MULPLIX: Um Sistema Operacional tipo UNIX para o Multiprocessador MULTIPLUS

Gustavo Peixoto de Azevedo (*)
Rafael Peixoto de Azevedo (+)
Norival Ribeiro Figueira (@)
Julio Salek Aude (#)

RESUMO

Este trabalho é um relato do estágio atual de desenvolvimento do MULPLIX, sistema operacional que está sendo projetado para atuar no MULTIPLUS, um multiprocessador científico de alto desempenho em desenvolvimento no NCE/UFRJ. Na sua versão inicial, o MULPLIX será resultado de extensões no PLURIX visando adequá-lo à arquitetura do MULTIPLUS a aos requisitos mínimos de aplicações científicas paralelizáveis. As principais extensões se referem a possibilidade de criação de processos leves, modificação das políticas de escalonamento e gerência de memória, colocação de primitivas de sincronização disponíveis para o usuário e implementação mais eficiente das primitivas de sincronização do tipo espera ocupada.

ABSTRACT

This paper describes the current development state of MULPLIX, an operating system which is being designed for MULTIPLUS, a high-performance scientific multiprocessor computer under development at NCE/UFRJ. In its initial version, MULPLIX will be a result of extensions to PLURIX aiming at the adaption of PLURIX to MULTIPLUS architecture and the requirements of parallel scientific applications. The main extensions are related to the possibility of creation of light-weight processes, changes in the scheduling and memory management policies, availability of synchronization primitives for the user and a more efficient implementation of busy waiting synchronization primitives.

(*) (+) (@) NCE/UFRJ; (#) NCE/UFRJ e IM/UFRJ.
CP 2324, CEP 20001, Rio de Janeiro, RJ; Tel: 290-3212 (r. 278).

(*) Mestrando Ciência da Computação, COPPE Sistemas UFRJ.
(+) Mestrando Ciência da Computação, COPPE Sistemas UFRJ.
(@) M.Sc. Ciência da Computação, COPPE Sistemas UFRJ.
(#) Ph.D. Ciência da Computação, Manchester University, UK.

1. INTRODUÇÃO

1.1 O Projeto MULTIPLUS

O projeto MULTIPLUS [3], atualmente em desenvolvimento no NCE/UFRJ, visa a concepção e construção de uma família de computadores paralelos de alto desempenho para aplicações científicas e de engenharia. A sua realização demanda pesquisa e desenvolvimento nas áreas de arquitetura de computadores, sistemas operacionais, microeletrônica e algoritmos paralelos.

O sistema operacional MULPLIX está sendo projetado para atuar no MULTIPLUS. Ele é uma evolução do PLURIX [8] visando propiciar ao usuário um ambiente para desenvolvimento e execução de aplicações intensamente paralelas. A sua compatibilidade com o PLURIX garante acesso a todo o conjunto de utilitários e aplicativos já desenvolvidos e/ou transportados para o PLURIX.

A arquitetura do MULTIPLUS está estruturada como um conjunto de "Clusters" interligados através de uma Rede de Interconexão (RI) multiestágio do tipo n-cubo invertido. Cada Cluster é normalmente composto de um número entre 1 e 8 de Nós de Processamento (NP), um Processador de Entrada e Saída (PES) orientado a blocos, um Processador de Entrada e Saída (PES) orientado a caracteres e uma Interface de Rede (IR). Os NP's, os PES's e a IR são interligados através de um barramento duplo, composto de um barramento de dados e de um barramento de instruções com 64 bits de largura. Ao todo o MULTIPLUS é capaz de suportar até 256 NP's.

Cada NP está equipado com um microprocessador SPARC, um coprocessador de ponto flutuante, um módulo de memória com 16 Mbytes, duas unidades de memória cache (uma para dados e a outra para instruções) e hardware de suporte à gerência de memória.

A memória do sistema é global apesar de estar fisicamente distribuída pelos NP's. Isto significa que os NP's podem acessar transparentemente tanto o seu próprio módulo de memória, quanto os módulos de memória localizados em outros NP's.

1.2 Os Sistemas Operacionais PLURIX e MULPLIX

O PLURIX é um Sistema Operacional com Filosofia Unix (SOFIX), inteiramente desenvolvido no NCE/UFRJ a partir de 1982. O PLURIX contrasta com o UNIX principalmente por ser capaz de controlar simetricamente computadores multiprocessados. Esta característica exige que o núcleo do PLURIX seja um programa paralelo e que a sua construção seja baseada em técnicas mais sofisticadas, normalmente não empregadas em sistemas monoprocessados.

