Ferramentas Gráficas de Apoio ao Sistema PARSYS

Tese submetida à Faculdade de Engenharia da Universidade do Porto para a satisfação parcial dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrotécnica e de Computadores

(Area de especialização em Telecomunicações)

por

Pedro Miguel Alves Sobral

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Setembro de 1996
Dissertação realizada sob a supervisão do

Doutor João Falcão e Cunha

Professor Auxiliar do Departamento de Engenharia Mecânica da
Faculdade de Engenharia da Universidade do Porto
Para a Luísa
Sumário

Este trabalho resume as principais abordagens à programação paralela em redes de estações de trabalho: a passagem de mensagens e a memória partilhada distribuída. Como exemplos destas abordagens são apresentados os principais sistemas de desenvolvimento de aplicações paralelas PVM (Parallel Virtual Machine) e Network-Linda. São também apresentadas as ferramentas gráicas de apoio à programação paralela que acompanham estes sistemas nas áreas do desenho, acompanhamento e correcção de erros nas aplicações paralelas e a da configuração e monitorização da rede de estações de trabalho.

Em seguida é feita uma descrição breve do PARSYS, um novo sistema de desenvolvimento de aplicações paralelas desenvolvido na FEUP. O PARSYS é um sistema de programação paralela simples, que utiliza memória virtual partilhada, tolerante a faltas e que pode ser utilizado num ambiente de computação heterogéneo. Foi com base neste sistema que foram desenvolvidas as ferramentas gráicas para a interface com o utilizador descritas nesta tese. A interface gráfica do PARSYS (IGP) foi desenhada com o objectivo de permitir:

- Gerir (adicionar e remover) o conjunto de máquinas que intervém na computação;
- Vir a monitorizar diversos parâmetros dos computadores do grupo tais como: número de utilizadores, carga da máquina, ocupação de memória, número de processos (entre outros);
- Iniciar e terminar programas paralelos;
- Monitorizar o conjunto de programas paralelos em execução no grupo de máquinas;
- Determinar as máquinas intervenientes em cada aplicação;
• Apresentar o número, estado e localização das tarefas de cada programa paralelo e,

• Acompanhar em tempo de execução a evolução dos valores das variáveis partilhadas.

Depois de apresentada a IGP são avaliadas as capacidades deste sistema de desenvolvimento, com o ênfase natural nas ferramentas de interface com o utilizador. De seguida são apontadas direcções para a evolução da interface gráfica com o utilizador e do sistema de desenvolvimento em geral.

Verifica-se que a IGP é muito útil pois possibilita uma fácil gestão do conjunto de máquinas e um acompanhamento muito próximo da execução das aplicações.
Abstract

This work presents the most important approaches to parallel computing in workstation clusters: the message passing and the distributed shared memory paradigms. As examples, we will describe the PVM (Parallel Virtual Machine) and the Network-Linda. We will also present the graphical tools available in this systems for designing, monitoring and debugging parallel applications and for cluster configuration and monitorization.

Next, we will do a brief description of the PARSYS system, a new parallel application development system built at FEUP. The PARSYS is a simple, distributed shared memory, fault tolerant system that can be used in an heterogeneous environment. The graphical user interface presented in this report was based in the PARSYS system. The PARSYS graphical user interface (pgui) was designed to:

- Manage the workstation cluster (add and delete hosts);

- Allow, in a near future, the monitorization of several cluster parameters. The number of users, load, memory occupation, number of processes (among others) of each host;

- Start and Stop parallel applications;

- Monitor the set of parallel programs running in the workstation cluster;

- Show the set of machines running each parallel application;

- Present the number, state and host of the tasks for all parallel programs and;

- Monitor, in runtime, the changes in the shared variables.
After presenting the pGUI we will compare it with the graphical tools described for the PVM and Network-Linda systems. Some ideas for the pGUI improvement are also presented.
Agradecimentos

Quero deixar aqui um agradecimento a todas as pessoas que me acompanham ao longo deste trabalho, muitos com o seu parecer técnico e ajuda na resolução de problemas, outros com o seu encorajamento e compreensão essenciais para levar a bom porto esta tarefa.

Ao Prof. João Falcão e Cunha, meu orientador, pelas sugestões, apoio científico e logístico que me dispensou bem como a sua ajuda na estruturação desta tese. Ao José Magalhães Cruz, pelo seu inestimável empenho e ajuda na interacção com o sistema PARSYS e na especificação dos objectivos do trabalho.

Aos colegas do Grupo de Optoelectrónica do INESC, instituição de acolhimento, pelas excelentes condições e ambiente de trabalho. Ao programa PRAXIS XXI pelo suporte financeiro.

À minha família e esposa pela compreensão e incentivo durante este projecto.
Índice

Índice de Tabelas x

Índice de Figuras xi

1 Introdução 1

1.1 A Era da Informática e das Telecomunicações 2

1.2 Normas e Ferramentas 4

1.2.1 As Comunicações 4

1.2.2 A Heterogeneidade e a Integridade dos Dados 5

1.2.3 Invocação de Procedimentos Remotos "RPC's" 5

1.2.4 Compiladores de "STUB's" 6

1.2.5 “Distributed Computing Environment (DCE)” 6

1.2.6 Sistemas de Operação Distribuídos 7

1.3 Sumário 8

1.4 Os Próximos Capítulos 8

2 A Computação Distribuída 10

2.1 Porque Computação Distribuída? 11
2.2 Objectivos e Expectativas .................................................. 12
2.3 Infra-estruturas Necessárias .............................................. 13
2.4 Limitações e Desafios ....................................................... 14
2.5 Conclusões ....................................................................... 16

3 Sistemas de Desenvolvimento de Aplicações Distribuídas ............. 17

3.1 Sistemas de passagem de mensagens .................................... 18
   3.1.1 PVM ...................................................................... 18

3.2 Sistemas de Memória Partilhada Distribuída ............................ 20
   3.2.1 Network-Linda ...................................................... 21

3.3 Conclusões ....................................................................... 23

4 O Sistema PARsys .................................................................. 25

4.1 Introdução ..................................................................... 25
4.2 O Modelo de Programação do PARsys ................................. 26
4.3 As Comunicações no Sistema PARsys ................................. 28
4.4 A Interface de Programação do PARsys ............................... 32
4.5 Conclusões ..................................................................... 33

5 Ferramentas Gráficas de Apoio à Programação Distribuída ........... 36

5.1 Introdução ..................................................................... 36
5.2 O Xpvm ..................................................................... 38
5.3 O HeNCE ................................................................... 40
5.4 O Tuplescope ............................................................... 43
6 A Interface Gráfica do PARSYS

6.1 Introdução ............................................. 46
6.2 Interacção com o PARSYS ......................... 47
6.3 A Janela Inicial ...................................... 48
6.4 Gestão das Máquinas ................................. 50
6.5 Gestão dos Programas ............................... 51
6.6 Monitorização da Execução dos Programas ......... 52
6.6.1 As Tarefas ......................................... 52
6.6.2 As Variáveis Globais .............................. 54
6.7 Comparação com outras Ferramentas ............... 56
6.8 Conclusões .............................................. 57

7 Conclusões e Aspectos para um Trabalho Futuro .... 58

7.1 Principais Contributos ............................... 58
7.2 Aspectos a Desenvolver Futuramente ............... 59
Lista de Tabelas

1.1 O Crescimento da Internet ............................................. 2

6.1 O Ficheiro /tmp/parsys.pid ........................................... 49

6.2 O Ficheiro .parsys.hosts .................................................. 50
### Lista de Figuras

<table>
<thead>
<tr>
<th>4.1</th>
<th>O Modelo de Programação do PARSYS</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Estrutura Geral das Mensagens no PARSYS</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Envio das Mensagens no PARSYS</td>
<td>31</td>
</tr>
<tr>
<td>4.4</td>
<td>Recepção das Mensagens no PARSYS</td>
<td>32</td>
</tr>
<tr>
<td>4.5</td>
<td>Exemplo de um Programa PARSYS</td>
<td>35</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>5.1</td>
<td>O Xpvm</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>O HeNCE</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>O Tuplescope</td>
<td>44</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>6.1</td>
<td>A Interface Gráfica e o PARSYS</td>
<td>47</td>
</tr>
<tr>
<td>6.2</td>
<td>A Janela Principal da Interface Gráfica</td>
<td>48</td>
</tr>
<tr>
<td>6.3</td>
<td>Adicionando uma Máquina ao PARSYS</td>
<td>50</td>
</tr>
<tr>
<td>6.4</td>
<td>Terminando a execução de um programa PARSYS</td>
<td>51</td>
</tr>
<tr>
<td>6.5</td>
<td>A Tabela de Tarefas</td>
<td>53</td>
</tr>
<tr>
<td>6.6</td>
<td>A Tabela de Variáveis</td>
<td>54</td>
</tr>
<tr>
<td>6.7</td>
<td>A Variável Count</td>
<td>55</td>
</tr>
</tbody>
</table>
Capítulo 1

Introdução

O uso de Estações de trabalho ligadas em rede para a implementação de sistemas de computação paralela tem vindo a crescer ao longo dos últimos anos. Muitas ferramentas de programação foram desenvolvidas com o objectivo de aumentar o desempenho de aplicações com grandes exigências de processamento. Cada vez mais a presença de uma interface amigável nessas ferramentas determina, em boa parte, o seu sucesso.

Neste trabalho são apresentados os resultados do estudo das diferentes abordagens à computação paralela em redes de estações de trabalho. Como exemplo são apresentados os sistemas PVM (Parallel Virtual Machine) e o Network-Linda. O principal objectivo deste estudo foi tornar claras as vantagens deste tipo de computação paralela e comparar os diferentes tipos de sistemas de desenvolvimento existentes. Um novo sistema de desenvolvimento de aplicações paralelas, o PARSYS, é também aqui apresentado.

Neste estudo comparativo foi dado realce às capacidades de cada sistema ao nível da interface com o utilizador. São apresentadas algumas ferramentas gráficas de interface com o utilizador e feito um estudo comparativo que inclui interface gráfica desenvolvida para o sistema PARSYS. De realçar na interface gráfica do PARSYS são as suas capacidades de monitorização e configuração do grupo de estações de trabalho e o acompanhamento das aplicações distribuídas, não só ao nível do conjunto de tarefas mas também das alterações das variáveis partilhadas.
1. Introdução

1.1 A Era da Informática e das Telecomunicações

Ao longo dos últimos anos, temos assistido a um crescimento exponencial nos recursos informáticos e nas telecomunicações. Este facto é demonstrado pelo crescente interesse demonstrado pela sociedade em geral por temas como a "Internet", o "World Wide Web" e o correio electrónico, por exemplo. Algumas previsões [33] apontam para que dentro de um a dois anos cerca de 200 milhões de utilizadores se servirão da Internet numa base diária. O crescimento exponencial da Internet (em particular o crescimento do número de máquinas ligadas à Internet) pode ser observado na Tabela 1.1.

<table>
<thead>
<tr>
<th>Data</th>
<th>Máquinas</th>
<th>Dominios</th>
<th>Resposta ao &quot;ping&quot;</th>
<th>Classe das Redes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 96</td>
<td>9,472,000</td>
<td>240,000</td>
<td>1,682,000</td>
<td>92 5655 87924</td>
</tr>
<tr>
<td>Jul 95</td>
<td>6,642,000</td>
<td>120,000</td>
<td>1,149,000</td>
<td>91 5390 56057</td>
</tr>
<tr>
<td>Jan 95</td>
<td>4,852,000</td>
<td>71,000</td>
<td>970,000</td>
<td>91 4979 34340</td>
</tr>
<tr>
<td>Out 94</td>
<td>3,864,000</td>
<td>56,000</td>
<td>1,024,000</td>
<td>93 4831 32098</td>
</tr>
<tr>
<td>Jul 94</td>
<td>3,212,000</td>
<td>46,000</td>
<td>707,000</td>
<td>89 4493 20628</td>
</tr>
<tr>
<td>Jan 94</td>
<td>2,217,000</td>
<td>30,000</td>
<td>576,000</td>
<td>74 4043 16422</td>
</tr>
<tr>
<td>Out 93</td>
<td>2,056,000</td>
<td>28,000</td>
<td>464,000</td>
<td>69 3849 12615</td>
</tr>
<tr>
<td>Jul 93</td>
<td>1,776,000</td>
<td>26,000</td>
<td>421,000</td>
<td>67 3728 9972</td>
</tr>
<tr>
<td>Abr 93</td>
<td>1,486,000</td>
<td>22,000</td>
<td>421,000</td>
<td>58 3409 6255</td>
</tr>
<tr>
<td>Jan 93</td>
<td>1,313,000</td>
<td>21,000</td>
<td>421,000</td>
<td>54 3206 4998</td>
</tr>
</tbody>
</table>

Tabela 1.1: O Crescimento da Internet

O rápido crescimento do número de computadores ligados em rede tem sido acompanhado por um incrível aumento no seu poder de cálculo. Os microprocessadores tem aumentado o seu desempenho a um ritmo muito mais elevado do que os supercomputadores tornando-os cada vez mais indicados (considerando a relação preço/desempenho) para cálculo científico avançado. Cheong [10] sumariza as cinco tecnologias chave para a computação científica de elevado desempenho: microprocessadores, redes de computadores, barramentos, memória de acesso aleatório e discos magnéticos rígidos. Cheong apresenta alguns dados para suportar estas afirmações:

- Desde 1985 que o desempenho dos microprocessadores baseados na tecnologia CMOS

---

1Estes dados foram estimados pelo contacto ("ping") de 1% de todas as máquinas; têm como fonte a empresa "Network Wizards" e estão disponíveis no endereço http://www.nw.com.
tem quadruplicado de três em três anos (na realidade nos últimos três anos tem sido até bastante superior). As velocidades de relógio dos processadores evoluíram desde os 200kHz em 1971 até 500MHz em 1996 (a Digital Equipment Corporation anunciou a comercialização a partir de Setembro de 1996 do processador Alpha 21164 RISC com a velocidade de relógio de 500Mhz [11]).

