Ambientes de Execução Paralela

Prof. Celso Maciel da Costa

Faculdade de Informática da PUCRS
Av. Ipiranga, 6681 -CEP 90619-900 - Porto Alegre, RS, Brasil -
Fone: +55 51 320-3611
E-mail: celso@inf.pucrs.br

Resumo

Por ambientes de suporte à execução paralela pode se entender os sistemas operacionais para máquinas paralelas e os ambientes de execução das ferramentas de programação paralela. Estes ambientes devem prover mecanismos de gerenciamento de recursos que permitam aos usuários desenvolver e executar as suas aplicações de maneira apropriada. Uma alternativa de implementação destes ambientes é a construção de um novo sistema operacional, com funções de gerenciamento de processos locais e remotos, com suporte à comunicação e a sincronização entre processos. Este sistema também necessita de mecanismos que permitam ao usuário acessar a máquina paralela. Outra alternativa é a utilização de um sistema operacional tradicional acrescido destas funcionalidades. Este artigo apresenta um estudo sobre estas alternativas e apresenta um panorama das soluções empregadas atualmente.

1. Arquiteturas Paralelas

As máquinas paralelas MIMD (Multiple Instruction, Multiple Data)[7] são compostas de processadores que operam de maneira independente. Pode-se classificar estas máquinas em memória compartilhada e memória distribuída. As máquinas com memória compartilhada UMA (Uniform Memory Architecture) preservam um tempo uniforme de acesso à memória. Nas máquinas NUMA (Non Uniform Memory Architecture) a memória é distribuída no conjunto de processadores e o tempo de acesso a memória é não uniforme. Isto é, o acesso é mais ou menos rápido dependendo se a memória endereçada é local ou não ao processador.

Nas máquinas MIMD sem memória comum, NORMA (No Remote Memory Architectutre), cada processador possui sua própria memória e não pode acessar a memória de um outro processador. Estas máquinas são formadas por processadores padrão e a noção deste tipo de máquina pode ser genérica, podendo incluir tanto uma máquina paralela com os processadores interconectados por um switch, como um cluster[9] de estações de trabalho.

Uma das tendências da nova geração de sistemas paralelos e distribuídos é integrar máquinas paralelas com redes de estações de trabalho. Neste sentido, as redes locais de alto desempenho têm recebido grande atenção, principalmente pelo constante aprimoramento e o baixo custo que apresentam, se comparadas às máquinas paralelas. O desempenho inferior que uma rede de estações de trabalho apresentava, em relação às máquinas paralelas, devido à velocidade de comunicação, tem diminuído sensivelmente com o surgimento de novas tecnologias, como Myrinet [17], dentre outras.

Para se escrever um programa paralelo se necessita de um modelo de programa. Os tipos de máquina (UMA, NUMA, NORMA), possuem uma influência fundamental sobre os modelos de programação existentes. No modelo de programa para máquinas com memória compartilhada os processos compartilham uma memória. Eles cooperam utilizando os dados compartilhados e a sincronização é obtida com os conceitos e operadores tradicionais (semáforos, monitores). Pela ausência de uma memória comum, a cooperação de processos em uma máquina com memória distribuída é feita por troca de mensagens. Se um processo deseja enviar uma estrutura de dados a um outro processo que executa em um outro processador ele deve enviar (operação send) estes dados ao processador no qual executa o processo destinatário. Reciprocamente, as operações do tipo receive permitem a um processo receber dados de um outro processo. Um modelo de programação paralela é implementado por uma nova linguagem (ex. Orca[8]), por extensões acrescentadas à uma linguagem já existente (Concurrent C [14]), ou por uma biblioteca paralela utilizada juntamente com uma linguagem clássica (ex. PVM[3]). Qualquer que seja a solução empregada, mecanismos de suporte à programação paralela devem ser providos, de maneira a permitir a execução de programas paralelos, sendo que aspectos fundamentais destes ambientes são a gerência de processos, a comunicação e a sincronização.

2. Ambientes de Suporte à Programação Paralela

Os ambientes se suporte à execução paralela devem oferecer serviços de gerência de processos (criação e destruição de processos locais e remotos), de comunicação entre processos no mesmo processador ou em processadores diferentes, de sincronização, de gerência de memória e de entrada e saída. O núcleo de execução pode ser organizado de maneira clássica. Cada processador possui um núcleo monolítico ao qual se incorporam a gerência de comunicações e o acesso aos serviços clássicos distantes. O conjunto de núcleos monolíticos cooperantes formam o núcleo monolítico paralelo.

