PROCESSAMENTO PARALELO

April 5, 2004

Contents

1 INTRODUÇÃO

Cerca de 90% dos computadores atuais são o que chamados de "Arquitetura de von Neumann", ou "Arquitetura Serial", pois empregam um único processador. Essa arquitetura, aliada aos avanços da microeletrônica, ofertou-nos o atual mercado de computadores, rápidos e baratos. Porém, tal arquitetura enfrenta um limite de velocidade que é ditado pelas leis da física. O tempo que um sinal elétrico gasta para trafegar entre dois pontos de um circuito eletrônico é muito pequeno, porém não é igual a zero. Em outras palavras, isto corresponde a dizer que existe um limite para a velocidade de relógio das CPUs e, infelizmente, ele não está muito distante. Como então continuar com a evolução dos computadores? Essa é a pergunta que tem ocupado a cabeça de muitos pesquisadores e desde a segunda metade desta década, várias soluções foram propostas. A principal resposta vem da comparação entre nosso cérebro e um processador. É sabido que o sinal elétrico trafegando por dentro de um CI é muito mais veloz que o trânsito de impulsos nervosos entre nossos neurônios. É claro que, para fazer operações numéricas, comparar e classificar, o computador é mais rápido. Mas, por outro lado, ele é inferior, pois não pensa, não inova e não aprende, apenas segue passos programados. Por exemplo, com um único olhar em uma sala identificamos imediatamente centenas de objetos. Já um computador, mesmo o mais sofisticado, apenas consegue identificar os objetos mais simples. Como será que o cérebro consegue ser superior aos processadores, se o nosso neurônio é muito mais lento que um circuito eletrônico? A resposta é óbvia: porque temos vários bilhões de neurônios operando em paralelo. Ora, por que, ao invés de construirmos CPUs velozes e gigantescas, não usamos várias CPUs, simples e confiáveis, operando em paralelo? Assim recentemente optou-se por construir máquinas com mais de um processador, estas máquinas tem um maior poder de processamento sem a necessidade de um único "chip" potente, ou seja, pode-se obter um computador com uma capacidade de processar um maior número de instruções sem ser necessário o projeto de um processador revolucionário com tal capacidade.Chegamos assim à idéia básica do processamento paralelo, que é a esperança para os próximo anos.

2 PROCESSAMENTO PARALELO

O processamento paralelo consiste em acelerar a execucão de um programa dividindo este em várias partes que podem ser executadas em simultâneo, cada uma delas em diferentes processadores. Na prática, é dificil dividir um programa de maneira a que cada um dos processadores execute as suas intruções sem que interfiram uns com os outros. Na maioria dos computadores apenas se usa um único processador, mas existem certos modelos que chegam a ter centenas de processadores. É tambem possivel realizar processamento paralelo através do uso de um conjunto de computadores(heterogêneos ou não) bastando para isso liguar os computadores em rede a que se dá o nome de cluster, ou seja, as máquinas são conectadas via rede para formar um único computador. Embora poucas tarefas possam ser paralelizadas com facilidade, um cluster de computadores é útil para uma enorme quantidade de problemas científicos.

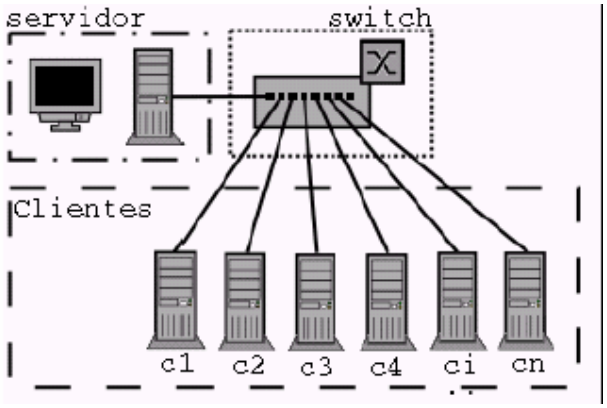

Figure 1: Diagrama de um cluster.

O princípio de funcionamento é simples. O servidor divide as tarefas em suas partes independentes (ditas concorrentes), a seguir, distribuí estas tarefas entre os vários computadores que fazem parte do cluster. As tarefas são processadas e então os resultados são enviados para o servidor. Para que o sistema funcione é necessário um servidor, vários clientes, uma biblioteca para troca de mensagens e o hardware para conexão via rede dos diversos computadores.