- As redes locais de computadores cresceram de 10Mbits/s em 1980 com a ethernet para 100Mbits/s em 1990 com o FDDI. Actualmente já existem redes locais com débitos da ordem dos Gbits/s (a Digital Equipement Corporation possui uma implementação comercial para interligação de estações de trabalho com débitos desta ordem de grandeza). Nos laboratórios da British Telecom já foram feitas experiências onde se alcançaram débitos da ordem das dezenas de Gbit/s!


- Ao nível dos discos magnéticos regista-se um crescimento na densidade de informação por polegada quadrada desde 1K bits por polegada quadrada em 1957 até 1Gbits nos anos 90. Representa uma duplicação da capacidade de armazenamento todos os três anos.

Todas estas evoluções apontam para uma quantidade crescente de recursos de computação disponíveis e com facilidade de acesso remoto. Contudo, a maioria dos recursos de computação (principalmente estações de trabalho de alto desempenho) estão inactivas durante cerca de 95% do tempo [29]. Esta situação levou à tentativa de aproveitar todo este poder de computação de uma forma transparente para o utilizador: A Computação Distribuída.
1. Introdução

1.2 Normas e Ferramentas

O pontos seguintes apresentam as principais normas e ferramentas que serviram de base ao desenvolvimento dos sistemas de computação distribuída. Estas ferramentas surgiram com o objectivo de minimizar os problemas resultantes da heterogeneidade das máquinas envolvidas na computação. As principais dificuldades que estas ferramentas visaram resolver eram ao nível das comunicações inter-máquinas e da conversão de dados entre arquitecturas com diferentes representações numéricas.

1.2.1 As Comunicações

Os sistemas de desenvolvimento de aplicações distribuídas têm obviamente que implementar de alguma forma um sistema de comunicações entre as diferentes estações de trabalho, através do qual toda a informação relativa às aplicações paralelas e ao funcionamento interno vai circular. Nos sistemas de operação UNIX, onde este tipo de abordagem à computação paralela inicialmente surgiu, o protocolo de rede mais utilizado é o IP (internet protocols). Existem dois tipos de comunicações que se podem estabelecer sobre a camada de rede IP:

- **UDP "User Datagram Protocol"**

  Neste tipo de comunicação não é explicitamente criado um canal de comunicação entre os dois interlocutores; a máquina emissor constrói os pacotes que pretende transmitir, endereça-os à máquina receptor e coloca-os na rede, ou seja, não há o estabelecimento de uma ligação efectiva entre as duas máquinas. Os pacotes podem não chegar ao destino se ocorrer congestão na rede ou algum erro na transmissão, pois não há mecanismos de correção de erros.

- **TCP "Transmisson Control Protocol"**

  Neste protocolo, as máquinas que pretendem trocar informação entre si, abrem um canal dedicado de comunicação. Com o TCP há garantia de entrega da informação ao outro interlocutor, pois há mecanismos de correção de erros. Este protocolo de comunicação é mais lento que o UDP.
1. **Introdução**

1.2.2 **A Heterogeneidade e a Integridade dos Dados**

Uma necessidade básica nos sistemas de desenvolvimento de aplicações paralelas é a comunicação entre um conjunto heterogêneo de estações de trabalho muitas vezes com diferentes representações dos dados. Este problema mostrou a necessidade da criação de um formato de dados independente da arquitectura das máquinas, que permitisse a codificação e descodificação de estruturas de dados. Os principais problemas são os formatos de dados dependentes do "hardware" e do "software":

- "Big Endian" versus "Little Endian" - existem processadores que trocam os bytes de uma palavra para operação e transmissão via rede enquanto outros processadores não o fazem, o que leva a óbvios conflitos de interpretação no caso de não se ter em conta este facto numa rede de estações de trabalho heterogênea.

- Representação em vírgula flutuante - a codificação das variáveis em vírgula flutuante não é a mesma em todos os sistemas de operação, o que, tal como no tópico anterior, é um factor a ter em conta na transmissão de informação numa rede heterogênea.

E as soluções encontradas:

- O XDR ("eXternal Data Representation") [31] da SUN. Protocolo desenvolvido pela SUN para possibilitar troca de dados entre máquinas com representações distintas. Os dados são codificados no emissor num formato independente da arquitectura sendo posteriormente transmitidos e descodificados no receptor.

- O ISO/OSI ASN.1 [8]. Este foi o protocolo definido pelo ISO "International Standard Organization" para normalizar a codificação básica dos dados de forma a possibilitar a comunicação entre máquinas com representações diferentes.

1.2.3 **Invocação de Procedimentos Remotos "RPC's"**

Os RPC's [18] definem um conjunto de rotinas que permitem que as aplicações em rede sejam desenvolvidas de forma a esconder os detalhes dos mecanismos de comunicação. As suas principais características são:
1. Introdução

- Utilizam o protocolo IP,

- Operam em modo síncrono (pergunta/resposta) em geral (existem sistemas experimentais que utilizam de forma assíncrona), o que pode ser uma importante limitação para a troca de mensagens em sistemas de desenvolvimento de aplicações paralelas. Estes sistemas geralmente tratam as comunicações de forma assíncrona (as mensagens são processadas quando na altura mais oportuna e não imediatamente após a chegada) para minimizar o número de interrupções no tempo de computação de cada tarefa e assim aumentar o desempenho.

- Os procedimentos remotos recebem parâmetros e retornam valores,

- Em geral utilizam o XDR para codificar e descodificar as mensagens de forma a passar, de forma transparente, mensagens entre sistemas que eventualmente possuam representações diferentes dos dados.

1.2.4 Compiladores de "STUB’s"

Aceitam uma descrição de alto nível da interface entre as máquinas que vão estabelecer comunicação e geram automaticamente os módulos necessários à comunicação via rede e à manutenção da integridade dos dados. Têm como principais características:

- São constituídos por uma coleção de compiladores e bibliotecas,

- Traduzem RPC em linguagem C,

- Incluem código para as funções de rede.

Por exemplo a SUN disponibiliza o rpcgen.

1.2.5 "Distributed Computing Environment (DCE)"

O DCE [25] é um ambiente de computação distribuída desenvolvido pela OSF ("Open Software Foundation") e tem como principal objectivo facilitar a construção de aplicações e a sua manutenção num ambiente de computação heterogéneo. Integra os serviços mais
importantes de uma forma distribuída como por exemplo os serviços de RPC’s, nomes, tempo, segurança e “threads”. Integra também ferramentas para a partilha de dados como o DFS “Distributed File System”, para o arranque de estações de trabalho sem disco rígido (“diskless”) e partilha de ficheiros e impressoras MS-DOS. Existe também uma ferramenta, DME (Distributed Management Environment) que auxilia a instalação de software, monitorização do desempenho, configuração, etc. num ambiente de computação heterogêneo.

1.2.6 Sistemas de Operação Distribuídos

Estes sistemas tem por base um núcleo do sistema de operação comum a todas as máquinas de uma rede de estações de trabalho heterogénea e fazem a gestão dos recursos em todo o grupo de uma forma automática. As principais características destes sistemas são:

- Transparência de localização. O utilizador acede a recursos remotos como se estivessem na máquina local,

- Migração dos dados e das tarefas. Os dados necessários a uma máquina são recolhidos de outra localização se necessário. As tarefas são migradas para as máquinas com as características e os dados necessários à sua execução. Tudo isto de forma transparente para o utilizador que pode estar em qualquer máquina do grupo,

- Controle de carga. Os processos são distribuídos pelas máquinas de forma a balancear a carga de todas as máquinas do grupo,

- Migração de tarefas na mesma arquitectura. Controla a execução de tarefas em que diversas partes são executadas por máquinas diferentes (embora com a mesma arquitectura),

- Migração de tarefas entre diferentes arquitecturas. Move as tarefas para as máquinas melhor ajustadas a cada tarefa.
1.3 Sumário

Este conjunto de ferramentas cada vez mais poderoso, tornou os programadores de aplicações distribuídas mais exigentes no sentido de obter portabilidade de código paralelo com a mesma facilidade com que é possível portar código sequencial entre diferentes arquiteturas. Em princípio, um algoritmo sequencial é portável para qualquer arquitetura que suporte o paradigma sequencial. Contudo os programadores querem mais do que isto: querem que a sua realização do algoritmo sob a forma de um programa particular seja portável — portabilidade de código fonte. O mesmo é verdade para os programas distribuídos que necessitam de trocar informação entre um conjunto heterogéneo de máquinas. A ideia para resolver este problema foi a de criar um conjunto de rotinas que permitissem tornar independente do código fonte a comunicação entre as máquinas. O primeiro sistema que alcançou grande aceitação foi o PVM e a sua biblioteca de rotinas ainda hoje é utilizada para este efeito, contudo, mais recentemente surgiu o MPI [32] (“Message Passing Interface”) com o objectivo de normalizar as rotinas de comunicação e proporcionar ainda maior portabilidade do código fonte. O MPI proporciona portabilidade ao nível do código fonte de programas que utilizem a passagem de mensagens escritos em C ou em Fortran entre um conjunto significativo de arquiteturas. Esta situação, tal como no caso sequencial, tem muitas vantagens, incluindo a de proteger os investimentos feitos em software e a de possibilitar o desenvolvimento e teste de um programa numa arquitetura de fácil acesso (por exemplo uma rede de estações de trabalho) antes de ser executado na máquina final (por exemplo Hardware paralelo específico).

1.4 Os Próximos Capítulos

O desenvolvimento da computação distribuída não está apenas dependente das evoluções na rapidez e capacidade dos processadores, memórias, redes, etc. mas também do aparecimento de ferramentas que permitam controlar e utilizar todo o poder de computação das redes de estações de trabalho. Com o aparecimento dos sistemas de desenvolvimento de aplicações paralelas surgiram diferentes abordagens para a distribuição das tarefas e passagem de dados entre as máquinas com objectivos muito distintos. No capítulo “Sistemas de Desenvolvimento de Aplicações Distribuídas” são apresentadas as duas principais abordagens à computação
1. Introdução

paralela em redes de estações de trabalho: os sistemas de passagem de mensagens e os sistemas de memória partilhada distribuída. São também analisadas as principais vantagens e limitações de cada tipo de sistema. Para ilustrar estas abordagens apresentamos os seus principais representantes: o PVM [12] (Parallel Virtual Machine) e o Linda [6].

No capítulo "O Sistema PARSYS" é feita uma descrição mais cuidada do sistema de desenvolvimento de aplicações distribuídas PARSYS [17] (PARallel SYstem), já que foi com base neste sistema que a maior parte deste trabalho foi desenvolvido. No PARSYS vai ser dado realce ao modelo de computação, comunicações e interface de programação.

Uma das dificuldades para a aceitação da computação distribuída para o cálculo científico era a ausência de ferramentas que permitissem utilizar o poder de computação das redes de estações de trabalho sem entrar em grandes detalhes de sintaxe e semântica essenciais para programar nos primeiros sistemas de desenvolvimento. Até porque um bom número de investigadores que necessita de grande poder de cálculo dos computadores não é da área de informática (mas sim de física, matemática, por exemplo) pelo que tem necessidade de ferramentas que facilitem o uso das redes de estações de trabalho. Actualmente já se encontram disponíveis algumas ferramentas que facilitam a interface entre os utilizadores e os sistemas de desenvolvimento. Algumas dessas interfaces acompanham o utilizador desde a fase de desenho da aplicação até à fase de correção de erros como é o caso do HeNCE [13] para o sistema PVM, por exemplo. Outras, como é o caso do Xpvm, tem como principal função a gestão e monitorização da máquina virtual paralela e da execução das aplicações. Estas e outras ferramentas são apresentadas neste trabalho no capítulo "Ferramentas Gráficas de Apoio à Computação Distribuída" como estudo comparativo com a interface desenvolvida para o sistema PARSYS.

