O SISTEMA ORIENTADO A OBJETOS MERLIN EM MÁQUINAS PARALELAS

Jecel Mattos de Assumpção Júnior
Laboratório de Sistemas Integráveis
Escola Politécnica da Universidade de São Paulo
tel / fax: (011) 543-4284
internet: jecel@lsi.usp.br

RESUMO

Os supercomputadores paralelos e as estações de trabalho pessoais podem resolver aspectos diferentes de uma aplicação: o primeiro como um processador numérico de alto desempenho e o outro com uma interface interativa e amigável com o usuário. Os dois ambientes de programação raramente são compatíveis, entretanto, o que complica esta "colaboração". O Sistema Merlin oferece um modelo computacional uniforme com a distribuição automática dos componentes da aplicação. As principais tecnologias que possibilitam isto são descritas neste trabalho: a linguagem orientada a objetos Self, o modelo de paralelismo de sincronismo por necessidade, a reflexão explícita, a proteção dos objetos e a compilação dinâmica adaptativa. A interação destas técnicas também é abordada.

ABSTRACT

Parallel supercomputers and personal workstations can solve different aspects of an application: one as a high speed number processor and the other with a friendly, interactive graphical user interface. The two programming environments are seldom compatible, however, which complicates this "collaboration". The Merlin System offers a uniform computational model with automatic distribution of the application components. The main technologies that make this possible are described in this paper: the Self object oriented language, the syncronism by necessity parallel model, explicit reflection, object protection and adaptive dynamic compilation. The interaction of these system features is also discussed.

1. Introdução

Computadores cada vez mais poderosos têm permitido o desenvolvimento de programas "amigáveis" onde a maior parte dos recursos de processamento é dedicada à interface com o usuário. A complexidade destes produtos alterou o desenvolvimento tradicional, tornando cada vez mais popular a técnica de programação orientada a objetos (que dá maior prioridade à facilidade de programação do que ao aproveitamento da máquina).

No caso dos supercomputadores o investimento em capacidade de processamento é substancial e este tipo de solução seria considerado um luxo inaceitável. O ambiente usual de usuário e de programação dessa classe de sistemas é semelhante aos usados no início da década de 60, quando cada ciclo de máquina tinha que ser aproveitado ao máximo.

Uma solução é combinar os dois tipos de sistemas. O usuário entra com os dados em um programa amigável em seu computador pessoal ou estação de trabalho e este programa formata esses dados e os envia para o supercomputador onde um exótico programa Fortran "mastiga os números" em alta velocidade. Os resultados são enviados ao programa original onde o usuário pode examiná-los da maneira que for mais conveniente. O supercomputador é aproveitado ao máximo enquanto o usuário trabalha de uma maneira razoavelmente decente. A grande vantagem deste mecanismo é aproveitar velhos conhecidos como o Spice, mas escrever dois programas (com estilos muito diferentes) para resolver cada novo problema não é uma proposta muito atraente.

O sistema operacional Merlin visa oferecer um ambiente único onde o software aplicativo é automaticamente distribuído entre os computadores de uma determinada instalação de acordo com os recursos computacionais necessários e o grau de interação com o usuário. Esta transparência de divisão de processamento implica em uma perda de desempenho, mas isto é considerado aceitável em face da facilidade de programação e uso.

Este trabalho apresenta aspectos do Sistema Merlin relevantes aos computadores paralelos de alto desempenho. A segunda parte apresenta o modelo de hardware implícito no projeto. A parte seguinte resume algumas características da linguagem Self, a base do Merlin. No próximo trecho é descrito o modelo de paralelismo adotado e seus efeitos sobre a linguagem Self. Os sistemas reflexivos são apresentados na parte 5 e o modelo de proteção é visto na parte 6. O tema da parte 7 é a compilação adaptativa e seu impacto na automatização do paralelismo. O último assunto abordado é como as técnicas apresentadas se combinam para tornar o sistema bastante eficaz. A parte 9 traz as conclusões do trabalho.

