Evolução e Tendências da Programação Paralela

Profa. Dra. Liria Matsumoto Sato

Escola Politécnica da Universidade de São Paulo
Depto. de Engenharia de Computação e Sistemas Digitais
Av. Prof. Luciano Gualberto, Trav. 3, núm. 158 CEP: 5508-900 São Paulo
E-mail: liria@pcs.usp.br

Apresentação

O uso de processamento paralelo na implementação de aplicações que envolvem computação intensiva e exigem uma resposta imediata, por exemplo, previsão meteorológica e sistemas de visualização volumétrica, assim como aplicações cuja computação intensiva exigem grande capacidade de processamento, tal como a simulação de fenômenos físicos (ex: evolução da galáxia, convecções de plasma na coroa solar [1]) é um fato, tanto em sistemas comerciais como nas pesquisas atualmente em desenvolvimento.
Duas classes de sistemas paralelos têm sido utilizadas: multiprocessadores e multicomputadores.
Os multiprocessadores são caracterizados por possuírem múltiplos processadores e memória compartilhada. São limitados quanto ao número de processadores pela contenção no acesso a esta memória.
Já os multicomputadores são escaláveis quanto ao total de processadores, mas têm o desempenho dependente da quantidade de comunicação efetuada na aplicação, como também dos recursos de comunicação, exigindo redes de interconexão de alta velocidade.
Uma combinação das duas classes estão , atualmente, sendo utilizadas, resultando em "clusters" de estações onde cada estação é um multiprocessador.
Contudo, tais sistemas requerem ambientes de programação e processamento, que permitam a exploração adequada dos seus recursos.
São, aqui, apresentadas a evolução e as tendências dos sistemas de programação e linguagens para computadores multiprocessadores e "clusters"de estações.

Computadores Multiprocessadores e "Clusters" de Estações

Na busca de soluções para atender a demanda de processamento solicitada pelas aplicações, computadores de alto desempenho têm sido desenvolvidos. Computadores vetoriais, multiprocessadores e multicomputadores têm sido propostos e disponibilizados comercialmente. Arquiteturas, especialmente das classes do multiprocessadores e multicomputadores, que visam oferecer melhores desempenhos, mantendo um custo acessível aos usuários são temas de pesquisa.

Os multiprocessadores são caracterizados por possuírem múltiplos processadores e memória compartilhada. São limitados quanto ao número de processadores pela contenção no acesso a esta memória. Contudo esta limitação está sendo reduzida com o uso da tecnologia de chaveamento na implementação de memória compartilhada por hardware. Atualmente já se tem disponível no mercado computadores multiprocessadores com 4 ou 8 processadores a baixo custo e outros mais potentes e com maior número de processadores e melhores acessos à memória compartilhada a custos mais altos. Já os multicomputadores são escaláveis quanto ao total de processadores, mas têm o desempenho dependente da quantidade de comunicação efetuada na aplicação, como também dos recursos de comunicação, exigindo redes de interconexão de alta velocidade.

O avanço das tecnologias de chaveamento de alta velocidade tem motivado a construção de "clusters" de estações como uma solução para a obtenção de sistemas de alto desempenho. O seu uso com o objetivo de obter alto desempenho na execução de uma aplicação única e monolítica, onde os nós de processamentos cooperam para atingir o objetivo desejado, tal como o tratamento de uma imagem volumétrica, é uma área atualmente em pesquisa, e é sob este enfoque que tais sistemas serão, aqui, abordados.

É possível construir "clusters" de estações com baixo custo, dependendo da sua configuração [1]. Os "clusters"de estações são constituídos de um conjunto de computadores, denominados nós, interconectados por uma rede de alta velocidade, e são classificados como multicomputadores [2].

Uma solução mista, que já vem sendo empregada, é a combinação das duas anteriores, resultando em "clusters" de estações, onde cada estação é um multiprocessador. Uma solução de mais baixo custo é um "cluster" heterogêneo de estações composto de multiprocessadores e de monoprocessadores.

