Simulação paralela em DSM usando variáveis compartilhadas

Marcelo Trindade Rebonatto
rebonatto@upf.tche.br
Mestrando UFRGS/UPF

Cláudio Fernando Resin Geyer
geyer@inf.ufrgs.br
UFRGS

Resumo

O conteúdo do presente texto detalha a especificação de uma proposta a um novo protocolo de simulação. Baseado nos protocolos conservadores de simulação paralela e diferente de todos os demais pelo fato de sua CI (Interface de Comunicação) utilizar variáveis compartilhadas ao invés de trocas de mensagens (MP - *Messaging Passing*). Ele foi implementado e testado sob um ambiente de estações de trabalho ligadas em rede, fazendo uso de memória compartilhada distribuída. Seu desempenho foi comparado com um protocolo tradicional implementado com uso MP, tendo ao final seus resultados discutidos.

Palavras chave: Paralelismo, Simulação de eventos, Simulação paralela

Introdução

Simulações são amplamente utilizadas para analisar o funcionamento de sistemas reais ou futuras modificações nos mesmos. Seu uso possibilita visualizar o comportamento de sistemas, sem a necessidade da realização efetiva das modificações ou a construção de, por vezes, caros protótipos. A simulação computacional é empregada com grande sucesso como elemento auxiliar na tomada de decisões, principalmente no planejamento a médio e longo prazos e em situações que envolvem custos e riscos elevados [1].

Com o passar do tempo, as novas aplicações tendem a consumir cada vez maior poder computacional, porém seus tempos de execução não podem crescer em escala semelhante. Desta forma, a simulação seqüencial pode ser insuficiente devido às restrições impostas pelo tempo de processamento. Como alternativa em poucos casos têm-se a possibilidade de melhorar os algoritmos, mas em outros, algum tipo de paralelismo pode ser necessário [2].

Para explorar o paralelismo, a carga computacional da simulação é dividida entre diversos processadores, transformando a aplicação em uma simulação paralela. Fujimoto [3] salienta que a utilização de vários processadores proporciona diversos benefícios, entre eles a redução do tempo de execução. Um menor tempo de processamento pode ser alcançado com a divisão de um processo de simulação, em vários sub-processos, podendo estes serem executados em paralelo.

Diversos protocolos de simulação paralela de propósito geral foram desenvolvidos durante as últimas duas décadas. Eles podem ser divididos em dois grandes grupos, os conservadores e os otimistas, dependendo do algoritmo empregado para preservar a correta ordem de execução dos eventos [4].

Durante a execução em paralelo de uma simulação, os processadores que a executam necessitam comunicar-se para que suas informações possam ser acessíveis aos demais. Os dados que necessitam ser trocados dizem respeito ao avanço da simulação, requisições oriundas de dependência de dados e solicitações de processamento de informações. A forma de comunicação usada tradicionalmente nos protocolos de simulação paralela é a troca de mensagens com um *timestamp* [5], não existindo dados compartilhados entre os processadores. Quase a totalidade dos protocolos de simulação paralela comunicam-se por MP.

A programação no paradigma de memória compartilhada é considerada mais simples, pois evita que o programador se preocupe com a comunicação entre os processos através de trocas

explícitas de mensagens [6]. Propor um novo protocolo de simulação, baseado em variáveis compartilhadas, facilitaria a implementação de simuladores, ficando o programador com maior tempo disponível para as questões relativas ao sistema a ser simulado.

Simulação paralela de eventos

Na simulação paralela de eventos, o sistema a ser simulado é dividido em subsistemas a serem executados de forma paralela nos processadores, dividindo também sua lista de eventos EVL (*Event List*) criando em cada unidade processadora um LP (*Logical Process*). Cada LP pode trabalhar com a evolução de seu tempo de processamento independente de outros LPs, mantendo assim seu próprio LVT (*Local Virtual Time*). O conjunto de LVTs dos LPs irá determinar o GVT (*Global Virtual Time*), que representa o avanço geral da simulação. A execução da simulação num LP pode necessitar ou alterar dados pertencentes a outros LPs [5].

Um dos principais problemas dos modelos de simulação paralela é a forte correlação entre os seus componentes. Para que a execução em paralelo de um modelo de simulação seja correto, um pré-requisito necessário que deve ser atendido é o da garantia de não ocorrência de Erros de Causalidade Local (*Local Causality Constraints* – LCC) [5, 7]).

Erros de causalidade local são provocados pelo tratamento de eventos em ordem temporal não crescente durante a simulação. Desta forma, eventos “futuros” podem, erroneamente, afetar o comportamento do modelo quando da execução de fatos “passados”. Para evitar tais erros, é necessário utilizar um protocolo de sincronização, de modo a certificar que o modelo está sendo executado corretamente com relação à ordenação temporal dos eventos [8].

Seqüências de processamento de eventos onde existe dependência entre dados devem ser executadas mantendo sua ordenação. Por outro lado, eventos que não possuam relação de causalidade podem ser executados em qualquer ordem sem que isto comprometa a veracidade dos dados oriundos do modelo [3].

Um LP possui duas funções distintas: a execução de eventos e a comunicação com outros LPs. A execução de eventos é realizada pela máquina de simulação (MS), que executa somente os eventos locais, podendo em sua execução agendar um novo evento a ser inserido em sua própria ou em outra lista. O controle do avanço local da simulação também é controlado pela MS. A comunicação é realizada pela interface de comunicação (CI), que utiliza um sistema para possibilitar a troca de informações bem como a sincronização dos LPs. A interface de comunicação, ligada à máquina de simulação, cuida ainda da propagação de efeitos causais em eventos a serem executados por LP remotos. A arquitetura básica de uma simulação paralela é mostrada na Fig.1.