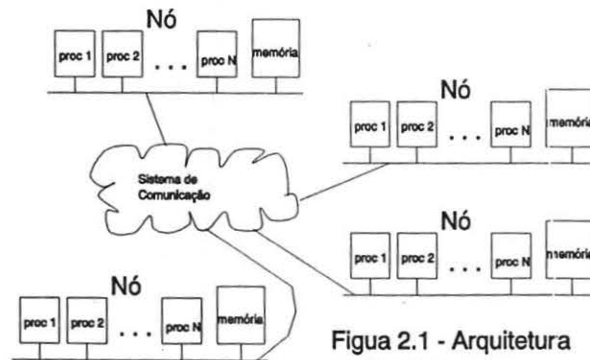
2. Arquitetura do Hardware

O sistema descrito neste trabalho representa a combinação de dois projetos muito semelhantes: o Merlin distribuído para redes de estações de trabalho de baixo custo (eLSI [AK90] - rebatizada de Merlin IV) e o Merlin paralelo para o computador MS8702 [As92] com 64 processadores.

O modelo de hardware do Merlin é um sistema constituído por uma coleção de nós computacionais que podem se comunicar entre si. Cada nó pode trocar informações com todos os outros nós, mas as taxas de comunicação não precisam ser uniformes e podem até variar muitas ordens de grandeza..

Cada nó tem uma memória local e um (ou mais) processadores. Todos os processadores de um nó precisam ser iguais, mas nós diferentes podem usar processadores diferentes. Um nó também pode ter recursos de entrada e saída.

Por esta definição, uma instalação com várias estações de trabalho e algumas máquinas paralelas ligadas em rede é apenas um sistema e tem seus recursos gerenciados de uma maneira global. As aplicações são distribuídas automaticamente pelos recursos computacionais disponíveis.



Figua 2.1 - Arquitetura do Hardware

3. Linguagem Self

A linguagem de programação Self [US87] é muito semelhante ao Smalltalk [GR83], mas usa um número menor de conceitos. Ela é puramente orientada a objetos - todos os dados e programas do sistema são agrupados em "caixas pretas" chamadas objetos que trocam mensagens entre si.

No Self cada objeto é completo em si mesmo, mas pode herdar de múltiplos outros objetos, o que permite organizá-los de modo a maximizar a flexibilidade de programação.

Como nenhum objeto pode acessar diretamente outro, eles podem ser colocados em máquinas diferentes pelo sistema ou até "migrar" durante a execução. Isto é muito difícil em linguagens como Pascal ou Fortran e totalmente impossível em C em função do uso livre de ponteiros.

4. Modelo de Paralelismo

A linguagem Self foi desenvolvida para aplicações seqüenciais e não possui um modelo de paralelismo que atenda a este projeto. Existem várias linguagens orientadas a objetos concorrentes que poderiam ter sido adotadas, mas os programas escritos nestes sistemas parecem exóticos e de difícil compreensão. A maturidade do Self vem de sua semelhança com o Smalltalk, representando duas décadas de experiência [Go72].

O modelo de paralelismo adotado, o sincronismo por necessidade [Ca90], visa alterar o mínimo possível a semântica do Self. Neste esquema, quando um objeto envia uma mensagem para outro recebe uma reposta provisória na forma de um "objeto futuro". Este objeto futuro pode ser manipulado como qualquer objeto e passado como parâmetro de mensagens para outros objetos. Qualquer envio de mensagem para um objeto futuro, entretanto, "congela" quem tentou enviá-la. Quando o objeto que recebeu a mensagem que provocou a criação do futuro

termina a execução este futuro é substituído pelo resultado e todos os que estiverem congelados em função deste futuro poderão prosseguir. Assim, quando um objeto envia uma mensagem para outro pode continuar em paralelo com este até que necessite da resposta para continuar. Neste instante ele para até que ela fique pronta.

