# Um Simulador Distribuído Baseado no Paradigma Espaço-Temporal

Nahri B. Moreano[1]

Valmir C. Barbosa[2]

## Resumo

Este artigo descreve o projeto, a implementação e uma avaliação experimental de um simulador distribuído de eventos discretos baseado no paradigma espaço-temporal de Chandy e Sherman. O ponto central do projeto do simulador foi a total separação das funções pertinentes ao simulador daquelas pertinentes à aplicação sendo simulada, permitindo assim o tratamento de uma vasta classe de problemas. A implementação foi realizada em Occam2 sobre um hipercubo de *transputers* com oito processadores. Uma avaliação experimental foi feita sobre a simulação de colisões de partículas em duas dimensões, um problema notoriamente difícil em termos de paralelização, e que ainda não havia sido tratado pelo paradigma espaço-temporal. Os resultados indicam valores de *speedup* compatíveis com os que têm sido obtidos para este problema com outros paradigmas.

## Abstract

In this paper we describe the design, the implementation, and an experimental evaluation of a discrete-event distributed simulator based on Chandy and Sherman's space-time paradigm. The key point in the design of the simulator has been the complete separation between simulator-related functions and those pertaining to the particular application being simulated, thereby allowing the treatment of a vast class of problems. The simulator was implemented in Occam2 on an eight-node transputer hypercube. An experimental evaluation was carried out on the simulation of colliding particles in two dimensions, a notoriously difficult problem in terms of parallel processing amenability, and which had not yet been approached via the space-time paradigm. Results indicate speedups comparable to what has been obtained for this problem under different paradigms.

[1]Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Caixa Postal 68511, 21945-970 Rio de Janeiro - RJ.

[2]Centro Científico Rio, IBM, Caixa Postal 4624, 20001-970 Rio de Janeiro - RJ.

## 1 Introdução

Este trabalho descreve o projeto e a implementação de um algoritmo de simulação distribuída de eventos discretos, baseado no modelo espaço-temporal proposto por Chandy e Sherman em [CS89]. O desempenho deste algoritmo é avaliado e resultados são apresentados.

Diversos sistemas físicos possuem modelagens matemáticas cuja solução analítica é desconhecida ou muito custosa. A simulação por computador é uma abordagem alternativa que viabiliza a análise de tais sistemas.

Na simulação de eventos discretos, o sistema físico é modelado como um conjunto de subsistemas, denominados processos físicos ($PF$s), que interagem entre si durante a simulação. O simulador é construído como um conjunto de processos lógicos ($PL$s), cada um correspondendo a um processo físico. As interações entre $PF$s são modeladas como eventos enviados entre os $PL$s correspondentes, estando a cada evento associado um instante de ocorrência.

A simulação de alguns sistemas físicos mais complexos consome uma quantidade enorme de tempo de processamento em computadores seqüenciais. A estrutura do problema de simulação descrita anteriormente — um sistema físico dividido em vários $PF$s que interagem entre si — é intrinsecamente paralela, o que sugere a utilização de computadores paralelos, com o objetivo de reduzir o tempo de processamento destas simulações. A simulação distribuída de eventos discretos refere-se à execução de um único programa de simulação de eventos discretos em um computador paralelo de memória e controle distribuídos.

As relações de causalidade do sistema físico determinam uma relação de dependência entre alguns eventos do sistema. Para que uma simulação reproduza o comportamento do sistema físico de forma correta, nenhum evento pode ser processado antes que todos os eventos dos quais ele depende já tenham sido processados. Caso contrário, ocorrem erros de causalidade.

O simulador descrito neste trabalho emprega um mecanismo de simulação otimista, ou seja, não evita a ocorrência de erros de causalidade, pois permite que a simulação prossiga sem ter certeza de que os eventos processados são seguros, isto é, sem saber se todos os eventos que podiam afetar um evento a ser executado já foram processados. Quando ocorre um erro de causalidade, o mecanismo realiza a detecção do erro e se recupera dele. Um erro de causalidade é detectado sempre que um $PL$ recebe uma mensagem atrasada, isto é, com tempo menor que o tempo da última mensagem processada. Para recuperar o erro, esse $PL$ tem que cancelar todos os efeitos da computação prematura. Este mecanismo é chamado de *rollback*. Ao contrário dos modelos otimistas, os modelos conservadores evitam completamente a possibilidade de ocorrer algum erro de causalidade. Para isso, baseiam-se em alguma estratégia para determinar quando é seguro processar um evento. $PL$s que não contêm nenhum evento seguro ficam bloqueados, bloqueios estes que podem levar a situações de *deadlock* se as precauções adequadas não são tomadas. Experimentos têm mostrado que os algoritmos conservadores só apresentam bons resultados se a aplicação permite que se explore *lookahead* [F90].

Uma inovação do modelo espaço-temporal em relação aos outros modelos otimistas propostos até então, como por exemplo o mecanismo de *Time Warp* baseado no paradigma de tempo virtual

e apresentado por Jefferson em [J85], é permitir que se explore paralelismo tanto pela decomposição espacial quanto pela decomposição temporal da aplicação. Na decomposição espacial, a aplicação é dividida em objetos, no domínio do espaço, onde cada objeto representa uma parte do sistema físico ao longo de toda a duração da simulação.