3 LIMITACÕES

Será que não existe um limite no uso de processamento paralelo? A resposta é sim, primeiro porque o trabalho dos processadores só será eficiente se estiverem perfeitamente sincronizados e equilibrados. Este ponto é importante: todos os processadores devem ter a mesma carga de trabalho de modo a se obter o melhor rendimento quando se usa mais que um processador. Além disso, deve haver um limite para a quantidade de processadores que podem trabalhar em paralelo. A partir deste limite, quanto mais processadores colocamos, pioramos o desempenho e em consequência, aumentamos o tempo de execucão do número de instruções de um programa. Temos então dois grandes problemas: até quanto podemos paralelizar uma tarefa e até quantos processadores devem ser alocados? A partir daí, surgem outras questões: como sincronizar esses processadores de forma a que um não repita o trabalho do outro e como garantir o balanceamento da carga de trabalho? Agora temos condições de entender porque se diz que as dificuldades presentes no projeto do hardware de máquinas paralelas não são tão complexas quando comparados com os problemas da sua programação isto porque há a necessidade de programas especificamente produzidos para estas máquinas de forma a aproveitar todo o seu potencial o que não é facil de encontrar pois a maioria absoluta dos programas escritos até hoje se destinam a computadores com apenas um processador. Diz-se que os computadores estão sempre uma geração atrasada em relação às nossas necessidades e os programas, duas gerações atrasadas. Em suma, um desafio maior que o projeto de supercomputadores é a sua programação.

4 TIPOS DE COMPUTADORES PARALELOS

É muito difícil a tarefa de classificar computadores paralelos. Já foram feitas diversas sugestões. A classificação definitiva ainda está por vir. Porém, a que trouxe melhores resultados e ainda hoje é usada, é a proposta por Flynn. Essa classificação está baseada em dois conceitos: fluxo de instruções e fluxo de dados. O fluxo de instruções está relacionado com o programa que o processador executa, enquanto que o fluxo de dados está relacionado com os operandos manipulados por essas instruções. O fluxo de instruções e o fluxo de dados são considerados independentes e por isso existem quatro combinações possíveis, como mostrado na Figura 2.

		Fluxo de Instruções (I)	
		Serial ou Único (**SI**)	Paralelo ou Múltiplo (**MI**)
Fluxo de Dados (D)	Serial ou Único (**SD**)	**SISD**	**MISD**
	Paralelo ou Múltiplo (**MD**)	**SIMD**	**MIMD**

Figure 2: Fluxo de instruções e dados.

SISD - Instrução Única, Dado Único (Single Instruction Single Data)

Essa arquitetura é usada nos computadores que temos em casa. Segue o proposto por von Neumann e é por isso denominada de "Arquitetura de von Neumann", também chamado de computador serial. Temos um único fluxo de instruções (SI), caracterizado pelo contador de programa da CPU, que opera sobre um único dado (SD) por vez. A Figura 3 apresenta um diagrama de blocos ilustrativo deste caso. Apesar de serem representados como módulos separados, a memória de instruções e a memória de dados são, para este caso, a mesma.

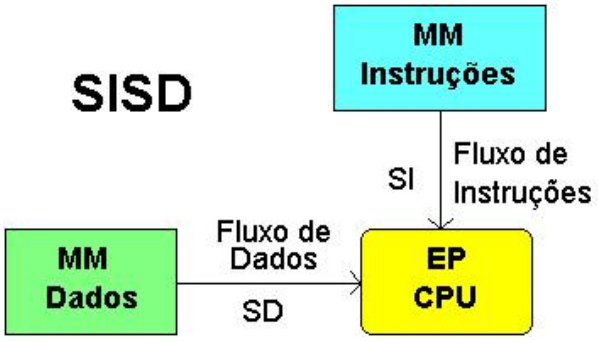

Figure 3: Arquitetura SISD (Single Instruction Single Data).

SIMD - Única Instrução, Múltiplos Dados (Single Instruction Multiple Data)

De início esta arquitetura paralela pode parecer estranha, mas como será constatado adiante, ela não só é conhecida, como também já foi muito utilizada. Para facilitar o conceito de um computador que utilize uma única instrução operando sobre uma grande quantidade de dados, devemos consultar o diagrama da Figura 4.