Uma outra solução consiste na abordagem dita microkernel. Um microkernel é essencialmente um núcleo de comunicação entre processos, de gerência de processos e de gerência de memória. Não oferece serviços clássicos de gerencia de arquivos, que são assegurados por processos servidores. Por exemplo, a abertura de um arquivo se traduz em uma mensagem ao servidor gerente de arquivos, solicitando o serviço. O servidor executa a solicitação e envia os resultados. As vantagens de um microkernel são a flexibilidade e a modularidade. Os recursos do sistema são acessados da mesma maneira, de acordo com um protocolo cliente/servidor. Para acrescentar um novo

serviço basta acrescentar um novo servidor. Um microkernel paralelo é composto por um conjunto de microkernel´s locais cooperantes, um em cada processador da máquina paralela. A função de comunicação de cada microkernel é estendida de maneira a permitir o acesso a serviços oferecidos por servidores distantes.

O ponto crucial de um microkernel paralelo é o microkernel de comunicação, que permite a comunicação entre processos sobre um mesmo processador ou em processadores diferentes, de acordo com um protocolo cliente/servidor. O núcleo de comunicação pode ser organizado por dois módulos:

a) Protocolo: neste nível são tratados os protocolos de comunicação entre processos, e a sincronização associada. Cada protocolo pressupõe um tratamento particular na expedição e na recepção de mensagens. Pode-se vislumbrar duas formas de organização desta camada. De acordo com a primeira, cada mensagem é tipada por um protocolo e cada tipo define um tratamento particular da mensagem correspondente ao protocolo. A outra solução é possuir um único protocolo privilegiado, suficientemente poderoso para suportar, sobre si, os outros protocolos. Por exemplo, um mecanismo RPC de base assegura as transferências de requisições de serviços e de respostas aos clientes e servidores. Os outros protocolos seriam implementados por servidores de comunicação.
b) Encaminhamento de mensagens: esta camada se encarrega de fazer as transferências de mensagens, isto é, expedir, receber e reexpedir. As seguintes situações podem ocorrer quando da emissão ou recepção de uma mensagem em um processador:
    - Emissão: quando a mensagem está em poder do núcleo de comunicação dois casos se apresentam: a) o processador destinatário é o originário da mensagem, e neste caso, trata-se de uma mensagem local, que será copiada localmente para o processo destinatário (servidor ou cliente). b) o processador destinatário é distante, e neste caso a mensagem deve ser encaminhada pela rede de comunicação.
    - Recepção: na recepção de uma mensagem, igualmente dois casos podem ocorrer:
      a) a mensagem é destinada ao processador receptor, e deve ser transmitida ao processo destinatário. b) a mensagem não é destinada ao processador receptor, e neste caso deve ser reenviada para um outro processador. Tratando-se de um cluster, o caso b) é resolvido pelos mecanismos de roteamento do protocolo de comunicação.

Para qualquer uma das abordagens, núcleo monolítico acrescido de um núcleo de comunicação, ou microkernel mais servidores especializados para oferecer os serviços, duas são as possibilidades de implementação. Desenvolver um novo sistema operacional e desenvolver um suporte de execução no topo de um sistema já existente. O projeto de um novo sistema operacional é uma tarefa de extrema complexidade. Necessitam ser definidos e implementados os subsistemas de entrada e saída, de gerência de processos, de comunicação e sincronização. Por outro lado, a utilização de

um sistema operacional já existente, por exemplo Linux, e desenvolver no topo deste sistema o ambiente de suporte à programação paralela tem-se mostrado uma alternativa muito atraente. Nas seções a seguir serão apresentados exemplos de utilização destas duas alternativas de implementação.