Ferramentas e Linguagens de Programação

O desenvolvimento de aplicações para sistemas que oferecem processamento paralelo requer ambientes de programação e processamento para a implementação e execução.

Diversas linguagens na abordagem procedimental, incluindo extensões de linguagens como a C e a Pascal foram propostas. Assim como sistemas de programação, incluindo compiladores com paralelização automática, foram propostos, e atualmente já se tem versões comerciais.

Na Escola Politécnica da Universidade de São Paulo, foi projetada a linguagem CPAR e implementado um sistema de programação, com o objetivo de disponibilizar uma linguagem e uma ferramenta que proporcionassem facilidades de programação e um alto ganho de desempenho.

A linguagem CPAR e sua biblioteca [3] foram originalmente definidas para o desenvolvimento de programas paralelos que executam em máquinas com múltiplos processadores e memória compartilhada física (Silicon Graphics SGI4D/480). Versões para sistemas operacionais LINUX e SOLARIS estão disponíveis. Uma versão encontra-se instalada em um multiprocessador Quad-Pentium Pro (4 processadores) e em multipropcessador Quad-Xeon, no Laboratório de Arquitetura e Software Básico do

Departamento de Engenharia de Computação e Sistemas Digitais (PCS /EPUSP), e no computador multiprocessador (SUN) no CENAPAD de Minas Gerais (32 processadores).

A biblioteca foi posteriormente implementada para executar sobre um DSM (Quarks) [4] [5]. Baseada no paradigma de variável compartilhada, CPAR é uma extensão da linguagem C e está em desenvolvimento um sistema CPAR para "cluster"de "clusters"de estações [6].

A linguagem CPAR [3] oferece um conjunto de construções poderosas que facilitam a definição, organização, e programação de aplicações paralelas. Um programa em CPAR pode ser organizado em blocos de execução paralela lógica aninhados, isto é, macrotarefas e microtarefas. Macrotarefas são unidades lógicas independentes na função principal do programa. Uma macrotarefa pode ser subdividida em uma ou mais microtarefas, que são responsáveis pelo paralelismo de granularidade fina, representadas no programa do usuário por laços ou blocos paralelos. Variáveis compartilhadas na linguagem CPAR podem ser definidas como globais ou locais a macrotarefa. Variáveis compartilhadas globais são acessíveis a quaisquer macrotarefas, enquanto as variáveis compartilhadas locais, declaradas no escopo de uma macrotarefa, são acessíveis apenas às microtarefas desta macrotarefa. Em um sistema CPAR desenvolvido para uma plataforma DSM (CPAR sobre o sistema DSM Quarks), para gerar um código executável, um programa fonte em CPAR é primeiro compilado pelo pré-compilador CCPAR, que converte o programa fonte em um programa C, incluindo chamadas para a biblioteca CPAR.

Um paradigma de programação que se mostra apropriado à programação paralela é a programação orientada a objetos, especialmente, devido ao encapsulamento de dados inerente a este paradigma. Diversas propostas nesta abordagem foram feitas, tais como a linguagem COOL [7] e a MENTAT [8]. Um aspecto importante na programação paralela é a questão dos mecanismos de sincronização, que combinado com o oferecimento de herança pelo paradigma de orientação a objetos cria um problema denominado de Anomalia da Herança [9], que vem sendo tratado. Uma proposta baseada em padrões apresenta um padrão denominado "objeto paralelo", implementado em JAVA, que trata estes aspectos [10] e que tem como objetivo a exploração efetiva de paralelismo com um alto grau de reutilização de código e programabilidade e pode ser utilizado na implementação orientada a objetos de aplicações Outro aspecto importante, nos sistemas multicomputadores, é a distribuição de carga entre os diversos nós de processamento. Neste sentido, algumas propostas têm sido apresentadas, visando promover o balanceamento de carga [11].