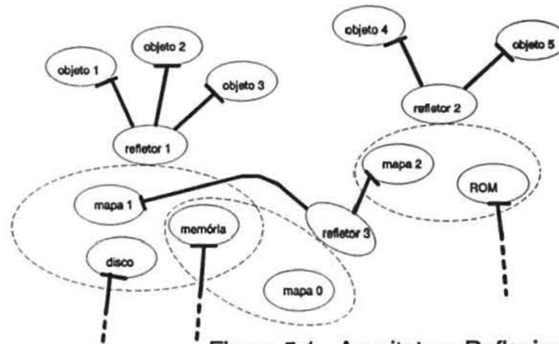


Figura 5.1 - Arquitetura Reflexiva

Este mecanismo tem a flexibilidade da mensagens assíncronas e a facilidade de programação das síncronas. Em princípio, o único efeito sobre a semântica dos programas em Self é que a ordem de avaliação de subexpressões passa a ser não determinísticas, mas são raros (talvez inexistentes) os programas que dependem da ordem de avaliação.

A versão anterior do sistema Merlin incluía dois outros tipos de objetos [As92]: os objetos passivos e os objetos distribuídos. Estes últimos não precisam fazer parte do modelo de paralelismo pois podem ser implementados a nível de aplicação. A parte 8 deste trabalho descreve o efeito da eliminação dos objetos passivos (que foram definidos por uma questão de eficiência de implementação, e não para compensar alguma limitação teórica).

Cada objeto atende a uma mensagem de cada vez, o que elimina a necessidade de mecanismos explícitos de sincronização, como semáforos (que obrigariam muitas alterações nos programas em relação a versões seqüenciais). Mensagens que chegam enquanto um objeto está ocupado ficam em uma fila de espera. O problema deste modelo é que não admite que um objeto envie mensagens para si mesmo, nem mesmo indiretamente. A solução adotada é a de permitir que um objeto bloqueado (congelado) esperando a resolução de um objeto futuro possa atender mensagens que estejam ligadas a este mesmo futuro. Este "furo" no modelo de paralelismo permite criar programas com erros, mas estes são os mesmos erros que apareceriam em programas recursivos seqüenciais.

5. Sistemas Reflexivos

Cada objeto representa um aspecto de um sistema e se identifica inteiramente com o conceito real. Um objeto retângulo, por exemplo, é realmente um retângulo para o resto do sistema e só pode ser manipulado como tal. Certas partes do sistema (o ambiente de programação e o depurador, entre outros) precisam ver e manipular o retângulo como uma lista de valores na memória, o que chamaremos de computação reflexiva (pois o sistema passa a ver si mesmo) ou metacomputação. Todo ambiente de programação ou sistema operacional oferece recursos reflexivos (como a função sizeof em C) mas de maneira arbitrária e pouco aproveitável.

Os sistemas orientados a objetos normalmente usam as classes para operações reflexivas (um retângulo não pode se manipular como uma seqüência de valores, mas a classe Retângulo pode). A linguagem Self eliminou as classes e teve que criar os "mirrors" (espelhos) que podem refletir outro objeto. Um retângulo só pode ser visto como um retângulo, mas podemos criar um espelho em cima deste objeto e enviar mensagens para o espelho pedindo para acrescentar ou eliminar funções em seu refletido.

Como a reflexão em Self se limita à estrutura dos objetos, o Sistema Merlin é baseado na organização do sistema operacional orientado a objetos Apertos [Yo92] como uma extensão dos espelhos do Self. Agora fica possível manipular aspectos dos objetos como tempo de processamento, posição na memória ou em disco, comunicação com outros objetos, distribuição de objetos no sistema e até a compilação do código de um objeto (veja a parte 7). Note a divisão explícita de níveis de programação: o código normal de um objeto envia mensagens para outros sem se preocupar com a localização real destes objetos; o algorítmo de balanceamento de carga, entretanto, é implementado em um "nível mais baixo" onde a localização dos objetos não é transparente.

A figura 5.1 mostra um conjunto de objetos e os meta-objetos necessários à sua operação. Os refletores organizam os meta-objetos em grupos chamados indicados pelas elipses pontilhadas. Os meta-objetos são também objetos e têm os seus refletores. O meta-objeto "mapa" equivale ao "mirror" do Self de Stanford.