As simulações também podem ser decompostas temporalmente. Cada objeto da simulação pode ser dividido em fases que são responsáveis pela simulação do objeto em diferentes intervalos de tempo da simulação, e essas fases podem ser executadas em paralelo. Esta decomposição temporal pode possibilitar um melhor desempenho, por extrair mais paralelismo da aplicação e por permitir que seja feito um melhor baleanceamento estático de carga.

O algoritmo desenvolvido foi implementado em um computador paralelo de memória distribuída, com os processadores interconectados por canais segundo a topologia de um hipercubo. O algoritmo foi projetado de forma a isolar da aplicação todos os detalhes do modelo de simulação. A aplicação fica responsável exclusivamente pela modelagem do sistema físico, sem se ater aos mecanismos que implementam o modelo otimista. Tal separação possibilita que as aplicações sejam modeladas e desenvolvidas de forma independente do simulador, e que sejam simulados diversos sistemas físicos diferentes, sem a necessidade de alterações no simulador. O modelo espaço-temporal aplica-se à simulação de sistemas físicos que atendam à propriedade de preditibilidade descrita em [M86].

Para avaliar o desempenho do simulador foi desenvolvida uma aplicação que modela um sistema físico de colisões de partículas. Este problema é de difícil tratamento e ainda não havia sido tratado através do modelo espaço-temporal. Os resultados obtidos foram satisfatórios, tendo sido compatíveis com os resultados obtidos para o mesmo problema através do uso de outros paradigmas de simulação [H89].

O modelo espaço-temporal é descrito na seção a seguir, enquanto que a seção 3 apresenta o simulador desenvolvido. Os resultados e as conclusões são apresentados nas seções 4 e 5, respectivamente.

## 2 O modelo espaço-temporal

Nesta seção o paradigma de simulação espaço-temporal é descrito. Este modelo foi apresentado em [CS89] e mais tarde formulado com mais precisão em [BCL91].

O sistema físico que se deseja simular é modelado por um conjunto de $PF$s, e uma interação entre dois $PF$s, por exemplo $PF_i$ e $PF_j$, é modelada por uma mensagem enviada entre $PF_i$ e $PF_j$. Deseja-se simular o sistema físico no intervalo de tempo $(0, H]$, onde $H$ é o horizonte final da simulação.

Este sistema físico pode ser representado por um diagrama onde a dimensão horizontal representa o espaço (isto é, diferentes $PF$s) e a dimensão vertical representa o tempo. Existe no diagrama uma linha vertical de tempo para cada $PF$, como na figura 1.

Este diagrama de linhas de tempo é chamado de diagrama espaço-temporal, com $PF$s ocu-

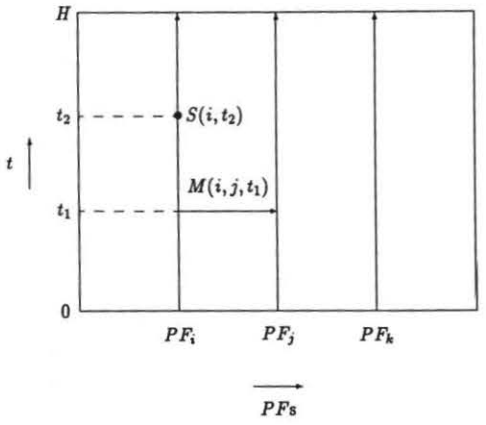


Figura 1: Diagrama espaço-temporal

pando a dimensão espaço. Cada ponto do diagrama representa uma porção do estado do sistema físico em um determinado instante. Um mecanismo de simulação é um método de preencher este diagrama, isto é, de determinar o estado de todos os $PF$s para todos os instantes em $(0, H]$. (Voltaremos a esta figura em breve para explicar o restante da notação nela contida.)

O diagrama espaço-temporal pode ser dividido em regiões de número e forma arbitrários, como na figura 2. Cada região representa uma porção do estado do sistema físico em um intervalo de tempo, isto é, representa a porção do sistema físico relativa aos $PF$s que estão incluídos nela, para o intervalo de tempo em que cada $PF$ está nela contido. Uma região pode incluir mais de um $PF$ e um $PF$ pode estar em mais de uma região. Dois $PF$s que interagem entre si devem estar na mesma região ou em regiões vizinhas.

A idéia básica que o modelo espaço-temporal propõe para preencher o diagrama é dividi-lo em regiões e determinar o comportamento de uma região dados os comportamentos das regiões vizinhas a ela, isto é, das regiões que fazem fronteira com ela. Assim, existe uma dependência cíclica, pois o comportamento de uma região $r_0$ depende do comportamento de uma região vizinha $r_1$, e o comportamento de $r_1$ depende do comportamento de $r_0$. Esta ciclicidade é resolvida utilizando um algoritmo de relaxação.

Seja $S(i,t)$ o estado local de $PF_i$ no instante $t$ e seja $M(i,j,t)$ a mensagem enviada por $PF_i$ para $PF_j$ no instante $t$ (figura 1). $M(i,j,t)$ é uma saída de $PF_i$ e uma entrada de $PF_j$ no instante $t$. Dado o ponto $(i,t)$ na altura $t$ na linha de tempo de $PF_i$ no diagrama espaço-temporal, a mensagem $M(i,j,t)$ é representada por uma linha horizontal direcionada do ponto $(i,t)$ para o ponto $(j,t)$. O ponto $(i,t)$ representa o estado $S(i,t)$ de $PF_i$ no instante $t$.

