SUPERCOMPUTADOR ORIENTADO A OBJETOS

Jecel Mattos de Assumpção Júnior
Laboratório de Sistemas Integráveis
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, no 158
05508 - São Paulo - SP
tel - (011) 815-9322 ramal 3589
e-mail: jecel@lsi.usp.br

RESUMO

A programação orientada a objetos é considerada um grande avanço no desenvolvimento de programas, mas está normalmente associada a baixos desempenhos. Uma das maneiras de contornar este problema é o aproveitamento de paralelismo. Este trabalho mostra a implementação de uma linguagem puramente orientada a objetos numa máquina com 64 processadores.

ABSTRACT

Object Oriented Programming is considered a great step forward in software development, but is usually associated with low performance. One way to solve this problem is to exploit parallelism. This paper describes the implementation of a pure object oriented language on a 64 processor machine.

1. INTRODUÇÃO

Entre os vários computadores paralelos desenvolvidos no Laboratório de Sistemas Integráveis da EPUSP está o MS8702. Seu objetivo é o estudo de arquiteturas paralelas escaláveis, de forma que é mais importante ter um número razoável de processadores do que um alto desempenho total. Mesmo assim, sua capacidade de 300 MIPS é suficiente para a maioria das aplicações práticas. A parte 2 deste trabalho apresenta maiores detalhes sobre a máquina em si.

Como programar máquinas paralelas? Diversas alternativas estão sendo estudadas no LSI: linguagens convencionais com paralelização explícita e bibliotecas de comunicação (Express e PICL) ou sistemas distribuídos (Helios), compiladores paralelizantes (CPar, Parafrase, Power C), linguagens concorrentes (Occam) e linguagens funcionais (SISAL). Este trabalho explora a semelhança entre o modelo de programação orientada a objetos e dos sistemas paralelos de memória distribuída a nível de mecanismos básicos: passagem de mensagens. A linguagem usada, Self [US87], é descrita na parte 3. A parte 4 detalha a implementação do Self no MS8702 e a parte 5 apresenta as conclusões obtidas na etapa atual do projeto.

2. ARQUITETURA

O MS8702 pode ser considerada uma implementação reduzida do projeto descrito em [AK91]. É uma máquina de memória distribuída com os processadores dispostos em uma matriz bidimensional ("mesh"). O processador usado é o CISC de 32 bits 68020 e seu coprocessador de ponto flutuante 68881 operando a 16 MHz. Cada nó de processamento possui 1 MByte de memória local. A comunicação com os quatro vizinhos é feita por um processador de 16 Bits T222 com programação fixa, no lugar do CI dedicado (que foi projetado e está em fase de fabricação do primeiro protótipo) para reduzir o tempo e custo do projeto.

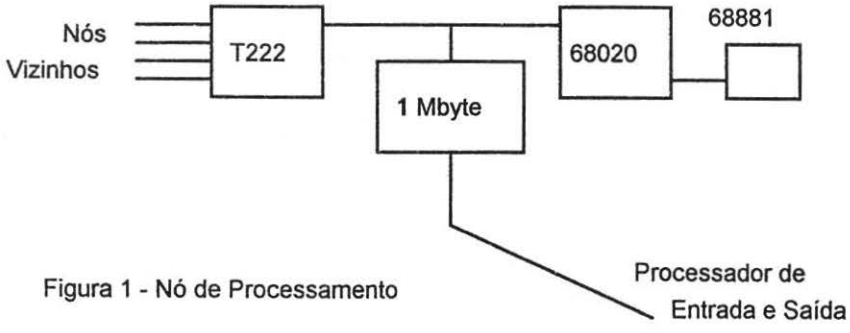
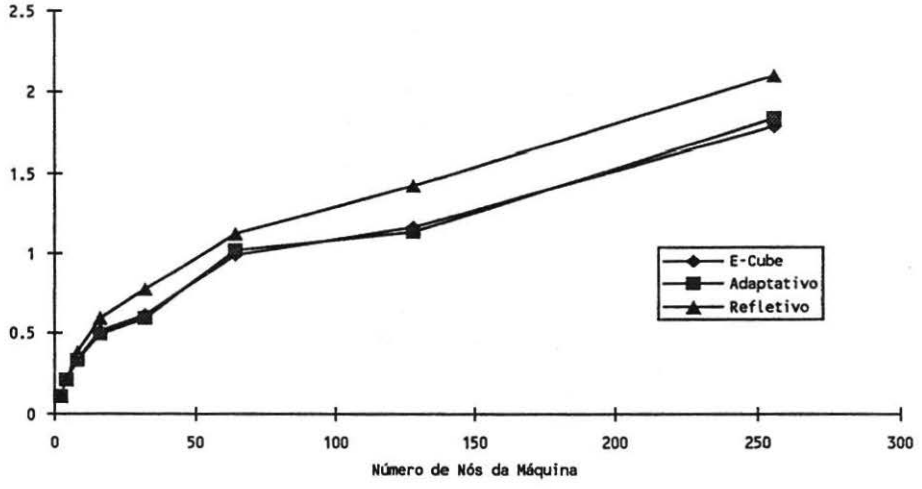


