AVALIAÇAO DE UM AMBIENTE UNIX COM MÚLTIPLOS PROCESSADORES

Gabriel P.Silva
Núcleo de Computação Eletrônica da UFRJ
Caixa Postal 2324, CEP 20001 - Rio de Janeiro, RJ

RESUMO

Este artigo apresenta uma avaliação de desempenho de um sistema multiprocessador, desenvolvido no NCE/UFRJ e denominado PEGASUS/PLURIX. O objetivo principal desta avaliação foi o de determinar o ganho obtido com o uso de múltiplos processadores em um ambiente do tipo UNIX. Um conjunto de programas portáteis foi utilizado para avaliar o sistema, permitindo uma comparação com resultados obtidos na execução destes programas em sistemas comerciais com ambiente UNIX.

ABSTRACT

In this paper, a performance evaluation of a multiprocessor system, named PEGASUS/PLURIX and developed at NCE/UFRJ, is presented. The main purpose of this evaluation was to verify the performance gain obtained with the use of multiple processors in a UNIX-like environment. A set of portable programs has been used and a comparision with some results obtained from commercial systems is also done.

1. INTRODUÇAO

Este artigo apresenta os resultados de uma avaliação do sistema multiprocessador PEGASUS/PLURIX [2]. O PEGASUS é um sistema com múltiplos processadores, barramento central, processadores dedicados de E/S e memória central compartilhada. Cada processador possui um microprocessador MC68020 de 32 bits, trabalhando a 12 Mhz, com memória cache de 4 Kbytes. O PLURIX é um sistema multiprocessador com filosofia UNIX, cujas principais características são a portabilidade e permitir o compartilhamento dos recursos do sistema entre mais de um processador.

A avaliação de qualquer sistema é um processo difícil devido ao grande número de variáveis envolvidas; processador, gerência de memória, barramento, organização dos arquivos, compilador e sistema operacional são algumas delas. No método utilizado neste trabalho, procurou-se não avaliar cada ítem, mas sim determinar o desempenho global do sistema, da maneira como se apresenta ao usuário.

Assim, foram escolhidos uma série de programas portáteis escritos em "C", para avaliação de ambientes do tipo UNIX. Estes programas foram coletados na literatura especializada [5,6,7] e, além de serem portáteis, exercitam as funções mais utilizadas em ambientes UNIX. Estes mesmos programas foram utilizados, como descreveremos mais adiante, para a avaliação do ganho obtido com o uso de múltiplos processadores.

Este artigo apresenta então, o método utilizado para avaliação de multiprocessamento no PEGASUS, uma descrição dos programas de avaliação, uma comparação com alguns sistemas comerciais, e os resultados da execução destes programas no PEGASUS, operando com um e com dois processadores.

2. MULTIPROCESSAMENTO

Existem vários níveis para se explorar o paralelismo em um computador. Isto pode-se dar, por exemplo, ao nível das instruções, ao nível de rotinas, ou ao nível de processos. Nos parágrafos que se seguem, descrevemos as opções que foram adotadas no PEGASUS/PLURIX e o método empregado para avaliar o ganho obtido com multiprocessamento.

O nível de paralelismo existente no PEGASUS/PLURIX é o de processo, ou seja, permite a existência de processos independentes executando simultaneamente nos vários processadores. Como o número máximo de processadores no PEGASUS/PLURIX é pequeno, não pode haver uma granularização excessiva (com processos muito pequenos), sob pena de perda de eficiência devido às trocas de contexto.

A comunicação entre processos de usuário é possível através da utilização de "pipes", que é um conjunto de "buffers" bidirecionais de comunicação, com capacidade limitada a 5 Kbytes, após o qual o processo que está escrevendo é bloqueado. Na ausência de dados no pipe, a leitura é sempre bloqueada. Uma outra ferramenta disponível é o "fork", que gera um processo "filho" que é a cópia exata do processo que chamou a função. Estas ferramentas utilizadas em conjunto permitem a implementação, embora com pouca flexibilidade, de algoritmos paralelos.

Quando um determinado processo deseja alocar um recurso, ou fazer uma operação de E/S, deve realizar uma chamada de serviço ("service call")ao núcleo do sistema operacional. Enquanto um processo, através do núcleo, executa instruções que manipulam um recurso, todos os outros processos ficam impedidos de manipular o mesmo recurso. Esta exclusividade é garantida no PLURIX, através do uso de semáforos [3].