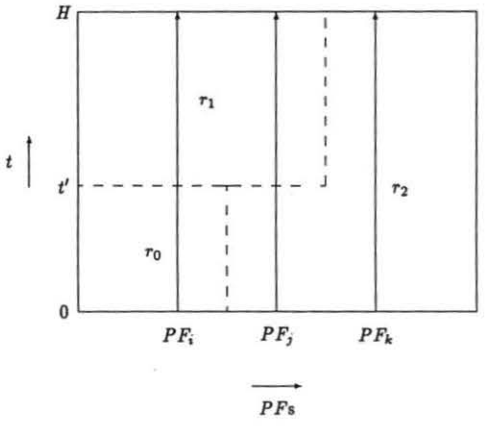


Figura 2: Divisão em regiões

$M(*,i,t)$ e $M(i,*,t)$ denotam, respectivamente, todas as entradas e saídas de $PF_i$ no instante $t$. Para quaisquer instantes $u$ e $v$ tais que $u < v$, seja $M(i,j,(u,v])$ o conjunto das mensagens $M(i,j,t)$ para todo $t$ no intervalo $(u,v]$. $S(i,(u,v])$ e $S(*,(u,v])$ possuem significados análogos.

Os estados e as saídas de um $PF$ em um intervalo de tempo são funções de seu estado no início do intervalo e de suas entradas durante o intervalo. Assim, o comportamento de um $PF_i$ é definido por

$$S(i,(u,v]) = f_i(S(i,u), M(*,i,(u,v])) \quad (1)$$

e

$$M(i,*,(u,v]) = g_i(S(i,u), M(*,i,(u,v])), \quad (2)$$

onde $f_i$ é uma função de atualização de estados e $g_i$ é uma função de determinação de mensagens a serem enviadas por $PF_i$.

O problema de simulação consiste em determinar os estados locais $S(*,(0,H])$, de todos os $PF$s para todos os instantes, e todas as mensagens $M(*,*,(0,H])$, entre todos os $PF$s e para todos os instantes, que satisfaçam as equações 1 e 2.

Uma região $r$ do diagrama espaço-temporal é um conjunto de pontos $(i,j)$ do diagrama. As entradas de uma região $r$ do diagrama espaço-temporal são :

- $M(i,j,t)$, se o ponto $(i,t)$ não está em $r$ e o ponto $(j,t)$ está em $r$; e
- $S(i,t)$, se o ponto $(i,t)$ não está em $r$ e o ponto $(i,t_+)$ está em $r$, onde $t_+$ é o exato instante

depois de $t$ [3].

As saídas da região $r$ são :

- $M(i,j,t)$, se o ponto $(i,t)$ está em $r$ e o ponto $(j,t)$ não está em $r$; e
- $S(i,t)$, se o ponto $(i,t)$ está em $r$ e o ponto $(i,t_+)$ não está em $r$, onde $t_+$ é o exato instante depois de $t$.

A cada região $r$ do diagrama espaço-temporal é associado um processo $P_r$ responsável por simular a porção do sistema físico que a região $r$ representa, isto é, por determinar o estado do sistema físico nos pontos do diagrama espaço-temporal pertencentes àquela região. Assim, $P_r$ simula o comportamento de cada $PF$ incluído em $r$, para o intervalo de tempo em que cada $PF$ está incluído em $r$. Por exemplo, na figura 2 o processo associado à região $r_2$ simula $PF_j$ no intervalo $(0,t']$ e $PF_k$ no intervalo $(0,H]$.

Pelo algoritmo de relaxação, cada processo $P_r$ determina o comportamento da sua região $r$, isto é, determina os estados $S(i,t)$ e as mensagens em $M(i,*,t)$, para todos pontos $(i,t)$ de $r$, utilizando estimativas sobre o comportamento das regiões vizinhas a $r$, de forma a satisfazer as equações 1 e 2. $P_r$ envia então estimativas sobre o comportamento de $r$ para os processos responsáveis pelas regiões vizinhas a $r$, ou seja, envia as saídas $M$ e $S$ de $r$.

Quando $P_r$ recebe novas informações sobre o comportamento de alguma região vizinha, isto é, recebe alguma entrada $M$ ou $S$ para $r$ (que são saídas da região vizinha), ele atualiza sua estimativa sobre o comportamento daquela região e redetermina o comportamento de $r$, sempre seguindo as equações 1 e 2. Se o seu comportamento foi alterado, $P_r$ envia então estimativas mais atuais das suas saídas para os processos responsáveis pelas regiões vizinhas.

Inicialmente, o valor inicial de $M(i,j,t)$ é arbitrário, para todo $PF_i$, $PF_j$ e $t$. O valor inicial de $S(i,0)$, para todo $PF_i$, é o estado inicial correto dado pelo sistema físico, enquanto que para $t > 0$ $S(i,t)$ é arbitrário.

À medida que a simulação progride, as estimativas vão se corrigindo. O algoritmo termina quando um ponto fixo é alcançado, isto é, quando o estado permanece inalterado ao evoluir a computação. Quando o ponto fixo é atingido, as estimativas das saídas de todas regiões são corretas para as estimativas dadas para as suas entradas (isto é, as equações 1 e 2 valem), para todas as regiões do diagrama espaço-temporal.