A capacidade de multiprocessamento do PLURIX é transparente aos usuários, ou seja, o PLURIX oferece ao programador um ambiente multiusuário essencialmente igual ao

UNIX. Entretanto, é possível acelerar até certo ponto determinadas tarefas, desde que elas possam ser decompostas em subtarefas de tamanho razoável e de interação bastante simples.

O utilitário "make" do PLURIX [4] é capaz de identificar e comandar operações que podem ser realizadas concorrentemente. Em computadores contando com mais de um processador esta concorrência se transforma em paralelismo. No caso da utilização típica do PLURIX em ambiente universitário - desenvolvimento de software - o exemplo mais claro de uso destas facilidades refere-se à aceleração da compilação de programas com fontes organizados em vários módulos. Esta tarefa envolve a compilação (concorrente) dos módulos cujos objetos são a seguir ligados.

O exemplo dado no parágrafo anterior sugere uma possibilidade de grande aumento da velocidade de processamento. Uma análise mais aprofundada, entretanto, revela que no caso mais otimista (compilação de muitos módulos e disponibilidade de muitos processadores) o tempo total de compilação tem um limite inferior dado pelo tempo de ligação dos módulos objetos, porque este processo é realizado seqüencialmente.

O grau de paralelismo que pode ser explorado pelo PLURIX é adequado à arquitetura multiprocessadora de pequena escala do PEGASUS [8]. Em casos específicos, ele proporciona ganhos de velocidade compatíveis com um pequeno número de processadores (até cerca de uma dezena). No caso mais geral, entretanto, os processadores são vistos no PLURIX como mais um recurso disponível para compartilhamento entre aplicações. Assim, a vantagem principal obtida através do uso de um número maior de processadores não é acelerar o processamento de uma aplicação, mas sim atender satisfatoriamente a um número maior de aplicações ou usuários.

Em termos de performance, o objetivo fundamental do MULTIPLUS e, conseqüentemente, do MULPLIX é viabilizar, através da exploração intensa de paralelismo, aplicações científicas e de engenharia que demandam uma enorme quantidade de processamento. Isto significa acelerar estas aplicações por um fator de dezenas ou centenas de vezes.

Considerando as diferenças de objetivos de projeto e as diferenças de arquitetura entre as máquinas base para desenvolvimento (MULTIPLUS e PEGASUS), o MULPLIX representará em alguns aspectos uma evolução técnica do PLURIX. Deste modo, o MULPLIX será um ambiente mais completo para o desenvolvimento e execução de programas paralelos, utilizará técnicas mais sofisticadas para gerência de memória e escalonamento de processos e tornará disponíveis para o usuário mecanismos de sincronização. A definição de um sistema operacional como o MULPLIX ainda é objeto de pesquisa a nível internacional. Por isso, a melhor estratégia para desenvolvimento do MULPLIX é realizá-lo em etapas; a experiência de uso do sistema com certeza influenciará fortemente o seu próprio desenvolvimento.

A versão inicial do MULPLIX tem o objetivo principal de

minimizar o tempo necessário a que o MULTIPLUS se torne operacional. Por operacional se entende que o MULTIPLUS possa ser utilizado como uma máquina SOFIX e possa ser demonstrado através da execução de aplicações paralelas cientificas e/ou de engenharia. Assim a versão 0.0 do MULPLIX é essencialmente uma adaptação do PLURIX, contendo modificações decorrentes da mudança de hardware e do suporte à execução de programas paralelos.

1.3 Sumário deste Artigo

O objetivo principal deste artigo é divulgar o estado atual de desenvolvimento do MULPLIX. As seções a seguir apresentam as extensões e adaptações que serão realizadas no PLURIX para a versão 0.0 do MULPLIX. A seção 2 descreve o modelo de execução concorrente do PLURIX, que é adequado aos programas seqüenciais, e apresenta as modificações necessárias à programação paralela. A seção 3 aborda as alterações na gerência de memória do PLURIX decorrentes do novo conceito de processo e da arquitetura NUMA. A seção 4 discute a necessidade da sincronização, analisa os métodos de sincronização no UNIX e no PLURIX e apresenta os mecanismos disponíveis no MULPLIX e a sua implementação no MULTIPLUS. A seção 5 apresenta os diferentes objetivos e métodos para escalonamento em sistemas de tempo compartilhado, como o PLURIX, e em sistemas para programação paralela, como o MULPLIX. Na seção 6 apresentamos o estágio atual de desenvolvimento e as perspectivas futuras para o MULPLIX.

2. MODELOS DE EXECUÇÃO CONCORRENTE

Uma aplicação paralela se distingue de uma aplicação seqüencial pela sua estruturação em unidades de execução concorrente. Unidades de execução concorrente têm a capacidade de serem executadas simultaneamente. O paralelismo ocorre quando há mais de um processador e esta capacidade é exercida.