Figura 1 - Nó de Processamento

Nas máquinas paralelas com arquitetura em mesh, é normal ligar os dispositivos de entrada e saída e o computador hospedeiro nas conexões que "sobram" nas bordas da matriz. Isto faz com

que o fluxo de mensagens entre os nós de processamento e os acessos aos periféricos se misturem, o que, além de degradar o desempenho, é indesejável numa plataforma de estudos onde se pretende extrair parâmetros para projetos futuros. Assim, os nós de processamento do MS8702 têm um canal separado para entrada e saída que liga os 16 nós de um bastidor ao controlador de discos, que é uma máquina tipo PC/AT.

Gráfico 1 - Mensagens por Unidade de Tempo para cada Algoritmo de Roteamento



O desempenho da rede de comunicação é um fator fundamental no desempenho da máquina. Foram feitas extensas simulações comparando as arquiteturas em hipercubo, mesh 2D e 3D, e matriz de barramentos 2D e 3D para uma série de programas. No caso do mesh 2D, a arquitetura escolhida, foram testados três algoritmos de roteamento diferentes: E-Cube (mensagens andam primeiro na horizontal e depois na vertical), Adaptativo (a rota pode variar para desviar de congestionamentos) e Refletivo (uma variação do E-Cube onde a ordem das dimensões é trocada se a mensagem vai para um nó numericamente inferior). O gráfico 1 mostra o resultado para o programa de comunicação concentrada, onde os processadores tentam mandar mensagens para o mesmo lugar. As curvas se desviam bastante da reta ideal, mas este programa representa um caso extremo. O algoritmo Refletivo mostrou uma ligeira vantagem, mas foi realmente escolhido por funcionar mesmo com processadores faltando numa borda (o que permite que a máquina continue operando enquanto alguns nós estão em manutenção).

O protótipo com 64 processadores é composto por quatro bastidores, cada um com seu processador de periféricos. As placas, ao serem encaixadas no bastidor, ficam automaticamente ligadas às vizinhas numa matriz 4 por 4. As ligações das bordas são transformadas em sinais diferenciais (para evitar ruídos) e podem ser levadas por um cabo de 16 vias para outro bastidor. Não existe, portanto, limite para o tamanho da máquina. É importante notar que a capacidade de

entrada e saída cresce proporcionalmente ao número de nós.

Os processadores de E/S também são ligados entre si, formando uma matriz com um quarto da resolução da matriz de processamento. Isto permite aos 68020 usarem, de maneira transparente, recursos de outros bastidores.