Como um $P_r$ só envia suas saídas para os processos responsáveis pelas regiões vizinhas a $r$ se o comportamento de $r$ foi alterado, quando o ponto fixo é atingido não existem mais mensagens em trânsito no sistema. Além disso, todos os $P_r$s estão ociosos, isto é, não possuem novas entradas para processar.

Em [BCL91] é apresentada a prova de corretude deste algoritmo. O algoritmo apresentado é extremamente não-determinístico, de modo que é possível derivar diversos algoritmos específicos

[3] Esta definição de $t_+$ é imprecisa quando se considera que o tempo é uma grandeza contínua. Todavia, na prática este tempo é discretizado, e então a definição se aplica.

a partir dele. A divisão em regiões de forma arbitrária permite que a aplicação seja decomposta tanto espacial quanto temporalmente, conforme já observamos.

## 3 O simulador

O simulador foi desenvolvido utilizando a linguagem de programação Occam2 no computador paralelo de memória distribuída NCP-I desenvolvido na COPPE/UFRJ [ACSC91]. Este computador possui oito nós de processamento, onde cada nó é um processador T800, e estes nós estão interconectados por canais de comunicação em uma topologia de hipercubo.

O simulador é replicado em todos os processadores da máquina. A cada processador é alocado um processo $P_r$ responsável pela simulação da região $r$ do diagrama espaço-temporal. Cada $P_r$ é composto pelos $PL$s correspondentes aos $PF$s incluídos na região $r$ (cada $PL$ pode então, em princípio, estar presente em mais de um $P_r$).

O modelo espaço-temporal é implementado no simulador, de forma isolada da aplicação, permitindo que os processos que simulam as regiões tratem exclusivamente da simulação do sistema físico.

O simulador é composto de duas camadas de programa, uma de mais baixo nível, de comunicação, que cuida da troca de mensagens entre dois processadores, e outra que realmente implementa o algoritmo de simulação. A camada de comunicação implementa os mecanismos de armazenamento e roteamento de mensagens necessários em um sistema distribuído, utilizando um processador virtual de comunicação [D90]. A segunda camada, por simplicidade chamada de simulador, implementa os mecanismos necessários para a realização da simulação otimista, que são escalonamento de eventos, gravação de estados, *rollback*, "preempção", detecção de convergência e coleta de fósseis[4].

Para cada $PL$ da região sob sua responsabilidade, o simulador possui uma lista de eventos destinados àquele $PL$, ordenados de forma não-decrescente de tempo. Sempre que não há nenhum $PL$ executando na sua região e existem eventos a serem processados, o simulador seleciona para processamento o evento de menor tempo, dentre todos os eventos ainda não processados de todos os $PL$s da região sob sua responsabilidade. Este evento é repassado para o $PL$ correspondente, que então realiza a sua simulação.

Um $PL$ executa, continuamente até o fim da simulação, um laço em que espera receber algum evento do simulador, trata este evento, isto é, simula a interação no sistema físico correspondente ao recebimento do evento, causando alterações nas variáveis de estado, e produz um ou mais eventos para outros $PL$s, modelando assim as relações de causalidade do sistema físico.

Os eventos produzidos por um $PL$ são repassados para o simulador, que se encarrega de encaminhá-los para o destino correto. Se o $PL$ destino do evento faz parte da mesma região alocada a este nó de processamento, o simulador simplesmente insere o evento na lista de eventos deste $PL$ destino. Caso contrário, tendo sido o $PL$ destino do evento alocado a outro nó, o simulador envia o

[4]Este termo vem do inglês *fossil collection*, um jargão da área que se refere à recuperação de espaço de memória através da liberação de estruturas que não são mais necessárias, sendo portanto um processo distinto do que se costuma chamar de *garbage collection*.

evento através da camada de comunicação para o simulador responsável pelo $PL$ destino, residente neste outro nó.

Sempre que recebe um evento através da camada de comunicação, vindo de um $PL$ alocado a outro processador, o simulador insere este evento na lista de eventos correspondente ao $PL$ destino do evento.

Por ser um algoritmo otimista, o simulador processa eventos sem ter certeza de que esses eventos são seguros e assim podem ocorrer erros de causalidade. Quando o simulador recebe um evento $e$ com tempo $t$ vindo de um $PL$ em outro processador destinado a um $PL_i$ sob sua responsabilidade, e o último evento que $PL_i$ recebeu tinha tempo maior que $t$, é detectado um erro de causalidade. Para recuperar o erro, o simulador cancela todos os efeitos da computação incorreta, isto é, os efeitos de todos os eventos que foram processados prematuramente (que são os eventos com tempo maior que $t$ processados por $PL_i$). Depois de cancelar os efeitos da computação incorreta, o simulador faz $PL_i$ retornar ao ponto correto onde ele possa receber o evento $e$ para processar na ordem correta. Este mecanismo é chamado de *rollback* e o evento que o causa é chamado de *straggler*. Como os eventos dos $PLs$ sob a responsabilidade do simulador são processados em ordem não-decrescente de tempo, apenas eventos que venham de $PLs$ em outros nós podem causar *rollback*.