O desempenho de uma aplicação paralela depende fundamentalmente da qualidade da sua concepção e da adequação e eficiência do ambiente de execução provido pelo sistema operacional.

A estruturação em unidades concorrentes de uma aplicação paralela bem concebida atende aos seguintes objetivos:

(1) Ser função da natureza do problema a ser resolvido e dos seus dados de entrada. Uma aplicação independente da arquitetura do computador para o qual foi inicialmente projetada pode aproveitar os melhoramentos de computadores paralelos mais poderosos.

(2) Maximizar o grau de concorrência, que é o potencial para paralelismo. Isto significa decompor a aplicação em um número máximo de unidades concorrentes.

(3) Minimizar a interdependência das unidades concorrentes. A interdependência está relacionada a necessidade de sincronização e comunicação, que são atividades que envolvem custos adcionais.

Os três objetivos acima podem acarretar dificuldades. A independência da arquitetura pode levar a ineficiência de execução em um computador com características discrepantes das requeridas pela aplicação. Há ainda um conflito entre os objetivos (2) e (3): a divisão de um sistema em um número maior de subsistemas normalmente implica em uma maior dependência entre os subsistemas.

Um sistema operacional adequado a aplicações paralelas deve prover um ambiente de execução que minore estas dificuldades. Por exemplo: um número de unidades concorrentes bastante superior ao número de processadores pode ser eficientemente suportado através de um adequado escalonamento destas unidades pelos processadores. Outro exemplo: mecanismos eficientes de sincronização e comunicação entre as unidades concorrentes possibilitam aumentar o grau efetivo de paralelismo da aplicação.

A subseção 2.1 analisa as características do ambiente provido pelo PLURIX para a execução de unidades concorrentes, que são decorrência da decisão de ser um sistema de uso geral, e mostra a sua inadequação para a programação paralela. A subseção 2.2 discute a solução adotada para o MULPLIX versão 0.0.

2.1 Modelo de Concorrência do PLURIX

A unidade de execução concorrente do PLURIX é o processo. As seguintes entidades, entre outras, estão associados a um processo:

- Espaço de endereçamento virtual composto no modo usuário de um segmento de texto, com o código em execução, um segmento de pilha e um segmento de dados.
- Contexto do processador, contendo os valores correntes dos registradores.
- Recursos alocados, entre eles os arquivos abertos.

O mínimo de operações efetivamente necessárias em qualquer sistema para que haja uma troca de contexto é o salvamento do estado atual do processador e a restauração do seu estado após a última instrução executada pela unidade concorrente que está retornando à execução. Uma troca de contexto no PLURIX envolve além destas operações, outras que podem ser extensas como a substituição das informações da gerência de memória relativas ao espaço virtual de endereçamento do processo que sai pelo que entra. A realização de uma troca de contexto no PLURIX exige a execução de várias centenas de instruções.

As aplicações paralelas bem concebidas apresentam um grau de concorrência muito grande. Este grau de concorrência tipicamente implica em um número maior de unidades de execução do que de processadores disponíveis e pode exigir a utilização mais freqüente de mecanismos de sincronização e comunicação, o que por sua vez tende a aumentar o número de trocas de contexto. Assim, aplicações bem concebidas demandam muitas trocas de contexto. Como no PLURIX uma troca de contexto é relativamente lenta, ela não é adequada à execução de aplicações paralelas.

A comunicação e sincronização entre processos no PLURIX pode se dar por herança (um processo novo recebe entre outros ítens uma cópia dos dados do processo que o originou), por envio e recepção de interrupções de software (sinais), por leitura e escrita de fifos e por leitura e escrita de arquivos. A comunicação por herança só ocorre no momento de criação de um novo processo. Os sinais são uma forma de comunicação muito restrita, sendo praticamente úteis apenas para informar a ocorrência de um evento. A troca de mensagens por leitura e escrita em fifos é restrita e é relativamente lenta. As operações sobre arquivos mesmo que realizadas em memória também utilizam muitas instruções. Portanto os mecanismos para a comunicação entre processos disponíveis no PLURIX não permitem uma eficiente cooperação intensiva entre processos. Esta caracteristica dificulta a utilização eficiente de um grau de concorrência maior. Desta forma, os mecanismos para comunicação e sincronização entre processos providos pelo PLURIX não são adequados à programação paralela.

2.2 Modelo de Concorrência do MULPLIX

A principal causa de ineficiência relacionada ao suporte provido pelo PLURIX para cooperação entre as unidades de execução concorrentes é que elas são processos distintos e portanto não compartilham recursos.