A interface gráfica desenvolvida para o sistema PARSYS é apresentada no capítulo "A Interface Gráfica do PARSYS" de uma forma mais detalhada. Vão ser descritas todas as funcionalidades desta ferramenta ao nível da gestão do conjunto de máquinas e ao nível do acompanhamento e correção de erros das aplicações paralelas. Vai também ser possível seguir a forma como a interface comunica com as máquinas do grupo e aplicações em execução, de forma a entender o seu funcionamento interno. Estas descrições são acompanhadas por figuras onde o leitor pode observar a interface em funcionamento. No capítulo final são apontadas as principais ideias para a evolução da interface gráfica e do sistema PARSYS.
Capítulo 2

A Computação Distribuída

O tema da computação distribuída, embora com outras designações, já se encontra bastante difundido na comunidade científica ligada à computação e tem cada vez mais aceitação como um meio eficaz de aproveitamento dos recursos existentes na rede. Termos como computação em rede ("cluster computing"), computação heterogénea ("heterogeneous computing") e máquina virtual heterogénea ("virtual heterogeneous machine") têm sido usados para descrever conceitos similares ou estreitamente relacionados com a computação distribuída.

Podemos definir a computação distribuída como a capacidade de agregar de forma transparente para o utilizador um conjunto heterogéneo de sistemas de computadores. Utilizando um sistema distribuído o programador tem a capacidade de envolver na computação uma multiplicidade de sistemas ligados em rede, diminuindo consideravelmente o tempo de resolução dos problemas.

A popularidade da computação distribuída tem vindo a ser reforçada com o aparecimento de pacotes de "software" bastante estáveis que possuem ferramentas para auxiliar o programador em todo o processo de desenvolvimento da aplicação, desde o seu desenho até ao acompanhamento da execução. Ao longo deste e dos próximos capítulos vão ser apresentadas as principais características disponíveis nestas ferramentas.
2. A Computação Distribuída

2.1 Porquê Computação Distribuída?

Ao longo desta secção vão ser expostas as principais justificações para o desenvolvimento da computação distribuída em redes de estações de trabalho. Nesta secção vão ser também discutidos os principais objectivos e expectativas resultantes da utilização destes recursos, as infra-estruturas necessárias e suas principais limitações.

A utilização de redes de estações de trabalho pode trazer um bom número de vantagens. Embora algumas opiniões sugiram que as redes de estações de trabalho podem substituir na prática os supercomputadores, a realidade é muito diferente; nada nos leva a crer que as estações de trabalho, mesmo interligadas com redes de muito alto débito, possam vir a substituir os grandes supercomputadores. Contudo, há muitas aplicações a serem executadas em supercomputadores que na realidade não necessitam dos recursos de um supercomputador. Eugene Brooks [2] identificou duas principais categorias de aplicações científicas com exigências muito diferentes ao nível dos recursos de computação que necessitam:

- Desempenho máximo ("Capability demand"). Cabem nesta designação os megaproblemas que requerem todas as capacidades de cálculo do sistema onde vão ser executadas, tanto ao nível da memória como do CPU. Este tipo de aplicações ("Grand Challenge Applications") foram a principal justificação para o aparecimento do processamento massivamente paralelo.

- Desempenho aceitável ("Capacity demand"). Este tipo de aplicações requerem um uso razoável dos recursos mas sem exigências de desempenho máximo e com um número moderado de acessos à memória. Esta descrição representa uma classe muito grande de aplicações que é necessário executar numa base diária. São tarefas particularmente talhadas para serem executadas em redes de estações de trabalho.

As principais razões que os apoiantes da computação distribuída [1] [23] [24] [34] apresentam como justificação para o uso de redes de estações de trabalho são:

- A necessidade de obter um número significativo de ciclos de CPU com um custo baixo para servir aplicações científicas com custos computacionais moderados,
2. A Computação Distribuída

- Devido aos contínuos cortes orçamentais esta é uma forma de satisfazer as crescentes necessidades de recursos de computação para fins científicos,
- O desejo de explorar o benefício da relação custo/desempenho dos processadores RISC,
- A necessidade de disponibilizar ciclos de CPU para aplicações interactivas durante as horas nobres de funcionamento e também de proporcionar um bom desempenho para aplicações do tipo "batch" nas horas de menor actividade,
- A necessidade de desenvolver e explorar aplicações paralelas sem custos adicionais,
- O desejo de rentabilizar a base instalada de estações de trabalho sem necessidade de investimento (habitualmente as estações de trabalho estão subaproveitadas durante 95% do tempo).
- A possibilidade de utilizar tecnologia normalizada (por exemplo as estações de trabalho e as redes) por forma a minimizar o risco económico e de proporcionar a construção de um sistema escalável,
- O aproveitamento das grandes capacidades de memória que são comuns nas estações de trabalho,
- O desejo de manter as facilidades de utilização, designadamente ao nível da interface com o utilizador, que já existem nos ambientes de estações de trabalho,
- As redes de estações de trabalho oferecem as maiores vantagens da computação paralela e sequencial; a velocidade do processamento sequencial e crescimento ilimitado do poder de computação do processamento paralelo, e
- O desejo de aproveitar os benefícios da computação heterogénea e assim produzir a solução mais eficiente para um dado problema.

2.2 Objectivos e Expectativas

Com a crescente aceitação pela comunidade científica da computação distribuída, os elementos dessa comunidade foram definindo ao longo do tempo um conjunto de objectivos e de expectativas sobre os sistemas de desenvolvimento de aplicações distribuídas. Embora alguns
desses objectivos ainda estejam a alguns anos de distância da sua concretização o facto de existir uma orientação para o desenvolvimento deste tipo de sistemas é muito importante para a previsão do ambiente de computação científica no futuro [28] [14] [23]. O sistema de desenvolvimento ideal terá de:

- Proporcionar transparência ao nível da arquitectura das máquinas,
- Proporcionar transparência ao nível das redes e comunicações,
- Proporcionar transparência no paralelismo,
- Proporcionar transparência da localização das tarefas,
- Proporcionar tolerância a faltas, tanto ao nível das redes como das estações de trabalho,
- Disponibilizar, de forma transparente, o acesso a todos os recursos e arquitecturas,
- Ser fácil de usar,
- Disponibilizar suporte para as linguagens de programação de alto nível mais comuns,
- Ser capaz de aumentar o seu desempenho utilizando a paralelização,
- Não apenas trabalhar com um conjunto de sistemas heterogéneo mas também explorar essa heterogeneidade,
- Proporcionar um ambiente portátil de processamento paralelo,
- O desempenho das aplicações paralelas tem de aumentar com a inclusão de mais estações de trabalho (o sistema deve ser escalável).

### 2.3 Infra-estruturas Necessárias

As redes de estações de trabalho para funcionarem como suporte para a computação distribuída colocam desafios únicos ao nível das infra-estruturas. As estações de trabalho podem estar geograficamente dispersas, pertencerem a organizações distintas, incluirmos "hardware" de múltiplos vendedores e terem grandes diferenças de "software" instalado (designadamente sistemas de operação, compiladores, etc.). Para que essas máquinas possam constituir um
2. A Computação Distribuída

recurso de computação distribuída alguns requisitos, designadamente ao nível das comunicações, são necessários [14](a itálico encontram-se tópicos que justificam a necessidade das ferramentas apresentadas nesta tese):

- Redes com grande largura de banda para suportar as necessidades de comunicação,
- Mecanismos de comunicação de baixa latência (100-500 μs entre máquinas),
- Boa escalabilidade,
- Possibilidade de comunicações do tipo "multicast" com grande largura de banda,
- Capacidade de recuperar automaticamente de faltas, tanto nas máquinas como na rede,
- Primitivas normalizadas para comunicação a baixo nível, sincronização e distribuição de tarefas entre todas as arquiteturas,
- Primitivas que escondam do utilizador as diferenças de arquitectura, protocolo e sistema de operação,
- Monitores de desempenho em tempo real,
- Distribuição fiável de tarefas "batch",
- Ferramentas para o desenvolvimento de aplicações distribuídas.
- Suporte das linguagens de alto nível tradicionais em computação heterogénea.
- Aplicações capazes de explorar o potencial das redes de estações de trabalho,
- Novas ferramentas de administração que possibilitem a gestão dos recursos de computação distribuída,
- Desenvolvimento de normas que protejam os investimentos em "software".

2.4 Limitações e Desafios

Nas secções anteriores foram expostas as principais características da computação distribuída. Há contudo algumas limitações que têm de ser ultrapassadas para que os benefícios desta
2. A Computação Distribuída

abordagem possam ser alcançados. Algumas dessas limitações não se restringem às redes de estações de trabalho mas também se apresentam aos criadores de arquiteturas massivamente paralelas. Algumas das interrogações que se apresentam à computação distribuída são [14]:

- O esforço necessário para melhorar o desempenho das aplicações pela paralelização (será que é preferível esperar por "hardware" mais rápido?),

- O futuro desconhecido de algumas ferramentas, devido ao seu estado inicial de desenvolvimento torna a escolha de plataformas muito arriscada (há no entanto um certo número de aplicações independentes da plataforma com créditos firmados, como é o caso do PVM e do Linda por exemplo),

- a necessidade do aparecimento de normas de facto que permitam proteger os investimentos no desenvolvimento de aplicações paralelas. O esforço conjunto de importantes investigadores e vendedores de "hardware" na especificação do MPI visou resolver este problema.

- Software de domínio público versus software proprietário,

- Heterogeneidade (sistemas de operação, software de rede, aplicações, hardware, etc.),

- O problema do acesso a recursos de computação geograficamente distribuídos,

- A habilidade para proporcionar segurança e autenticação nos grupos de estações de trabalho,

- A necessidade de contabilizar os recursos de computação utilizados por cada aplicação de forma precisa,

- A necessidade de ferramentas de programação (detectores de erros "debuggers", ferramentas para controlar o desempenho e controlar as aplicações).

- A necessidade software para controlar o sistema de operação e fazer a gestão de aplicações distribuídas em ambientes de computação heterogêneos.

- A necessidade de unificar os modelos de programação distribuída e paralela para os vários tipos de sistemas (uniprocessador, multiprocessador de memória partilhada, multiprocessador distribuído, etc.).
2. A Computação Distribuída

- A necessidade de software que de forma transparente distribua os recursos, mantenha a consistência dos dados, possa gerir a comunicação e sincronização em diferentes plataformas.

2.5 Conclusões

Fica claro (tendo em conta as afirmações da secção 2.1) que há muitas vantagens no desenvolvimento da computação distribuída em redes de estações de trabalho heterogéneas, principalmente se considerarmos que quase todas as organizações têm condições para implementar sistemas deste tipo sem custos adicionais. Um outro ponto importante (apresentado na secção 2.2) é a evolução dos sistemas de desenvolvimento de aplicações distribuídas no sentido de tornarem cada vez mais transparente ao utilizador o facto de estar a criar e/ou utilizar aplicações distribuídas numa rede heterogénea de máquinas. Em relação às infra-estruturas podemos concluir (secção 2.3) que existem algumas restrições ao nível das comunicações para a utilização destes sistemas principalmente em problemas que exijam utilização intensa da rede. Contudo, a falta de normas para as comunicações e ferramentas para o desenvolvimento de aplicações distribuídas, é ainda mais limitativa do que a baixa largura de banda da generalidade das infra-estruturas de comunicações. Embora com algumas limitações e interrogações (secção 2.4) a crescente utilização deste tipo de sistemas possibilitará que uma boa parte dos problemas dos sistemas de desenvolvimento de aplicações distribuídas sejam ultrapassados.
Capítulo 3

Sistemas de Desenvolvimento de Aplicações Distribuídas

Para podermos falar de computação distribuída num conjunto de estações de trabalho ligadas em rede, temos de tomar em consideração que as máquinas por si só não conseguem agregar o seu poder de computação de forma a contribuirem para a resolução de um problema. É necessária a presença de um sistema que faça a gestão do grupo de estações de trabalho. Estes sistemas são o tema deste capítulo.