3. Sistemas Operacionais Paralelos

No início da década de 90, com o acentuado barateamento dos processadores, começaram a ser desenvolvidas máquinas paralelas. O processador Transputer [11] permitiu a construção de máquinas paralelas sem memória comum e para estas máquinas foi desenvolvido um novo sistema operacional, Helios[13], baseado na tecnologia microkernel. Por outro lado, a IBM em suas máquinas SPx decidiu-se pela utilização do sistema operacional AIX, ao qual foram acrescentadas funcionalidades de acesso a uma rede de comunicação baseada em um Switch. A programação nestas máquinas é feita com a ferramenta MPI[10], que permite a programação com o paradigma de troca de mensagens. Amoeba[1], também baseado na tecnologia microkernel, foi um novo sistema desenvolvido para suportar distribuição e paralelismo. A construção dos programas paralelos no Amoeba é feita com a linguagem Orca, uma nova linguagem de programação orientada à objetos que permite a construção de programas paralelos utilizando-se de objetos compartilhados. Outro sistema operacional importante é o Mach[1]. O Mach é um microkernel que oferece mecanismos básicos de comunicação e de gerência de memória que permitem a sua utilização em ambientes sem memória compartilhada.

Em relação a estes sistemas, o sistemas operacional Helios praticamente desapareceu juntamente com a utilização do processador Transputer para máquinas paralelas. Este processador não evoluiu e não foram produzidas novas máquinas. O Amoeba e o Mach servem de referência para a construção de sistemas operacionais baseados na tecnologia microkernel, porém a utilização destes sistemas é muito restrita. O que se constata é que o desenvolvimento de um sistema operacional dedicado à uma máquina paralela específica, além de ser uma tarefa complexa, não satisfaz um requisito importante que é a portabilidade.

4. Ambientes de Execução no Topo de Sistemas Operacionais Tradicionais

Diversos são os exemplos de ambientes de execução paralela construídos no topo de sistemas operacionais já existentes. Estes ambientes podem ser um sistema operacional, como por exemplo DCE[1], ou um núcleo que oferece suporte de execução de ferramentas de programação paralela, por exemplo, PVM. Este ambiente deve oferecer mecanismos para à comunicação, à sincronização e a gerência de processos paralelos. Os demais serviços deste ambiente são oferecidos pelo sistema operacional tradicional utilizado.

DCE é um microkernel desenvolvido para executar no topo de sistemas operacionais UNIX em uma rede de estações de trabalho. Possui um mecanismo RPC que permite a comunicação entre clientes e servidores locais e distantes, um servidor de arquivos distribuídos, um servidor de tempo global e suporte a programação

multithread. PVM é uma ferramenta de programação paralela baseada em troca de mensagens. PVM é um ambiente que permite ao usuário ver uma coleção de máquinas heterogêneas, interligadas por uma rede de comunicação, como uma única máquina paralela. Um núcleo PVM executa em cada máquina da "máquina virtual". Este núcleo atua como núcleo de comunicação e controlador. O núcleo PVM na máquina na qual foi disparada a execução do programa paralelo é o "master", e os demais "slaves". As principais funções do núcleo PVM são a gerência de tasks e a gerência de comunicação. Solaris MC[18] é uma extensão do sistema operacional Solaris para cluster de workstations e que parece ao usuário ser um único computador utilizando o sistema operacional Solaris. Este sistema consiste de um conjunto de extensões do kernel Solaris, de um sistema de arquivos distribuído e suporte a criação e destruição de processos locais e remotos.

MDX[6] é um ambiente de programação que suporta dois modelos de programação paralela: memória compartilhada e troca de mensagens. No modelo com memória compartilhada o programador escreve o seu programa como se fosse um programa multithread tradicional, e a sincronização é obtida com o uso de semáforos e barreiras. No modelo baseado em troca de mensagens, os processos se comunicam através de portas de comunicação e são usadas barreiras para sincronização. O suporte de execução deste sistema é baseado no modelo cliente/servidor e executa no topo do sistema operacional Linux em um cluster de estações de trabalho. Um núcleo de comunicação permite a troca de mensagens entre os núcleos locais e servidores especializados oferecem os serviços de gerência de processos, de comunicação e de sincronização. Um servidor de nomes é utilizado para fornecer o serviço de resolução de endereços dos servidores, de portas e de processos do sistema.

5. Conclusão

Uma tendência atual é a utilização de cluster´s, que permitem que se tenha processamento de alto desempenho a baixo custo, e a construção de ambientes de execução para programas paralelos no topo de um sistema operacional tradicional, com algumas modificações. Normalmente o sistema que serve como base é um Unix, embora a opção de uso do Windows NT[2] já esteja sendo considerada. Neste sentido, a utilização do Linux têm-se mostrado atrativa, pelo fato de ser um sistema livre, com código fonte disponível. Outra alternativa que está sendo explorada é o uso de Java[19] para suporte ao processamento de alto desempenho, que no entanto apresenta problemas relacionados a performance da JVM.