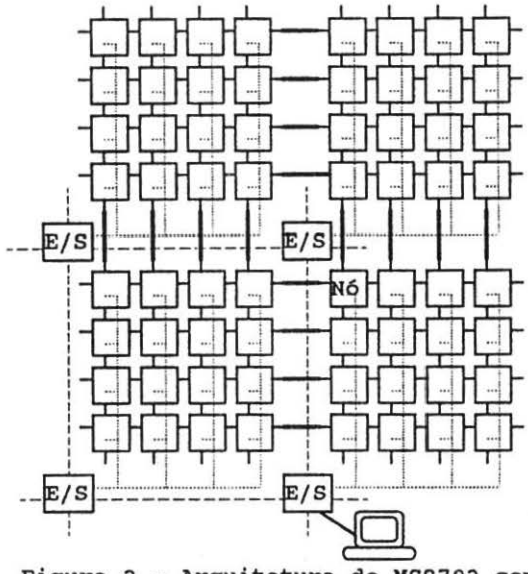


Figura 2 - Arquitetura do MS8702 com 64 Nós

3. LINGUAGEM SELF

A linguagem Self, muito semelhante ao Smalltalk-80 [GR83], é puramente orientada a objetos: toda a informação do sistema é encapsulada em "objetos" que juntam dados e programas. A única maneira de se alterar o estado do sistema é enviando uma mensagem a algum objeto para que este tome as providências necessárias. No Smalltalk, os objetos são "caixas pretas" para o resto do sistema mas têm um modelo interno mais ou menos convencional com variáveis locais e globais. No Self as coisas são mais uniformes e o objeto deve enviar mensagens para si mesmo (daí o nome da linguagem) para alterar seu estado.

Não existe o conceito de classes em Self. Cada objeto contém sua própria descrição (o que evita meta-objetos, meta-meta-objetos, etc.). Existe um esquema bastante sofisticado de herança múltipla que permite que um objeto repasse mensagens que ele mesmo não implementa. Novos objetos são criados como cópias de objetos já existentes (que servem de protótipos de tipos desejáveis de objetos) e podem ser alterados posteriormente.

O uso de mensagens para tudo parece ser muito ineficiente, mas técnicas sofisticadas (e inéditas) de compilação [Ch89] podem eliminar os custos envolvidos e produzir programas com desempenho próximo do C otimizado.

4. IMPLEMENTAÇÃO

A analogia entre processadores que não partilham memória, mas podem se comunicar, e objetos que são "caixas pretas" e trocam mensagens é evidente. Parece muito tentador dividir os objetos entre os processadores disponíveis e encarar as mensagens como uma implementação elegante das chamadas de procedimentos remotas (RPCs). Isto realmente foi feito, mas apenas para objetos selecionados, chamados "servidores". Quando é enviada uma mensagem para este tipo de objeto ocorre uma transferência de dados entre tarefas (que podem estar no mesmo processador ou não). Se a mensagem for enviada a um objeto normal que não esteja presente no nó, é trazida uma cópia para a memória local e a execução do programa continua.

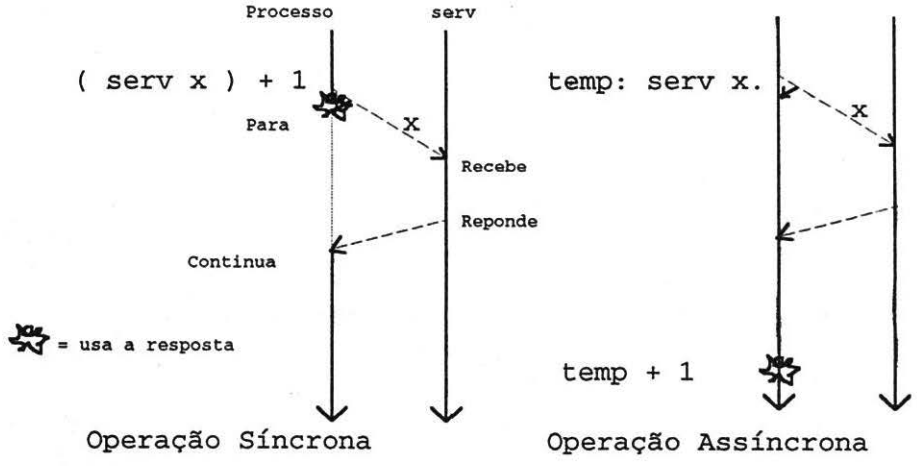


Figura 3 - Sincronismo por Necessidade

As mensagens para os servidores seguem o modelo de sincronismo por necessidade [Ca90]. As mensagens síncronas reduzem bastante a paralelização possível por serem muito restritivas. Já as assíncronas são mais complexas, o que leva a erros freqüentes de programação. No sincronismo por necessidade, as mensagens aos servideres devolvem imediatamente um objeto especial, chamado objeto futuro, que será substituído pela resposta do servidor quando esta estiver disponível. O processo que enviou a mensagem pode continuar sua execução, mas se tentar enviar qualquer mensagem para o objeto futuro será obrigado a esperar pela resposta do servidor. Assim, o sincronismo ocorre só quando houver necessidade, indicada implicitamente no próprio programa.