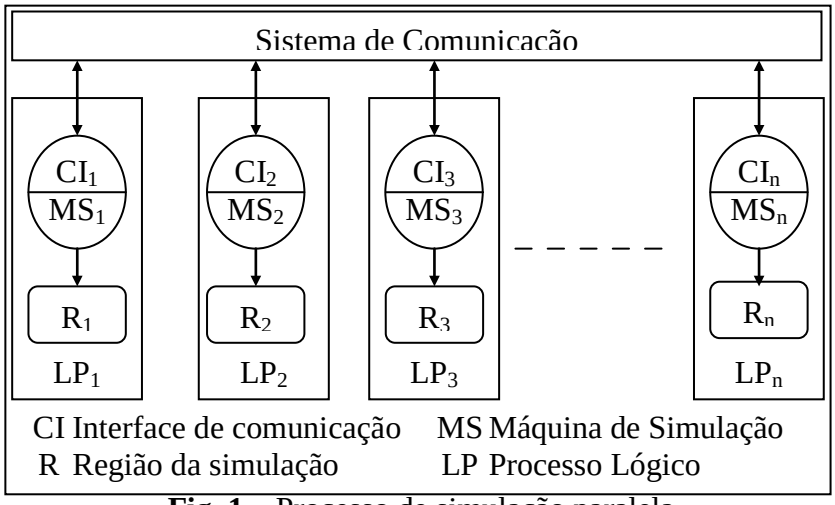

Fig. 1 – Processo de simulação paralela

Diversos protocolos de simulação paralela foram propostos com o objetivo de tratar a aderência aos LCC. O desempenho de tais protocolos é intrinsecamente dependente do ambiente, da finalidade e do tipo de modelagem utilizado no processo de simulação, não tendo sentido sua classificação. Porém, duas linhas claras de abordagem podem ser determinadas:

- protocolos conservadores: inicialmente propostos por Chandy e Misra (1979) e Bryant (1984) *apud* [3, 5], são assim denominados por se aproximarem muito dos modelos de execução tradicional de simulação. Neste tipo de protocolo, um evento somente é executado quando todos os eventos que podem afetá-lo já o foram. Assim, sua funcionalidade inibe a ocorrência de LCC.
- protocolos otimistas: buscam explorar a violação aos LCC, baseados na observação empírica de que a ocorrência de erros de causalidade tende a ser minimizada pelo próprio processo de simulação. Desta forma, é permitida a execução de eventos sem que se tenha a garantia que sua execução seja temporalmente segura, tratando os LCCs através de processos de *rollback* [9].

Os protocolos conservadores, também conhecidos como CMB devido aos seus criadores, não permitem a ocorrência de LCC. O seu princípio de funcionamento consiste em determinar quando é seguro executar um evento. Mais precisamente, se um LP possui um evento não processado E_1, com *timestamp* T_1 não existindo nenhum outro com tempo para execução inferior ao seu, e o LP puder determinar que é impossível o recebimento posterior de evento a ser executado com tempo menor que o T_1, o evento E_1 é um evento seguro [7].

Em seu funcionamento, o LP repetidamente seleciona um dentre os eventos seguros a serem processados o de menor *timestamp* e o executa. Os LPs contendo eventos a processar não seguros, devem ser bloqueados enquanto permanecerem nesta situação, podendo originar situações de *deadlock*, que devem ser tratadas e/ou evitadas [5, 7].

Os protocolos CMB diferem ainda na forma como suas variações são implementadas. Abordagens síncronas, como "janela de tempo" e "*lookahead*", além de possibilitar um melhor desempenho, podem explorar os mecanismos de sincronização em *hardware* dos ambientes multiprocessados. Por outro lado, abordagens assíncronas como, "prevenção" e "detecção e recuperação" de *deadlock*, adaptam-se melhor a ambientes de memória distribuída [3, 8].

Os protocolos otimistas, apresentados inicialmente por Jefferson e Zowizral em 1985 *apud* [3, 5], detectam e recuperam erros de causalidade, mas não impedem que os mesmos ocorram. Diferindo dos conservadores, eles não necessitam determinar eventos seguros. Eles permitem a ocorrência de erros LCC e, na ocorrência de um, chamam um procedimento para recuperar o processamento perdido. Uma das vantagens destes mecanismos é permitir ao simulador explorar o paralelismo em situações nas quais é possível a ocorrência de erros de causalidade, mas que de fato não ocorrem. Desta forma, ao tratar a ocorrência de LCC como uma exceção e não como uma regra, pode-se explorar o paralelismo intrínseco dos modelos e apresentar melhores desempenhos.

O maior problema dos protocolos otimistas é no momento da ocorrência de erros LCC. Nesta situação os LPs devem retornar a um estado seguro, anterior ao tempo atual simulado para re-executar os eventos a partir de tal estado. O problema pode tornar-se ainda maior quando o retrocesso de eventos inclui comunicações entre LPs. Neste caso, todos os LPs vão retornando até encontrarem um estado seguro, podendo sofrer de efeito dominó. O mecanismo de *rollback* necessita que os LPs registrem os estados da simulação, acumulando informações de seus eventos de forma cronológica [7].

Diversas variações são encontradas, diferindo quando as operações de *rollback* são efetuadas, do cancelamento agressivo (*Time Warp*) em direção a técnicas onde o retrocesso de ações é menos freqüente. Recursos como janelas de tempo e *lookahead* também são utilizados nas implementações.

Os protocolos otimistas sofrem restrições em relação ao uso de memória, pois seu consumo elevado é apontado como problema. Um dos principais motivos é a obrigatoriedade de periodicamente realizar o armazenamento do estado dos processos, que pode causar um *overhead* capaz de levar a uma séria degradação de desempenho. Outro problema é a preocupação com erros arbitrários que podem ocorrer, pois estes podem ser causadores de novas execuções. Tais

incorreções computacionais podem entrar em laços infinitos, necessitando de mecanismos de controle para uma intervenção a fim de interromper o sistema [5, 7].

Um dos grandes problemas das técnicas conservadoras é que não podem explorar de forma completa o paralelismo nas aplicações de simulação. A possibilidade da ocorrência de *deadlock* é outro fator a ser levado em conta na construção de uma aplicação de simulação paralela conservadora. Por outro lado, os proponentes de métodos conservadores asseguram que os protocolos otimistas são mais complexos de implementar do que os conservadores, particularmente se tentarem encontrar erros arbitrários [7].

Modelo de simulação usando variáveis compartilhadas

Os protocolos de simulação paralela usam trocas de mensagens para a comunicação entre os LPs. Neste contexto, dois tipos de mensagens podem ser encontradas: as do mecanismo de sincronização e as utilizadas para o compartilhamento de informações entre os LPs [10]. No primeiro caso, as mensagens podem ser encaradas como simples troca de informações entre os processadores. Por outro lado, as que são utilizadas para intercâmbio de dados, emulam um ambiente onde informações seriam compartilhadas por diversos processadores. No caso, o mundo real seria representado com maior naturalidade se fossem utilizadas técnicas de variáveis compartilhadas [11].