Existe um grande número de sistemas de desenvolvimento de aplicações distribuídas. Alguns exemplos são o PVM, o Network-Linda, o DQS, o HeNCE, o Entreprise, o Express e o TreadMarks. Uma excelente referência para uma análise das funcionalidades destes e de outros sistemas pode ser encontrada em Turcotte [21]. Contudo, não é de forma alguma o objectivo deste trabalho a descrição de todos os sistemas de desenvolvimento de aplicações distribuídas que existem. Neste trabalho vão ser apresentados os sistemas cujos modelos de programação representam as duas principais abordagens à computação distribuída, a saber, os sistemas de passagem de mensagens e os sistemas de memória partilhada distribuída. Como exemplo de um sistema de passagem de mensagens vai ser apresentado o PVM (Parallel Virtual Machine) já que é um sistema ainda em pleno desenvolvimento e se tem imposto como uma referência para os outros sistemas de passagem de mensagens. No âmbito dos sistemas de memória partilhada distribuída a escolha recaiu sobre o Network-Linda que na realidade foi o primeiro sistema a abordar o paradigma da computação distribuída desta
forma. Outros sistemas vão também ser apresentados mais à frente neste texto, mas com uma abordagem muito diferente e mais relacionada com o tipo de ferramentas que têm disponíveis para programação distribuída e da interface homem-máquina que apresentam.

3.1 Sistemas de passagem de mensagens

Os processadores num grupo de estações de trabalho não partilharam a mesma memória física, logo toda a comunicação entre processos tem de ser feita através da rede. Os sistemas de passagem de mensagens são implementados com base em bibliotecas de rotinas como o PVM, o TCGMSG [15] e o Express [26]. Uma norma para a passagem de mensagens, o MPI, foi também desenvolvida. Nos sistemas de passagem de mensagens a natureza distribuída da memória do sistema de computação, é totalmente visível ao programador. O programador necessita de ter atenção:

- à localização dos dados que pretende enviar,
- **quando** deve comunicar com outros processos,
- **quais** os processos com que necessita de comunicar e
- **o que** comunicar.

Esta forma de comunicação não facilita a programação nestes sistemas especialmente para aplicações com estruturas de dados complexas.

3.1.1 PVM

PVM ("Parallel Virtual Machine") é um projecto ainda em desenvolvimento que começou em 1989 no "Oak Ridge National Laboratory” (ORNL) e agora envolvendo pessoas do ORNL da Universidade do Tennesee e da Universidade Carnegie Mellon. O projecto produziu uma ferramenta de programação paralela que é útil a outros membros da comunidade científica e, no interesse do avanço da investigação, este sistema está disponível na Internet sem encargos de licenciamento.

¹para informações acerca da forma de obtenção do PVM basta enviar correio electrónico para netlib@ornl.gov com subject: send index from PVM3.
3. Sistemas de Desenvolvimento de Aplicações Distribuídas

O Modelo de Programação

O modelo de programação do PVM é bastante simples, contudo muito geral e acomoda uma grande variedade de estruturas de programação. A interface de programação é de fácil compreensão, com o objectivo de que estruturas simples de programação sejam implementadas de forma intuitiva. O utilizador escreve a sua aplicação como uma colecção de tarefas cooperantes. As tarefas acedem aos recursos do PVM através de uma biblioteca de rotinas com interfaces normalizadas. As primitivas de passagem de mensagens do PVM estão orientadas para operação em ambiente heterogéneo envolvendo construções para o armazenamento ("buffering") e transmissão de informação.

O PVM

O PVM é um sistema que possibilita que um grupo (possivelmente heterogéneo) de estações de trabalho ligadas por um conjunto (possivelmente heterogéneo) de redes actuem como um recurso de computação paralela. Com o PVM um conjunto de computadores sejam eles, sequenciais, paralelos, ou vectoriais apresentam-se como uma única máquina virtual com memória distribuída. Para configurar uma máquina virtual paralela pessoal é simplesmente necessário listar os nomes das máquinas num ficheiro quando o PVM começa a executar (parâmetros adicionais nesse ficheiro permitem diferentes nomes de "login" em cada uma das máquinas e diferentes localizações de ficheiros necessários ao PVM). Esta configuração, se necessário, pode ser mais tarde alterada ou pelo uso de um programa do sistema PVM chamado pvm ou por um programa. O programa pvm também permite inspecionar o estado da máquina virtual paralela. O PVM inclui também ferramentas para controlar e fazer a procura e correcção de erros em programas paralelos, por exemplo. Estas ferramentas serão apresentadas no capítulo seguinte.

A máquina virtual do utilizador pode sobrepôr-se com a de outro utilizador e cada máquina virtual pode executar muitas aplicações. Contudo, um utilizador só pode ter uma máquina virtual paralela activa em cada momento. As aplicações podem ser escritas em C ou Fortran e são paralelizadas utilizando rotinas de passagem de mensagens comuns à maior parte dos computadores com memória distribuída. O sistema Fortran é construído em cima do sistema C.

O PVM providencia rotinas para iniciar e terminar tarefas na máquina virtual do utilizador.
e permite que essas tarefas possam comunicar e sincronizar entre elas. As primitivas de comunicação incluem rotinas para a troca de estruturas de dados e também rotinas de mais alto nível para difusão ("broadcast"), sincronização de barreiras ("barrier synchronization") e sincronização de acontecimentos ("event synchronization"). Em particular o PVM trata da conversão das mensagens se no grupo de máquinas existem diferentes representações dos dados, por exemplo se as máquinas ordenam os bytes de forma diferente ou têm uma representação em vírgula flutuante diferente. O sistema PVM é constituído por duas partes:

- um "deamon", o pvmrd3. Este reside nos computadores que fazem parte da máquina virtual paralela. Providencia, entre outras coisas, um ponto de contacto entre os computadores.

- Uma biblioteca de rotinas de interface com o PVM. Esta biblioteca contém rotinas para a passagem de mensagens, lançamento de processos, coordenação de processos e modificação da máquina virtual paralela. As aplicações têm de ser compiladas junto com esta biblioteca.

No PVM, em qualquer ponto da execução de uma aplicação concorrente, os processos que existirem podem comunicar e/ou sincronizar entre si livremente. As aplicações no PVM consistem em uma ou mais "instâncias" de um ou mais "componentes". Uma "componente" é um programa sequencial convencional mas normalmente com chamadas às rotinas das bibliotecas do PVM. Cada "instância" de cada "componente" é um processo que pode ser visto como uma subtarefa da aplicação.

### 3.2 Sistemas de Memória Partilhada Distribuída

Os sistemas de memória partilhada distribuída por software [1] apresentam o modelo de programação de memória distribuída sobre um conjunto de rotinas de passagem de mensagens. Uma aplicação pode ser escrita da mesma forma como seria desenhada num multiprocessador com memória partilhada, acedendo aos dados partilhados com as operações tradicionais de leitura e de escrita. A coerência da memória é mantida através da troca de mensagens entre as máquinas que constituem o sistema de programação distribuída. Embora seja mais
fácil de programar desta forma (em relação à programação por passagem de mensagens), os sistemas de memória distribuída (SMD) tendem a gerar mais comunicações e como tal a ser menos eficientes que os sistemas de passagem de mensagens. Nos sistemas de passagem de mensagens, as comunicações são totalmente controladas pelo programador, que tem um conhecimento exacto das estruturas de dados que é necessário partilhar. Em contraste, os SMD têm pouca noção da aplicação que estão a executar e como tal têm poucas possibilidades de optimizar as comunicações. Como o custo de trocar mensagens entre estações de trabalho é muito elevado, estas comunicações a mais podem causar uma séria degradação no desempenho.

3.2.1 Network-Linda

O Modelo de Programação

O modelo de programação do Linda [6][5] [7] foi desenvolvido como parte de um projecto de investigação na universidade de Yale e é um dos sistemas em que os processos comunicam e se sincronizam através de uma memória partilhada distribuída que é referida como "tuple space". "Tuples" são coleções de campos ligados logicamente para formar zonas de armazenamento da informação. De alguma maneira, fazem lembrar as estruturas das linguagens de programação tradicionais.

Um processo que necessite de comunicar informação gera um novo "tuple" e coloca-o no "tuple space". Um processo que necessite de informação procura-o no "tuple space", fazendo uma procura associativa onde um "tuple" apropriado é encontrado pela sua estrutura e pelo valor de um dos seus campos. Para actualizar um "tuple", o "tuple" é removido, modificado e posto no "tuple space". Restringindo a modificação dos "tuples" desta forma, um mecanismo implícito de bloqueamento de acesso ("locking") assegura a sincronização necessária no caso de múltiplos acessos ao mesmo "tuple". Em situações apropriadas, a transmissão de dados entre os processos implicada pelas operações no "tuple space" pode ser optimizada por uma primitiva de difusão ("broadcast").

Um processo activo pode delegar uma subcomputação gerando um "live tuple". O resultado é um novo processo capaz de executar concurrentemente com os outros processos, incluindo o seu criador. Quando completada a tarefa o "live tuple" é então transformado num "tuple"
ordinário, caracterizado pelos resultados da sua computação, e é colocado no "tuple space" onde pode ser acedido pelos outros processos.

As operações no "tuple space" são as seguintes:

- **out.** Coloca um "tuple" no "tuple space",
- **in.** Remove um "tuple" do "tuple space",
- **rd.** Lê um "tuple" do "tuple space" mas não o retira,
- **eval.** Cria um "live tuple". Como consequência um novo processo (ou processos) é (são) lançado(s) o que eventualmente tem como resultado a adição de "tuple(s)" ao "tuple space".

Os "tuples" são selecionados pelas primitivas **in** e **rd** com base nos valores dos seus campos. O **in** e **rd** são operações bloqueantes, ou seja, se não existe nenhum "tuple" nas condições especificadas o processo que invocou a operação é suspenso até que um "tuple" nessas condições surja. As chamadas **inp** e **rdp** são as versões não bloqueantes do **in** e **rd**. Se um "tuple" nas condições exigidas é encontrado no momento da chamada destas primitivas **inp** ou **rdp**, o seu comportamento é exactamente igual ao do **in** e **rd** e retornam o valor 1 (sucesso). Contudo, se não existir nenhum "tuple" nas condições exigidas estas rotinas não bloqueiam o processo mas retornam o valor 0 (insucesso).

O Linda é um modelo de programação paralela e não apenas uma linguagem de programação, mas a adição das operações do Linda a uma linguagem de programação como o C ou o Fortran torna a linguagem numa variante do Linda que pode ser usada para programação paralela. A linguagem que é o resultado desta junção funciona como uma linguagem de coordenação e proporciona as condições necessárias para combinar diferentes processos num programa paralelo completo. As operações do Linda são ortogonais às da linguagem de programação de suporte e apresentam capacidades complementares que possibilitam a programação paralela.

**O Network Linda**

O SCA’s ("Scientific Computing Associates) Network Linda é uma implementação do modelo de programação paralela Linda que é executado num grupo de estações de trabalho, não
necessariamente com a mesma arquitectura. O Network-Linda inclui os compiladores C-Linda e Fortran-Linda para C e Fortran.

Os programas compilados para executar na rede de estações de trabalho são iniciados por um utilitário ntsnet que, entre outras tarefas, trata do lançamento e finalização dos programas, espera pela normal ou anormal terminação das tarefas, consulta as máquinas da rede para receber informação sobre a sua carga e inicia processos remotamente. O utilitário ntsnet tem um grande número de opções e também ficheiros de configuração com o objectivo de tornar a execução de um programa Linda tão flexível quanto possível. Utilizando as opções e ficheiros de configuração o utilizador pode especificar, entre outras coisas:

- O número de máquinas (nós) onde o programa vai executar,
- A velocidade relativa dessas máquinas,
- O número mínimo e máximo de processos por máquina,
- A localização dos ficheiros executáveis nas máquinas remotas (copiando-os se necessário),
- A localização dos directórios de trabalho nas máquinas remotas,
- vários parâmetros relacionados com a distribuição das tarefas ("Scheduling").

Quando uma versão de rede de um programa executa, o sistema Linda determina em que máquinas os diferentes processos vão ser executados. Se a tarefa requerer menos processadores do que o total licenciado, o ntsnet selecciona os que estão menos carregados nesse momento. Normalmente o utilizador não controla que processador vai receber um processo particular. Contudo, incluindo uma identificação do processador entre os campos dos "tuples", torna possível que uma dada máquina identifique tarefas que lhe são exclusivamente atribuídas.