Pode haver múltiplas cópias de objetos normais mas, neste caso, elas são marcadas como "apenas para leitura". Para alterar um objeto o processador deve obter uso exclusivo deste, pedindo que os outros nós invalidem suas cópias. Os objetos realmente residem em disco, sendo a

memória dos nós apenas um "cache" num esquema de memória virtual. Os processadores de entrada e saída mantêm um diretório para saber onde existem cópias dos objetos para manter o esquema de coerência. O endereço global de um objeto é sua posição no disco. Ao receber uma cópia, no entanto, um nó gera um endereço local, como no LOOM [KK83], que é usado internamente. Isto é invisível para o resto da máquina, pois apenas endereços globais são usados em mensagens entre nós.

Estado do Objeto	Operação	Conseqüência	Próximo Estado
Não Presente	Leitura	Pede uma Cópia	Partilhado ou Exclusivo
Não Presente	Escrita	Pede uma Cópia Pede para Invalidar outras Cópias	Modificado
Partilhado	Pedido de Invalidação		Não Presente
Partilhado	Leitura		Partilhado
Partilhado	Escrita	Pede para Invalidar outras Cópias	Modificado
Exclusivo	Pedido de Invalidação		Não Presente
Exclusivo	Leitura		Exclusivo
Exclusivo	Escrita		Modificado
Modificado	Pedido de Invalidação	Manda de Volta para o Disco	Não Presente
Modificado	Leitura		Modificado
Modificado	Escrita		Modificado

Tabela 1 - Funcionamento da Memória Virtual

O esquema de "invalidação na escrita" parece muito lento, mas muitos dos objetos são apenas lidos e os que são alterados raramente são partilhados. Mesmo assim, a pausa para se buscar uma cópia de um objeto pode reduzir significativamente o desempenho. O segredo é esconder esta "latência" dando outra coisa para o processador fazer enquanto aguarda. Isto só é possível se o grau de paralelismo da aplicação for bem maior que o número de processadores. O modelo de paralelismo apresentado até aqui, o dos servidores (que, a propósito, difere do modelo do Self de Stanford [Ho91]), é muito flexível mas normalmente leva a graus de parelelismo da ordem de poucas dezenas. Pode-se dizer que equivale à paralelização de módulos em linguagens tradicionais.

Além dos objetos normais e servidores, podemos criar "objetos distribuídos" [Da87][Da92].

Muitos dos objetos mais importantes de qualquer programa Self são coleções como: vetor, conjunto, lista, etc. As versões distribuídas destes objetos podem substituí-los para se obter um paralelismo na ordem de centenas ou milhares (semelhante ao paralelismo de "loops" em linguagens convencionais). Estes objetos parecem existir em todos os nós da máquina simultaneamente, mas apenas um pedaço do objeto fica presente fisicamente em cada nó. Quando uma mensagem é enviada para o objeto de qualquer nó, ela é distribuída para todos os nós e cada um executa o que foi pedido para sua parte do objeto. Quando o último nó terminar, quem enviou a mensagem original receberá uma única resposta. É interessante notar que a semântica do envio de mensagens independe do tipo de objeto. Isto faz com que o programa possa ser desenvolvido só com objetos normais e mais tarde paralelizado quase que automaticamente ao se trocar objetos chaves por servidores e/ou objetos distribuídos.