A arquitetura mostrada apresenta N processadores (EP), sendo que cada processador trabalha sobre um dado distinto, que vem de cada um dos N módulos de memória (MD). O ponto importante é que todos os processadores (EPi) trabalham sincronizados e executam todos a mesma instrução, ou seja, cada instrução é passada, ao mesmo tempo, para os N EPs. Assim, os processadores executam a mesma instrução, porém sobre um dado diferente. Como é fácil de concluir, um computador com essa arquitetura é capaz de operar um vetor de dados por vez. Vem daí seu nome de Computador Vetorial, ou "Array Processor".

Um grande exemplo desta arquitetura são os famosos computadores Cray. Outro exemplo é o conjunto de instruções MMX. Eles são muito usados quando um mesmo programa deve ser executado sobre uma grande massa de dados, como é o caso de prospeção de petróleo. Note que essa arquitetura não sofre com problemas de sincronização, pois existe um único programa em execução.

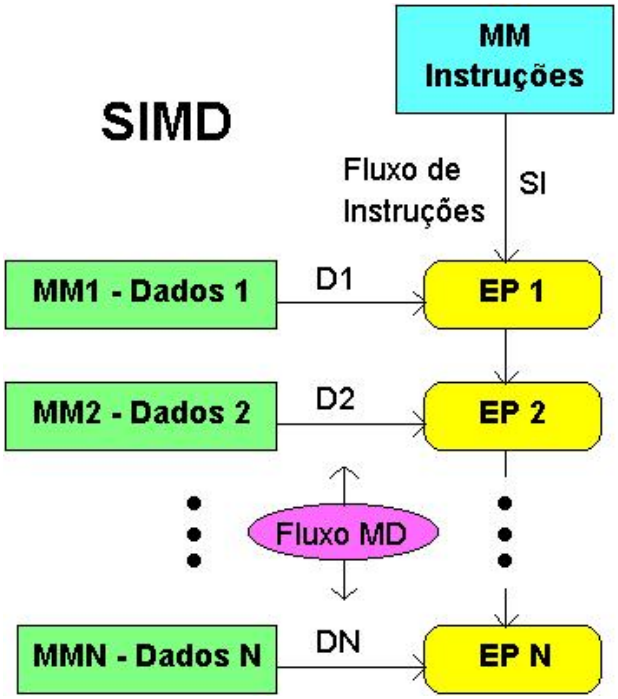

Figure 4: Aquitetura SIMD (Single Instruction Multiple Data).

MISD - Múltiplas Instruções, Dado Único (Multiple Instruction Single Data)

Essa arquitetura é um pouco mais difícil de ser explicada. Tentemos imaginar como é que se pode fazer múltiplas operações (MI) sobre um mesmo dado (SD). Os próprios pesquisadores têm opiniões divergentes sobre esse assunto. De forma simples, vamos estudar a Figura 5, onde pode ser visto que, apesar de existir um único fluxo de dados, existem vários dados sendo operados ao mesmo tempo. Essa figura é conhecida na literatura especializada com o nome de "pipeline" ou linha de produção.

A figura mostra que N processadores operam sobre K diferentes dados. Façamos uma analogia com uma linha de montagem de carros. Vamos supor que um carro leve 6 horas para ser montado e que a tarefa de montagem possa ser dividida em 12 equipas, numeradas em sequência de 1 até 12, cada uma gastando meia hora. É fácil de ver que não precisamos esperar que a última equipa termine de montar um carro para a equipa 1 inicie a montagem do carro seguinte. Na verdade, todas elas trabalham simultaneamente, montando vários carros ao mesmo tempo. Por isso, com alguma liberdade, é costume classificá-la de paralelismo temporal. Então, apesar do tempo de montagem consumir 6 horas, a saída da linha de produção entrega um carro a cada meia hora. Agora é fácil voltar aos computadores e ver que temos um único fluxo de dados (SD), porém vários instruções (MI) processam esses dados, ao mesmo tempo.