A prevenção de "deadlock" é conseguida através da manutenção de uma ordem pré-estabelecida para a alocação dos diversos tipos de recurso. Esta ordem, porém, nem sempre pode ser respeitada. Nestes casos, usa-se um esquema de detecção e posterior recuperação de "deadlock". Uma vez que se verifica que um recurso já está alocado para outro processo, e que pode haver "deadlock", a recuperação do "deadlock" é feita através da liberação de todos os recursos alocados por este processo.

A distribuição de tarefas entre os vários processadores é responsabilidade de um processo denominado despachante ("dispatcher"), sendo que existe uma cópia para cada processador ativo no sistema. Como a fila de processos é única, a função deste processo é, de tempos em tempos, verificar se existe algum processo para ser executado. Neste caso, o processador correspondente realiza uma troca de contexto e passa a executar o novo processo. O acesso à fila de processos é mutuamente exclusivo, ou seja, só é permitido a um processador por vez.

O uso de mais de um processador oferece um ganho imediato nas configurações multiusuário, pois o tempo de resposta para a execução das tarefas dos usuários diminui, ou seja, os processos aguardam menos tempo na fila, já que há mais servidores para executá-lo.

No caso de haver apenas um usuário no sistema, com uma alta demanda de capacidade computacional, o ganho pode ser obtido através de algoritmos paralelos, utilizando-se processos distintos, comunicando-se entre si. Neste caso, a eficiência com que o problema é resolvido, depende ainda da quantidade de mensagens e da maneira como se processa esta comunicação.

Optou-se por uma forma simples de determinar o ganho que se obtém com o uso de mais um processador: executou-se um processo idêntico em cada processador e comparou-se com o caso em que apenas um processador executa um processo. Esta situação simula o ganho máximo que deve ser obtido no sistema, sendo que em uma aplicação normal, com o uso ou não de algoritmos paralelos, os resultados obtidos serão mais modestos.

Os programas, embora simples, servem para determinar a eficiência do sistema para lidar com três tipos de necessidades básicas existentes em um programa: entrada/saída, processador e chamadas de serviço. Um dos parâmetros para a escolha destes programas também foi a portabilidade, para permitir comparação com sistemas, multiprocessadores ou não, já existentes ou que viessem a ser elaborados. Maiores detalhes sobre a adaptação de ambientes UNIX para uso de processamento paralelo podem ser encontrados nas referências [1] e [8].

3. OS PROGRAMAS DE AVALIAÇAO

3.1. Considerações.

Nesta seção, destacamos alguns pontos que acreditamos úteis na compreensão dos resultados obtidos:

. Na medição dos tempos utilizamos a rotina "time" do UNIX, que apesar de introduzir um pequeno "overhead", oferece um mecanismo padrão de medição de tempo muito mais preciso, p.ex., que a utilização de um cronômetro;

. Na comparação com o desempenho de alguns computadores, normalizamos os resultados em relação ao PEGASUS. Nesta normalização, foi utilizada a média geométrica, e não a média aritmética, pois a utilização desta última pode levar a uma interpretação errada dos resultados [4];

. O tempo de "usuário" da rotina "time" é o tempo gasto pelo processo executando instruções não privilegiadas (p.ex., cálculos aritméticos). O tempo de "sistema" é o tempo gasto executando comandos privilegiados (p.ex., comandos do sistema) e mais algum "overhead" a nível de sistemas (p.ex., trocas de contexto). O tempo "real" é o que o nome indica, não é a soma dos tempos de sistema e usuário. A diferença é o tempo perdido aguardando E/S, sinais de outros processos, ou na fila de processos, entre outros;

. No caso do PEGASUS/PLURIX o menor intervalo de tempo discriminável é 16ms;

. A soma dos tempos de usuário e sistema costuma também ser apresentada como tempo de processador.

3.2. Descrição.

A seguir fazemos uma descrição sumária dos programas selecionados. Uma listagem completa pode ser conseguida com o autor.

1. Syscall.

Este programa determina as perdas nas chamadas ao sistema. Realiza várias iterações de um pedido para retornar o número de identificação do processo.

Nesta chamada estão envolvidos as seguintes perdas: o tempo gasto para preparar e executar o desvio para o núcleo; o tempo para realizar a função pedida; o tempo gasto para o contexto de usuário ser restaurado; e a eventual perda de tempo quando uma troca de contexto entre processos é necessária.