A preocupação de se dispor de um padrão de recursos para programação, e mesmo para implementar ferramentas, que sejam compatíveis para os computadores multiprocessadores disponibilizados no mercado por diversos fabricantes, levaram as empresas a se unirem em um esforço conjunto, gerando a proposta de uma interface baseada em um conjunto de diretivas para o compilador, denominada OPENMP.

Programação em "cluster de estações

Visando oferecer um paradigma de programação que não torne a implementação das aplicações em clusters de estações demasiado complexa, tem se desenvolvido uma linha de pesquisa que oferece o paradigma de programação com memória compartilhada, tendo sido propostos os denominados sistemas com memória compartilhada distribuída. Programabilidade é uma das linhas promissoras de pesquisa na área de sistemas de memória compartilhada distribuída [12]. Outro aspecto importante é a necessidade de convergência de programação para os diversos sistemas propostos. Pesquisas nesta linha, tais como oferecer as diretivas de OpenMP para "cluster" de estações, encontram-se em andamento [13]. Buscando oferecer uma solução que promova a convergência de programação sobre sistemas multiprocessadores e "cluster" de estações e programabilidade no nível de linguagem, encontra-se em desenvolvimento no Laboratório de Arquitetura e Software Básico da Escola Politécnica, tendo atualmente um protótipo preliminar um sistema de programação que oferece a linguagem CPAR para "cluster"de estações, implementado sobre o sistema de memória compartilhada distribuída (DSM) QUARKS [4]. Deverá ser implementado sobre outros DSM's atuais.

Sistemas DSM: conceitos e evolução

Considerando que a escrita de programas utilizando passagem de mensagem é muito complicada e sensível a erros, é necessário oferecer suporte de software apropriado para o desenvolvimento e execução de aplicações paralelas em "clusters"de estações.

É conhecido que o paradigma de programação com variáveis compartilhadas, uma vez que mantém as características da programação seqüencial, é um paradigma bastante intuitivo, além de não adicionar dificuldades à implementação de aplicações. Sistemas DSM [14][4][15][16][17] escondem a visão distribuída de um sistema com memória distribuída, oferecendo ao programador um estilo semelhante ao de memória compartilhada para escrever aplicações.

Duas abordagens são utilizadas nos sistemas DSMs.

A primeira abordagem, e atualmente ainda em pesquisa por diversos núcleos de pesquisa, oferece uma abstração de um espaço de endereçamento compartilhado em uma rede de computadores que forma um sistema distribuído. Para tanto, um sistema DSM deve migrar ou replicar os dados, alocados em páginas, mantendo a coerência dos seus valores, durante a execução das aplicações. Tais sistemas são classificados como DSM baseado em bloco [18] . O principal fator determinante do desempenho de um sistema DSM é o número de mensagens que trafegam na rede. Nos últimos anos, a comunidade internacional tem se dedicado intensivamente em criar estratégias para reduzir o número de mensagens. Para tanto, alguns buscam esta redução através da exploração da localidade dos dados e da sua distribuição [19]. Outros buscam-na nas estratégias para implementação de mecanismos de sincronização, basicamente semáforos e barreiras [13][20] [15]. E outros se fixam nos modelos de consistência [13][15][21][22], procurando reduzir o número de mensagens, mantendo a consistência