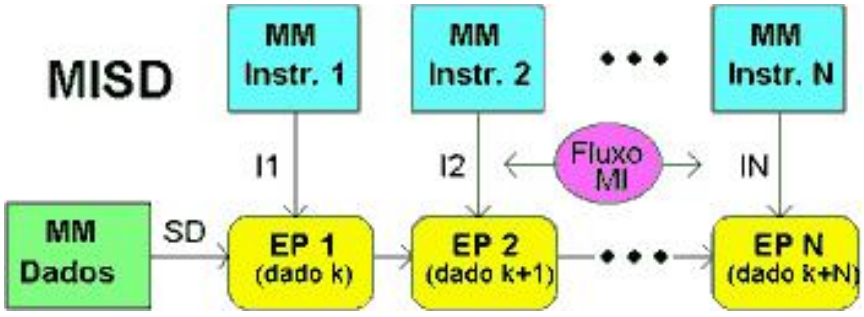

Figure 5: Arquitetura MISD (Multiple Instruction Single Data).

MIMD - Múltiplas Instruções, Múltiplos Dados (Multiple Instruction Multiple Data)

Essa é a arquitetura que esperaríamos encontrar em um computador paralelo. Temos vários dados (MD) sendo operados por vários instruções (MI), simultaneamente. Essa é a arquitetura mais usada pelos modernos supercomputadores. Nesse caso, é importante que os processadores possam se comunicar entre si para fazer a sincronização e trocar informações. Além disso, é necessário ter uma memória, chamada de global, onde todos processadores possam disponibilizar, para os demais, os resultados intermediários. A Figura 6 apresenta uma solução genérica para essa arquitetura MIMD.

Note que agora temos N processadores e dois tipos de memória: a local e a global. A

memória global pode ser acessada por qualquer um dos processadores, por isso existe a chamada "Estrutura de Comunicação", que disponibiliza a memória global para qualquer um dos processadores. Uma única memória global criaria um grande gargalo, pois só um processador poderia acessá-la por vez. Por isso, a figura apresenta duas memórias globais e, com isso, dois processadores podem ser atendidos ao mesmo tempo.

Para evitar uma quantidade excessiva de acessos a essa memória, os processadores possuem a chamada memória local, onde está a maioria das suas instruções e dos dados que devam ser operados. Essa memória local evita que a estrutura de comunicação se transforme num enorme gargalo. Os processadores precisam trocar informações e, no caso desta figura, a própria estrutura de comunicação se encarrega desta tarefa.

Uma simples análise da arquitetura MIMD da Figura 6 mostra que agora existe um fluxo de múltiplos dados (MD) sendo operado por um fluxo de múltiplas instruções (MI). Agora sim é necessária a genialidade dos programadores, pois como conseguir que uma estrutura deste tipo, imagine 1.024 processadores, trabalhe de forma sincronizada para gerar algum resultado? Se já é difícil escrever e depurar programas seriais, imagine fazer isso em um computador com 1.024 diferentes programas trabalhando sobre 1.024 dados diferentes.

Figure 6: Arquitetura MIMD (Multiple Instruction Multiple Data).

NOTA:Nas Figuras 3 a 6 apresentamos os diagramas de blocos para essas 4 combinações. Nessas figuras, a sigla MM representa o "Módulo de Memória" e a sigla EP representa o "Elemento de Processamento", ou seja, o processador.

5 TIPOS DE PROCESSAMENTO PARALELO

Descreve-se a seguir os diferentes tipos de estruturas utilizadas para implementar o processamento paralelo.

1. *Processamento paralelo com Swar(Simd Withn a Register)*
   Consiste em utilizar as instruções MMX disponibilizadas nos novos processadores (Pentium MMX), para realizar tarefas em paralelo. Requer programação em baixo nível. Observe que com swar você pode fazer processamento paralelo em uma máquina com um único processador.

2. *Processamento paralelo com SMP(Symetric Multi Processor)*
   SMP é uma sigla que designa computadores com mais de um processador com as mesmas características, daí o termo Symetric Multi Processor. Os processadores compartilham o mesmo BUS e a mesma memória. Veja na Figura 7 uma placa mãe modelo ASUSCUV4X- DLS com dois slots para processadores PIII de 1000MHz.