2. Funcall.

Neste programa avaliamos o tempo gasto em uma chamada de função, que realiza uma soma de três variáveis. Uma outra maneira de realizar este teste, verificando a eficiência do compilador "C", é medir a diferença do tempo de usuário entre dois programas: em um deles realizamos uma atribuição a uma variável diretamente, no outro fazemos esta atribuição através de uma chamada de função. Tanto melhor o compilador, quanto menor a diferença entre os tempos de "usuário".

3. Piper.

Neste programa, com o uso do "fork", se obtém dois processos idênticos que se comunicam através de um duto ("pipe"). Isto serve para medir a eficiência do sistema para usar o "pipe". O "pipe" é uma facilidade muito importante em um sistema UNIX, sendo utilizado por vários utilitários (p.ex., "lint") e pela interface de comandos ("shell").

4. Shell.

Este programa é na realidade um conjunto de diversos utilitários frequentemente utilizados em ambientes do tipo UNIX. Este programa faz uso do "shell" e de alguns utilitários para ordenar, salvar em disco, manipular e finalmente remover do disco uma lista de palavras em inglês. Se a implementação do "shell" em teste não for idêntica à do UNIX, pode haver problemas na execução deste programa.

5. Sieve.

O programa "Sieve of Eratosthenes" (crivo de Eratósthenes) é um programa amplamente utilizado. Ele verifica a capacidade máxima do processador e a eficiência do compilador determinando 1899 números primos entre 1 e 8190 [5].

6. Disktest.

Este programa é destinado a verificar a velocidade do sistema em operações de E/S. O teste contém 3 sequências:

1. Cria um arquivo e escreve 1000 registros, de 512 bytes cada;

2. Lê este arquivo sequencialmente em 1000 iterações;

3. Faz 1000 buscas aleatórias e lê um registro de 512 bytes para cada busca.

A escrita e a leitura sequencial representam a movimentação de grande volume de dados, como em cópia de arquivos. A busca aleatória representa a atividade de pesquisa em banco de dados, onde a busca de um determinado registro é o objetivo principal. Um fator que influencia fortemente o desempenho deste programa, é o número de "buffers" de E/S existentes no sistema.

4. ANÁLISE DOS RESULTADOS

Existem 2 tabelas de resultados. Na primeira delas apresentamos dados colhidos na referência [6]. Estes dados são referentes ao desempenho de alguns supermicros na execução dos programas apresentados. Já na tabela II apresentamos os dados referentes a alguns minis e superminis obtidos na referência [7]. Observe que para alguns programas o número de iterações é diferente daquele existente na referência [6]. Nas duas tabelas apresentados resultados do PEGASUS operando com um e com dois processadores.

Nas tabelas de resultados, os tempos do PEGASUS operando com dois processadores estão divididos por dois. Isto permite uma melhor comparação com os tempos medidos com apenas um processador.

A seguir relacionamos os pontos significativos observados nas medidas realizadas:

. Com o PEGASUS configurado com um processador, e a partir dos resultados da execução de programas com uso intensivo de processador, obteve-se a seguinte relação:

PEGASUS = 1.0 VAX-780
PEGASUS = 1.4 VAX-750
PEGASUS = 1.6 PDP-11/70
PEGASUS = 1.5 ZILOG MODEL 11+
PEGASUS = 2.0 AT&T 382
PEGASUS = 2.1 ALTOS 586

. Estes dados permitem situar o PEGASUS na faixa dos superminis, mostrando que os microprocessadores apresentam hoje um desempenho equivalente aos superminis de 32 bits, no que diz respeito à capacidade de processamento. Na comparação com supermicros de mesa (de 16/32 bits), o desempenho foi significativamente superior, o que indica a influência da arquitetura utilizada no desempenho final. O uso de um barramento padrão rápido e eficiente, memória cache e periféricos inteligentes pode ser destacado;

. Com a colaboração de mais um processador, e utilizando-se o método descrito na seção dois, verificou-se que na execução dos programas "sieve" e "funcall" obteve-se um fator de ganho entre 1.8 e 2.0. Estes programas fazem uso intensivo de funções de usuário (aritmética inteira, ordenação, busca, etc.) e são diretamente favorecidos com a adição de mais um processador;

