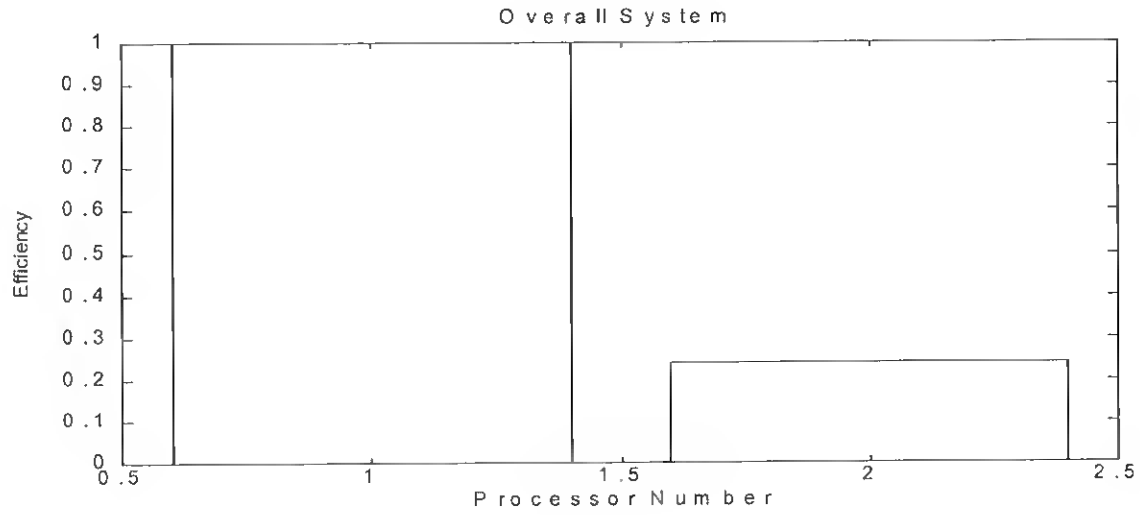


fig. a)

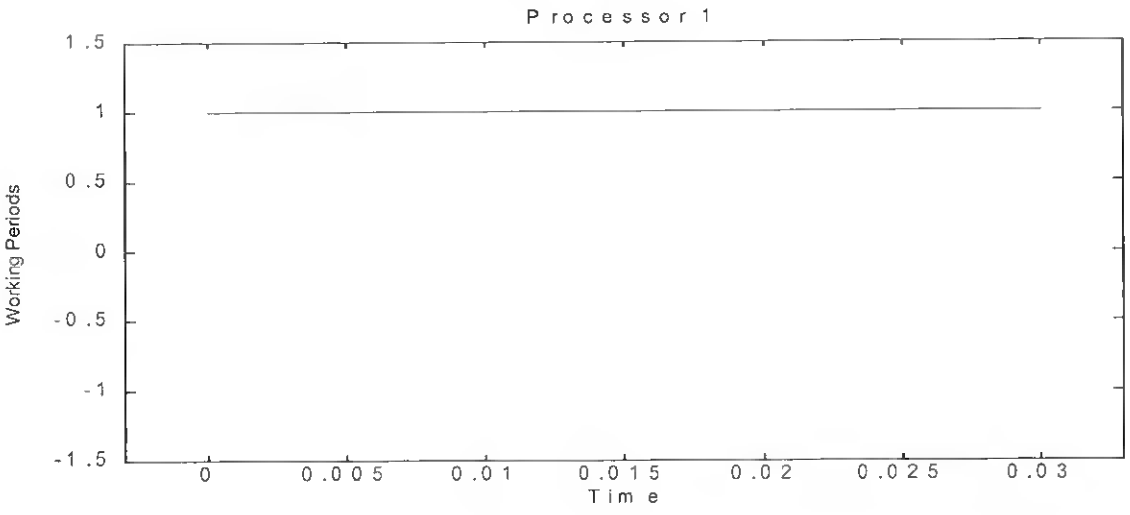


fig. b)