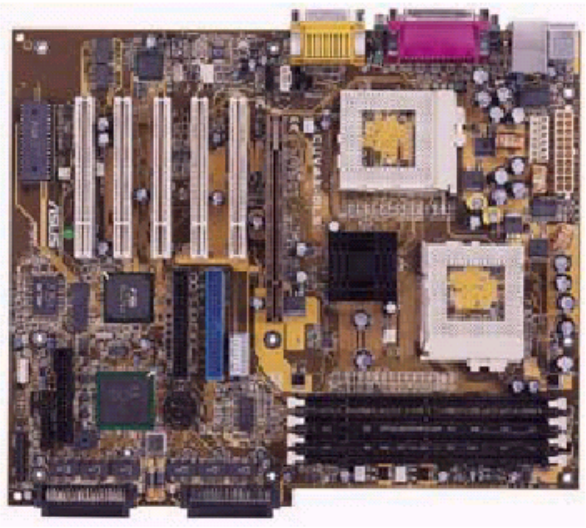

Figure 7: Placa mãe ASUS-CUV4X-DLS com slot para 2 processadores.

- Requisitos: Os programas devem ser desenvolvidos com o uso de múltiplas threads (multi-threadings) ou múltiplos-processos (multi-processing).
- Vantagens: Relativamente fácil de programar.
- Desvantagens: Requer máquinas com dois ou mais processadores (são máquinas caras).

3. *Processamento paralelo com cluster Beowulf*
   Beowulf é uma tecnologia de cluster que agrupa computadores rodando GNU/Linux para formar um supercomputador virtual via processamento paralelo (distribuído). Veja nas Figuras 8 e 9 exemplos de clusters tipo Beowulf.

   - Requisitos: Conjunto de computadores (sem teclado, sem monitor e sem mouse) conectados em rede para processamento paralelo (uso exclusivo). Requer o uso de uma biblioteca de mensagens como PVM ou MPI, ou o uso de múltiplos processos com o Mosix.
   - Vantagens: Manutenção facilitada, redução do número de problemas ocasionados pela instalação de pacotes desnecessários. Menor custo das máquinas e de manutenção.
   - Desvantagens: As máquinas tem seu uso limitado ao processamento definido pelo servidor.

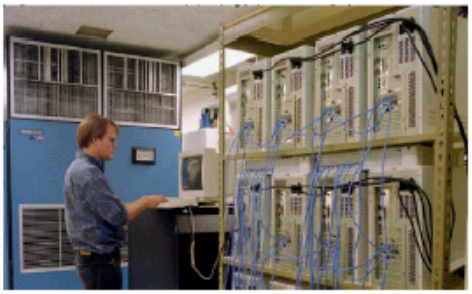

Figure 8: Exemplo de cluster estilo Beowulf

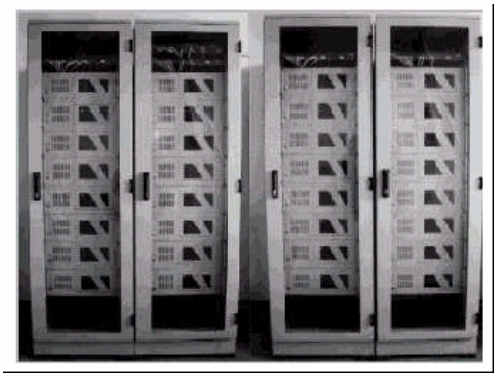

Figure 9: Cluster da University Koblenz-Landau Institute of Physics.

4. *Processamento paralelo com cluster de workstation*
   Um cluster de workstation é um conjunto de computadores completos (com teclado, monitor, mouse), conectados em rede, e que cumprem duas funções:

   i) o uso diário, com diversos tipos de programas como processadores de texto e planilhas,

   ii) o uso para processamento paralelo pesado no final do dia e/ou nos fins de semana. A Figura 10, mostra um exemplo de cluster de workstation.

   - Requisitos: As máquinas devem ser completas e independentes. Requer o uso de uma biblioteca de troca de mensagens como PVM ou MPI.
   - Vantagens: Possibilita o uso das máquinas por diferentes usuários para realização de suas tarefas rotineiras.
   - Desvantagens: Como vários usuários estão utilizando os processadores para outras tarefas, o desempenho do sistema é reduzido. Na prática reduz o uso do cluster ao final do dia e nos fins de semana. Tem um custo maior por máquina e mais problemas com a manutenção do sistema.

Figure 10: Exemplo de um Şcluster de workstationŤ (Distributed Computing Laboratory, Department of Mathematical Sciences, Cameron University, Lawton).