O conceito de variáveis compartilhadas em memória distribuída é muito semelhante à lógica empregada nos protocolos conservadores de simulação, onde os acessos devem ser realizados em ordem, a fim de não incorrer em erros LCC, sem a possibilidade de nova execução de ações. Algoritmos para simulação paralela utilizando variáveis compartilhadas foram propostos em [12], porém apenas conceitos gerais foram abordados, como controlador central de variáveis e técnicas para replicação, aplicados em protocolos conservadores e otimistas.

O modelo de simulação proposto neste trabalho utiliza características dos protocolos conservadores de simulação, mas apresenta modificações em pontos como gerenciamento da memória, avanço do tempo local de simulação e sincronização dos processos lógicos. Sua funcionalidade assemelha-se muito à dos protocolos conservadores com execução síncrona.

Conceitos utilizados

Sendo considerado como conservador, o modelo a ser apresentado traz os benefícios e problemas de seus antecessores. O fraco paralelismo dos protocolos conservadores, evidenciado como problema desde a sua concepção, foi amenizado com a utilização de conceitos como *lookahead* e janelas de tempo. O modelo a ser descrito faz uso destes componentes, característicos dos algoritmos de simulação paralela conservadora de execução síncrona, com o objetivo de melhorar o desempenho.

As vantagens dos protocolos CMB, como maior facilidade na implementação, baixo consumo de memória e fácil obtenção do GVT, são mantidas. A seguir serão descritas as principais características dos protocolos conservadores utilizadas:

- Princípio operacional — Não ocorrência de LCC
- Sincronização dos LP — A cada barreira, definida pela janela de tempo
- *Deadlock* — Não ocorre
- GVT — Obtido de forma direta
- *Lookahead* — Utilizado de forma constante
- Estrutura do LP — Mantém a estrutura de dois sub-sistemas: MS e CI.

O modelo de simulação com variáveis compartilhadas apresenta diversas modificações em relação ao modelo CMB inicialmente proposto. Várias não são exclusividade sua, visto que ao

longo do tempo muitas novas variações foram introduzidas nos protocolos CMB. Ele apresenta uma mudança em nível conceitual, ocasionando diversas outras no funcionamento do processo de simulação, tanto em nível de sincronização dos LPs quanto no agendamento de novos eventos.

A principal modificação do protocolo ocorre em nível conceitual, proporcionado pelo acesso direto de um LP a parte da memória local pertencente a outro LP. Uma das características de ambos os grupos de protocolos de simulação Paralela originalmente descritos é que, um LP somente poderia acessar diretamente sua memória local [3]. O acesso a memórias não locais deveria ser efetuado através de trocas de mensagens. A Fig.2a, indica como tradicionalmente é realizado o acesso a memória de outros LPs, havendo uma troca de mensagens entre os processos envolvidos. Já no caso da Fig.2b, o acesso a memória utilizado neste modelo.

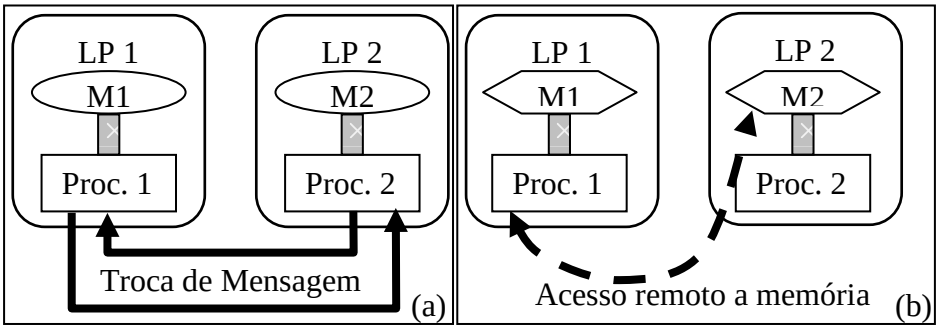

Fig. 2 – Comunicações entre 2 LPs no modelo com DSM

Cabe salientar, que nem toda a memória local dos LPs necessita ser acessada pelos demais; apenas uma parte. Havendo somente algumas regiões de memória compartilhada, todo um novo mecanismo de sincronização pode ser construído. Desta forma diversas mudanças podem ser realizadas no protocolo de simulação paralela, entre elas:

- tratamento do *deadlock*: o modelo proposto, virtualmente nunca entra em *deadlock*, visto que há a prevenção para que tal não aconteça;
- avanço do tempo local de simulação: por assemelhar-se a um protocolo de execução síncrona, o avanço do LVT, pode ocorrer de forma livre até a chegada a uma barreira de sincronização.
- mecanismo de sincronização: a sincronização é facilmente efetuada usando estruturas de dados compartilhados nos LPs.
- agendamento de novos eventos oriundos de outros LPs: é realizado quando o LP chega a uma barreira de sincronização.

Estrutura de memória

Grande parte da memória dos LPs é utilizada com a manipulação de sua EVL que não precisa ser compartilhada. O que necessita ser acessível a outros LPs são as informações relativas a sincronização dos processos. São encontrados diversos mecanismos de sincronização baseados em MP, nestes, no conteúdo das mensagens encontram-se informações relativas ao avanço do tempo local de simulação e agendamento de novos eventos nos outros LPs.

A fim de reduzir o tempo de ociosidade dos processadores, apenas o LVT de cada LP necessitaria ser compartilhado. Desta forma, quando um LP alcançasse uma barreira de sincronização, poderia “espiar” os demais LVTs e atualizar o seu. Porém, o LP não poderia seguir seu processamento sem as informações relativas aos eventos a serem criados nele por outros LPs. Assim, não apenas o LVT necessita ser compartilhado, também regiões de memória compartilhada devem ser fornecidas para o controle da criação de novos eventos oriundos de outros LPs.

Desta forma, são utilizadas estruturas de memória compartilhada para controlar o LVT e a criação de novos eventos. Porém, para o uso destas variáveis, outras de acesso local são necessárias a fim de que os mecanismos descritos possam ser corretamente implementados. A seguir um esquema das principais estruturas de dados utilizadas pelo modelo.