Uma vez que os recursos computacionais que dão suporte para um objeto são representados explicitamente, fica possível definir uma única operação para trocar estes recursos: a migração de objetos. A migração pode ser de um nó do sistema para outro, como é normal em sistemas distribuídos, ou pode ser entre "meta espaços" de um mesmo nó (um objeto que existe inicialmente em ROM pode migrar para a memória volátil se precisar ser alterado; um objeto que existe na memória virtual pode migrar para o espaço de objetos permanentemente residentes para executar uma função de tempo real como a geração de som).

Ao representar explicitamente a reflexão no sistema ao invés de escondê-la em um misterioso núcleo, o Sistema Merlin permite a experimentação de diversos algoritmos de distribuição de objetos e a evolução para acompanhar novas técnicas.

6. Proteção

O Sistema Merlin procura ser o mais aberto possível no sentido que todos as facetas da implementação podem ser examinadas e alteradas pelos programadores (eliminando, até certo ponto, a distinção entre programadores de sistema e de aplicações). A liberdade tem o seu preço: um programador ou usuário pode fazer alterações desastrosas - uns novatos aprendendo Smalltalk estavam examinando e alterando objetos do sistema até modificar o objeto que representava a cor preta; a tela começou a ficar cada vez mais branca e eles não puderam desfazer o estrago. Isto é uma irritação em um sistema experimental dedicado a um usuário, como era o caso, mas é impensável em um ambiente de produção partilhado por toda uma empresa.

A solução do Sistema Merlin é fazer cada objeto pertencer a um usuário, que é o único que pode alterá-lo. Outros usuários poderão "ler" o objeto (se o dono colocá-lo em um lugar visível aos outros) e até criar cópias para si, mas não

podem afetar o objeto original. Além de usuário correspondentes a pessoas, existem usuários criados apenas para serem donos dos objetos de uma aplicação ou de partes (módulos) do sistema. No lugar de uma "sopa" de objetos que tanto confunde programadores iniciantes, o sistema divide claramente os objetos relacionados e evita que partes importantes do sistema sejam acidentalmente (ou maliciosamente) alterados atrapalhando a vida de todos os usuários.

Esta estrutura não é "fechada" pois o usuário pode copiar e alterar (só para si) partes críticas do sistema. Se for necessário fazer uma alteração que afete todos, basta mostrar ao sistema que se possui permissão para tal.

Para permitir o partilhamento controlado de objetos são criados grupos, que são usuários com uma lista de membros. Qualquer membro de um grupo tem as mesmas permissões em relação aos objetos do grupo que o próprio grupo. Um determinado usuário pode ser membro de quantos grupos precisar. Existe um grupo "todos" no qual são criados os objetos públicos, e todos os usuários servem como grupos do "superusuário" que pode manipular todos os objetos do sistema.

7. Compilação Adaptativa

A maioria das implementações do Smalltalk traduzem os programas fontes para uma linguagem de máquina fictícia chamada de "bytecodes" Cada computador real tem um interpretador que simula a operação da "máquina virtual" Este método tem a vantagem da portabilidade entre máquinas diferentes e a redução do espaço na memória e em disco (pois os bytecodes são muito compactos). O desempenho nunca foi um ponto forte destes sistemas, o que levou à compilação dinâmica nas implementações mais modernas. No lugar de interpretar uma rotina em bytecodes, a primeira vez que é chamada ela é traduzida para linguagem de máquina nativa e guardada em um "cache" em uma parte da memória. Toda vez que a mesma rotina voltar a ser chamada, é usada a versão do cache com um ganho significativo de desempenho (em troca de usar mais memória, como sempre).

Graças à herança, os mesmos bytecodes podem ser executados por objetos diferentes, o que reduz bastante as otimizações que podem ser feitas. A solução do Self é a compilação personalizada [Ch92] que gera uma versão diferente de linguagem de máquina nativa para cada tipo diferente de objeto. Para cada versão customizada, boa parte das características dinâmicas se tornam fixas e podem ser eliminados boa parte dos envios de mensagens. Quando alguma parte do sistema que o compilador tratou como constante é alterada, os códigos afetados são despejados do cache e voltam a ser compilados com a nova "constante" na próxima vez em que for chamado.