5. *Processamento paralelo em um cluster com MOSIX*
   O MOSIX é um adendo ao kernel do GNU/LINUX que adiciona ao mesmo capacidades de computação com cluster. Possibilitando que as estações do cluster, baseadas em X86/Pentium/AMD, trabalhem de forma cooperativa, como sendo um único sistema. A migração dos processos entre as várias máquinas do cluster é automática. Isso permite que programas antigos funcionem num ambiente de cluster com pouquíssimas alterações.

   Características do MOSIX: Balanceamento dinâmico e inteligente de carga, uso com cluster heterogêneo, transparência, escabilidade, descentralização e autonomia dos nós. Migração preemptiva dos processos com uso de algoritmos probabilísticos. Comunicação entre núcleos eficiente, controle descentralizado.

   - Requisitos: Requer a recompilação do kernel com a inclusão do Mosix ou instalação de kernel em pacote (como os pacotes rpm do GNU/Linux/RedHat).
   - Vantagens: O trabalho de programação é reduzido, exigindo apenas a implementação dos mecanismos de troca de mensagens entre os diferentes processos. Otimização do uso das máquinas do cluster com a migração automática dos processos.
   - Desvantagens: Exige a recompilação do kernel.

6 COMUNICACAO ENTRE PROCESSOS

Uma aplicação paralela ou multiprocessada é formada por um conjunto de processos que cooperam na obtenção de um resultado a que se dá o nome de multitasking (múltiplos processos sendo executados simultaneamente). Tipicamente, esses processos precisam se comunicar para trocar dados e pedidos entre si. Como cada processo progride de forma independente dos demais, muitas vezes é necessário definir pontos de sincronização entre eles, de forma a garantir a coerência da computação global. Nos últimos anos, vários sistemas operacionais passaram a oferecer suporte a um tipo de processos denominados threads, ou processos de peso leve, que podem se comunicar através de variáveis globais compartilhadas. Do ponto de vista do programador paralelo, threads são importantes em máquinas com vários processadores, onde eles permitem o desenvolvimento de aplicações paralelas cujas partes se comunicam por memória compartilhada (memória do sistema que pode ser acessada por cada processador individualmente) e, além disso, muitas vezes executam de forma mais eficiente do que processos tradicionais. A comunicação por memória compartilhada requer um grande cuidado na maneira como se programa para ela e com o problema de sincronização caso se queira utilizar simultaneamente mais de um processador na execução de um programa, ou seja, dois ou mais processadores não devem acessar simultaneamente a mesma posição da memória se, pelo menos um deles, desejar escrever sobre aquela posição só devendo ser permitido os processadores acederem a mesma posição da memória se ambos desejarem ler aquela posição. Para garantir uma relação de precedência

entre comandos de processos distintos, o programador deve utilizar comandos explícitos de sincronização, que devem ser providos pela linguagem ou sistema operacional de forma a que as regras acima sejam obedecidas na maior parte possível do tempo. Existem meios de se impedir que outro processador acesse uma dada posição da memória enquanto outro a está acessando, porém este método força o segundo processador a esperar que o primeiro termine de usar aquela posição, isto pode ser feito com threads, onde o mecanismo de sincronização mais frequentemente oferecido é o de variáveis de condição, que correspondem a um serviço que permite que um processo permaneça bloqueado até que uma condição seja verdadeira. O desenvolvimento de aplicações baseadas em variáveis compartilhadas e condições pode ser bastante complexo. Os bloqueios e sinalizações associados a variáveis de condição podem estar espalhados por um código extenso, e qualquer acesso a uma variável global pode representar uma necessidade de sincronização, tornando árdua a tarefa de depuração do programa paralelo. Muitos autores acreditam que a forma alternativa de comunicação entre processos é a utilização de memória distribuida onde cada processador possue a sua própria memória e quando um processador necessita de informações armazenadas na memória de outro processador estes "trocam mensagens", o que no final acaba por representar uma base mais segura para o desenvolvimento de aplicações paralelas. De qualquer maneira, a troca de mensagens é a forma natural de comunicação entre processos que não compartilham memória (por exemplo, entre processos que executam nos vários nós de um cluster de computadores). Em sua forma mais básica, um mecanismo de troca de mensagens fornece uma primitiva para envio de dados para um processo específico (send) e de recebimento de dados de um processo específico ou de qualquer processo que tenha enviado alguma mensagem (receive). Muitas abstrações e expansões podem ser oferecidas em cima desse modelo básico. Por exemplo, pode-se citar bibliotecas de programação paralela que, além de fornecerem diversas variantes de send e receive, muitas vezes oferecem mecanismos para comunicação e sincronização de grupos.