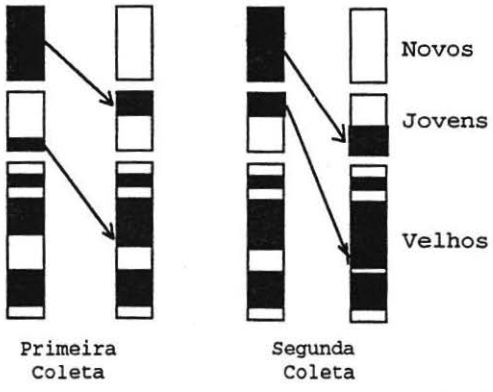


Figura 4 - Funcionamento do "Coletor de Lixo"

O programador pode criar novos objetos em Self, mas não eliminá-los explicitamente. Existe um esquema de "coleta de lixo" [WM89] que reaproveita o espaço dos objetos sem uso. Os objetos, na memória de um nó, são divididos em novos, jovens e velhos. Cada tipo tem sua própria área na memória. Quando a área dos novos se esgota, estes são movidos para a região dos jovens (que, por sua vez, se tornaram velhos). Estas mudanças envolvem apenas os objetos sobreviventes, que são uma minoria. Linguagens como Self, Smalltalk e LISP geram muitos objetos temporários e alguns de longa duração. Quando um objeto chega a ser velho, é muito pequena a chance de se tornar lixo e é ignorado pelo "coletor de lixo". Apenas objetos velhos residem em disco e periodicamente (semanalmente, por exemplo) deve ser executado um coletor de lixo especial para recuperar o espaço do disco. Se for necessário o endereço global de um objeto novo ou jovem (para ser enviado como parâmetro de uma mensagem, por exemplo), este é amadurecido prematuramente para manter a simplicidade da memória virtual.

Quando acaba o espaço de memória dos objetos velhos em um nó, os objetos que estiverem mais tempo sem uso são escritos de volta para o disco (se tiverem sido modificados). Isto é feito até existir espaço suficiente para receber o objeto que estiver vindo do disco ou da área jovem.

Não é feito nenhum tipo de relocação na área velha, o que provoca a fragmentação do espaço disponível. Espaços contíguos aparecem quando um número razoável de objetos volta para o disco.

O MS8702 é normalmente usado para a execução de programas desenvolvidos em outras máquinas. Será colocado, entretanto, um ambiente completo de programação para que o usuário tenha a possibilidade de usar o próprio computador interativamente (ou da própria console, ou remotamente através da rede local) aproveitando horários em que a máquina ficaria sem uso. No futuro será desenvolvido um ambiente gráfico completo equivalente às excelentes ferramentas disponíveis no Smalltalk.

5. CONCLUSÃO

A programação orientada a objetos não precisa ser sinônimo de ineficiência. O uso de paralelismo e técnicas avançadas de compilação permitem o aproveitamento das vantagens dos objetos mesmo em aplicações como simuladores de circuitos e computação gráfica.

6. AGRADECIMENTO

Este trabalho foi apoiado pela FINEP através do convênio 52-87.0125.03. Agradeço a Sérgio Takeo Kofuji pelas ideias para a definição da arquitetura (especialmente a parte de entrada e saída) e pelo trabalho de implementação do hardware. Agradeço ao grupo do Self de Stanford pelo sistema para a SUN 4 e por todas as idéias de implementação e discussões dos detalhes internos.

7. BIBLIOGRAFIA

[AK91] Assumpção Jr., Jecel M. & Kofuji, S. T.: "Proposta de um Computador Paralelo de Alto Desempenho com Memória Distribuida", Anais do XXIV Congresso Nacional de Informática, São Paulo, Setembro 1991

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

[Ch89] Chambers, Craig & et al.: "An Efficient Implementation of Self, a Dynamically-Typed Object-Oriented Language Based on Prototypes", OOPSLA'89 Conference Proceedings, New Orleans, LA, October 1989

[Da87] Dally, William J.: "A VLSI Architecture for Concurrent Data Structures", Kluwer Academic Publishers, 1987

[Da92] Dally, William J.: "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms", IEEE Micro, Vol 12, Num 2, April 1992

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

[Ho91] Hölzle, Urs & et al.: "The Self Manual", Stanford University, 1991

[KK83] Kaehler, Ted & Krasner, Glenn: "LOOM - Large Object-Oriented Memory for Smalltalk-80 Systems", Capítulo 14 de Smalltalk-80 Bits of History, Words of Advice, Addison-Wesley, 1983

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

[WM89] Wilson, Paul R. & Moher, Thomas G.: "Design of the Opportunistic Garbage Collector", OOPSLA'89 Conference Proceedings, New Orleans, LA, October 1989