### 3.3 Conclusões

Depois de apresentadas as duas principais abordagens à programação distribuída, podemos concluir tal como Petrie e Carr [27] que é menos sujeita a erros a programação de problemas complexos (que necessitem de partilhar estruturas de dados complexas entre os diferentes processos) em sistemas de memória partilhada distribuída. A clareza da programação paralela e
3. Sistemas de Desenvolvimento de Aplicações Distribuídas

a forma fácil de expressar a concorrência nos sistemas de memória partilhada distribuída tem sempre um custo em desempenho em relação aos sistemas de passagem de mensagens onde cabe ao programador definir as comunicações e sincronização entre processos. Contudo, têm surgido sistemas do tipo memória partilhada distribuída com desempenhos comparáveis aos sistemas de passagem de mensagens, um bom exemplo desta situação é o TreadMarks [20]. Em [22] encontra-se um conjunto de comparações de desempenho entre o TreadMarks e o PVM. Este aumento de desempenho deve-se fundamentalmente a melhoramentos nos algoritmos utilizados para manter a consistência da memória distribuída o que diminui as necessidades de comunicação entre as máquinas, principal limitação deste tipo de sistemas.
Capítulo 4

O Sistema PARSYS

4.1 Introdução

O sistema PARSYS [17] é uma ferramenta de programação de aplicações paralelas direc- 
cionada para problemas com granularidade elevada. No PARSYS as tarefas são distribuídas 
por um conjunto de estações de trabalho ligadas em rede. As principais características do 
PARSYS são:

- **SPMD ("Single Program Multiple Data")** – O modelo natural de programação 
do PARSYS é o modelo SPMD. Isto resulta principalmente do facto de todas as tarefas 
serem idênticas (com a exceção natural da primeira tarefa que inicia a execução do 
programa). Contudo outros modelos de programação podem ser usados com expressões 
do tipo "If-Else" no código das tarefas.

- **Programação Paralela de Elevada Granularidade** – O sistema PARSYS foi 
desenhado para ser utilizado com um modelo de programação onde cada tarefa par-
ticipante no programa resolve uma parte apreciável do problema. A programação 
paralela com granularidade fina, que necessita de muitos recursos de comunicação não 
é adequada para o PARSYS.

- **Simplicidade** – Uma vez entendido o problema, e a forma de o paralelizar, é relativa-
mente fácil criar código paralelo com o PARSYS, pois existe um pequeno número de
rotinas de sistema que é necessário utilizar e o conjunto de restrições que é imposto na
organização do código é muito pequeno.

- Tolerância a Faltas – No sistema PARSYS, as tarefas de um mesmo programa
paralelo trocam mensagens entre elas; quando alguma tarefa não está a desempenhar
o seu papel (i.e., não troca mensagens com as outras tarefas durante um período de
tempo especificado) a tarefa que a tentou contactar replica-a noutra máquina.

- Memória Partilhada Distribuída – O sistema PARSYS utiliza a memória par-
tilhada distribuída como forma de partilha dos dados necessários à resolução dos
problemas. Embora utilizando a troca de mensagens entre as tarefas a nível interno, o
acesso a dados partilhados é directo na perspectiva do programador.

O funcionamento do sistema PARSYS vai ser apresentado ao longo deste capítulo, no entanto
é de notar que o PARSYS ainda está em desenvolvimento e tem portanto algumas limitações
nas facilidades de programação que são oferecidas. Principalmente ao nível da representação
dos dados está a ser feito trabalho de forma a possibilitar que o PARSYS possa tirar partido
de uma máquina, independentemente da sua arquitectura. Como no âmbito desta tese foi
necessária uma forte interacção com o PARSYS designadamente ao nível das comunicações
e organização da informação interna ao sistema, são apresentadas algumas sugestões para o
melhoramento deste sistema.

4.2 O Modelo de Programação do PARSYS

O modelo de programação do sistema PARSYS é bastante simples. Quando o utilizador, da
linha de comando lança um programa PARSYS, a máquina local começa a executar o código
(a primeira tarefa é criada). Se for necessária a criação de novas tarefas para a execução do
programa, uma nova tarefa é criada numa das máquinas disponíveis para executar programas
do PARSYS, e a tarefa mãe prossegue com a sua parte da computação. A nova tarefa tem
exatamente o mesmo comportamento da tarefa que a criou, e o processo de criação de
novas tarefas pôr quando todas as partes do problema estiverem entregues à sua tarefa. As
tarefas do mesmo programa estão ligadas entre si por um anel lógico de "sockets" (portas de
software através das quais o sistema de operação encaminha e recebe informação para um
4. **O Sistema PARSYS**

dado processo ou grupo de processos) usando o protocolo UDP ("User Datagram Protocol"). Embora todas as tarefas de um mesmo programa possam comunicar directamente entre si, isto só acontece em situações particulares, em geral, as mensagens circulam através do anel lógico da primeira para a última tarefa (direcção dos ponteiros do relógio na figura 4.1)

![Diagrama de PARSYS](image)

**Figura 4.1:** O Modelo de Programação do PARSYS.

Todas as tarefas de um programa PARSYS possuem tabelas internas onde está armazenada informação importante acerca do programa paralelo, incluindo o endereço do porto de contacto de cada uma das outras tarefas, valores das variáveis partilhadas do programa, e outra informação de controlo. Periodicamente as tarefas de um mesmo programa trocam mensagens que incluem toda esta informação; cada tarefa pode então actualizar a sua informação interna (no caso da informação que lhe chega ser mais recente que a sua) mantendo-se desta forma a coerência da memória partilhada distribuída. No PARSYS a memória partilhada distribuída é apresentada sob a forma de dois tipos de variáveis: as Variáveis Dependentes da Tarefa (VDT) e as Variáveis Independentes da Tarefa (VIT). As VDT's são variáveis que embora partilhadas não podem ser acedidas livremente por todas as tarefas. Cada tarefa tem um campo dessa variável que lhe pertence e no qual pode efectuar
operações de leitura e escrita livremente. No entanto, nos campos pertencentes a outras tarefas apenas pode efectuar operações de leitura. As VIT's, por outro lado, são variáveis partilhadas sem restrições de acesso, em que todas as tarefas podem efectuar operações de leitura e escrita.

4.3 As Comunicações no Sistema PARSYS

![Diagrama de Estrutura Geral das Mensagens no PARSYS]

**Figura 4.2: Estrutura Geral das Mensagens no PARSYS**

Como todos os sistemas do seu tipo, o PARSYS implementa a memória partilhada distribuída à custa da passagem de mensagens entre os processos que constituem um determinado programa paralelo. Contudo este processo é totalmente transparente ao programador. No sistema PARSYS estão definidos alguns tipos de mensagens que cobrem basicamente todas as necessidades de comunicação deste tipo de sistemas de desenvolvimento de aplicações paralelas. As mensagens no PARSYS têm como estrutura base a apresentada na figura 4.2. Nesta figura estão apresentados todos os possíveis campos de uma mensagem. Os campos incluídos em cada mensagem variam conforme o tipo de mensagem em questão. Basicamente podemos dizer que uma mensagem pode conter até três grandes grupos de informação:

1. **Dados para monitorização.** Esta informação é utilizada para a monitorização da máquina virtual paralela e do programa paralelo a que pertence a tarefa. É com estes
dados que a interface de monitorização observa o que se está a passar ao nível da execução do programa. Este grupo de informação inclui:

- O número de variáveis partilhadas definidas no programa paralelo ao qual a tarefa pertence (representado na figura 4.2 pelo conjunto [N,Nval] onde o N representa apenas um identificador e o Nval é o valor propriamente dito).

- O número de variáveis partilhadas independentes da tarefa (representado na figura 4.2 pelo conjunto [I,Ival]).

- O número de variáveis partilhadas dependentes da tarefa (representado na figura 4.2 pelo conjunto [D,Dval]).

- O número de passos definidos no programa paralelo. É possível utilizar uma rotina PARSYS em certos pontos importantes do código para tirar conclusões acerca do estado do programa (representado na figura 4.2 pelo conjunto [S,Sval]).

- Outra informação sobre as variáveis partilhadas do programa representada pelos conjuntos [1,Dados],[2,Dados],etc. Cada conjunto contém informação sobre uma determinada variável, variável 1, variável 2, etc. Os dados de cada variável são, o seu nome, o seu tipo e o seu tamanho em bytes.

2. Dados dependentes das tarefas. Neste campo é transportada a informação de todas as tarefas que estão definidas no programa paralelo em causa, ou seja um conjunto do tipo [T,Tinfo] (ver figura 4.2) para cada tarefa. A informação que existe nas mensagens para cada tarefa inclui:

- Informação interna relativa à tarefa, ou seja o Identificador do Programa a que a tarefa pertence, o Identificador da Tarefa dentro do programa, uma estrutura do tipo sockaddr_in (que contém informação de contacto da tarefa através do seu "socket" de comunicações), a idade da tarefa (tem que ver com o número de actualizações que foram feitas na sua tabela de variáveis), o pid ("process identificador") da tarefa e o estado da tarefa (no PARSYS estão definidos os estados: BABY - tarefa em início de execução, ACTIVE-tarefa em actividade, STALLED - tarefa em estado de espera e DEAD - tarefa terminada).

- Informação relativa às variáveis partilhadas dependentes da tarefa, que inclui para cada variável desse tipo: o número da variável, o número de alterações que
4. O Sistema PARSYS

já sofreu e o *valor* da variável. Este campo termina com o separador [000].

3. Dados independentes das tarefas. Neste campo é veiculada informação relativa às variáveis partilhadas de acesso livre a todas as tarefas; é o conjunto [D,Dinfo] da figura 4.2. Neste campo, para cada variável independente da tarefa é apresentado: o *número da variável*, o *número de alterações* que já sofreu e o *valor* da variável. Este campo também termina com o separador [000].

No PARSYS actualmente, estão definidos os seguintes tipos de mensagens:

- **INI.** Uma nova tarefa pede ao seu criador informação inicial sobre o programa paralelo (inclui o campo de dados dependentes das tarefas),

- **AIN.** Resposta a uma mensagem INI com todos os dados disponíveis sobre o programa (inclui os campos de dados dependentes e independentes das tarefas),

- **ORI.** Envio de dados gerados na tarefa local (inclui os campos de dados dependentes e independentes das tarefas),

- **FOR.** Reenvio de dados novos recentemente recebidos pela tarefa (inclui os campos de dados dependentes e independentes das tarefas),

- **STE.** Passo de programa concluído: contacto em sentido contrário ao normal (inclui os campos de dados dependentes e independentes das tarefas),

- **RES.** Retorna resultados finais para a primeira tarefa (inclui os campos de dados dependentes e independentes das tarefas),

- **KIL.** Ordena a uma tarefa que termine.

- **URG.** Mensagem a requisitar resposta urgente (ainda não implementada, inclui os campos de dados dependentes e independentes das tarefas),

- **AUR.** Resposta a pedido de urgência (ainda não implementada, inclui os campos de dados dependentes e independentes das tarefas),

- **MOR.** Mensagem a anunciar uma tarefa moribunda, uma substituição é requerida.
4. O Sistema PARSYS

- **GII.** Mensagem enviada pela interface de monitorização a pedir o envio de informação sobre as variáveis existentes no programa.
- **GIA.** Resposta à mensagem GII (inclui o campo de dados para a monitorização),
- **GIQ.** Mensagem enviada pela interface de monitorização a pedir o envio de informação ao PARSYS sobre estado do programa.
- **GIP.** Resposta à mensagem GIQ (inclui os campos de dados dependentes e independentes das tarefas).

![Diagrama do envio de mensagens no PARSYS]

**Figura 4.3: Envio das Mensagens no PARSYS**

No sistema PARSYS o envio e a recepção de mensagens é tratada de forma distinta. Enquanto o envio de mensagens é feito de forma síncrona a recepção processa-se de forma assíncrona.

Na figura 4.3 está apresentada a forma de envio das mensagens: A mensagem é construída com base na informação presente nas tabelas internas das tarefas, ou seja, a tabela de variáveis e a tabela de tarefas. Uma vez empacotada segundo a sintaxe atrás apresentada (dependente do tipo da mensagem), a mensagem é enviada através do porto UDP para a rede em direção à tarefa destinatária.

Por sua vez, a recepção de mensagens tende a ser bastante mais complexa, já que não é prático que estas sejam recebidas de forma síncrona uma vez que o processamento da tarefa seria interrompido de cada vez que uma mensagem chegasse. A forma ideal de tratar este problema é receber as mensagens de forma assíncrona: Quando uma mensagem chega à tarefa é gerado um sinal do tipo SIGIO que implica uma chamada assíncrona à rotina
Figura 4.4: Recepção das Mensagens no PARSYS

`sigioasync()`: esta rotina guarda a mensagem numa zona de memória e retorna. Mais tarde, de forma síncrona a tarefa trata a mensagem e actualiza as suas tabelas com a nova informação, se for caso disso. Este mecanismo está ilustrado na figura 4.4.

### 4.4 A Interface de Programação do PARSYS

A interface de programação do sistema PARSYS inclui um pequeno conjunto de rotinas em C que, adicionadas com esta linguagem, possibilitam a distribuição de um programa por várias máquinas ligadas em rede. As rotinas de interface com o sistema PARSYS a que o programador tem acesso são:

- **int writeglob(int globvar, char *addr)**. Esta rotina permite a escrita de uma variável partilhada.