A execução de um evento pode causar duas coisas que precisam ser desfeitas quando ocorre um *rollback*. O evento pode modificar o estado do $PL$ que o processou e pode enviar um ou mais eventos para outros $PLs$. Para desfazer a alteração no estado do $PL$, o simulador utiliza um mecanismo de gravação dos estados dos $PLs$. Periodicamente, as variáveis de estado de um $PL$, que representam o estado do $PF$ correspondente em um determinado instante, são salvas. Estes estados salvos são inseridos em uma lista de estados também ordenada em forma não-decrescente de tempo. Quando o $PL$ sofre *rollback* por receber um *straggler* para o tempo $t$, o simulador restaura um estado antigo do $PL$ da sua lista de estados, mais precisamente, o último estado salvo com tempo menor que $t$.

O simulador cancela o envio de eventos para outros $PLs$ enviando uma mensagem de controle para os simuladores responsáveis pelas regiões vizinhas à região $r$ que engloba o $PL$ que sofreu *rollback*. A função desta mensagem, chamada $inf\_rb$, é informar os simuladores das regiões vizinhas que a região $r$ sofreu *rollback* para um determinado instante $t$ (que consta de um campo da mensagem), e que portanto todos os eventos que essas regiões receberam de $r$ com tempo maior que $t$ eram estimativas prematuras. Como pelo modelo espaço-temporal o comportamento de uma região depende de e afeta apenas o comportamento das regiões vizinhas a ela, basta que o simulador envie esta mensagem de controle para os simuladores das regiões vizinhas.

O *rollback* em um ambiente distribuído é custoso pois o processo lógico que recebe um *straggler* pode ter enviado vários eventos para outros $PLs$, causando efeitos colaterais neles e levando-os a enviar ainda mais eventos para mais outros $PLs$, e assim por diante. Alguns desses eventos podem inclusive estar fisicamente em trânsito e portanto fora do controle do sistema por um período de tempo arbitrário. Todas esses eventos porém, em trânsito ou não, e independentemente de quão longas sejam as cadeias causais envolvidas, devem ser efetivamente cancelados e seus efeitos, se houver, revertidos.

Assim, quando o simulador recebe uma mensagem $inf\_rb$ com o tempo $t$, se ele já havia processado eventos com tempo maior que $t$, esta mensagem vai cancelar das listas de eventos dos $PL$s ali alocados todos os eventos com tempo maior que $t$ oriundos da região que enviou a mensagem, e causar *rollback* na região sob sua responsabilidade. O tratamento deste *rollback* é o mesmo de quando o simulador recebe um *straggler*, propagando desta forma o *rollback* para todas as regiões afetadas pela computação incorreta. Se nenhum evento com tempo maior que $t$ foi processado, a mensagem $inf\_rb$ não tem este efeito de *rollback*. Repetindo recursivamente este procedimento, todos os efeitos da computação incorreta são cancelados. Com este mecanismo a simulação sempre progride, pois no pior caso todas as regiões fazem *rollback* para o tempo $t$ (que é o tempo do evento que iniciou o *rollback*).

De maneira diferente do mecanismo *Time Warp*, que cancela os eventos enviados prematuramente um a um de forma explícita, precisando para isso enviar um anti-evento para cada um deles, o mecanismo implementado envia apenas uma mensagem para cada uma das regiões vizinhas. Com isso há uma grande redução no volume do fluxo de mensagens, mas exige-se que os canais de comunicação respeitem a ordem FIFO de envio das mensagens.

A freqüência com que o estado de um $PL$ é salvo pode ser variada, podendo ser a cada evento processado pelo $PL$ ou adotando como espaçamento um número maior de eventos processados. A escolha desta freqüência é uma questão de compromisso, pois quando os estados são salvos muito freqüentemente, torna-se necessária a disponibilidade de bastante memória. Por outro lado, se a distância entre dois estados salvos é muito grande, isto é, muitos eventos foram processados por este $PL$ desde a última vez em que ele teve um estado seu gravado, quando ocorre um *rollback* o $PL$ pode precisar voltar atrás mais do que o necessário inicialmente para restaurar um estado antigo.

Quando o simulador recebe um evento atrasado para o tempo $t$ para um $PL$ sob sua responsabilidade, é possível que este $PL$ esteja processando um outro evento com tempo maior que $t$. Neste caso, não faz sentido permitir que este $PL$ prossiga com o processamento deste outro evento, pois já é sabido que este processamento será desfeito. Sempre que essa situação ocorre, o simulador utiliza um mecanismo de "preempção" para interromper o $PL$ e suspender o processamento daquele evento. Isto foi implementado com algumas alterações no mecanismo de escalonamento do processador T800, que oferece dois níveis de prioridade de processos. O simulador foi implementado como um processo de alta prioridade enquanto que os $PL$s são processos de baixa prioridade, e sempre que necessário o simulador altera a lista de processos de baixa prioridade do processador.

Uma outra tarefa do simulador é determinar o quanto a simulação já progrediu, isto é, a partir do instante 0 inicial da simulação, até que instante $t$ as estimativas sobre os comportamentos de todas regiões já estão corretas. Quando isso é determinado, diz-se que a simulação já convergiu para o intervalo $(0, t]$. A detecção de convergência é utilizada para determinar a terminação da simulação, que corresponde à convergência para todo o intervalo $(0, H]$ da simulação, e para o gerenciamento do espaço de armazenamento.