. Na execução dos programas com grande uso de chamadas ao sistema (syscall), o ganho foi bastante pequeno (1.1 a 1.2), apesar de haver bastante uso de processador. Note que, a chamada ao sistema utilizada (getpid), faz acesso a um recurso de exclusão mútua (tabela de processos) onde só é permitido o acesso de um processador por vez. Como o programa é bastante pequeno, esta perda de tempo passa a ser significativa;

. Poderia-se esperar que, com a adição de um outro processador, as operações de E/S fossem prejudicadas, devido ao aumento da disputa pelo uso do barramento. Na execução do programa "disktest", o fator de ganho obtido foi 1.0,ou seja, não houve perdas com a inclusão de outro processador. Isto porque os processadores possuem memória "cache", o que faz com que a taxa de ocupação do barramento seja significativamente diminuída;

. Na tabela de resultados, com o uso de dois processadores, o tempo de "processador" de algumas tarefas é maior que o tempo "real", ou seja, o tempo de processamento seria maior que o tempo total de execução da tarefa. Na realidade, o tempo de "processador" mostrado é a

soma dos tempos de processamento da tarefa em cada um dos processadores, explicando-se assim os valores obtidos;

. Considerando-se um "mix" típico com os programas utilizados, obtêm-se um ganho médio de 1.4. Isto é uma indicação de que em uma aplicação real, com um ambiente multiusuário, o ganho ainda seria significativo;

. Considerou-se nesta análise que os processadores são exatamente iguais. Na realidade, dada a constituição do sistema, um dos processadores é ligeiramente mais rápido que o outro. E em sistemas com vários processadores, sempre haverá diferenças na capacidade de processamento de cada um. Assim, um processo executado em um processador mais lento é prejudicado,processos idênticos teriam tempo de execução diferentes, dependendo de onde fossem executados. E preciso que o sistema operacional tenha mecanismos para compensar as diferenças existentes entre os processadores, eventualmente através da atribuição de "pesos" a cada processador na hora de calcular a prioridade com que um processo volta à fila de execução;

. O uso de algumas facilidades de "hardware", tais como "cache" e processadores de E/S, permite o uso eficiente de múltiplos processadores. Contudo, com a adição de mais processadores, a contenção do barramento tende a ser significativa, limitando o número máximo de processadores que podem ser utilizados. Seria necessário considerar o uso de outras formas de interconexão, para permitir um grau maior de paralelismo que o obtido atualmente.

5. CONCLUSAO

As arquiteturas com múltiplos processadores e barramento central são, em minha opinião, uma ponte entre os sistemas centralizados e os totalmente distribuídos, devendo ser convenientemente exploradas.

Este artigo apresentou os resultados de uma avaliação do sistema PEGASUS/PLURIX, um sistema multiprocessador com memória central compartilhada. Algumas dificuldades, tais como a manutenção da consistência dos dados na cache,tiveram que ser superadas, até que pudessemos contar com um sistema funcionando adequadamente.

A avaliação realizada procurou mostrar o sistema de uma maneira global, e não apenas o "hardware" ou "software" isoladamentes. Os resultados até agora obtidos, mostram que os dois processadores trabalham sem contenção no barramento. Mostram também uma implementação correta do sistema operacional, que permite o compartilhamento dos recursos com bom desempenho e sem ocorrência de "deadlock".

O trabalho aqui apresentado não pretende ser exaustivo. Existe a necessidade de extender as medidas para ítens não abordados, tais como: tolerância a falhas, desempenho em computação científica e algoritmos paralelos. Para este último ítem, está sendo desenvolvido um compilador "C" com extensões concorrentes, para facilitar a implementação destes algoritmos.

Finalmente, espera-se em um futuro próximo apresentar outros resultados, que venham a se acrescentar ao já realizado em termos de avaliação de desempenho.