- **int peekglob(int globvar, int taskid)**. Esta rotina permite verificar se uma dada variável partilhada pode ou não ser lida.
4. O Sistema PARSYS

- char *readglob(int globvar, int taskId). Esta rotina é utilizada para efectuar a leitura de uma variável partilhada. É uma chamada bloqueante.
- int gettaskid(void). Retorna a identificação da tarefa que a invocou.
- int new_task(void). Cria uma nova tarefa.
- int step_done(void). Marca a conclusão de uma parte da tarefa.

A escrita de programas no PARSYS não obriga a grandes restrições na organização do código. As principais restrições são:

- é necessário no início de cada programa PARSYS invocar uma rotina que executa inicializações (fst_init()),
- as variáveis partilhadas têm de ser declaradas num bloco comentado para que o pré-processador as identifique como tal,
- as variáveis dependentes da tarefa são declaradas como vectores de dimensão MAXTASKS,
- Em vez de declarar uma função main() como é tradicional na linguagem C, no PARSYS o utilizador declara uma função user_main()
- Tem de ser incluído no programa PARSYS um ficheiro de cabeçalho de nome parsys-user.h

Na figura 4.5 é apresentado um pequeno programa escrito no sistema PARSYS que apenas faz uma troca de mensagens entre duas tarefas. Neste exemplo podemos ter a noção exacta da forma de um programa no sistema PARSYS.

4.5 Conclusões

O sistema PARSYS apresenta em alguns pontos uma abordagem inovadora. O seu modelo de programação onde todas as tarefas se encontram ao mesmo nível (ou seja, não existem relações do tipo mestre-escravo) e a capacidade de tolerar faltas (em parte consequência do seu modelo de programação sem pontos sensíveis) são os de maior destaque. Contudo, pelo facto de ser ainda um sistema experimental e ainda estar em evolução tem ainda algumas
4. O Sistema PARSYS

limitações. Do meu contacto com o sistema PARSYS ficaram algumas ideias para a sua evolução que passo a referir:

- Tentar usar um conjunto normalizado de rotinas para efectuar a passagem de mensagens, por exemplo o MPI, para evitar problemas na conversão de dados entre máquinas com representações diferentes dos dados.

- Tentar desenvolver um modelo de dados mais versátil para as tabelas internas por forma a eliminar as restrições ao número de tarefas e tamanho de variáveis.

- Pensar em implementar mecanismos de coerência de memória que permitam minimizar a quantidade de informação transmitida e dessa forma melhorar o desempenho global.
#include <stdio.h>
#include "parsys-user.h"

/*
GLOBVAR char buff[MAXTASKS][100];
*/

tst_init()
return 0;
}

user_main()
char *pbuf;
int id;
id = gettaskid();
if (id == 0) /* first task */
{
int tsender;
printf("i'm t\%d\n", gettaskid());
new_task();

if ((pbuf = readglob(buf, tsender)) == NULL) {
fprintf(stderr, \nCould not read globvar \%d\n", buff);
return(-1);
}
printf("from t\%d: \%s\n", tsender, pbuf);
}
else /* other task */
{
char obuf[100];
strncpy(obuf, "hello, world from ");
gethostname(obuf strlen(obuf), 64);+
writeglob(buf, obuf);
}
return 0;
}

Figura 4.5: Exemplo de um Programa PARSYS.
Capítulo 5

Ferramentas Gráficas de Apoio à Programação Distribuída

5.1 Introdução

Ao longo deste trabalho já foi possível tirar a conclusão de que existe um conjunto bastante amplo de tarefas que o utilizador tem de efectuar de forma a poder tirar partido do potencial representado pelas suas estações de trabalho ligadas em rede. As principais necessidades são:

- Instalar e configurar um sistema de desenvolvimento nas estações de trabalho ligadas em rede,
- Tomar conhecimento com a sintaxe do sistema em que pretende trabalhar e das características particulares do sistema (esta pode ser uma tarefa bastante demorada, dependendo do sistema em questão),
- Escrever código paralelo baseado num determinado algoritmo,
- Alterar a configuração do grupo de estações de trabalho dependendo de cada problema particular,
- Monitorizar em tempo real o estado das máquinas que constituem o grupo de trabalho, de forma a controlar a execução dos programas paralelos,
5. Ferramentas Gráficas de Apoio à Programação Distribuída

- Monitorizar em tempo real a execução dos programas paralelos de maneira a tirar conclusões sobre o seu desempenho, correção, estado, etc.,

- Fazer o "debugging" das aplicações paralelas.

Na realidade, muitas destas necessidades também existem nos ambientes de computação tradicionais. A dificuldade em utilizar as ferramentas existentes para a computação em série nos sistemas de desenvolvimento de aplicações paralelas é que, nestes sistemas toda a informação se dispersa por um conjunto de máquinas, ou seja, não está centralizada. A gestão de todo este processo tornava bastante difícil ao utilizador não familiarizado com estas tarefas a utilização do potencial de computação adormecido nas suas máquinas. Este foi um motivo forte para muitos elementos da comunidade científica que enfrentavam problemas passíveis de resolução utilizando esta abordagem mostrarem relutância em aceitá-la. Esta fase, no entanto, foi ultrapassada pela aparição de ferramentas gráficas de uso intuitivo que transformaram um bom número das tarefas atrás citadas em poucos "cliques" de rato. Em nossa opinião uma boa ferramenta gráfica de apoio ao utilizador da programação distribuída deverá ter as seguintes características:

- Ser fácil de usar e configurar (mostrar apenas o que o utilizador necessita).

- Ser escalável - eficiente mesmo com o aumento do número de máquinas ou complexidade do problema.

- Não alterar de forma significativa o desempenho das máquinas na resolução dos problemas.

- Recolher informação dos programas paralelos em tempo de execução ("on the fly").

- Auxiliar o programador na:

  - Criação dos programas paralelos,

  - Monitorização das máquinas envolvidas na computação,

  - Monitorização da execução dos programas paralelos e respectivas tarefas,

  - Detecção e correção de erros dos programas paralelos,

  - Tarefa de aumentar o desempenho dos programas paralelos.
5. Ferramentas Gráficas de Apoio à Programação Distribuída

Nas próximas seções vamos tomar conhecimento com as mais importantes ferramentas que estão disponíveis nesta área, algumas delas, como vamos ver, criaram inclusive uma nova forma de pensar a programação paralela. Partindo de um algoritmo paralelizado, é agora bastante simples obter um programa a funcionar num sistema de programação distribuída do tipo dos que temos vindo a descrever.

5.2 O Xpvm

O Xpvm foi desenvolvido pelo mesmo grupo de pessoas que desenvolveu o PVM [12], com o objectivo de disponibilizar ao utilizador do PVM uma interface gráfica que combinasse as capacidades da interface de texto do PVM (o pvm) com um monitor de desempenho e um “debugger” numa única interface de uso fácil. Para analisar um programa com o Xpvm, o programador apenas tem de o compilar utilizando a biblioteca de rotinas do PVM, versão 3.3 ou posterior, a qual já foi concebida para capturar informação de monitorização em tempo real. Assim, qualquer tarefa iniciada pelo Xpvm, irá retornar informação de forma a ser possível fazer uma análise da execução em tempo real ou depois de o programa terminar, através da consulta de informação guardada em ficheiros durante a execução do programa. O Xpvm proporciona ainda ao utilizador um acesso do tipo “apontar-e-clicar” a todos os comandos da consola do PVM.

Um menu permite que os utilizadores adicionem ou removam máquinas do grupo de computação, os programas paralelos podem ser iniciados de uma caixa de diálogo onde é possível configurar todas as opções de execução, incluindo as rotinas da biblioteca do PVM que pretendemos monitorizar. O Xpvm serve também de monitor de desempenho para as tarefas do PVM. As tarefas iniciadas pelo Xpvm retornam automaticamente informação sobre os acontecimentos que descrevem a actividade do PVM que desejamos monitorizar. Os programas não necessitam de ser recompilados ou anotados para monitorização, uma vez que a biblioteca de rotinas do PVM 3.3 inclui essa possibilidade. As diversas vistas sobre os programas paralelos proporcionadas pelo Xpvm são alteradas em conjunto pela acção do rato e estão correlacionadas no tempo, permitindo que o utilizador mais facilmente compare informação de diferentes origens relacionada com uma ocorrência particular na execução do programa. O Xpvm permite observar a informação recolhida da execução dos programas de
várias perspectivas:

- **Vista de Rede**

  A perspectiva de rede tem como objectivo mostrar ao utilizador a actividade das máquinas que participam na máquina paralela virtual. Cada máquina é representada por um ícone que inclui a arquitectura e o nome da máquina. Estes ícones são iluminados a cores diferentes para indicarem o estado das tarefas em execução em cada máquina. A cor "activa" significa que pelo menos uma tarefa naquela máquina está em actividade executando trabalho útil. A cor "sistema" significa que nenhuma tarefa está a executar trabalho útil, mas que pelo menos uma está a executar rotinas do PVM. Quando não há tarefas a executar na máquina o seu ícone é deixado sem cor ou branco.

- **Vista Espaço–Tempo**

  Esta perspectiva mostra o estado de tarefas individuais enquanto elas são executadas independentemente das máquinas. Cada tarefa é representada por uma barra horizontal ao longo do eixo temporal, onde a cor da barra em determinado instante indica o estado da tarefa. A cor de "computação" é utilizada para os instantes de tempo em que a tarefa está a executar trabalho útil. A cor de "sistema" indica os instantes de tempo em que a tarefa está a executar rotinas do PVM para comunicação, controlo de tarefas, etc. A cor de "espera" representa instantes de tempo em que a tarefa está à espera de mensagens de outras tarefas ou seja o tempo perdido em sincronização. A comunicação entre as tarefas é também apresentada nesta perspectiva à custa de linhas desenhadas entre as barras que representam as tarefas nos tempos correspondentes ao envio e recepção das mensagens. Alguma informação mais precisa sobre os tempos de envio e recepção das mensagens e sobre os tempos de execução das tarefas pode ser obtida clicando nos respectivos ícones.

- **Vista de Utilização**

  Aqui encontramos um sumário da vista espaço–tempo em cada instante de tempo através de um gráfico mostrando em cada instante de tempo o número de tarefas em "computação" em "sistema" e "espera". Esta informação é representada por
três rectângulos sobrepostos, sendo a utilização da máquina virtual paralela deduzida através da área ocupada por cada uma das cores.

- Chamadas ao PVM e Resultados das Tarefas

Esta informação tem alguma utilidade para assistir na correção de erros nos programas paralelos. Cada linha do ficheiro de monitorização contém os parâmetros de chamada e os resultados retornados das chamadas às rotinas do PVM.

O Xpvm é muito útil na gestão de alto nível que é necessário ter em atenção para executar programas paralelos no PVM. Contudo, o Xpvm não contempla a monitorização de certos parâmetros das máquinas envolvidas na computação tais como carga da máquina, número de processos e utilizadores etc., que são importantes para decidir o grupo de máquinas que vão receber tarefas, principalmente se tivermos em conta que muitas dessas máquinas podem não estar dedicadas a este tipo de aplicações. Outro ponto em que o Xpvm tem algumas lacunas é na detecção e correção de erros de programas paralelos, já que não é possível, por exemplo, fazer um acompanhamento das alterações nos valores das variáveis pois apenas está disponível monitorização nos pontos de interface com as rotinas do PVM. A figura 5.1 ilustra esta ferramenta depois de executar um pequeno programa incluído nos exemplos do PVM onde cada máquina troca uma mensagem com outra ("hello").

5.3 O HeNCE

O HeNCE [13] é um ambiente gráfico de desenvolvimento de aplicações paralelas escrito para o X-Windows sobre a biblioteca de rotinas do PVM, que permite aos utilizadores desenvolverem programas paralelos numa rede de máquinas. Neste ambiente, o programador é responsável por especificar o paralelismo desenhando grafos que expressam as dependências e o controlo de fluxo num programa. O HeNCE inclui uma classe de grafos de forma a auxiliar o programador na especificação do paralelismo. O HeNCE inclui um editor de grafos no qual o utilizador "desenha" o seu programa. Cada nodo no HeNCE representa uma subrotina escrita em C ou Fortran. Os arcos nos grafos do HeNCE representam dependências e controles de fluxo. A interface para a passagem de parâmetros é um dos pontos fortes do HeNCE. Os programadores têm unicamente que especificar os parâmetros que vão ser utilizados para
Figura 5.1: O Xpvm
invocar cada nodo de subrotina. Estes parâmetros são especificados quando o programador associa uma subrotina a um nodo do grafo. As rotinas são mapeadas em máquinas com base numa matriz de custos definida pelo utilizador. Existem seis tipos de construções nos grafos do HeNCE:

- **Nodos de subrotinas.** Os nodos de subrotina representam uma função particular com uma lista de parâmetros, a ser invocada durante a execução do grupo. Estas subrotinas não têm qualquer estado senão a sua lista de parâmetros, ou seja, não lhes é possível consultar informação das outras subrotinas, nem podem escrever em nenhuma variável global (fora da sua lista de parâmetros) que possa ser acessada por outra subrotina.