No MULPLIX o conceito de processo foi estendido para suportar mais de uma linha de execução ("thread"). Nesta nova organização, um processo é um ambiente onde podem coexistir várias linhas de execução. Estão associadas a um processo um conjunto de recursos providos pelo sistema operacional e compartilhados por suas linhas de execução, tais como o texto (instruções) de um programa, um espaço de endereçamento, arquivos abertos, etc... Uma linha de execução basicamente tem associado um ponto de execução do programa. Uma aplicação paralela corresponde a um processo e o seu conjunto de linhas de execução.

O compartilhamento de recursos entre linhas de execução de um mesmo processo simplifica, e conseqüentemente acelera, a troca de contexto entre estas linhas de execução.

As seções 3 (Gerência de Memória) e 4 (Sincronização) analisam mais detalhadamente as facilidades para cooperação entre linhas de execução de um mesmo processo.

A pluralidade de linhas de execução em um mesmo processo é uma idéia também adotada no sistema operacional Mach [1] da Universidade Carnegie Mellon e em sistemas operacionais de tempo real [11].

3. GERÊNCIA DE MEMÓRIA

A gerência de memória do PLURIX é segmentada e utiliza o mecanismo de "swap" em disco para prover a ilusão de uma disponibilidade de memória superior à realmente existente. A área de memória de um processo é basicamente dividida em segmentos de dados, pilha e código, tanto para o modo usuário quanto para o modo supervisor. O compartilhamento de memória entre processos não é previsto.

A estrutura hierárquica da memória do MULTIPLUS o classifica como uma arquitetura NUMA ("Non Uniform Memory Access"), o que resulta em tempos de acesso à memória dependentes da sua localização. Segundo LeBlanc et al [9], um sistema operacional para um multiprocessador NUMA com centenas de processadores, deve prover gerência de memória virtual, facilitar o compartilhamento dinâmico geral de memória e minimizar a diferença nos tempos de acesso. Esta minimização exige a exploração efetiva da localidade de memória.

Visando explorar a localidade de memória e suportar o novo conceito de processo, a gerência de memória do MULPLIX difere da gerência do PLURIX nos seguintes pontos:

- Adoção de um esquema de memória paginada com alocação esparsa.
- Replicação do código do núcleo para todos os NP's.
- Replicação do código do processo para os Clusters onde o processo esteja sendo executado.
- Definição de um segmento adicional de dados locais (não compartilhado) para cada linha de execução.
- Definição de segmentos de pilha (usuário e supervisor) para cada linha de execução.

4. SINCRONIZAÇÃO

Esta seção define o termo sincronização, analisa a sincronização interna dos sistemas operacionais UNIX, PLURIX e MULPLIX, apresenta e discute os mecanismos de sincronização disponiveis para a programação de aplicações paralelas no MULPLIX e, finalmente, apresenta a implementação dos mecanismos de sincronização na arquitetura do MULTIPLUS.

4.1 Definição de Sincronização

Quando duas ou mais atividades concorrentes cooperam entre si, faz-se necessária uma coordenação para que esta cooperação se processe corretamente. O termo "sincronização" abrange os aspectos temporais desta coordenação, ou seja, atividades concorrentes são sincronizadas quando o seu progresso no tempo obedece a determinadas restrições.

Um sistema concorrente pode ser modelado no tempo como um conjunto de operações seqüenciais, sujeitas a uma relação de ordem parcial e a uma relação de exclusão mútua.

A relação de ordem parcial é uma relação transitiva, que define pares ordenados de operações na forma "(a, b)", significando que a operação "b" somente pode ser executada após a operação "a". O adjetivo "parcial" indica que a relação não abrange todos os possíveis pares de operações (*).

A relação de exclusão mútua define pares de operações cuja execução não pode se sobrepor no tempo. A relação de exclusão mútua está diretamente associada ao compartilhamento de recursos.

No caso do modelo básico de programação paralela oferecido pelo MULPLIX, as operações de um sistema concorrente são agrupadas em diversas linhas de execução. As operações que compõem cada linha de execução são executadas seqüencialmente, assim estão totalmente ordenadas entre si e obviamente são mutuamente exclusivas e portanto já estão implicitamente sincronizadas. A sincronização envolvendo operações em linhas de execução distintas é realizada através de mecanismos de sincronização. Quanto maior o número de linhas de execução, maior o potencial de paralelismo, mais intensa será a cooperação entre elas e, conseqüentemente, maior a necessidade de sincronização explícita.

4.2 Sincronização Interna ao Sistema Operacional

Em ambientes SOFIX a interação entre o núcleo do sistema e um processo ocorre através de chamadas ao sistema, que envolvem a alteração do estado do processador do modo usuário para o modo supervisor. Em modo supervisor o processo executa o núcleo (código e estruturas de dados). Assim, os processos executam não apenas o programa preparado pelo usuário, mas também o núcleo do sistema operacional.