Um aspecto importante é a possibilidade da existência de ambientes que, além de suportar a execução de programas paralelos, permitem a execução de programas seqüenciais, que são distribuídos na rede de processadores, com o redirecionamento das operações de entrada e saída. Trata-se de uma solução que propicia uma utilização mais genérica dos cluster´s, e que aponta para uma direção interessante na produção de tais ambientes.

Ainda em relação aos cluster´s, alguns temas que merecem atenção são a carga de sistema e programas de aplicação em paralelo em sistemas compostos por um grande

número de nós, a coexistência de diferentes tecnologias de redes rápidas, um sistema de arquivos distribuído, entre outros.

Bibliografia

1. A. S. Tanenbaum. Distributed Operating System. Prentice-Hall, 1995.
2. Abrahan Silberschatz and Peter Baer Galvin. Operating System Concepts. John Wiley & Sons, Inc. 1999.
3. Al Geist et al. PVM: Parallel Virtual Machine – A Users´ Guide and Tutorial for Network Parallel Computing. The MIT Press.
4. A. S. Tanenbaum, M. Franz Kaashoek and Henri Bal, (1992). E. Parallel Programming Using Shared Objects and Broadcasting. In IEEE Computer, August 1992.
5. Celso Maciel da Costa. Environnement D´éxécution Parallèle: Conception et Architecture. *In.* Laboratoire de Génie Informatique, Université Joseph Fourier, Grenoble France. Thése de Doctorat, 1993.
6. Celso Maciel da Costa, Dotti, Fernando Luís, Copetti, Alessandro and Preuss, Evandro. MDX: A Parallel Programming Environment supporting Distributed Shared Memory and Message Passing. AST2000 - Argentine Symposium on Computing Technology - JAIIO2000-XXIX Jornadas Argentinas de Informática e Investigacions Operativas. Tandil, Argentina, September 4 - 9, 2000.
7. Flynn, M. J. Some Computer Organization and Their Effectiveness. IEEE Transactions on Computers, vol. C-21, pp.948-960, sept. 1972.
8. H. E. Bal., M.F. Kaashoek and A.S Tanembaum. Orca: A Language For Parallel Programming of Distributed Systems. In IEEE Transactions on Software Engineering, Vol. 18, No.3, March 1992 pp. 190-205.
9. Hwang, Kai. Zhiwei, Xu. Scalable parallel computing : technology, architecture, programming. Editora McGraw-Hill, 1998.
10. Ian Foster. Designing and building parallel programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley, 1995.
11. Inmos Ltd. Transputer Development System. Prentice-Hall, 1988.
12. Message Passing Interface Forum. MPI: A message passing interface standard, version 1.0. Mai. 1994.
13. N. H. Garnet. Helios: An Operating System For The Transputer. Proc. of OUG-7. IOS, Springfield 1987.
14. N. H. Gehani and W. D. Roome. The Implementing Concurrent C. In IEEE Software-Practice and Experience , Vol. 22(3), March 1992, pp. 265-285.
15. R. Pinheiro Bianchini. Nomad: um sistema operacional eficiente para clusters de uni e multiprocessadores. In anais do Simpósio Brasileiro de Arquitetura de Computadores, Setembro 1998.
16. Raymond Namyst. PM^2: un environnement pour unce conception protable et une exécutin efficace des applications parallèles irregurlières. Laboratoire d'Informatique Fondamentale de Lille, Lille, Dez. 1996.
17. S. Pakin , M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Message for Myrinet. In Proceedings of SuperComputing '95, IEEE Computer Society Press, 1996.
18. Solaris MC – http://www.sunlabs.com/research/solaris-mc
19. Yelick, Semenzato, Pike, Miyamoto, Liblit, Krishnamurthy, Hilfinger, Graham, Gay, Colella, Aiken. Titanium: A High-Performance Java Dialect. ACM 1998. Workshop on Java for High-Performance Network Computing, Stanford, California, February 1998.
20. Willy Zwaenepoel et al. *TreadMarks: Shared Memory Computing on Network of Workstations*. IEEE, Fev 1996. pp. 18-28.