Com a compilação ocorrendo a tempo de execução aparece um dilema: compiladores símples podem tomar pouco tempo mas gerar código lento enquanto compiladores sofisticados podem provocar longas pausas na execução para gerar código de alto desempenho. A compilação adaptiva é uma opção: usa-se um compilador rápido mas burro inicialmente; quando uma rotina passa a ser muito usada ela é recompilado com outro compilador mais sofisticado. Ao se recompilar um programa que já vinha sendo executado, é possível aproveitar informações sobre a dinâmica do sistema para gerar código otimizado para o caso real [HCU91].

8. Combinação das Técnicas

Muitas outras técnicas fazem parte do Sistema Merlin (como o modelo gráfico, a interface com usuário, a memória vritual), mas as descritas aqui são as mais relevantes para os sistemas paralelos de alto desempenho. O objetivo desta parte do trabalho é mostrar o resultado da combinação das idéias descritas.

8.1. Proteção + Modelo de Paralelismo

Os objetos ativos definidos no modelo de paralelismo devem estar presentes em apenas um nó do sistema. Qualquer objeto residente em outro nó que precise dele deve enviar uma mensagem pelo sistema de comunicação entre nós. Como objetos básicos (a forma da letra A, por exemplo) se tornariam um gargalo ao gerar uma intensa troca de mensagens pela rede, versões anteriores do Merlin definiam objetos passivos que podiam existir simultaneamente em vários nós. A coerência entre as cópias era mantida por um esquema de invalidação na escrita [As92].

Em um sistema com o esquema de proteção adotado, uma parte dos objetos pertencem ao próprio usuário que está executando a aplicação (normalmente objetos do mais alto nível) e boa parte são objetos de sistema, que são apenas para leitura neste contexto. Como dificilmente o "dono" destes objetos básicos vai estar presente no sistema, podemos permitir múltiplas cópias dos objetos ativos sem o perigo de incoerências. Se o dono entrar no sistema (o superusuário invocar o editor de formas de letras, neste caso) todas as cópias são eliminadas e os outros objetos passam a ter que enviar mensagens pela rede com uma enorme perda de desempenho.

Este sistema é análogo à invalidação na escrita mas opera em uma escala totalmente diferente (sessões inteiras no lugar de objetos individuais) o que resulta em um desempenho típico muito melhor. Note que o desempenho no pior caso é muito inferior ao esquema anterior mas este caso é tão raro que seu impacto é desprezível. As estruturação do sistema em grandes domínios de proteção permitiu a adoção de um modelo uniforme de paralelismo com apenas objetos ativos.

8.2. Compilação Adaptativa + Modelo de Paralelismo

Como todos os objetos são ativos (servidores), todo envio de mensagem pode iniciar paralelismo e implica na criação de um objeto futuro. Na compilação adaptativa, o compilador que vai otimizar uma rotina recebe dados colhidos em tempo de execução sobre o uso da rotina. Assim, boa parte dos envios de mensagens podem ser convertidos em símples chamadas de subrotina (possível pois a semântica do sincronismo por necessidade é exatamente igual ao das mensagens síncronas). Só as mensagens enviadas a objetos remotos ou que provoquem o início de uma atividade demorada cujo resultado não é imediatamente usado incorrem no custo da criação de objetos futuros, todas as verificações, o empacotamento e enfileiramento das mensagens.

8.3. Sistemas Reflexivos + Compilação Adpatativa

A compilação adaptativa está intimamente ligada ao estado de execução do sistema. Este estado é normalmente inacessível a programas normais, o que impede os vários compiladores de

serem escritos dentro do próprio ambiente. A reflexão explícita cria uma estrutura organizada para interagir com o estado de execução, de forma que os compiladores do Merlin são programas normais escritos em Self. Um interpretador bem símples é usados para dar início à operação do sistema: o compilador é interpretado enquanto compila a si mesmo e, em seguida, passa a compilar outras rotinas. A administração do cache a nível da linguagem de alto nível (por reflexão) garante que código crítico, como o do compilador, não é descartado na falta de espaço, o que provocaria uma terrível degradação no desempenho.