7 ALGUMAS FERRAMENTAS DE PROGRAMAÇÃO PARALELA

Um número crescente de ferramentas para programação paralela estão disponíveis para o desenvolvimento de aplicações paralelas. Em 1995, o IEEE definiu um padrão de interface de programação com threads, conhecido como Posix threads ou Pthreads. Desde então, diversos sistemas operacionais vêm incluindo suporte a esse padrão, também implementado por uma biblioteca disponível publicamente. No modelo Pthreads, vários threads podem ser definidos dentro de um único processo. O processo funciona como o espaço de endereços de memória que uma determinada aplicação pode utilizar. Os diversos threads associados a um processo compartilham memória: quando um thread realiza escritas na memória (ex: atribuições a variáveis), outro thread pode ler o resultado. O sistema operacional gerencia a alocação de processadores para a execução dos threads como se eles fossem processos independentes. Em uma máquina com múltiplos processadores, diferentes threads de uma mesma aplicação podem ser simultaneamente executados em diferentes CPUs. A interface Pthreads define basicamente os seguintes grupos de rotinas: rotinas de controle, que permitem criar, eliminar e suspender a execução de threads, rotinas de sincronização, entre as quais as que manipulam as variáveis de condição descritas na seção anterior, e rotinas para controle de escalonamento, que permitem alterar as prioridades relativas dos diversos threads de uma aplicação. Com o aumento da banda passante oferecida pelas redes locais, passou a ser comum o uso de redes de estações de trabalho como se fossem máquinas multiprocessadoras. Surgiram então desde o final dos anos 80 uma série de bibliotecas de programação paralela para este tipo de ambiente. Entre elas, provavelmente a que obteve a maior divulgação e difusão de uso foi o PVM (Parallel Virtual Machine). PVM foi inicialmente implementada para máquinas Unix que se comunicassem por TCP/IP, aproveitando assim o impulso de crescimento da Internet. Atualmente existem versões para diversos outros sistemas. A biblioteca PVM é disponível publicamente, através da Internet. Em 1995, autores e usuários de bibliotecas de troca de mensagens fomaram um grupo com o intuito de definir uma interface de programação paralela comum, de modo a evitar a proliferação de programas escritos em diferentes "dialetos". Esse grupo envolveu aproximadamente 40 pessoas de 60 instituições. O resultado desse esforço foi a definição do MPI (Message Passing Interface), que inclui a definição de interfaces de um conjunto de rotinas para comunicação por troca de mensagens. Desde a publicação do MPI, em 1995, diversas implementacões já foram desenvolvidas, sendo várias delas disponíveis publicamente. O desenvolvimento dessas várias implementacões totalmente compatíveis com a definição original vem fazendo do MPI um padrão ad hoc para bibliotecas de troca de mensagens. O núcleo do MPI é formado por rotinas de comunicação entre pares de processos (tipo send e receive). Além desse núcleo, o MPI define várias outras rotinas, entre as quais se destacam rotinas de comunicação e sincronização em grupo. A instalação das bibliotecas das diferentes versões de MPI costuma ser simples. Embora exija uma certa intimidade com os conceitos de programação paralela, a programação com MPI não apresenta maiores dificuldades.

8 BIBLIOTECAS PARA DESENVOLVIMENTO DE PROGRAMAS UTILIZANDO PROCESSAMENTO PARALELO

1. **Processos**
   De um modo geral, os computadores com sistemas operacionais multi-tarefa disponibilizam um conjunto de funções para divisão e compartilhamento do processador e da memória. Estes sistemas costumam disponibilizar chamadas ao kernel que possibilitam a criação de múltiplos processos. Se a máquina tem mais de um processador, o sistema operacional distribui os processos pelos processadores. No GNU/Linux e nas variantes do Unix, um processo pode ser clonado com a função fork(). A comunicação entre os processos é feita de forma simplificada com o uso de pipes.

   - Requisitos: Requer o aprendizado do uso das instruções fork (para clonar processos) e pipe (para comunicação entre os processos).
   - Vantagens: Pode ser utilizado com Mosix, não sendo necessário acrescentar mecanismos de distribuição dos processos.
   - Desvantagens: O Mosix só é disponível na plataforma GNU/LINUX.