apenas em pontos estratégicos, que garantem uma execução correta do programa dada a corretude do programa paralelo. Tais pontos são aqueles onde ocorrem sincronização, tais como barreiras e obtenção e liberação de semáforos, dado que o paralelismo exige sincronização para a correta execução de programas paralelos.
A maioria dos sistemas DSM nesta abordagem não oferece suporte de alto nível, como uma linguagem, para escrever tais aplicações.
Um sistema que implementa a linguagem CPAR sobre o sistema DSM QUARKS [4] foi desenvolvido no Laboratório de Arquitetura e Software Básico da Escola Politécnica da USP [23][24] para um "cluster" homogêneo de computadores.
A segunda abordagem é representada pelos DSMs baseados em estrutura. Esses sistemas mantêm em pequenas unidades, como por exemplo objetos, dados relacionados logicamente. São implementados pelas camadas de compilador , biblioteca e linguagem, exigindo a definição de modelo e linguagens de programação com características novas. Em algumas linguagens como a JADE [25], Emerald [26] e Clouds [27],a memória é estruturada como objetos distribuídos, ou como variáveis como no Munin [28]. O sistema Linda [29] estrutura a memória como se fosse um banco de dados, ordenando a memória compartilhada como uma memória associativa chamada espaço de tupla. Nesta abordagem não ocorre o falso compartilhamento. Em geral, nos DSMs da primeira abordagem, como a consistência é efetuada por página, ou seja a granularidade de dados é uma página, dados que não são manipulados no nó são atualizados, configurando um falso compartilhamento [28]. Quanto menor o tamanho da página menor a possibilidade de falso compartilhamento. Contudo neste caso um maior número de páginas poderão ser atualizadas, aumentando a demanda de comunicação. Entretanto, na segunda abordagem, dependendo da granularidade das estruturas compartilhadas, se estas forem enviadas uma de cada vez, implicará em grande sobrecarga de comunicação.

A exploração da memória compartilhada física para a computação intensiva, presente em nós multiprocessadores já vem sendo tratada. Entretanto, nos sistemas propostos que provêm esta exploração, a estratégia de coerência adotada se baseia em página, dificultando a característica de heterogeneidade entre as estações que compõem o "cluster". Por exemplo, se enquadra aqui, o sistema proposto por Samanta et al. [30], entre outros.
A idéia de implementar um espaço de endereçamento sobre uma rede de computadores já foi proposta a um certo tempo. Desde o início, os pesquisadores da área tiveram que lidar com sérias questões que afetavam o desempenho de tais sistemas: problemas com a granularidade de coerência que podem levar ao falso compartilhamento ou à fragmentação, custo de cada operação de comunicação, sincronização cara feita com troca de mensagens, falta de página e a sobrecarga gerada pelo processamento do protocolo.
Basicamente, três camadas afetam o desempenho de um DSM: a aplicação, protocolo/modelo e a arquitetura de comunicação. Para aumentar o desempenho, vários trabalhos foram propostos, que concentram-se principalmente nas duas últimas camadas e que visam[31]:

- redução do tráfego e freqüência da comunicação através de novos modelos de consistência relaxada e implementações de protocolos;
- redução dos custos de comunicação através de suporte adicional de hardware na arquitetura de comunicação.

Tem-se chegado a uma certa maturidade de protocolos, levando agora os esforços da comunidade científica para a camada de aplicação e para a sinergia entre as camadas. Novas áreas estão sendo enfocadas, tais como [31]: avaliação de desempenho direcionado à aplicação; reestruturação da aplicação para DSM e portabilidade de desempenho para diversas plataformas; melhorias no protocolos direcionadas aos gargalos de aplicações reais; interações de protocolo e sinergia com suporte da arquitetura; ferramentas de software para aumentar a programabilidade.

Mais especificamente sobre a pesquisa de DSM por software, importantes direções apontam para:

- comparação de DSM com recentes abordagens de software de memória compartilhada com granularidade fina que não dependam tanto de modelos de
- consistência relaxada;
- utilização de nós multiprocessados ao invés de nós monoprocessados;
- escalabilidade.

A crescente facilidade de construção de sistemas compostos por sistemas multiprocessadores, demanda por pesquisas que confirmem ou não a expectativa de aumento de desempenho do DSM sobre essa plataforma. Intuitivamente vê-se o ganho, pois em multiprocessadores a coerência e a sincronização são feitas via hardware. Apresentaremos aqui apenas algumas, a proposta apresentada por Samanta et al. [30] e o sistema Nomad apresentado em [32].