Quando o simulador determina que a simulação já convergiu até o tempo $t$, todo o espaço de armazenamento utilizado com eventos e estados correspondentes a instantes menores que $t$ pode ser liberado, pois não é possível que haja *rollback* para um tempo menor que $t$. Isto pode ser afirmado

dado que não existem mais estimativas incorretas para instantes menores que $t$, logo não existem mais mensagens em trânsito com tempo menor que $t$. Este mecanismo de liberação de espaço de armazenamento, chamado de coleta de fósseis, é necessário para a reutilização da memória.

Para realizar a detecção de convergência foi implementado um algoritmo de controle centralizado baseado na idéia proposta em [CS89]. O problema consiste em determinar o instante $t$ da simulação para o qual não existam mensagens em tânsito com tempos menores ou iguais a $t$, e os simuladores estejam ociosos com relação a eventos naquele intervalo. O algoritmo utiliza um processo chamado detector, existente em apenas um processador da rede, que pode se comunicar com os processos simuladores residentes em todos os processadores da rede. O simulador responsável pela região $r$ do diagrama espaço-temporal possui contadores $rec[r', t]$ e $env[r', t]$ que são, respectivamente, os números de mensagens com tempo $t$ que ele recebeu de e enviou para a região $r'$, para cada região $r'$ vizinha a $r$ e cada tempo $t$ de $(0, H]$. Quando o simulador fica ocioso, isto é, quando ele não tem nenhum evento para processar e todos os $PL$s sob sua responsabilidade estão ociosos também, ele envia estes contadores para o detector. O detector recebe estes contadores de cada região e os armazena nas variáveis $rec[r, r', t]$ e $env[r', r, t]$, que são os números de mensagens com tempo $t$ que a região $r$ recebeu de e enviou para a região $r'$, respectivamente. O detector determina que a simulação convergiu até o tempo $T$ se para todos os tempos $t \leq T$ e todas as regiões vizinhas $r$ e $r'$ $rec[r, r', t] = env[r', r, t]$[5]. Sempre que determina uma nova convergência, o detector avisa todos os simuladores, que se encarregam de acionar os mecanismos necessários para a coleta de fósseis. Quando a simulação converge para todo o intervalo $(0, H]$, ela está terminada.

O simulador desenvolvido possui mecanismos que tratam apenas da decomposição espacial da aplicação. Para utilizar também a decomposição temporal, é necessário um mecanismo de migração de estados. Por exemplo, quando um $PL$ está em duas regiões, na primeira para o intervalo $(0, t]$ e na segunda para o intervalo $(t, H]$, o simulador responsável pela primeira região, quando determina o estado do $PL$ no instante $t$, deve enviá-lo para a segunda região, que até o momento vinha trabalhando com uma estimativa deste estado.

Em [RBJ91], o modelo *Time Warp* foi estendido para aplicar decomposição temporal além da espacial. O trabalho sugere que a decomposição temporal melhora o desempenho de simulações em que a maioria dos eventos são apenas *read-only*, isto é, eventos cujo processamento não altera as variáveis de estado dos $PL$s que os processam. Neste caso, o recebimento de um estado migrado não necessariamente causaria *rollback*, pois possivelmente os eventos não precisariam ser reprocessados. Porém, a maioria dos sistemas físicos simulados não possuem essas características.

Dado que o recebimento de um estado quase sempre causa *rollback* no $PL$ correspondente (supondo que os eventos processados não sejam *read-only*), pois este $PL$ vinha trabalhando com uma estimativa para este estado, para que a simulação distribuída tenha bom desempenho é necessário um mecanismo para consertar a computação errada mais eficiente do que simplesmente desfazer tudo e recomeçar. Em [LL91] e [BCL92] foram implementados mecanismos que fazem esse

[5] A demonstração deste resultado baseia-se na noção de um estado global de um sistema distribuído. Especificamente, pode-se argumentar que se tem um estado global sempre que todos os canais estão vazios. Em particular, se neste estado global os processos estão ociosos, então uma condição de terminação foi atingida. Ao restringir estes conceitos a intervalos de tempo da simulação, esta terminação se traduz em convergência dentro daqueles intervalos [CL85].

conserto de forma eficiente, eliminando a necessidade de *rollback*, e bons resultados foram obtidos. Porém, estes mecanismos são específicos para a aplicação simulada, que nos dois casos era um sistema estocástico de rede de filas. Para outras aplicações, inclusive para o sistema de colisões de partículas simulado neste trabalho, tal mecanismo não parece viável.

## 4 Resultados

Para avaliar o desempenho do simulador distribuído desenvolvido com base no modelo espaço-temporal e implementado no computador paralelo, foi empregado o sistema lógico que modela o comportamento de um sistema físico de colisões de partículas em duas dimensões. O estudo deste sistema é interessante porque ele possui diversas aplicações práticas e porque sua simulação é custosa computacionalmente.

Neste sistema de colisões de partículas, um determinado número de partículas esféricas estão em movimento dentro de um domínio de duas dimensões limitado por bordas, e colidem entre si e com as bordas. Testes variando o número de partículas no domínio foram realizados para avaliar o comportamento do simulador.

O estado do sistema é composto pela posição e velocidade de cada partícula. Inicialmente são conhecidas a posicão e a velocidade iniciais das partículas, ou seja, o estado inicial do sistema. Neste estado inicial, a posição das partículas é gerada aleatoriamente de forma que elas fiquem distribuídas uniformemente pelo domínio. Deseja-se simular as colisões que ocorrem durante um determinado intervalo de tempo, e com isso determinar as posições e as velocidades das partículas no final do intervalo. Nos testes realizados, todas as partículas possuem o mesmo raio, porém este parâmetro pode ser facilmente modificado.