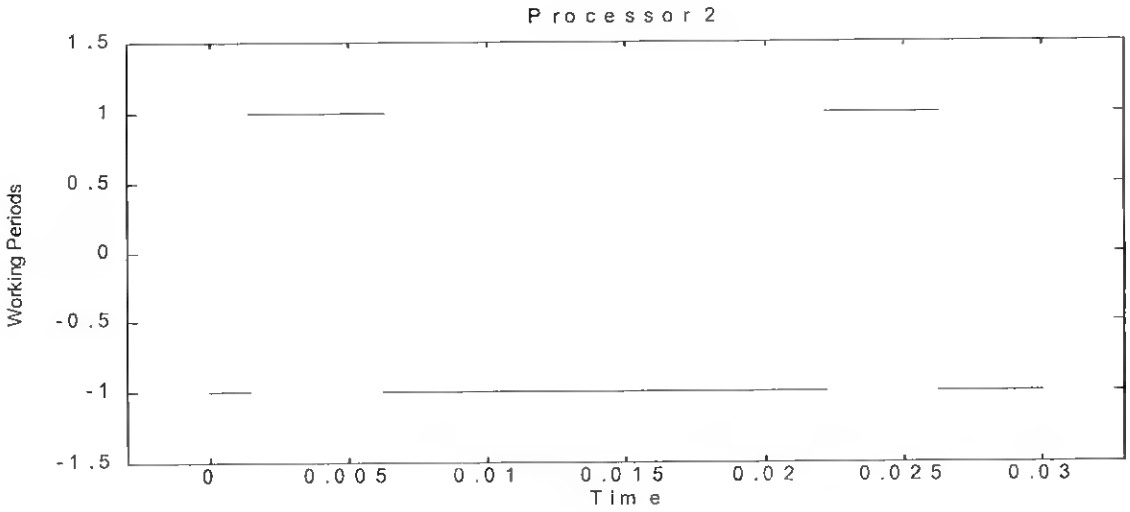


fig. c)

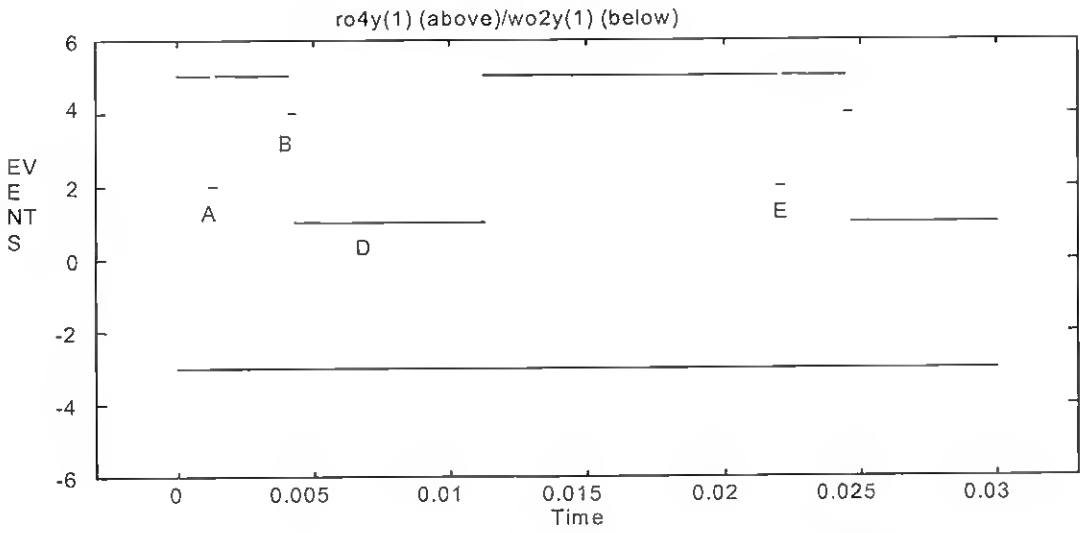


fig. d)