O compartilhamento do núcleo pelos processos define linhas de execução concorrente para o mesmo e portanto existe a necessidade de sincronização do núcleo. Os parágrafos a seguir analisam as soluções para sincronização do núcleo adotadas para os sistemas UNIX, PLURIX e MULPLIX.

O problema de sincronização do núcleo do UNIX [5] é

(*) Por isto o sistema é concorrente !

bastante simplificado em razão de duas restrições de seu projeto: (1) aplicação voltada para sistemas uniprocessadores e (2) incapacidade de preempção. Em sistemas uniprocessadores as possibilidades de concorrência do núcleo são restritas aos diversos processos inativos em modo supervisor e ao processo ativo. A incapacidade de preempção significa que quando o núcleo é ativado por uma chamada ao sistema, ele normalmente executa continuamente até o término da chamada; o processo ativo em modo supervisor somente é substituído em situações bem definidas. Assim é necessário considerar a sincronização apenas nestas situações: (1) quando uma subrotina do núcleo decide substituir o processo ativo e (2) quando há uma interrupção para atendimento de um dispositivo de E/S.

A substituição do processo ativo exige do programador do núcleo a garantia de que todas as estruturas de dados manipuladas por este processo estão em um estado consistente. Do ponto de vista da engenharia de software esta solução tem a desvantagem de exigir um conhecimento global do núcleo mesmo para alteração de uma característica específica do sistema.

No caso de atendimento de interrupções, é suficiente ordenar os diversos tipos de atendimentos de modo a desabilitar interrupções que poderiam trazer concorrência na manipulação de estruturas de dados.

O problema de sincronização do núcleo do PLURIX [7] foi resolvido de um modo mais completo do que no UNIX. O núcleo de PLURIX utiliza sincronização explícita baseada nos seguintes mecanismos: (1) semáforos binários em espera ocupada (família de primitivas "spin"), (2) semáforos binários preemptíveis (família de primitivas "sleep"), (3) semáforos generalizados (família de primitivas "sema") e (4) eventos (família de primitivas "event"). A situação de bloqueio fatal ("deadlock") é eliminada através da ordenação dos recursos (para impedir circularidade na alocação) e, quando não é possível obedecer a esta ordem, através da liberação de todos os recursos alocados pelo processo e início de uma nova tentativa de alocação no caso da detecção da possibilidade de bloqueio fatal.

O método adotado para sincronização interna do PLURIX será também adotado para o MULPLIX. A implementação das primitivas será adaptada para a obtenção de um melhor desempenho no MULTIPLUS (veja a subseção 4.4).

4.3 Primitivas de Sincronização Oferecidas pelo MULPLIX

As primitivas de sincronização oferecidas pelo MULPLIX na versão 0.0 foram projetadas de modo a atender aos seguintes objetivos:

(1) Suficiência. Todas as diversas necessidades de sincronização por parte das aplicações paralelas devem ser atendidas direta ou indiretamente através das primitivas.

(2) Simplicidade: generalidade e eficiência. O núcleo

deve implementar com eficiência um pequeno número de primitivas bastante genéricas. A utilização destas primitivas deve ser feita preferencialmente através de bibliotecas de subrotinas e compiladores, apesar do seu uso direto pelo programador mais experiente também ser possível.

(3) Segurança. As primitivas se mal utilizadas poderão levar uma aplicação a um estado de bloqueio fatal, sem que isto afete gravemente as outras aplicações ou o próprio sistema.

Embora o sistema não restrinja o uso de mecanismos potencialmente perigosos, os programas de sistema que eventualmente os utilizarem devem ser implementados de modo a não permitir uma interferência danosa em seu funcionamento por parte de usuários.

As primitivas de sincronização oferecidas pelo MULPLIX são extensões às primitivas baseadas em semáforos e eventos utilizadas para sincronização do núcleo. Por simplicidade, as primitivas serão implementadas totalmente em modo supervisor, cabendo ao núcleo alocar as estruturas de dados necessárias aos mecanismos.

As primitivas baseadas em semáforos foram estendidas para fornecer diretamente a identificação do recurso liberado. As primitivas são as seguintes: "semaalloc" e "semadealloc" para alocação de um semáforo, "semainit" para o estabelecimento de um valor inicial para o semáforo, "semalock" para a obtenção de um recurso, "semafree" para a liberação de um recurso e "sematest" que obtém o recurso apenas se imediatamente disponível e retorna o sucesso da operação.