- **Arcos de dependência simples.** Os arcos de dependência representam relações entre subrotinas num grafo HeNCE. Um arco de um nodo para outro representa o facto de que o nodo da cauda do arco tem de ser executado antes do que se encontra à cabeça. Durante a execução do grafo, as rotinas são automaticamente iniciadas quando as suas predecessoras, tal como definido nos arcos de dependência, forem completadas.

- **Arcos de dependência condicionais, ciclos, "fan" e "pipeline".** Estas construções adicionam subgrafos ao grafo de programa, com base em expressões a serem avaliadas em tempo de execução.

A janela inferior da ferramenta de construção de grafos do HeNCE da figura 5.2 mostra-nos um exemplo de teste (incluso na distribuição do HeNCE) onde estão representados nodos de subrotinas, arcos de dependência e "fans". O programa apresentado na figura, é executado de baixo para cima (norma no HeNCE), realizando uma série de testes de sincronização temporal entre um conjunto de rotinas dependentes entre si.

Embora a capacidade do HeNCE produzir código paralelo a partir de um grafo anotado seja a sua característica mais enovadora, o HeNCE é um sistema com muito mais potencialidades. Inclui também ferramentas para a monitorização da máquina virtual paralela apresentando ao utilizador imagens com as máquinas e a sua arquitectura, o estado em que se encontram e a sua utilização. O utilizador pode percorrer todo o processo de desenvolvimento de uma aplicação sem sair da interface gráfica, desde o "desenho" do programa até à recolha de
5. Ferramentas Gráficas de Apoio à Programação Distribuída

Figura 5.2: O HeNCE

resultados.

5.4 O Tuplescope

O Tuplescope é uma ferramenta baseada no X-Windows para a correcção de erros e visualização da execução de programas desenvolvido para o Network-Linda. Em adição às facilidades dos "debuggers" tradicionais tais como a execução passo a passo e a definição de pontos de paragem, o Tuplescope possibilita a monitorização de classes de "tuples", dados em "tuples" específicos e indicações visuais de interações entre processos ao longo da execução do programa.
A figura 5.3 apresenta um esquema onde é apresentada a aparência gráfica do Tuplescope. Esta ferramenta está dividida em duas partes: a primeira, representada na zona superior da figura, mostra uma série de botões que têm associadas diversas funções para o controlo do Tuplescope; a segunda, na zona inferior da figura, apresenta as classes de “tuples”. Dentro das classes de “tuples” são apresentados todos os “tuples” definidos sob a forma de pequenos círculos sendo possível a visualização do seu conteúdo com a ajuda do rato. Dentro das classes de “tuples” são também apresentadas as interacções entre processos e o estado de cada processo em determinado momento.

Esta ferramenta está exclusivamente vocacionada para a deteção de erros nos programas do Network–Linda. Não inclui facilidades para a monitorização do grupo de máquinas envolvidas na computação, nem outras funções existentes em outras ferramentas atrás apresentadas. Além disso, a interface gráfica não é um programa autónomo que permita o acompanhamento simultâneo de vários programas em execução. A monitorização é definida por uma opção de compilação que obriga ao lançamento do Tuplescope por invocação do nome do executável criado. Apesar de restringida à deteção de erros não deixa de ser uma ferramenta útil aos utilizadores do sistema Network–Linda.
5.5 Conclusões

Este conjunto de ferramentas aqui apresentado têm objectivos diferentes. O Xpvm foca as áreas da configuração e gestão da máquina virtual paralela e o acompanhamento da execução dos programas; o HeNCE é um sistema gráfico de desenho de aplicações paralelas que inclui ferramentas para muitas outras tarefas (executadas pelo Xpvm e outras); o Tuplescope mais direccionado para a detecção e correção de erros em aplicações distribuídas. No entanto estas ferramentas têm em comum o facto de o seu principal objectivo ser facilitar a utilização dos sistemas de desenvolvimento de aplicações paralelas para os quais foram criadas. Cada vez mais um sistema de desenvolvimento de aplicações paralelas sem ferramentas deste tipo recebe pouca aceitação dos utilizadores.
Capítulo 6

A Interface Gráfica do PARSYS

6.1 Introdução

Este trabalho resultou da necessidade de dotar o sistema PARSYS com um conjunto de ferramentas para o utilizador executar as tarefas de configuração e gestão do conjunto de máquinas envolvidas na computação da forma mais transparente possível. Outro factor importante nos sistemas de desenvolvimento de aplicações distribuídas é a monitorização da execução dos programas. Não é simples para o utilizador controlar uma aplicação distribuída, uma vez que está envolvido, quase sempre, um conjunto vasto de máquinas onde são executadas as tarefas do programa paralelo. A IGP possibilita a monitorização de programas paralelos, acompanhando em tempo de execução vários parâmetros de todas as tarefas de um programa paralelo. Outra tarefa que é possível na interface do PARSYS é o acompanhamento, também em tempo de execução, das alterações sofridas pelas variáveis partilhadas definidas em cada programa paralelo, o que é sem dúvida muito útil não só para se ter a noção clara do estado do programa mas também para detectar eventuais erros na codificação. Se tivermos em mente que algumas aplicações de cálculo científico (que obrigam à resolução de problemas complexos de grande dimensão) têm tempo de execução de semanas ou até meses, verificamos a utilidade desta ferramenta que nos mostra o andamento da computação com muita precisão.

A interface gráfica do sistema PARSYS foi codificada de forma a ser portável entre uma
grande variedade de estações de trabalho cujos sistemas de operação obedecem às normas POSIX. A codificação foi feita numa estação de trabalho HP730 com o sistema de operação HP-UX 9.05. A linguagem de programação utilizada foi o C e foram utilizadas as bibliotecas gráficas do X-Window e Motif1.2. Para o desenho das janelas da interface foi escolhido o construtor de interfaces UNIX da "Visual Edge Software". Nas próximas secções vamos descrever em pormenor as facilidades oferecidas pela IGP e a forma como foram implementadas. Mais informação sobre a IGP pode ser obtida em [30].

6.2 Interacção com o PARSYS

A interface gráfica é totalmente independente das aplicações paralelas em execução. Esta afirmação pode ser visualizada na figura 6.1 onde se pode ver a interface gráfica, tal como todas as outras tarefas, a estabelecer um canal de comunicação ("socket") com uma tarefa do programa a monitorizar. As trocas de informação entre as tarefas e a interface são feitas utilizando as mensagens definidas para esse efeito e apresentadas no capítulo onde é descrito...
o sistema PARSYS. Como esta comunicação é do tipo pergunta/resposta a degradação de desempenho causada pela monitorização na tarefa contactada é directamente proporcional ao intervalo de tempo entre pedidos de informação. O intervalo de tempo entre pedidos representa uma solução de compromisso entre a actualidade da informação e a degradação de desempenho imposta à tarefa contactada.

6.3 A Janela Inicial

A janela de entrada da interface gráfica do sistema PARSYS é apresentada na figura 6.2. A barra de menus no topo da figura contém diversos itens cujas funções vamos apresentar mais à frente. Na restante parte da janela encontramos dois quadros: o da esquerda apresenta-nos os programas paralelos em execução na máquina ("programs"), enquanto o da direita mostra-nos o conjunto de máquinas definidas pelo utilizador para executarem tarefas nos programas PARSYS. Para que a interface possa contactar com todas as aplicações em execução, cada tarefa em início de execução, regista no ficheiro /tmp/parsys.pid informação sobre o programa a que pertence, a sua identificação dentro do programa, o seu porto.
6. A Interface Gráfica do PARSYS

("socket") de contacto, a identificação do seu processo e a identificação do utilizador (quando a tarefa termina retira essa informação). É a partir deste ficheiro que o quadro "Programs" da figura 6.2 é construído. A tabela 6.3 mostra o conteúdo desse ficheiro quando algumas aplicações (a aplicação running com quatro tarefas e a aplicação letters com onze tarefas) se encontram em execução.

<table>
<thead>
<tr>
<th>Program</th>
<th>Prgid</th>
<th>Taskid</th>
<th>Port</th>
<th>Pid</th>
<th>Eff. Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>running</td>
<td>15371</td>
<td>0</td>
<td>15371</td>
<td>15371</td>
<td>100</td>
</tr>
<tr>
<td>running</td>
<td>15371</td>
<td>2</td>
<td>15386</td>
<td>15386</td>
<td>100</td>
</tr>
<tr>
<td>running</td>
<td>15371</td>
<td>5</td>
<td>15406</td>
<td>15406</td>
<td>100</td>
</tr>
<tr>
<td>running</td>
<td>15371</td>
<td>8</td>
<td>15422</td>
<td>15422</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>0</td>
<td>15454</td>
<td>15454</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>1</td>
<td>15457</td>
<td>15457</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>2</td>
<td>15460</td>
<td>15460</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>3</td>
<td>15463</td>
<td>15463</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>4</td>
<td>15466</td>
<td>15466</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>5</td>
<td>15469</td>
<td>15469</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>6</td>
<td>15472</td>
<td>15472</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>7</td>
<td>15475</td>
<td>15475</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>8</td>
<td>15478</td>
<td>15478</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>9</td>
<td>15481</td>
<td>15481</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>10</td>
<td>15484</td>
<td>15484</td>
<td>100</td>
</tr>
<tr>
<td>letters</td>
<td>15454</td>
<td>11</td>
<td>15487</td>
<td>15487</td>
<td>100</td>
</tr>
</tbody>
</table>

Tabela 6.1: O Ficheiro /tmp/parsys.pid

O quadro "Machines" da figura 6.2 mostra-nos um conjunto de ícones (que incluem o nome da máquina e a sua arquitectura) que representam as máquinas onde vão ser executadas as aplicações paralelas. O utilizador controla o conjunto de máquinas através da interface gráfica como vamos ver mais à frente. A interface armazena a informação relativa às máquinas no ficheiro .parsys.hosts (apresentado na tabela 6.3) no directório casa do utilizador. A sintaxe deste ficheiro é semelhante à do ficheiro de sistema /etc/hosts apenas inclui mais um campo para a identificação da arquitectura da máquina.

Para o utilizador acompanhar a execução de uma qualquer aplicação, basta que a selecione na lista de programas. Como consequência o quadro "Machines" da figura 6.2 passa a mostrar o conjunto de máquinas onde se encontram tarefas do programa seleccionado (o que implica a possível remoção e adição de máquinas no quadro). O quadro referido é automaticamente actualizado sempre que o conjunto de máquinas a executar o programa
6. A Interface Gráfica do PARSYS

Figura 6.3: Adicionando uma Máquina ao PARSYS

escolhido seja alterado.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Host Name</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>194.117.27.101</td>
<td>maat.inescn.pt</td>
<td>NEXT</td>
</tr>
<tr>
<td>192.82.214.124</td>
<td>crazy.fe.up.pt</td>
<td>ALPHA</td>
</tr>
<tr>
<td>192.35.246.9</td>
<td>bart.inescn.pt</td>
<td>SUN4</td>
</tr>
<tr>
<td>194.117.27.52</td>
<td>maggoo.inescn.pt</td>
<td>NEXT</td>
</tr>
<tr>
<td>194.117.27.105</td>
<td>yang.inescn.pt</td>
<td>HPPA</td>
</tr>
<tr>
<td>194.117.27.100</td>
<td>bee.inescn.pt</td>
<td>NEXT</td>
</tr>
</tbody>
</table>

Tabela 6.2: O Ficheiro .parsys.hosts

6.4 Gestão das Máquinas

A gestão das máquinas onde o utilizador pretende executar os seus programas paralelos é feita a partir da interface gráfica. A partir do menu Hosts visível na figura 6.2 o utilizador pode adicionar e remover máquinas desse conjunto. A figura 6.3 ilustra a adição de uma nova máquina ao quadro “Machines” da figura 6.2. A adição de uma nova máquina implica a procura de várias informações:

- A sua identificação na rede (o número IP). Obtida através de uma chamada à rotina gethostbyname() e
Figura 6.4: Terminando a execução de um programa PARSYS

- A arquitetura da máquina. Obtida através da execução de um ficheiro de comandos que determina a arquitetura da máquina remota.

Depois de obtidas essas informações, um novo ícone é adicionado ao quadro "Machines" da figura 6.2 e uma nova linha é adicionada ao ficheiro .parsys.hosts. A remoção de uma máquina do grupo é bastante mais simples. Implica apenas a remoção do ícone do quadro "Machines" da figura 6.2 e a remoção da linha referente à máquina em questão do ficheiro .parsys.hosts.