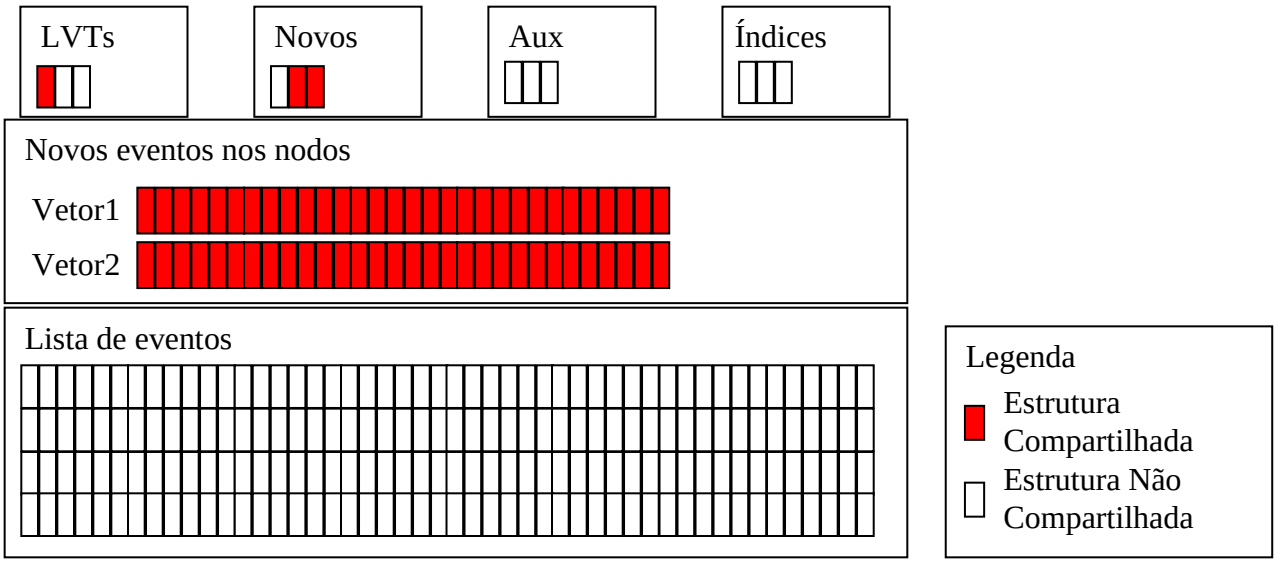

Fig. 3 – Estrutura de memória de um LP

A Fig.3 apresenta a fotografia da memória de um LP do simulador com uso de variáveis compartilhadas. Foi extraída do LP_0 de um modelo de simulação envolvendo três LPs. Assim, as primeiras estruturas estão representadas com três posições (o número de LPs é limitado somente pelo número de processadores disponíveis).

Antes de detalhar as estruturas de dados compartilhadas, deve-se descrever a forma dos compartilhamentos, juntamente com os tipos de acessos possíveis. Na Fig.3, encontram-se diversas estruturas de dados compartilhadas, algumas com compartilhamento total e outras parcial. Os tipos de acessos à memória compartilhada previstos são de leitura e escrita para o LP que contem fisicamente a região compartilhada, e apenas de leitura para os demais. Desta forma, o funcionamento ocorre com o LP responsável pela variável compartilhada fornecendo os valores que os demais LPs podem ler. A escolha de acessos remotos a memória somente de leitura foi para não onerar o modelo com acessos remotos de escrita, que necessitam de maior controle para serem realizados e consequentemente maior *overhead* de processamento.

Duas formas de compartilhamentos são permitidas: o acesso a todos os LPs e a um só. Para exemplificar a divisão toma-se por base o LVT e a quantidade de eventos a serem criados no LP_x. No primeiro caso, todos os LPs, devem enxergar o valor dos demais enquanto no segundo, somente LP_x necessita acessar remotamente a variável compartilhada.

A estrutura descrita como "**LVTs**" é um vetor de inteiros que armazena os LVTs de cada nodo (LP). Apenas a posição relativa ao número do nodo em questão necessita ser compartilhada, sendo visível a todos os LPs. No exemplo da figura acima, apenas a posição zero do vetor é compartilhada. As demais posições, armazenam o LVT de outros nodos, atualizados por acessos remotos e utilizados para o controle do avanço local da simulação. A manutenção em apenas uma estrutura de dados os LVTs da simulação, proporciona de forma facilitada o cálculo do GVT.

A estrutura descrita como "**Novos**", também é um vetor de inteiros, que armazena a quantidade de novos eventos que um LP irá criar em outro dentro de uma barreira de sincronização. Diferente de "LVTs", suas porções compartilhadas são aquelas que não possuem correspondência com o nodo atual. Cada posição deste vetor somente é acessível de forma remota pelo nodo correspondente a sua posição, ou seja, a posição de número 1 (um) somente poderá se acessada remotamente pelo LP_1.

As estruturas de dados descritas em "**Novos eventos nos nodos**" são vetores de inteiros, que trabalham em conjunto com cada uma das posições do vetor "Novos". Nelas, são guardados os dados referentes aos novos eventos. Cada "Vetor" é compartilhado por inteiro e acessível de forma remota somente pelo nodo que referencia, da mesma forma que uma posição de "Novos".

Diversas estruturas de dados com acesso local são necessárias no modelo de simulação paralela descrito. Na Fig.3 encontra-se uma grande estrutura de dados referenciada por "**Lista de Eventos**". A EVL é mantida ordenada, com o tempo de execução do evento Ev_i, sendo sempre menor ou igual ao tempo em que o evento Ev_{i+1} deverá ocorrer. Todas as operações relativas a lista de eventos são realizadas localmente sobre esta estrutura.

A estrutura "**Índices**" é auxiliar às variáveis compartilhadas "Vetor n". Nela, em cada posição, é armazenado o índice do vetor em que o próximo evento a ser agendado em outro LP deverá ser inserido. Por exemplo, se em LP_0 a variável `INDICES[1]` armazenar o valor 20, indica que o próximo evento agendado por LP_0 em LP_1 deverá ser colocado a partir da 20^a posição na estrutura "Vetor 1".