Este sistema físico foi modelado como um conjunto de $PF$s, através da divisão do domínio em setores. Cada $PF$ corresponde a um setor do domínio e o seu $PL$ correspondente é responsável pela simulação do comportamento das partículas que estão posicionadas dentro daquele setor. Uma colisão entre duas partículas ou entre uma partícula e uma borda do domínio é modelada como um evento do $PL$ onde ocorre a colisão destinado a ele mesmo. A passagem de uma partícula de um setor para outro é modelada como uma seqüência de eventos trocados entre os $PL$s correspondentes aos setores origem e destino da partícula.

Cada $PL$ se comporta da seguinte maneira. Quando recebe um evento do simulador para o tempo $t$, que pode ser uma partícula entrando em seu setor ou uma colisão, o $PL$ atualiza a posição das partículas em seu setor para o tempo $t$ e trata o evento de forma adequada, seja inserindo aquela partícula no seu conjunto de partículas, no primeiro caso, ou determinando as velocidades das partículas envolvidas após a colisão, no segundo caso. Em seguida ele determina o próximo evento a ocorrer com alguma partícula do seu setor, que pode ser uma colisão ou a saída de uma partícula do setor, e envia este evento para o simulador.

Para determinar estes eventos, cada $PL$ possui também em seu estado uma lista de possíveis colisões ou saídas a ocorrer. No início da simulação, o $PL$ determina, para cada partícula de seu setor, se é possivel que ela colida com cada outra partícula do setor ou com as bordas do domínio

durante o intervalo da simulação, dadas suas posições e velocidades iniciais. O $PL$ determina também, para cada partícula, se é possível que ela saia de seu setor durante o intervalo da simulação, dada sua posição e sua velocidade inicial. Essas possíveis colisões e saídas são inseridas na lista, em ordem não decrescente de tempo.

Quando recebe um evento de colisão, após tratar o evento o $PL$ redetermina as possíveis colisões e saídas, mas agora apenas das partículas envolvidas na colisão. Se o evento recebido é a entrada de uma partícula, ele determina as possíveis colisões e saídas apenas daquela partícula. Para determinar o próximo evento que enviará para o simulador, o $PL$ seleciona a colisão ou saída de menor tempo de sua lista. Se não utilizasse esta lista, o $PL$ teria que determinar, a cada evento executado, as possíveis colisões e saídas de todas as partículas de seu setor, e não apenas das partículas envolvidas no evento processado.

O diagrama espaço-temporal foi dividido de forma a haver uma região por processador. Cada região corresponde a um $PF$ e a todo o intervalo da simulação. Assim, o número de processadores determina o número de setores em que o domínio é dividido. A figura 3 mostra a divisão do domínio em setores para o caso de oito processadores. Note que neste caso o diagrama espaço-temporal deve ser visto em três dimensões, sendo a figura 3 a parte "espacial" do diagrama. As regiões foram mapeadas na rede de processadores interconectados sob a topologia de um hipercubo de forma que regiões vizinhas fossem alocadas a processadores vizinhos. Desta forma, a troca de mensagens contendo eventos e mensagens para a propagação de *rollback*, $inf\_rb$, é feita apenas entre processadores que se comunicam diretamente, sem necessidade de roteamento.

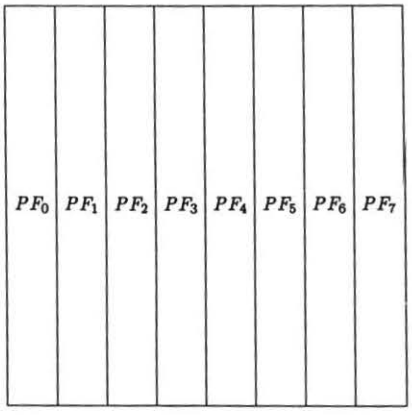


Figura 3: Divisão do domínio em setores

Foram realizados experimentos variando o número $N$ de partículas no domínio e mantendo o tamanho do domínio e o raio das partículas constantes. A simulação distribuída destes casos

foi executada na rede em topologia de hipercubo utilizando 2, 4 e 8 processadores, e o tempo de execução destas simulações foram medidos. Foram executadas simulações para $N = 25$, 50, 75, e 100. Esses mesmos testes foram executados em um simulador seqüencial, e os tempos destas execuções medidos. O desempenho do simulador distribuído foi avaliado através do *speedup* obtido por ele, dado pela razão entre o tempo de execução do melhor algoritmo seqüencial[6] e o tempo de execução do algoritmo paralelo. O gráfico da figura 4 apresenta estes *speedups* em relação ao número de processadores utilizado nas simulações (o *speedup* para 100 partículas em dois processadores requereria, para ser obtido, uma simulação com uso de memória além da disponível, não sendo então apresentado).