É de notar que estas alterações devem ser feitas antes de colocar em execução o programa paralelo. Programas em execução não são afectados pelas alterações feitas no conjunto de máquinas, uma vez que a migração de tarefas por acção directa do utilizador não se encontra ainda implementada (contudo, há de facto migração de tarefas, mas é apenas implementada pelo mecanismo de tolerância a faltas).

6.5 Gestão dos Programas

A interface gráfica possibilita a terminação de um programa paralelo. Terminar uma aplicação sequencial é uma tarefa simples, visto que basta enviar um sinal de terminação ao programa
em questão. Contudo num ambiente distribuído o processo é um pouco diferente; é necessário localizar todas as tarefas de um programa paralelo no grupo de máquinas e então enviar-lhes um sinal de finalização. A IGP mantém internamente uma tabela onde guarda a informação de contacto de cada tarefa de um programa paralelo, quando o utilizador decide terminar a execução de um determinado programa paralelo uma mensagem de terminação é enviada para cada tarefa, possibilitando assim a terminação normal da aplicação. Um exemplo é apresentado na figura 6.4.

Neste momento não é ainda possível iniciar programas paralelos da interface gráfica. Esta facilidade implica o redireccionamento dos ficheiros de saída ("stdout") e erro ("stderr") para uma janela da interface gráfica tarefa ainda não concluída.

6.6 Monitorização da Execução dos Programas

Uma parte muito importante deste trabalho, que se torna difícil de resumir, foi dedicada à criação de ferramentas que permitissem a monitorização de programas paralelos. Especialmente em programas distribuídos por um conjunto vasto de máquinas é importante para o programador poder centralizar numa única ferramenta toda a informação do comportamento da aplicação. A IGP tem a capacidade de acompanhar a evolução de todas as tarefas de cada programa paralelo, e as alterações sofridas por todas as variáveis partilhadas. Nas próximas secções podemos ver a forma como essa informação é apresentada ao utilizador.

6.6.1 As Tarefas

Ao nível das tarefas a interface gráfica tem a possibilidade de acompanhar diversos parâmetros da execução da aplicação paralela. A figura 6.5 apresenta a tabela de tarefas de um programa de teste em execução. A informação afixada neste quadro é a seguinte:

1. **Taskid**—Este campo contém a identificação interna de cada tarefa, ou seja, o seu número dentro do programa (na figura 6.5 são apresentadas as tarefas com a identificação 0,1,2,...6).
2. **Host** – Este campo apresenta-nos a máquina onde cada tarefa se encontra a executar (na figura 6.5 são apresentadas tarefas a executar nas máquinas *yang*, *romeo* e *juliet*).

3. **Status** – Aqui podemos ver o estado em que se encontram as tarefas do programa paralelo. Como já foi descrito mais atrás (no capítulo: O Sistema PARSYS) as tarefas podem estar nos estados: BABY – tarefa em início de execução, ACTIVE – tarefa em actividade, STALLED – tarefa em estado de espera e DEAD – tarefa terminada). Na figura 6.5 todas as tarefas estão em actividade.

4. **Work Done** – O programador pode definir no código da aplicação um conjunto de "marcos" em zonas importantes do programa utilizando a rotina do PARSYS `step_done()`. O campo **Work Done** na tabela de tarefas procura dar uma ideia do estado em que a aplicação se encontra pela análise do número de barreiras que já foram transpostas.

Esta informação não é estática, varia com o tempo. No início da aplicação o número de tarefas vai crescendo, à medida que a computação vai decorrendo as tarefas vão alterando
6. A Interface Gráfica do PARSYS

![Figura 6.6: A Tabela de Variáveis](image)

Figura 6.6: A Tabela de Variáveis

O seu estado e na finalização o número de tarefas vai diminuindo até a última tarefa terminar. A interface gráfica acompanha todo este processo apresentando ao utilizador, de forma automática, as alterações no número das tarefas, sua localização, estado, e barreiras ultrapassadas.

### 6.6.2 As Variáveis Globais

Na tabela de variáveis o utilizador pode consultar informação acerca das variáveis globais definidas no programa paralelo que está a monitorizar. A figura 6.6 mostra o conteúdo desta tabela na monitorização de um programa de teste. A tabela de variáveis inclui, para cada variável global:

1. **Type** – Este campo apresenta o tipo das variáveis globais definidas no programa paralelo. No PARSYS existem variáveis dependentes da tarefa ("Task Dependent Variables" - TDV) e variáveis independentes da tarefa ("Task Independent Variables" - TIV).
2. **Name** - Este campo contém os nomes das variáveis globais, tal como definidos pelo programador no código da aplicação (Na figura 6.6 temos definidas as variáveis: Global, Duration, Zero, Count, Incr e Localstr).

3. **Changes** - Aqui são contabilizadas as alterações feitas a cada variável.

4. **Value** - Este campo mostra o valor da variável propriamente dito. No caso da variável ainda não ter sido alterada por nenhum processo é apresentado o identificador **NOT SET** (como é o caso da variável Global na figura 6.6), no caso de uma variável que não possa ser apresentada nesta tabela por razões de espaço apresenta-se o identificador **SET** ou os primeiros valores das variáveis (no caso dos "arrays").

5. **...** - Este botão é utilizado quando a variável a inspeccionar não pode ser exibida na tabela por razões de espaço. Esta situação acontece sempre com variáveis dependentes da tarefa (já que são "arrays" com uma zona de memória para cada tarefa do programa) e com variáveis do tipo "string". Ao utilizar este botão um quadro é apresentado com o valor da variável que o utilizador pretende observar. Na figura 6.7 é apresentado o quadro completo da variável **Count** da figura 6.6.

Da mesma forma que na tabela de tarefas, também os valores do quadro das variáveis são alterados à medida que a aplicação vai decorrendo. Este comportamento dinâmico permite que o utilizador tenha a noção exacta do comportamento da sua aplicação, ajudando-o
6.7 Comparação com outras Ferramentas

Existem implementadas uma quantidade apreciável de ferramentas gráficas de apoio à programação paralela em redes de estações de trabalho. Alguns exemplos, que consideramos mais relevantes, foram apresentados neste trabalho: o Tuplescope, o Xpmv e o HeNCE. É de notar que estas ferramentas têm objectivos bastante distintos, pelo que apenas é possível uma comparação em pontos comuns.

Observando as funcionalidades do Tuplescope, por exemplo, verificamos que esta ferramenta foi desenvolvida com o objectivo de ajudar o utilizador na procura e correcção das aplicações do Network–Linda. Contudo esta tarefa não é feita na rede de estações de trabalho, mas sim numa única estação de trabalho a simular o comportamento de computador paralelo. Tem facilidades para executar o programa passo a passo, colocar pontos de paragem no código e inspecionar o conteúdo da memória partilhada. A IGP não permite executar código passo a passo nem a colocação de pontos de paragem no código já que é executada no ambiente real de computação (multiprocessador) e não num ambiente simulado. Contudo, como vimos atrás, é possível na IGP o acompanhamento total das alterações da memória partilhada. Além disso, a ferramenta aqui apresentada possui facilidades para configuração e monitorização do grupo de máquinas a executar código e algumas facilidades de acompanhamento das tarefas que não estão presentes no Tuplescope.

O Xpmv é uma ferramenta com objectivos bastante próximos dos da IGP. As informações mais importantes da execução dos programas PVM recolhidas pelo Xpmv são apresentadas na vista de rede ("network view") e vista espaço-tempo ("space-time view"); esta informação é apresentada aos utilizadores do PARSYS nas tabelas de máquinas (quadro "machines" na figura 6.2) de tarefas da interface gráfica. A vista de rede do Xpmv e a tabela de máquinas têm sensivelmente as mesmas características uma vez que ambas possuem ferramentas para a gestão do conjunto de máquinas. Contudo, na monitorização de um programa paralelo, a IGP acompanha apenas as máquinas envolvidas nesse programa o Xpmv apresenta todas as máquinas definidas ao utilizador, mesmo que não estejam a executar tarefas do programa em questão. Se estivermos a trabalhar com um número apreciável de máquinas (algumas
dezenas) esta facilidade da IGP pode ser bastante útil. A comparação da vista espaço-tempo do Xpvm (onde o Xpvm afixa informação sobre as diferentes tarefas) e a tabela de tarefas da IGP é favorável ao Xpvm; não por o Xpvm apresentar mais informação, mas sim pela forma como a apresenta. O Xpvm apresenta esta informação por meio de gráficos animados que não tivemos oportunidade de implementar na nossa interface até ao momento. Por outro lado, a interface gráfica aqui descrita apresenta facilidades de monitorização das variáveis partilhadas que não estão presentes no Xpvm (no Xpvm é apenas possível acompanhar os parâmetros de entrada e resultados das chamadas às rotinas da biblioteca do PVM).

O HeNCE, como vimos atrás, é um sistema gráfico de desenvolvimento de aplicações distribuídas. Possui facilidades para o desenho dos programas paralelos que não estão ainda desenvolvidas no sistema PARSYS. É um sistema muito completo também na área da monitorização das máquinas envolvidas na computação com uma apresentação muito semelhante à do Xpvm e superior à da IGP. Na área da monitorização das variáveis partilhadas é bastante mais pobre que a IGP mostrando apenas as informações de controle especificadas pelo utilizador no código e sob a forma de texto.

6.8 Conclusões

A utilização da IGP ao longo dos testes que executámos e da opinião de outras pessoas ligadas a esta área revelou que, apesar de ainda existirem algumas lacunas, a interface é de uso intuitivo e permite ao utilizador configurar a informação que pretende observar. As possibilidades desta ferramenta nas áreas da monitorização e configuração do grupo de máquinas, do acompanhamento dos programas paralelos e das alterações das variáveis distribuídas facilitam a utilização do sistema PARSYS.
Capítulo 7

Conclusões e Aspectos para um Trabalho Futuro

7.1 Principais Contributos

Este trabalho teve como principal objectivo o desenvolvimento das ferramentas gráficas de apoio ao sistema PARSYS. Devido ao modelo de programação do sistema PARSYS não era possível utilizar ferramentas já existentes na interface com o utilizador deste sistema. Embora algumas funcionalidades da IGP sejam comuns a outras ferramentas (a que resulta naturalmente das necessidades dos utilizadores deste tipo de sistemas), a nossa abordagem em pontos como o acompanhamento das tarefas e principalmente na monitorização da memória partilhada distribuída apresenta ideias inovadoras. O conjunto das ferramentas da IGP, potência a utilização do sistema PARSYS na medida em que confere aos utilizadores deste sistema, um controlo efectivo sobre as aplicações paralelas e sobre o conjunto de máquinas responsáveis pela computação.

Uma parte do trabalho a montante deste desenvolvimento, que não posso deixar de referir, foi o estudo das ferramentas de trabalho utilizadas, que permitiram a aquisição de conhecimentos muito úteis em projectos futuros. Outra parte, encontra-se apresentada no início deste documento e está relacionada com o estudo do estado da arte na área dos sistemas de desenvolvimento de aplicações distribuídas em redes de estações de trabalho. Este estudo,
sem pretender ser uma referência exaustiva para todos os sistemas de desenvolvimento existentes, procurou apresentar as principais direções existentes nesta área. Foram também apresentadas as ferramentas de interface com o utilizador mais representativas nesta área, e comparadas as suas funcionalidades com as da IGP.

7.2 Aspectos a Desenvolver Futuramente

A implementação da IGP não está de forma alguma concluída. Estão definidas algumas orientações para a melhoria das facilidades oferecidas pela IGP e adição de novas ferramentas. As principais ideias são:

- **Monitorização da Rede** – Vamos procurar apresentar ao utilizador um conjunto de informações acerca das máquinas onde vão ser executadas as aplicações distribuídas. Vamos mostrar ao utilizador informação sobre: O número de utilizadores na máquina, a sua carga, o número de processos e a memória livre, por exemplo. Uma parte desse trabalho já foi elaborado no sistema MONSYS [16] e será adaptado para ser incluído na IGP.

- **Monitorização de Programas Paralelos** – Está também previsto um estudo de viabilidade com vista à implementação na interface de ferramentas para a monitorização do desempenho das aplicações paralelas. O melhoramento das facilidades de detecção e correção de erros está também previsto.

- **Ambiente de programação gráfico** – Um objectivo ambicioso é o desenvolvimento de um ambiente de computação inteiramente gráfico comparável por exemplo ao HeNCE. Deverá possibilitar executar as tarefas de criação do programa, compilação, correção de erros, monitorização da execução e recolha de resultados, por exemplo, sem abandonar a interface gráfica do sistema de desenvolvimento.
Bibliografia


Enterprise: An interactive graphical programming environment for distributed software 


Equipment Corporation, July 1996.


Research Center for Computational Field Simulation, P.O.Box 6176, June 1994.

heterogeneous distributed computing systems. In Proceedings 1992 Workshop on 


system for the support of a parallel processing environment. In Computing Systems in 

Engenharia (FEUP), September 1995.