As primitivas baseadas em eventos foram estendidas para possibilitar a manipulação de eventos cuja ocorrência depende da sua sinalização por mais de uma linha de execução. As primitivas são as seguintes: "eventalloc" e "eventdealloc" para alocação de um evento, "eventinit" para estabelecimento do número de linhas de execução que precisarão sinalizar este evento para que ele seja considerado ocorrido, "eventcount" para a sinalização por uma linha de execução, "eventdone" para a sinalização forçada do evento (desconsiderando o valor estabelecido pela primitiva "eventinit"), "eventwait" para esperar a ocorrência de um evento e "eventtest" para verificar a ocorrência de um evento.

4.4 Implementação na Arquitetura do MULTIPLUS

Esta seção descreve a implementação das operações "F&I" ("Fetch and Increment") e "TAS" ("Test and Set") e das primitivas de sincronização baseadas em espera ocupada de semáforos binários. Estas operações e primitivas formam a base para a implementação das demais primitivas para sincronização em modo supervisor (entre linhas de execução do núcleo) e em modo usuário (entre linhas de execução de um mesmo processo), por isso a eficiência das suas implementações no MULTIPLUS é fundamental para o desempenho geral do sistema.

A operação "TAS" pode ser implementada a partir da operação "F&I". A operação "F&I" não é diretamente suportada pelo SPARC. As instruções atômicas definidas pelo SPARC são: "LDSTUB" (lê um byte sem sinal de uma posição de memória e escreve uma constante em seu lugar) e "SWAP" (lê uma palavra de uma posição de memória e escreve outra em seu lugar). Considerando que o projeto MULTIPLUS também envolve o desenvolvimento de um novo microprocessador, segundo as definições do SPARC, a conveniência da implementação de uma instrução "F&I" foi analisada para este novo processador. Esta opção foi descartada por duas razões: (1) a disponibilidade do novo processador está prevista apenas para uma etapa posterior do projeto MULTIPLUS e (2) é importante conservar a compatibilidade deste processador com a definição do SPARC, de modo a não limitar a sua utilização ao MULTIPLUS. Diante da impossibilidade de alteração do processador, foi adotada uma solução externa. Ela se baseia nos seguintes princípios:

- O programador usa "LDSTUB" como "F&I" de um byte e "SWAP" como "F&I" de uma palavra. Conseqüência: estas instruções originais não são implementadas no MULTIPLUS. Avaliação: provavelmente não há perda, porque, considerando a natureza específica destas instruções, elas não devem estar presentes em programas de usuário.

- A memória tem a inteligência necessária à realização de um "F&I" quando da recepção de uma leitura para a "F&I".

- Os circuitos externos ao SPARC ignoram os pedidos de escrita decorrentes de operações de "F&I".

A viabilidade da solução proposta depende das seguintes questões:

- Reconhecimento e operação transparente dos pedidos de leitura decorrentes de operações "F&I" pela Rede de Interconexão. Resolvido pela criação de um comando de rede para a leitura "F&I".

- Interação correta com os caches. Solução: as variáveis operadas por "F&I" são armazenadas em espaços de endereçamento definidos como não possíveis de estar no

cache.

- Serialização das leituras "F&I". Isto é garantido porque as operações "F&I" são realizadas pelo módulo de memória em que o operando se encontra.

A forma de implementação da espera ocupada em multiprocessadores de larga escala com memória compartilhada deve ser muito eficiente [2]. Uma má implementação pode sobrecarregar demasiadamente as vias de acesso à memória e assim diminuir sensivelmente o desempenho efetivo do multiprocessador.

Os requerimentos para uma implementação eficiente da espera ocupada são:

- Provocar o mínimo de aumento do tráfego nas vias de acesso à memória global.

- Ser rápida, de modo a contribuir para que o tempo de utilização do semáforo binário seja mínimo e portanto permitir uma maior taxa de utilização.

Para que a utilização do semáforo binário seja rápida é necessário impedir a preempção nos processadores aguardando-o, de modo que eles possam utilizá-los imediatamente após sua aquisição. Além disso, é claro que deve-se minimizar as instruções necessárias a obtenção do semáforo binário, quando ele já estiver livre e as instruções necessárias a sua liberação quando houver processadores esperando-o.

O algoritmo "Queueing in shared memory" apresentado em [2] foi utilizado como base para desenvolvimento de um novo algoritmo mais adequado ao MULTIPLUS. As idéias básicas deste novo algoritmo são:

- Enfileiramento dos processadores esperando por um semáforo binário em um buffer circular;

- Percepção da disponibilidade de um semáforo binário através da alteração de uma variável local no cache;

- Se houver um processador esperando pelo semáforo binário, o processador que o libera deve passá-lo diretamente ao próximo, sem perturbar os demais.