2. **Threads**
   Threads são múltiplos caminhos de execução que rodam concorrentemente na memória compartilhada e que compartilham os mesmos recursos e sinais do processo pai. Uma thread é um processo simplificado, mais leve ou "light", custa pouco para o sistema operacional, sendo fácil de criar, manter e gerenciar. O padrão de implementação de threads é o POSIX 1003.1c threads standard.

   - Requisitos: Requer o conhecimento da biblioteca de programação com threads (também conhecida como PThreads).
   - Vantagens: Em poucas palavras é o pacote definitivo para o desenvolvimento de programação em larga escala no Linux. Relativamente fácil de programar. O GNU/Linux tem total suporte ao Pthreads.
   - Desvantagens: Não pode ser utilizado com MOSIX.

3. **PVM (Parallel Virtual Machine)**
   É a biblioteca mais utilizada para processamento distribuído. É o padrão de fato da indústria de software. O PVM se baseia em duas primitivas básicas:
   i) envie mensagem
   ii) receba mensagem.
   É de fácil utilização, mas não é tão poderoso quando comparado com o MPI. O usuário deve configurar as máquinas para que sejam o mais idênticas possível, facilitando a manutenção e estabelecendo uma relação de confiança entre elas. Usar

rhosts e rsh é a forma mais simples de conseguir isso. O usuário roda o gerenciador do PVM, adiciona máquinas ao cluster e depois simplesmente executa o programa feito usando as bibliotecas PVM.

- Requisitos: Para o desenvolvimento dos programas é necessário conhecer a biblioteca PVM. É um sistema explicito, ou seja, cabe ao programador dividir as tarefas através da troca de mensagens.
- Vantagens: Possibilita o uso do processamento distribuído. É o mais utilizado. Alguns programas de engenharia e matemática geram código automaticamente para o PVM.
- Desvantagens: Não é mais o padrão. O desenvolvimento dos programas fica bem mais complicado quando comparado com threads.

4. **MPI (Message Passing Interface)**
É um método que inclui conceitos novos como rank (cada processo tem uma identificação única, crescente), group (conjunto ordenado de processos) e communicator (uma coleção de grupos), que permitem um gerenciamento mais complexo (e inteligente) do uso de cada máquina do cluster. O MPI tem opções mais avançadas (que o PVM), como envio de mensagens broadcast (para todas as máquinas do cluster) e multicast (para um grupo específico de máquinas), assim como um melhor controle sobre o tratamento que cada mensagem terá ao ser recebida por outro ponto do cluster. A configuração do MPI depende da implementação utilizada e algumas delas chegam a instalar front-ends para compiladores em C e Fortran, mas a forma geral de uso é semelhante.

   - Requisitos: Requer o conhecimento de um sistema bastante complexo de troca de mensagens, o MPI. É um método explicito.
   - Vantagens: É o novo padrão para processamento distribuído, embora ainda seja menos utilizado que o PVM.
   - Desvantagens: Na prática significa aprender uma nova linguagem de programação. É um padrão da indústria com várias implementações individuais. É complicado.

9 CONCLUSÃO

A primeira conclusão que se pode tirar é que o processamento paralelo é a malhor maneira para continuar com a evolução dos computadores visto haver um limite para o processamento e que o uso do processamento paralelo (principalmente em um clusters)revela ser uma ferramenta extremamente poderosa, possibilitando o desenvolvimento de simulações avançadas em sistemas de baixo custo. Como visto, os computadores podem ser utilizados para processamento comum de dia e para processamento pesado a noite e nos finais de semana, aproveitando melhor o parque de máquinas instaladas. Viu-se também que os programadores precisam aprender os conceitos básicos de processamento paralelo e as diferentes formas de distribuição do processsamento (processos, threads, PVM, MPI). O mecanismo mais fácil de desenvolver processamento paralelo envolve a utilização de multiplas threads, e a seguir multiplos processos sendo aconselhável iniciar com estes mecanismos. Posteriormente, pode-se trabalhar com bibliotecas especializadas como PVM e MPI. Em ambos os casos procure utilizar uma biblioteca padrão e multi-plataforma.Verificou-se que a grande vantagem do uso de threads é o compartilhamento da memória pelas threads. Sendo a memória compartilhada a implementação de códigos paralelos é extremamente simplificada.