BIBLIOGRAFIA

1. CAJANI, V. "Architectural Issues in Designing a Unix Multiprocessor System". **Microprocessing and Microprogramming**, Amsterdan, North-Holland, **20** (1-3): 79-84 , Apr 1987;
2. FALLER, N.; SALENBAUCH, P. "Técnicas de Projeto Utilizadas na Construção do Supermicro PEGASUS-32X e do Sistema Operacional PLURIX", **DATA NEWS**, São Paulo, SP, CWB,Ano X (269): pp. 12-16, 30 Abr. 1985;
3. FALLER, N.; SALENBAUCH, P. "Plurix, o Sistema Operacional Multiprocessador do NCE/UFRJ: Sincronização de Processos", **DATA NEWS**, São Paulo, SP, CWB, Ano X (290): 26-35, 24 Sep. 1985;
4. FLEMING, P.J. & WALLACE, J.J. "How not to Lie with Statistics: The Correct Way to Summarize Benchmarks Results". **Communications of the ACM**, New York, **30** (3): 218-21, Mar 1986;
5. GILBREATH, J. and GILBREATH, G. "Erathostenes Revisited", **Byte**, Peterborough, NH, McGraw Hill, **9** (2): 283-326, Jan 1983;
6. HENDRICKS, S.; DENNEY, M.; HALLIDAY,D."Performance Tests on Unix Micros", **UNIQUE**, Denville, N.J., **3** (7): 20-28, Summer 1984;
7. HINNANT, D.F. "Benchmarking Unix Systems". **Byte**, Peterborough, N.H., McGraw Hill,(8): 132-5, 400-9, Aug. 1984;
8. JACOBS, HERB & TEST, JACK, A. "The Unix System Adapts to a Parallel Processing". **Unix World**, Mountain View, CA,Tech Valley, **3** (16): 48-52, 54-55, June 1986.

TABELA I

Nome do Programa	AT&T 3B2	ALTOS 586-30	ZILOG 11+	PEGASUS 32.1	PEGASUS 32.2	Ganho Relat.
Sieve	7.1 (7.0)	7.0 (6.9)	5.0 (4.9)	3.1 (3.0+0.0)	1.7 (2.6+0.1)	1.8
Syscall	3.5 (3.4)	5.5 (4.8)	4.0 (4.0)	1.5 (0.1+1.4)	1.4 (0.2+2.4)	1.1
Funcall	48.0 (47.8)	37.0 (36.8)	15.0 (14.9)	18.9 (18.8+0.1)	10.6 (19.4+0.3)	1.8
Piper	10.2 (5.3)	11.0 (6.5)	9.0 (3.9)	10.5 (0.1+4.6)	7.9 (0.2+6.9)	1.3
Disktest	30.3 (12.5)	70.0 (15.1)	30.0 (10.7)	33.8 (0.2+15.1)	33.9 (0.2+15.7)	1.0

AT&T 3B2 = 2 Mbytes de memória, 32 Mbytes de disco, CPU 32100, Unix Versão V.
ALTOS 586-30 = 512 Kbytes de memória, 30 Mbytes de disco, CPU 8086, Xenix Versão 7.
ZILOG 11+ = 512 Kbytes de memória, 52 Mbytes de disco, CPU Z8000, Zeus Versão III.
PEGASUS = 4 Mbytes de memória, 85 Mbytes de disco, CPU 68020, Plurix Versão 1.0.
Tempos em segundos: real (user + sys)

TABELA II

Nome do Programa	VAX-780	VAX-750	PDP-11/70	PEGASUS 32.1	PEGASUS 32.2	Ganho Relat.
Sieve	1.7 (1.5+0.1)	2.4 (1.7+0.1)	2.3 (1.6+0.1)	1.6 (1.5+0.0)	0.8 (1.3+0.1)	2.0
Syscall	4.8 (0.4+4.0)	7.0 (0.8+6.2)	8.0 (0.2+7.5)	3.8 (0.3+3.5)	3.1 (0.3+6.1)	1.2
Shell	3.3 (0.3+1.8)	3.8 (0.4+1.5)	4.0 (0.2+1.7)	3.6 (0.3+2.8)	3.1 (0.3+2.0)	1.2
Piper	3.2 (0.1+1.2)	4.6 (0.2+2.1)	8.1 (0.0+3.4)	4.5 (0.0+2.0)	3.8 (0.1+3.3)	1.2

VAX-780 = 4 Mbytes de memória, 256 Mbytes de disco, Unix 4.1 BSD.
VAX-750 = 2 Mbytes de memória, 121 Mbytes de disco, Unix 4.1 BSD.
PDP-11/70 = 1 Mbyte de memória, 400 Mbytes de disco, Unix 2.8 BSD.
PEGASUS = 4 Mbytes de memória, 85 Mbytes de disco, Plurix 1.0
Tempos em segundos: real (user + sys)