Iniciação de um Semáforo Binário:

```
para cada posição i de buffer
    buffer[i] := ESPERANDO

atual := 0

POS (atual) := DOANDO
```

Obtenção de um Semáforo Binário:

```
local[ID] := ESPERANDO

p := POS (F&I (atual))

buffer[p] := ID

se buffer[ANT (p)] <> DOANDO
    espera (local[ID] = DOANDO)
```

Liberação de um Semáforo Binário:

```
buffer[p] := DOANDO

se buffer[SUC(p)] = id
    local[id] := DOANDO

buffer[ANT(p)] := ESPERANDO
```

A figura acima mostra os algoritmos para a obtenção, liberação e iniciação de um semáforo binário. Estes algoritmos realizam a espera ocupada observando uma palavra presente no módulo de memória local ao processador, o que reduz significativamente a contenção nos barramentos.

5. ESCALONAMENTO

A função básica do escalonamento é a alocação de linhas de execução de processos aos processadores.

5.1 Objetivos

Os escalonadores dos sistemas multiprogramados interativos tem por objetivos prover a ilusão da existência de um processador para cada unidade de execução e maximizar a utilização dos processadores. Como forma de atingir a estes objetivos, eles são dinâmicos (tomam decisões a tempo de execução), e realizam um compartilhamento do tempo dos processadores entre aquelas unidades. As políticas adotadas para este compartilhamento visam distribuir com justiça o tempo de processador entre aquelas unidades, minimizar os seus tempos de reação aos comandos dos usuários e evitar esperas indefinidas.

O objetivo fundamental do MULTIPLUS é tornar viável a execução de aplicações científicas ou de engenharia que demandem muito processamento, através da exploração intensiva de paralelismo. A viabilidade está relacionada ao tempo real necessário à finalização das aplicações paralelas em execução, conseqüentemente o objetivo central do escalonamento no MULPLIX é minimizar este tempo.

A maior parte das máquinas para processamento paralelo tem um escopo de utilização limitado devido a dificuldade para programá-las. Esta dificuldade decorre das limitações ou das particularidades dos modelos de programação paralela presentes nestas máquinas. Assim normalmente as aplicações são programadas especificamente para uma máquina particular, com uma configuração de hardware pré-estabelecida.

O MULTIPLUS pretende se diferenciar das demais máquinas por prover um ambiente de programação mais genérico e confortável. Esta generalidade envolve uma maior independência do programador em relação à configuração de hardware disponível para utilização. O conforto é atingido pela disponibilidade de um SOFIX, com um ambiente para desenvolvimento de software ao qual a maioria dos programadores sofisticados está habituado, e pela existência de um mesmo ambiente para execução e desenvolvimento de aplicações paralelas.

Os escalonadores estáticos podem proporcionar desempenho superior aos dinâmicos, quando se conhece detalhadamente os tempos de execução das unidades de execução. No entanto para a maior parte das aplicações paralelas a obtenção deste conhecimento é impossível ou demasiadamente onerosa. Por outro lado, os escalonadores dinâmicos podem se adaptar a qualquer carga de processamento e podem apresentar um desempenho próximo do ótimo quase sempre. Assim um escalonador dinâmico é mais adequado a um ambiente mais genérico.

Como forma do MULPLIX ser um ambiente de execução e desenvolvimento confortável e genérico, a versão 0.0 do MULPLIX provê um escalonador dinâmico.

5.2 Escalonamento no MULPLIX

Visando atingir o objetivo central do escalonamento, as linhas de execução decorrentes de aplicações paralelas devem ser privilegiadas. Além disto, deve ser buscada a utilização eficiente da hierarquia de memória do MULTIPLUS, pois este fator influencia fortemente o desempenho geral do sistema.

As linhas de execução pertencentes a aplicações paralelas são privilegiadas através do tratamento diferenciado à evolução das suas prioridades e da não realização de compartilhamento de tempo na maioria dos processadores. A diferenciação acima refere-se a não alteração daquelas prioridades. As vantagens advindas desta diferenciação são a manutenção da prioridade inicial (alta) da aplicação paralela apesar da sua utilização de processadores e a economia do processamento que seria necessário para o cálculo de sua evolução. O não compartilhamento de tempo em um processador permite a execução com o mínimo de interferências de uma linha de execução; ela executará sem interrupções até que termine ou necessite de um recurso não disponível imediatamente. A existência de processadores com tempo compartilhado garante que os processadores não terão sua alocação monopolizada por aplicações paralelas, conseqüentemente permitindo a execução de processos interativos.

A utilização eficiente da hierarquia de memória do MULTIPLUS depende da exploração da localidade de memória. A localidade de memória cresce a medida que a posição de memória a ser acessada por um NP está na memória local de um NP em outro Cluster, na memória local de outro NP no mesmo Cluster, na memória local do próprio NP, ou no Cache do próprio NP. A próxima linha de execução a ser escalada pertence ao processo de maior prioridade e maximiza a localidade de memória.