A estrutura "**Aux**" também auxiliar no uso de variáveis compartilhadas. Ela armazena em cada posição, a quantidade de eventos já lidos remotamente no nodo referente a sua posição. Sua utilização é indispensável para que os acessos a estrutura "Vetor n" possam ser realizados somente como leitura. A cada acesso remoto, um cálculo é efetuado para saber qual é a posição em "Vetor n" que contêm os eventos a serem agendados.

Funcionamento do modelo

O modelo de simulação paralela com uso de variáveis compartilhadas possui um funcionamento semelhante aos protocolos conservadores de simulação paralela de execução síncrona. Ele pode ser dividido em duas partes: execução de eventos e sincronização. A fase de execução está inserida na "Máquina de Simulação" enquanto que a de sincronização faz parte da "Interface de comunicação".

A fase de execução inicia a simulação e executa todos os eventos até a chegada a uma barreira de sincronização. Neste ponto, a parte referente à sincronização assume o controle, executando tarefas como a definição de uma nova barreira de sincronização e atualização de eventos agendados. Após serem realizadas as tarefas de sincronização, o LP volta a executar a simulação de eventos. O processo de simulação segue alternando entre as partes enquanto existirem eventos a serem simulados. Como o trabalho descreve um novo protocolo, apenas a parte relativa a interface de comunicação será descrita a seguir, a máquina de simulação varia a cada simulação realizada, apenas porém deve utilizar o conceito de janelas de tempo.

Lookahead

O modelo descrito utiliza o conceito de *lookahead* com o objetivo de diminuir o tempo de simulação. Ele é tratado como um parâmetro de entrada, sendo facilmente modificado de uma execução para outra. Dentro do modelo, é utilizado para fins de cálculo do *timestamp* de novos eventos e como fator determinante no cômputo da janela de tempo.

Como os sistemas do mundo real são diferentes em comportamento, o valor do *lookahead* varia de acordo com o sistema a ser simulado. Este valor é mantido sem alterações durante toda a execução. Uma opção em relação ao *lookahead* constante na simulação é o seu cômputo de maneira dinâmica, podendo ser diferente em vários pontos da simulação. Porém, conforme descrito em [13,14] a simulação é uma representação de um único ambiente do mundo real, podendo permanecer com um valor igual na predição do tempo para criação de novos eventos.

Ao início de uma simulação, o *lookahead* (L) do modelo é conhecido, e ele representa qual a quantia de tempo em que o LP poderá "olhar a frente" em relação a criação de novos eventos. Quando a execução de um evento determina que o mesmo criará um novo evento, o tempo em que o novo deverá ocorrer deve ser calculado. Para este cálculo, é gerado um valor aleatório dentro do intervalo [1..L]. Este valor representa o adicional a ser incluído no T_{atual} para a execução do novo evento. Porém, se um sistema possui *lookahead* L é dito que não pode receber novos eventos com

tempo inferior a T_{atual}+L, caso o valor aleatório seja diferente de L, o modelo irá agendar um novo evento com uma diferença de tempo de execução em relação ao atual menor que L. Conforme descrito, a única possibilidade de não infringir o conceito do *lookahead* do sistema seria a geração do valor aleatório igual a L, fato que ocorreria com pouca probabilidade a medida que L aumente.

Para a solução deste problema, ao valor gerado sempre é adicionado o valor de L, transformando o intervalo para a geração do valor aleatório em [0..(L-1)]. O primeiro valor adicionado a T_{atual} corresponde ao valor do *lookahead* presente no modelo, enquanto que o aleatório é o incremento do mesmo, podendo não ocorrer nenhum incremento ou no máximo, o valor de L menos 1. A subtração de uma unidade no cálculo do valor aleatório é efetuada para evitar que um novo evento seja criado com um tempo T_{atual}+(2*L), o que pode não corresponder aos sistemas simulados. A fórmula abaixo exemplifica o cálculo do *timestamp* de um novo evento.

$$T_{novo} := T_{atual} + L + random(0, (L-1))$$

A visão do cálculo pode ser exemplificada quando analisado o agendamento de um novo evento na própria EVL. Se o sistema assume que não pode receber novos eventos com tempo inferior a T_{atual}+L, não pode gerar um tempo inferior a este para um novo evento a ser criado em sua EVL. O cálculo descrito é utilizado tanto para a criação de novos eventos na própria EVL quanto em outra, pois em ambas estão sendo gerados novos eventos de um mesmo sistema simulado.

Barreiras de sincronização

O valor do *lookahead* também é utilizado para a determinação das barreiras de sincronização, delimitadoras das janelas de tempo. O modelo utiliza o valor de L para definir a primeira barreira de sincronização, possibilitando, assim, ao final do tempo em que o primeiro evento pode utilizar o *lookahead*, que uma sincronização seja efetuada para receber os novos eventos de outros LPs.

O modelo proposto utiliza janelas de tempo para execução dos eventos a fim de diminuir a comunicação entre os LPs na sincronização dos mesmos. Dentro das janelas, o avanço do LVT é livre, devendo o processo ao chegar na barreira delimitadora do final da janela, efetuar a sincronização. Neste ponto, o protocolo conservador com uso de janelas de tempo (CTW), obriga os LPs que alcançaram a barreira de sincronização a esperar pelos demais. Após, todos os LPs alcançarem a barreira, o processo segue adiante, com a nova barreira sendo delimitada.

A figura ao lado mostra o gerenciamento das barreiras no novo protocolo proposto. A barreira inicialmente definida para LP_2 era 50, e ao alcançá-la, ela foi redefinida para 80, enquanto que as demais permaneceram inalteradas. Para efetuar o avanço da barreira de sincronização de LP_2, foi utilizado o menor LVT entre os LPs. A este valor foi adicionado o valor constante do *lookahead*, possibilitando que LP_2 avance até o tempo de simulação 80. Com este avanço, LP_2 não necessita esperar pelos demais para prosseguir com a simulação. Neste Ponto os eventos agendados nos outros nodos a serem executados em LP_2 devem ser efetivados.