A questão da escalabilidade é um dos terrenos mais inexplorados da pesquisa, pois a escalabilidade não é somente no aspecto do desempenho, mas também em relação à memória utilizada pelos protocolos cujas necessidades variam de acordo com o número de nós do sistema.

A heterogeneidade entre os computadores que compõem o “cluster” é desejável visto que poderiam ser incluídos alguns computadores com recursos de processamento especiais que em uma aplicação poderiam ser utilizados para processar tarefas que seriam executadas com maior desempenho se disponibilizados tais recursos. Por exemplo poder-se-ia ter no “cluster” um computador com processadores distintos dos demais nós do “cluster”. Em sistemas DSM que oferecem uma abstração de um espaço de endereçamento compartilhado entre os nós, através da migração ou replicação de dados, alocados em páginas, mantendo a coerência dos seus valores, durante a execução das aplicações, a heterogeneidade é uma questão difícil de ser tratada . Como a consistência de dados se baseia em páginas de dados, a compatiblização dos dados é uma tarefa complexa.

A questão da heterogeneidade é mais fácil de ser tratada quando um sistema é estruturado em variáveis ou objetos na linguagem fonte, como na segunda abordagem. Neste caso, o compilador adiciona rotinas de conversão para todos os acessos à

memória compartilhada. O sistema Agora [29] trabalha sobre essa abordagem estruturando a memória em objetos compartilhados entre máquinas heterogêneas, não tratando, entretanto, as características de se ter disponível múltiplos processadores e memória compartilhada de nós multiprocessadores.
Uma proposta recente, o projeto InterWave que se encontra em andamento, apresentado em [33], trata a heterogeneidade, provendo o compartilhamento entre os processos, através de segmentos persistentes compartilhados independentes, os quais podem ser alocados dinamicamente e receber nomes simbólicos.
Em [30] Samanta et al. apresentam resultados interessantes quanto à exploração do desempenho em máquinas multiprocessadoras. São colocados protocolos SVM (Shared Virtual Memory) que suportam nós simétricos multiprocessados (SMP, symetric multiprocessor). Além disso são apresentados os compromissos-chave do projeto, sendo também discutidas as escolhas feitas. Como complementação é apresentada uma implementação sobre uma rede de 4 SMPs quad Intel Pentium Pro interconectados com Myrinet, bem como seus resultados de desempenho. As questões da heterogeinedade , porém não são tratadas nesse trabalho.
Nomad é uma ferramenta em desenvolvimento na UFRJ [32]. Trata-se de um sistema operacional distribuído para clusters de estações mono e multiprocessadas que oferece características necessárias a um sistema operacional moderno, tais como: escalabilidade, escalonamento de aplicações, entrada e saída distribuídas, detecção e recuperação de falhas, distribuição da demanda por recursos do cluster, proteção entre processos e compatibilidade retroativa.

Conclusões

Para computadores multiprocessadores dispõem-se atualmente de compiladores que provêem a paralelização automática de aplicações, que entretanto não permitem a exploração total do processamento paralelo nas aplicações. Uma interface padrão, a OPENMP, foi apresentada como resultado de um esforço conjunto dos fabricantes na busca de uma padronização. Contudo, esta interface oferece um conjunto de diretivas de compilador. Algumas extensões da linguagem C e FORTRAN foram propostas.
Para que se tenha portabilidade das aplicações, considerando computadores multiprocessadores e “cluster”de estações, a disponibilidade de uma linguagem para as duas classes de sistemas é necessária. Versões de OPENMP para “cluster”de estações foram propostas [13]. Compiladores e bibliotecas para a linguagem CPAR para “cluster” de estações compostos de microcomputadores monoprocessadores ou incluindo microcomputadores mono e multiprocessadores , baseados em sistemas DSM, estão em desenvolvimento.
Alguns aspectos da programação paralela de aplicações ainda precisam ser melhor tratados, especialmente aqueles relacionados com a otimização das implementações, através da redução de acessos à memória em computadores multiprocessadores e de número de mensagens em “cluster”de estações. Outro ponto que ainda deve ser explorado é o uso adequado dos recursos de processamento oferecidos por nós

multiprocessadores inclusos em “cluster” de estações. A questão da programabilidade é uma questão relevante que também deve ser tratada.