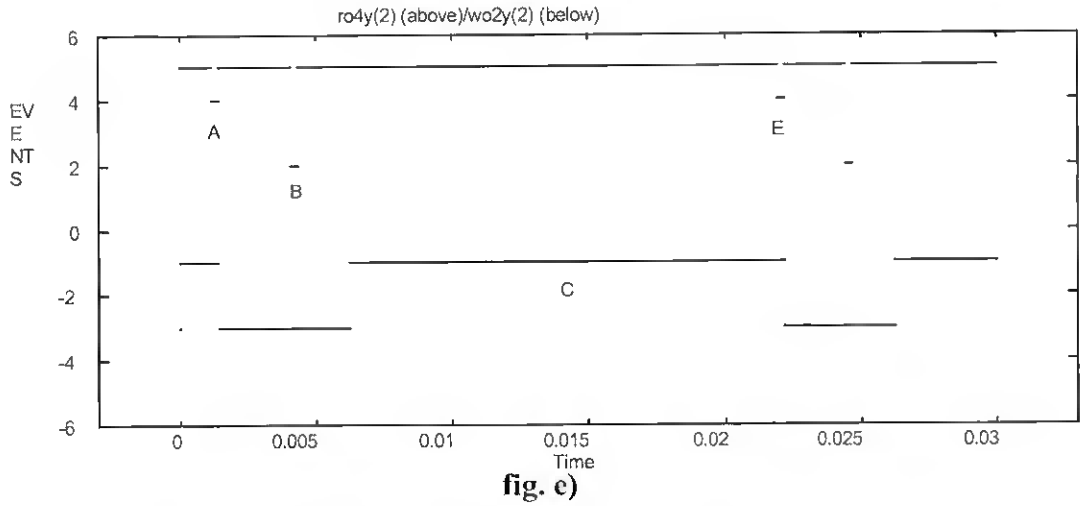


fig. e)

Fig. 39 - Gráficos da *toolbox Matlab* relativos à topologia mista

O programa paralelo deste exemplo é o apresentado no Anexo K.

5.3.3.2 Tempos obtidos na comunicação externa

Para o exemplo da arquitectura heterogénea consideraram-se não apenas os tempos gastos para comunicar vectores de reais (entre 10 e 100 pontos) entre 1 *transputer* e 1 C40, mas também para converter estas dados nos respectivos formatos. Estes tempos estão expressos na fig. 40, onde a escala do eixo y está expressa em mseg. O código respectivo é o apresentado no Anexo L .

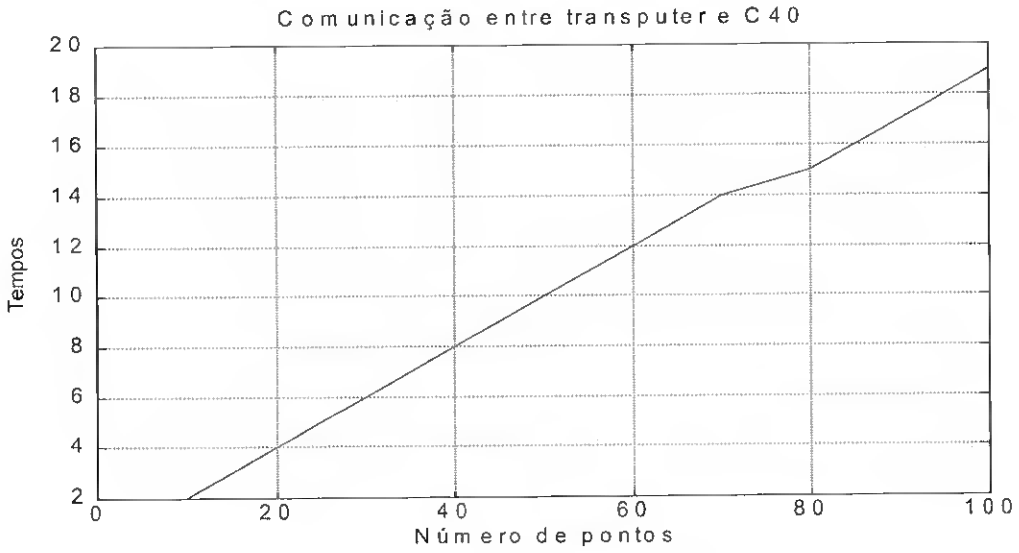


Fig. 40 – Comunicação entre 1 *Transputer* e 1 *C40*

Observa-se que os valores obtidos são muito próximos de uma linha recta. Neste caso o *setup time* é desprezável, e o tempo assumido para comunicar um real foi de 100 μs. Considerou-se tanto o *flop_time* do *Transputer* de 7,4 μs como o *flop_time* do *C40* de 1,6 μs dado se tratar de uma arquitectura com dois processadores de diferentes tipos.

5.4 Conclusões

Neste capítulo o funcionamento do simulador foi ilustrado com exemplos, tendo os resultados obtidos sido interpretados com base nos gráficos disponíveis. O simulador apresenta resultados bastante aproximados com os efectivamente obtidos nas plataformas paralelas, para todos os casos de teste. O maior erro na execução simulada foi observado para o caso de arquitecturas heterogéneas sendo de cerca de 15%. Para as arquitecturas homogéneas, este nunca foi superior a 10% para os casos experimentados.