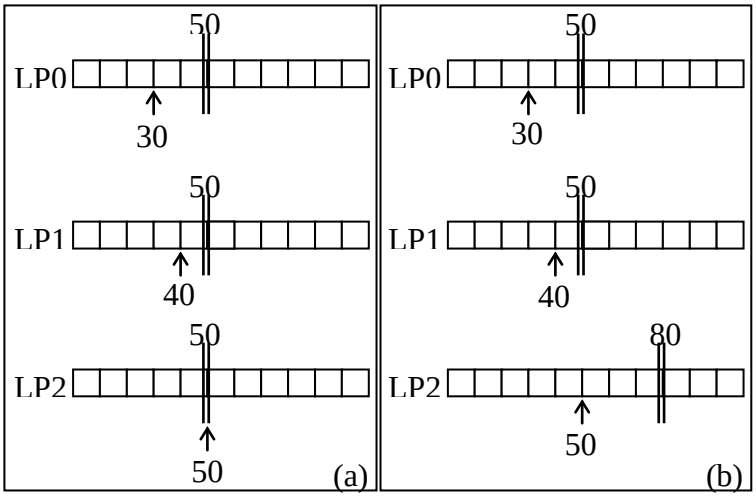

Fig. 4 – Definição de uma nova barreira

Cada LP deve controlar de forma independente sua barreira de sincronização e seu LVT. Porém, a diferença entre a menor barreira presente no sistema em relação à maior, nunca ultrapassa o valor do *lookahead*, assim o princípio da não ocorrência de erros LCC é preservado.

A prática de avançar a barreira, tomando por base o menor LVT existente no modelo, oferece um novo período em que o LP pode continuar a executar seus eventos. Saliente-se que este período nunca é igual ao *lookahead* do modelo, porém pode beneficiar o processo de simulação. O LP pode executar eventos que o fariam atrasar a execução de outros contidos nas próximas janelas de tempo, Desta forma, além de diminuir o tempo ocioso, pode contribuir de forma efetiva na redução do tempo total da simulação.

O controle dos novos eventos agendados pelo próprio LP é realizado ainda no momento da execução dos mesmos. Tais eventos são inseridos na EVL logo após a execução dos que os criaram e antes de executar o seguinte. Desta forma, como o agendamento de eventos na própria EVL funciona de forma semelhante a uma simulação seqüencial, deve-se apenas calcular o *timestamp*, encontrar seu lugar na EVL e inserí-lo em ordem. Por outro lado, o agendamento de eventos oriundos de outros LPs é efetuado a cada sincronização, pois não serão gerados eventos com *timestamp* inferior a barreira de um LP enquanto o mesmo estiver com o LVT dentro de uma janela que tenha ao seu final a referida barreira. Somente os eventos já agendados até o momento em que um LP irá realizar a sincronização serão criados; os demais o serão na próxima sincronização. Não há problemas, pois a barreira não será acrescida com um valor em que possam ocorrer erros LCC.

O processo de agendamento de eventos oriundos de outros LPs é dividido em duas etapas: a preparação e a efetivação. Na preparação, é realizado o cálculo do *timestamp* do evento, logo após o mesmo é armazenado na estrutura de memória compartilhada referente ao nodo em que ele deve ser criado. Na efetivação, os eventos propriamente ditos são obtidos e inseridos na EVL.

Uma distinção do presente modelo em relação aos baseados em trocas de mensagens, é que um LP não recebe dos outros os eventos a serem executados. Ao invés, ele os lê remotamente nos nodos correspondentes. Esta ação só é possível com o uso da memória compartilhada, fundamento em que se baseia o presente modelo. A vantagem desta abordagem é que o LP não precisa esperar pela informação, mas quando a necessita pode buscá-la diretamente em sua origem. Desta forma, o LP pode avançar seu processamento sem depender que os demais se comuniquem com ele.

Foi criado um algoritmo para a leitura das informações nos LPs. A cada sincronização um ou dois acessos às estruturas de dados compartilhadas são realizados: um para conhecer a quantidade de eventos e outro para efetivamente pegá-los. Assim, é sempre realizado um acesso à estrutura “Novos”, na posição referente ao nodo atual. Caso retorne um valor diferente de zero, um acesso é realizado à estrutura “Vetor n” para obter informações relativas aos eventos. O segundo acesso é sempre em diferentes posições do “Vetor n”. Este algoritmo de leitura deve ser realizado em todos os demais LPs presentes na simulação. A definição da posição do “Vetor n” em que será realizada a segunda leitura remota é efetuada utilizando a variável local “Aux”.

Ainda na sincronização dos processos, outra atribuição é evitar que o *deadlock* aconteça. Em uma execução normal, o modelo não entra em *deadlock*, visto que sempre existe um evento sendo executado ou a simulação chegou ao final. Os protocolos que se utilizam de duas fases, geralmente na sincronização, podem eleger um evento que pode acabar com uma situação de *deadlock*. No presente modelo isto não é necessário, visto que com o uso do menor LVT para o avanço da barreira local de sincronização, poderá ser encontrado um incremento mínimo de barreira de sincronização. Por menor que seja o avanço, proporciona novos eventos seguros para execução.

Implementação e resultados

O ambiente de execução utilizado na implementação do modelo descrito é baseado em estações de trabalho ligadas através de uma rede de conexão, pois além de ser escalável, apresenta-se como

promissor na execução de simulações [15]. O modelo foi implementado e testado sobre três máquinas SUN monoprocessadas, duas de 270MHz e outra de 300MHz, todas contendo 192Mb de RAM. A comunicação é proporcionada por um Hub FastEthernet, sendo que durante os testes elas constituíram uma LAN isolada. O uso de apenas três processadores foi devido ao equipamento disponível, porém, o programa pode ser executado utilizando um número maior de processadores.

A biblioteca *Athapascan0* [16] foi escolhida para a exploração do paralelismo na implementação do modelo. Desenvolvida no LMC-IMAG (*Laboratoire de Modélisation et Calcul - Institut d'Informatique et de Mathematiques Appliquees de Grenoble*), pelo grupo APACHE, permite a programação paralela em computadores de memória distribuída, com as características da memória compartilhada. Além da programação em SM, ela também possibilita a criação de aplicações paralelas baseadas em MP. A linguagem de programação utilizada foi C.

Para testar o desempenho do modelo proposto outros dois simuladores usando técnicas tradicionais foram desenvolvidos. Um foi desenvolvido sem a exploração do paralelismo, com seu código escrito na linguagem C sem utilização de nenhuma biblioteca. Outro, paralelo, utilizando MP foi implementado com o uso da mesma biblioteca. Um protocolo clássico de simulação paralela com janelas de tempo conservadoras foi utilizado. Ele manteve a forma de cálculo do *timestamp* para um novo evento e o *lookahead* constante, idênticos ao modelo descrito.