Referências

1. D. Ridge, D. Becker, P. Merkey, T. Sterling. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs.
2. G. Bell. Scalable Parallel Computers: alternatives, issues, and challenges. International Journal of Parallel Peogramming, v.22, n.1, pp. 3-47, 1994.
3. L. M. Sato. Ambientes de programação para sistemas paralelos e distribuídos. Tese de Livre-docência. Escola Politécnica da Universidade de São Paulo., 1995.
4. M. Swanson, L. Stoller, J. Carter. Making distributed shared memory simple, yet efficient. Technical report, Computer Systems Laboratory, University of Utah, 1998.
5. L. B. Arantes, L. M. Sato. CPAR-DSM: A Support for Parallel Programming on Top of DSM. Anais do International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, p. 859-866, julho/1998.
6. L. B. Arantes, L. M. Sato. A proposal for a parallel programming support for multi-lan platforms. XI Symposium on Computer Architecture and High Performance Computing , Natal, RN, p.197-204, setembro, 1999.
7. R. Chandra, A. Gupta, J.L. Hennessy. COOL: an object-based language for parallel programming. Computer, p. 13-26, 1994.
8. A.S. Grimshaw. Easy-to-use object-oriented parallel processing with Mentat. Computer, v.26, n.5, p.39-51, 1993.
9. L. N. Salvador, L. M. Sato. Anomalia da Herança em programação concorrente orientada a objetos. Anais do IX Simpósio Brasileiro de Arquitetura de Computadores, Campos do Jordão, páginas 431-446 , outubro/1997.
10. L. N. Salvador, L. M. Sato. Objeto Paralelo: um padrão de projeto para programação paralela. In: Congreso Argentino de Ciencias de la Computacion (IV CACiC), Ushuaia, Argentina, Outubro, 2000.
11. H. Senger, L. M. Sato, L. N. Salvador. Aplicando um sistema de execução de programas no suporte a linguagens paralelas orientadas a objetos. Anais do IX Simpósio Brasileiro de Arquitetura de Computadores, Campos do Jordão, páginas 493-508 , outubro/1997.
12. L. Iftode, J.P. Singh. Shared Virtual Memory: Progress and Challenges
13. H. Lu, Y. C. Hu, W. Zwaenepoel. OpenMP on Networks of Workstations. Proceedings of SC’98, Orlando, FL, November 7-13, 1998.
14. P. Keleher. Lazy release consistency for distributed shared memory. Phd thesis, Rice University, Houston, Texas, January ,1998.
15. B. N. Bershad, M. J. Zekauskas, W. A. Sawdon. The Midway Distributed Shared Memory System. In COMPCOM, 1993.
16. C. Amza, A. L. Cox, S. Dwarkads, P. Keleher, L. Rajamony, W. Yu, W. Zwaenepoel. Trademarks: shared memory computing on networks of workstations. IEEE Computer, 29 (2): 18-28, February, 1996.
17. D. Khandekar. Quarks: distributed shared memory as a building block for complex parallel and distributed systems. Master thesis, Dept of Computer Science, University of Utah, 1996.
18. W. Y. Liang. Adsmith: an obect-based distributed shared memory system on PVM. http://archil.ee.ntu.edu.tw/wyliang/qdsmith/.
19. M. Stum, S. Ahou. Algorithms Implementong Distributed Shared Memory. IEEE Computer, v.23,n.5, pp. 54-64, May, 1990.
20. D. Iguma. Aspectos de implementação de um ambiente para programação paralela com variáveis compartilhadas em sistemas distribuídos. Tese de Mestrado. Escola Politécnica da Universidade de São Paulo, 1997.