Em cada uma das secções e parágrafos correspondentes deste capítulo foram indicados os programas sequenciais, paralelos e de comunicações desenvolvidos em *Parallel C*, os quais se encontram num anexo a esta tese. Algumas das listagens estão comentadas possibilitando o esclarecimento de passos envolvidos na programação desses algoritmos.

5 CONCLUSÕES E COMENTÁRIOS FINAIS

Nesta tese de Mestrado foi descrita uma nova versão de um simulador em *Matlab* para aplicações paralelas em arquitecturas homogéneas a heterogéneas. Foram integradas nesta versão novas possibilidades e capacidades para simulação de sistemas homogéneos constituídos por vários elementos de processamento como sejam Inmos *transputers* ou *Texas C40 DSPs*, além de ser estendida a sua aplicabilidade para arquitecturas heterogéneas. Foi ainda melhorada a geração de código automático de funções em *Matlab* que implementam os processos simulados, tornando assim a utilização do simulador mais amigável. O simulador implementado permite, conforme foi ilustrado ao longo deste trabalho, a comparação de implementações paralelas alternativas e a monitorização da execução paralela simulada através da visualização de gráficos de eventos da execução paralela.

Esta abordagem foi concebida com base em estudos prévios realizados sobre possíveis esquemas de paralelização de algoritmos numa rede de *transputers*.

Este trabalho poderia ser melhorado, caso o tempo o permitisse. Assim, embora seja fácil ao utilizador definir a topologia a utilizar, através da execução do programa *SecSim*, poder-se-ia utilizar as facilidades gráficas do ambiente *Simulink* para definir a topologia da rede de processamento. Desta maneira a topologia iria sendo criada pelo utilizador numa forma gráfica, que por certo seria mais agradável e evidente ao utilizador. Outro aspecto que poderia ter sido melhorado está relacionado com o aspecto dos gráficos obtidos, que, com a utilização das novas facilidades introduzidas na nova versão de *Matlab*, poderiam ser mais explícitos para o utilizador. Finalmente a aplicabilidade do simulador poderá ser estendido a outros tipos de processadores, tais como os processadores *SHARC*, da *Analog Devices*, e para outros sistemas paralelos heterogéneos.

REFERENCIAS

[1] Tokhi, M., Hossain, M., Baxter, M. and Fleming, P., *Performance Evaluation of Homogeneous and Heterogeneous Architectures for Real-Time Processing and Control*, 3rd IFAC Workshop on Algorithms and Architectures for Real-Time Control, Ostend, Belgium, June 1995

[2] Bass, J., Browne, A., Hajji, M., Marriot, D., Croll, P., and Fleming, P., *Automating the Development of Distributed Control Software*, IEEE Parallel and Distributed Technology, **2**, 4, 1994, pp. 9-19

[3] The Math Works Inc.: *Matlab user's Guide*, The Math Works Inc., South Natick, MA 01760, USA, 1991

[4] Ruano, A. E., *A Matlab Toolbox for Simulating Transputer Applications*, Proc. 2nd Internacional Meeting on Vector and Parallel Processing (VECPAR'96), Porto, Portugal, September 1996

[5] Texas Instruments *TMS320C40 User's Guide*, Texas Instruments, USA.

[6] Inmos Ltd., *The transputer Databook*, 2nd ed. Inmos Ltd., 1989

[7] Ruano, A., Fleming, P., Jones, D., *An Efficient Parallel Implementation of a Least Squares Problem*, Computing Systems in Engineering, Vol. 6, Nº 4/5, 1995, pp. 313-318

[8] Amdahl, G., *Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities*, AFIPS Conf. Proc., 30, 1967, pp. 438-485

[9] Bräunl, T., *Parallel Programming - an Introduction*, Prentice Hall, 1993

[10] Tokhi, M., Ramos-Hernandez, D., Chambers, C., Hossain, M., *Performance Evaluatio of DSP, Risc and Transputer based Systems in Real-Time Implementation of Signal Procesing and Control Algorithms*, 4th IFAC Workshop on Algorithmsand Architectures for Real-Time Control, Algarve, Portugal, 1997, pp. 287-292

[11] Galletly, J., *Occam 2*, Pitman, 1990

[12] 3L Parallel C User's Guide, version 2.2.4, 1991

[13] Inmos Ansi C, Inmos Ltd., 1989

[14] Single Processor 3LC4X Parallel C, version 2.0.2, 1991

[15] C40 Technical Note , *Using the HET4030tl with Parallel C*, 3L, 1995