Na realização dos testes, os simuladores foram executados tendo como parâmetros uma lista de eventos[1] e um valor para o "*lookahead*". Os parâmetros de criação das listas de eventos e *lookahead* são de simulações reais. Na obtenção de resultados, os processos foram repetidos em séries de cinco execuções, sendo após extraída a média entre elas. Os simuladores foram executados em várias listas de eventos com diversos valores de *lookahead*. Os resultados de *speedup* e eficiência do novo simulador estão na tabela abaixo, eles foram calculados com 3 processadores.

TABELA 1 – *Speedup* e eficiência dos simuladores paralelos

Eventos	*Lookahead*	Tempo Seqüencial	Tempo Paralelo	*Speedup*	Eficiência
500 000	20	3.817,59	2.576,80	1,4815	0,4938
500 000	30	3.817,59	2.245,43	1,7002	0,5667
500 000	40	3.817,59	2.069,90	1,8443	0,6148
1 000 000	20	15.233,23	8.351,84	1,8239	0,6080
1 000 000	30	15.233,23	7.482,54	2,0358	0,6786
1 000 000	40	15.233,23	7.091,13	2,1482	0,7161
1 500 000	20	34.222,92	17.303,37	1,9778	0,6593
1 500 000	30	34.222,92	15.720,34	2,1770	0,7257
1 500 000	40	34.222,92	15.010,28	2,2800	0,7600
2 000 000	20	60.802,06	29.264,61	2,0777	0,6926
2 000 000	30	60.802,06	27.002,87	2,2517	0,7506
2 000 000	40	60.802,06	25.849,92	2,3521	0,7840

Fonte: Autor.

Os valores de *speedup*, variam de um mínimo 1,4815 até o máximo de 2,3521. Analisados dentro de uma mesma lista, eles crescem na medida que o valor do *lookahead* utilizado sobe. Na medida que a quantidade de eventos aumenta, o *speedup* de todos os processos também aumenta, levando os processos com 2 000 000 de eventos a terem os maiores valores. A eficiência, como é diretamente dependente do *speedup*, segue a variação do primeiro. Iniciando com um mínimo de 49,38% e crescendo até 78,40%, demostra que os processos paralelos gastam quantia significativa de tempo para operações de comunicação e sincronização. Como poderia ser esperado, os processos menos eficientes são aqueles com um menor número de eventos.

[1] As listas não são de nenhuma simulação real, foram utilizadas apenas para avaliar o desempenho dos simuladores

Uma comparação direta entre os resultados dos simuladores pode ser visualizado na Fig.5. Ela identifica a variação do *lookahead* nas listas com os diversos simuladores implementados. O simulador com uso de memória compartilhada é identificado por “Par_SM” enquanto o que usa troca de mensagens é identificado por “Par_MP”.

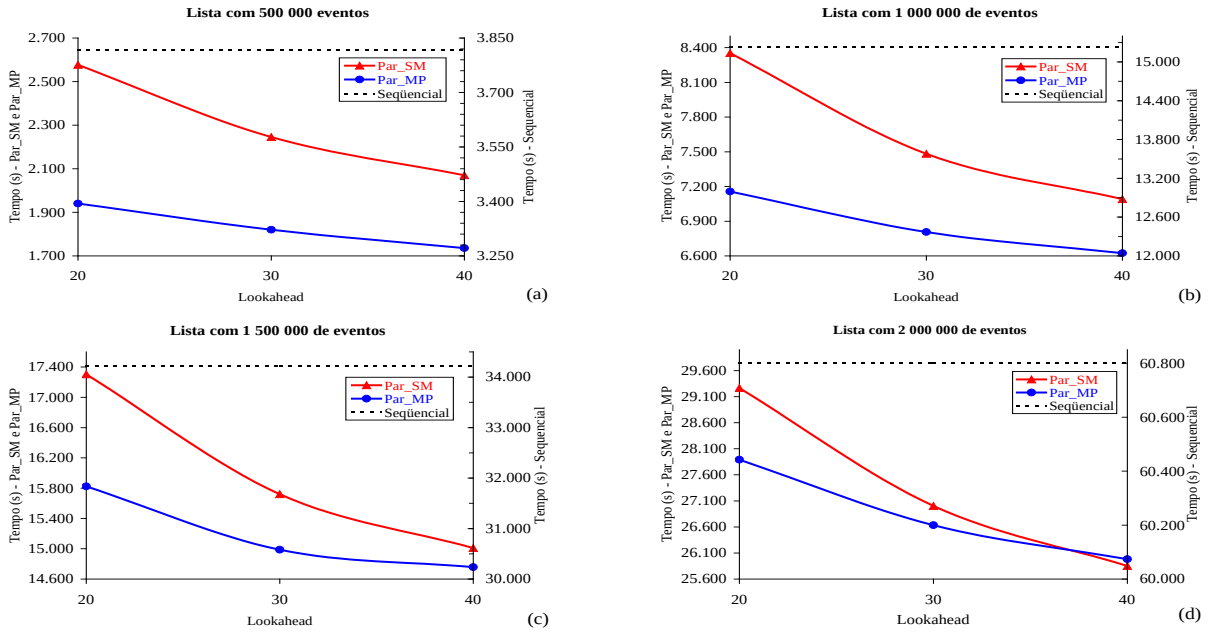

Fig. 5 – Resultados entre simuladores com diferentes *lookahead*

A variação do *lookahead* origina variações no tempo de execução dos simuladores paralelos. A Fig.5a, retrata as variações ocorridas em execuções de 500 000 eventos, tendo o simulador baseado em trocas de mensagens um menor tempo de execução para todos os valores de *lookahead* analisados. O mesmo ocorre com a lista de 1 000 000 de eventos, representada na Fig.5b, assim como na lista de 1 500 000, demonstrada na Fig.5c. Uma diferença entre as figuras citadas é que quanto mais cresce o número de eventos e o *lookahead*, gradualmente se aproximam os tempos de execução dos simuladores paralelos. A aproximação entre os valores, culmina na Fig.5d, que demostra a evolução dos processos de simulação em listas com 2 000 000 de eventos. Mantendo a tendência da listas com menos eventos a diferença entre os valores de tempo de execução ficam cada vez menores, sendo que na execução com *lookahead* 40, o simulador baseado em variáveis compartilhadas obtém um desempenho melhor em relação ao simulador com MP.