8.4. Sistemas Reflexivos + Modelo de Paralelísmo

A administração automática da distribuição dos objetos e do paralelismo simplifica enormemente o desenvolvimento de aplicações. Isto pode se tornar uma limitação, entretanto, quando o programador tem informações sobre o funcionamento global de sua aplicação que podem ter um enorme impacto no desempenho, mas não tem como aproveitar isto no programa.

O sistema de reflexão permite ao programador a substituição seletiva dos sistemas automáticos (bastando ir até o nível de metacomputação onde a operação ocorre e trocá-la).

9. Conclusão

Um centro de processamento de dados pode ter dezenas de estações de trabalho de baixo custo e algumas máquinas paralelas de alto desempenho. Uma aplicação normalmente é dividida em partes diferentes para aproveitar da melhor maneira possível as características de cada máquina. O uso de diferentes ambientes de programação (algumas vezes de linguagens diferentes) e a ligação explicita dos pedaços por protocolos de rede tornam este tipo de desenvolvimento muito demorado.

O Sistema Merlin visa automatizar esta distribuição de tarefas oferecendo um ambiente uniforme de objetos que se comunicam. Este trabalho apresentou algumas técnicas que reduzem a perda de desempenho em relação aos sistemas mais tradicionais. A idéia é que não seja necessário abrir mão de décadas de avanços na área de software para se programar a nova geração de supercomputadores paralelos.

10. Agradecimento

Agradeço ao Prof. Sérgio Takeo Kofuji e ao pessoal do LSI pelo apoio e pelas máquinas paralelas. Aos meus pais, entre tantas outras coisas, pelo financiamento do projeto. Ao grupo Self da Sun por ter criado a linguagem Self e pelas muitas idéias que se tornaram parte do Merlin. Ao Dr. Yasuhiko Yokote por ter enviado cópias dos trabalhos do Sony Computer Science Laboratory e pelas idéias dos sistemas operacionais reflexivos.

11. Bibliografia

[AK90] Assumpção Jr., Jecel M. & Kofuji, Sergio T.: "eLSI - Estação Pessoal Gráfica", Anais do SIBGRAPI'90, Gramado, RS, 1990, p. 319-322

[As92] Assumpção Jr., Jecel M.: "Supercomputador Orientado a Objetos", Anais do IV SBAC-PAD, São Paulo, SP, 1992, p. 335-345

[Ca90] Caromel, Dennis: "Concurrency And Reusability: From Sequencial To Parallel", Journal of Object Oriented Programming, Vol. 3, No. 3, Sep/Oct 1990, p. 34-42

[Ch92] Chambers, Craig: "The Design and Implementation of the Self Compiler, an Optimizing Compiler for Object-Oriented Programming Languages", dissertação de Ph. D., Computer Science Department, Stanford University, March 1992

[GK76] Goldberg, Adele & Kay, Alan (Editores): "Smalltalk-72 Instruction Manual", Xerox PARC Technical Report SSL-76-6, 1976

[GR83] Goldberg, Adele & Robson, David: "Smalltalk-80: The Language and Its Implementation", Addison-Wesley, Reading, MA, 1983

[HCU91] Hölzle, Urs, Chamber, Craig & Ungar, David: "Optimizing Dynamically-Typed Object-Oriented Programming Languages with Polymorphic Inline Caches", ECOOP'91 Conference Proceedings, Geneva, Suiça, July 1991

[US87] Ungar, David & Smith, Randall B.: "Self: The Power of Simplicity", OOPSLA'87 Conference Proceedings, Orlando, FL, 1987

[Yo92] Yokote, Yasuhiko: "The Apertos Reflective Operating System: The Concept and Its Implementation", OOPSLA'92 Conference Proceedings, 1992