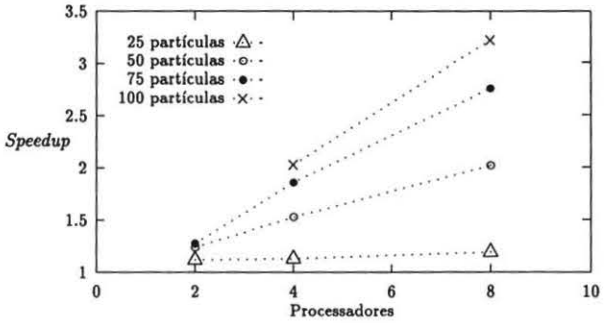


Figura 4: *Speedup* do simulador para o sistema de colisões de partículas

Como mencionamos anteriormente, estes resultados podem ser considerados muito bons no contexto da simulação de colisões de partículas. Este problema já havia sido submetido a outros métodos otimistas de simulação paralela, com *speedups* semelhantes (um exemplo é dado em [H89], onde são descritos os resultados obtidos com o *Time Warp*). Conforme era esperado, o desempenho do algoritmo melhora à medida que o tamanho da entrada (o número de partículas no domínio) cresce. Quando este número cresce, aumenta a densidade de partículas, isto é, o número de partículas por área no domínio, aumentando assim a proporção de eventos de colisão em relação aos eventos de entrada e saída de partículas. Os eventos de colisão são eventos de processamento interno a um processador, enquanto que eventos de entrada e saída de partículas demandam o envio de mensagens entre processadores.

Nestes testes, os estados dos $PL$s foram salvos a cada evento executado. Não foi possível aumentar mais o parâmetro $N$, por limitações de memória, pois essa aplicação é dispendiosa quanto à demanda de espaço de armazenamento para a gravação dos estados. Os estados dos $PL$s são

[6] A definição de "melhor" algoritmo seqüencial aqui utilizada merece uma explicação. Não se trata apenas do algoritmo mais rápido, mas sim de um algoritmo que reproduza os mesmos resultados do algoritmo paralelo. No caso da aplicação específica em questão, há uma grande sensibilidade a erros numéricos, e por isto adotou-se no caso seqüencial a mesma divisão do domínio em regiões utilizada no caso paralelo.

consideravelmente grandes, pois incluem a posicão e a velocidade de cada partícula do setor, bem como a lista de possíveis ocorrências.

## 5 Conclusões

Os experimentos realizados para a avaliação do desempenho do algoritmo de simulação distribuída baseado no modelo espaço-temporal permitem concluir que o algoritmo pode ter um desempenho muito bom se o volume de processamento que o simulador tem para realizar se sobrepõe ao volume de comunicação. Quando isso não acontece, a eficiência do algoritmo é baixa.

Para testes com entradas pequenas, a quantidade de mensagens em trânsito para a propagação dos *rollbacks* torna-se muito grande em relação à quantidade de processamento interno aos processadores, fazendo com que a simulação progrida de forma muito lenta, pois o simulador passa a maior parte do seu tempo desfazendo e reprocessando.

Uma característica atraente do modelo espaço-temporal é que as trocas de eventos se dão apenas entre regiões vizinhas, o que diminui o volume de mensagens e pode evitar a necessidade de roteamento.

## Referências

[ACSC91] Amorim, C. L.; Citro, R.; de Souza, A. F. & Chaves Filho, E. M., *The NCP-I Parallel Computer System.* COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Relátorio Técnico ES-241/91, 1991.

[BCL91] Bagrodia, R.; Chandy, K. M. & Liao, W. T., "A unifying framework for distributed simulation", *ACM Transactions on Modeling and Computer Simulation* **1** (4), 348–385, 1991.

[BCL92] Bagrodia, R.; Chandy, K. M. & Liao, W. T., "An experimental study of the performance of the space-time simulation algorithm", *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, 159–168, 1992.

[CL85] Chandy, K. M. & Lamport, L., "Distributed snapshots: determining global states of distributed systems", *ACM Transactions on Computer Systems* **3** (1), 63–75, 1985.

[CS89] Chandy, K. M. & Sherman, R., "Space-time and simulation", *Proceedings of the SCS Multiconference on Distributed Simulation*, 53–57, 1989.

[D90] Drummond, L. M. de A., *Projeto e Implementação de um Processador Virtual de Comunicação.* COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Tese de Mestrado, 1990.

[F90] Fujimoto, R. M., "Parallel discrete event simulation", *Communications of the ACM* **33** (10), 30–53, 1990.

[H89] Hontalas, P.; Beckman, B.; DiLoreto, M.; Blume, L.; Reiher, P.; Sturdevant, K.; van Warren, L.; Wedel, J.; Wieland, F. & Jefferson, D., "Performance of the colliding pucks simulation on the Time Warp operating systems (part 1: asynchronous behavior and sectoring)", *Proceedings of the SCS Multiconference on Distributed Simulation*, 3–7, 1989.

[J85] Jefferson, D. R., "Virtual time", *ACM Transactions on Programming Languages and Systems* **7** (3), 404–425, 1985.

[LL91] Lin, Y. B. & Lazowska, E. D., "A time-division algorithm for parallel simulation", *ACM Transactions on Modeling and Computer Simulation* **1** (1), 73–83, 1991.

[M86] Misra, J., "Distributed discrete-event simulation", *ACM Computing Surveys* **18** (1), 39–65, 1986.

[RBJ91] Reiher, P.; Bellenot, S. & Jefferson, D., "Temporal decomposition of simulations under the Time Warp operating system", *Proceedings of the 5th Workshop on Parallel and Distributed Simulation*, 47–54, 1991.