A figura ao lado, retrata a execução dos três simuladores em 500 000 até 2 000 000 de eventos com *lookahead* 40. Nota-se um desempenho semelhante dos simuladores paralelos, com leve vantagens do implementado com MP nas listas e menores e também pequena vantagem do que utiliza variáveis compartilhadas na lista com 2 000 000 de eventos.

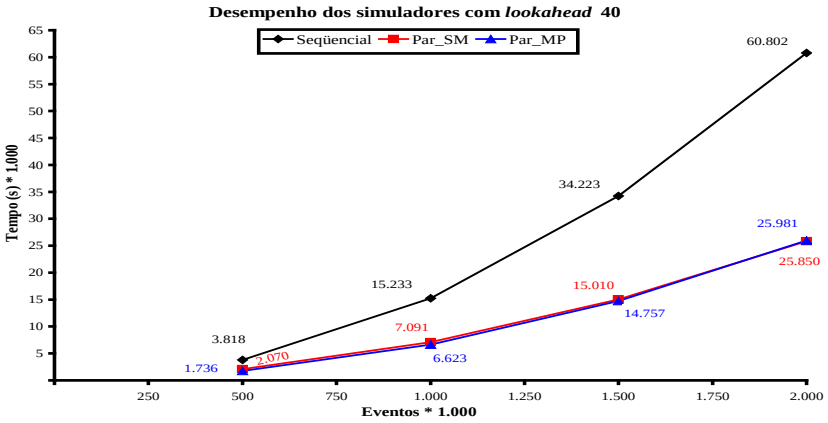

Fig. 6 – Resultados entre simuladores com lookhaead 40

Conclusões

O presente texto especificou uma proposta a um novo protocolo de simulação paralela, fazendo uso de variáveis compartilhadas. Além de uma implementação facilitada, propiciada pelo uso de memória compartilhada ao invés de trocas de mensagens, oferece a possibilidade de melhor ocupar o tempo ocioso dos processadores. Nos testes realizados, o novo protocolo obteve um desempenho semelhante ao protocolo tradicional utilizando trocas de mensagens, provando que é viável a implementação de simulação paralela de eventos em memória distribuída usando DSM.

Bibliografia

[1] SHANNON, Robert E. Introduction to Simulation. In: WINTER SIMULATION CONFERENCE, 1992, Arlington. **Proceedings...** San Diego: SCS, 1992. 1410p. p.65-73.

[2] IKONEN, Jouni; PORRAS, Jari. Load Balancing in conservative simulation. In: EUROSIM, 11th, 1999, Erlangen. **Proceedings...** Ghent: SCS, ASIM, 1999. 744p. p.250-257.

[3] FUJIMOTO, Richard M. **Parallel and Distributed Simulation Systems**. New York: John Wiley & Sons, 2000. 300p.

[4] PORRAS, Jari *et al*. Reducing the Number of Logical Channels in the Chandy-Misra Algorithm. In: EUROPEAN SIMULATION SYMPOSIUM, 9th, 1997, Passau. **Simulation in Industry** Ghent: SCS, 1997, 755p. p.252-256.

[5] FERSCHA, Alois. Parallel and Distributed Simulation of Discrete Event Systems. In: ZOMAYA, Albert Y. H. **Parallel and Distributed Computing Handbook.** New York: McGraw-Hill, 1996. 1199p. chap.35, p.1003-1041.

[6] ARAUJO, Edvar Bergmann. **Um estudo sobre Memória Compartilhada Distribuída**: Trabalho Individual. Porto Alegre: PPGC/UFRGS, 1999. 59p. (TI-868).

[7] FUJIMOTO, Richard M. Parallel Discrete Event Simulation. **Communications of the ACM,** New York, v. 33, n. 10, p. 31-53, Oct. 1990.

[8] VACCARO, Guilherme Luis Roehe. **Simulacao Paralela e Distribuida com vistas ao co-Design**: Trabalho Individual. Porto Alegre: CPGCC/UFRGS, 1999, 50p. (TI-811).

[9] MUSSI, Philippe. **Parallel Discrete Event Simulation**. Natal: SBAC'99, 1999. Internation School on Advanced Algorithmic Techniques for Parallel Computation with Aplications. Lectures Notes.

[10] REBONATTO, Marcelo Trindade. Simulação paralela de eventos discretos em DSM. In: SEMANA ACADEMICA DO PPGC, 4., Porto Alegre, 1999. **Anais...** Porto Alegre: PPGC/UFRGS, 1999, 391p. p.87-90.

[11] HIRATA C.; KRAMER J. On the Optimisation of Shared Variables in Time Warp. In: EUROPEAN SIMULATION SYMPOSIUM, 9th, 1997, Passau. **Simulation in Industry** Ghent: SCS, 1997, 755p. p.264-268.

[12] MEHL, Horst; HAMMES, Stefan. Shared Variables in Distributed Simulation. In: PARALLEL AND DISTRIBUTED SIMULATION, 7th, 1993, San Diego. **Proceedings...**. San Diego: ACM/SCS/IEEE, 1993, 168p. p.68-75.

[13] PORRAS, Jari; IKONEN, Jouni; HARJU; Jarmo. Applying a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 12th, 1998, Alberta **Proceedings...** Los Alamitos: IEEE Computer Society, 1998, 197p. p.188-195.

[14] NETSA, Alexandre; KHALIFA, Nabil Ben. Static and dynamic lookahead computation for conservative distributed simulation of Timed Petri nets. In: EUROPEAN SIMULATION SYMPOSIUM, 1998, Nottingham. **Simulation Technology: Science and Art** Ghent: SCS, 1998. 766p. p.195-199.

[15] PHAM, CongDuc. High performance clusters: A promising environment for parallel discrete event simulation. In: PDPTA'99, 1999, Las Vegas. **Proceedings...** Las Vegas:, 1999.

[16] BRIAT, Jacques; GINZBURG, Ilan; PASIN, Marcelo. **Athapascan-0 Reference Manual**.

Grenoble: LMC-IMAC, 1998. 82p. Disponível por WWW em http://www-apache.imag.fr/software/ath0/manuals.html (19 Mar. 1999)