Como forma de maximizar a localidade de memória, há uma fila de linhas de execução prontas por Cluster. Cada uma destas filas pode ser acessada por todos os processadores, mas um processador disponível só procura uma linha de execução pronta para executar na fila de outro Cluster se a fila do seu Cluster estiver vazia.

O escalonamento no PLURIX [10] apesar de mais flexível que o do UNIX não é adequado a exploração de paralelismo e a uma arquitetura NUMA com dezenas ou centenas de processadores, como o MULTIPLUS. Sua inadequação à exploração de paralelismo decorre do não tratamento diferenciado à execução de aplicações paralelas. A inadequação à arquitetura do MULTIPLUS advém da existência de um fila única de processos prontos e da não consideração da localidade de memória. A utilização de uma fila única de prontos em uma arquitetura com centenas de processadores pode provocar uma contenção das vias de acesso à memória e a ela própria. Não há no PLURIX o objetivo de maximizar a localidade de memória. A escolha do processador da próxima execução de um processo no PLURIX independe da localização de seus acessos à memória, ocasionando mesmo em arquiteturas UMA ("Uniform Memory Access") o desperdício do cache nas trocas de contexto.

6. ESTÁGIO ATUAL E PERSPECTIVAS FUTURAS

A versão 0.0 do MULPLIX está definida funcionalmente. A estratégia para implementação abrange duas fases. A fase inicial utilizará supermicros com o PLURIX como máquinas de desenvolvimento e abrangerá dois esforços paralelos: implementação das extensões ao PLURIX e adaptação do compilador para a linguagem "C" do PLURIX para os processadores SPARC. Na fase seguinte, após a disponibilidade de um protótipo do MULTIPLUS, o sistema será transportado e seu desenvolvimento prosseguirá neste ambiente fortemente influenciado pela realimentação decorrente da sua própria experiência de uso.

As perspectivas mais imediatas são de conclusão e apresentação pública da versão 0.0 ao final do segundo semestre de 1991. Já está em estudos a evolução do MULPLIX para melhor refletir o paralelismo da arquitetura do MULTIPLUS. Versões posteriores suportarão uma variedade de modelos de programação paralela, a nível de compiladores, bibliotecas de subrotinas ou diretamente pelo núcleo.

7. AGRADECIMENTOS

Os autores agradecem ao CNPq e à FINEP o apoio ao desenvolvimento deste projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M., "Mach: A New Kernel Foundation for UNIX Development"; Proc. of the Summer 1986 USENIX Technical Conference and Exhibition, pp.93-112.

[2] Anderson, T. E., "The Performance of Spin Lock Alternatives for Shared Memory Multiprocessors", IEEE Trans. on Parallel and Distributed Systems, January 1990, Vol 1, Number 1, pp. 6-16.

[3] Aude, J. S. et al, "MULTIPLUS: Um Multiprocessador de Alto Desempenho", Anais do X Congresso da Sociedade Brasileira de Computação, Julho de 1989, pp. 93-105.

[4] Azevedo, G. P., "Make, a Ferramenta Essencial de um SOFIX", Boletim do PLURIX, Novembro de 1987, Ano 1 Número 2.

[5] Bach, M., "The Design of the UNIX Operating System", New Jersey, Prentice-Hall, 1986.

[6] Faller, N., Azevedo, G. P., Azevedo, R. P., Barbosa, S. M. A., e Salenbauch, P., "O PLURIX Versão 2.0", Boletim do PLURIX, Agosto de 1988, Ano 2 Número 5.

[7] Faller, N. e Salenbauch, P., "PLURIX, O Sistema Operacional Multiprocessador do NCE-UFRJ: (1) Sincronização de Processos", Data News, 24 de setembro de 1985.

[8] Faller, N. e Salenbauch, P., "A Multiprocessing UNIX-like Operating System", Proc. of the Second IEEE Workshop on Workstation Operating Systems, IEEE Computer Society Press. Washington, DC, E.U.A, pp. 29-36

[9] LeBlanc, T. J., Marsh, B. D. e Scott, M. L., "Memory Management for Large-Scale Multiprocessors", Technical Report, Computer Science Department, University of Rochester, 1989.

[10] Salenbauch, P., "O escalador de processos do PLURIX", Boletim do PLURIX, Agosto de 1989, Ano 3 Número 9.

[11] Technical Committee on Operating Systems of the IEEE Computer Society, "POSIX: IEEE trial-use standard portable operating systems for computer environment", New York, Wiley-Interscience, 1986.