Modelos de Computação Paralela e Projeto de Algoritmos

Ronaldo Parente de Menezes e João Carlos Setubal

Relatório Técnico DCC–95-13

Agosto de 1995

Modelos de Computação Paralela e Projeto de Algoritmos

Ronaldo Parente de Menezes e João Carlos Setubal

Sumário

The PRAM model [FW78] has been the main vehicle for parallel algorithm design and analysis for more than 15 years. Its simplicity makes it unrealistic, and this has led many researchers to propose extensions and other models. Among the new models are the BSP model [Val90] and the LogP model [CKP+93]. We present here short descriptions of these models and show that they belong to different abstraction levels. We point out both positive aspects and shortcomings of each model, particularly with respect to algorithm design and analysis. We discuss the relevance of each model with regard to present-day parallel processing and argue that the LogP model is in a better position to gain wide acceptance.

Sumário

O modelo PRAM [FW78] vem sendo já há mais de 15 anos o modelo principal para projeto e análise de algoritmos paralelos. A sua simplicidade o torna um modelo pouco fiel à realidade, o que motivou o aparecimento de extensões e de outros modelos, dentre os quais BSP [Val90] e LogP [CKP+93]. Apresentamos uma descrição sucinta desses 3 modelos e mostramos os diferentes níveis de abstração em que se situam. São apresentadas as vantagens e desvantagens de cada um, particularmente com relação ao projeto e análise de algoritmos. A relevância de cada um frente ao panorama atual de máquinas paralelas é discutida, concluindo-se que o modelo LogP, apesar de ser de mais baixo nível, é o que tem mais chance de se disseminar em situações práticas.

1 Introdução

Existem ainda hoje arquiteturas bastante diversas para processamento paralelo. Essa diversidade é tão grande que sistemas de classificação foram inventados para categorizá-las. Um dos mais populares é aquele que divide as máquinas em SIMD e MIMD [Fly72]. Outra forma de separar as máquinas leva em conta a forma de comunicação entre os processadores: por memória compartilhada ou por troca de mensagens. Essa situação contrasta com o caso seqüencial, onde a arquitetura de von Neumann permanece como o denominador comum da vasta maioria das máquinas seqüenciais.

Essa diversidade de arquiteturas paralelas criou uma grande brecha entre a teoria e a prática da computação paralela. Por teoria entendemos aqui o projeto e análise de algoritmos paralelos. Por prática, entendemos o uso de máquinas paralelas reais para a solução efetiva de problemas.

A teoria tem usado nos últimos 15 anos primariamente o modelo PRAM[FW78]. Trata-se de um modelo bastante simples e que permitiu o desenvolvimento de um grande corpo de literatura em algoritmos paralelos. No entanto, essa mesma simplicidade torna o modelo PRAM

muito distante das máquinas reais, fazendo com que grande parte dos algoritmos paralelos da literatura não possa ser implementada (constituindo-se esta situação a "brecha" citada acima).

Este estado de coisas fez com que esforços de pesquisa fossem dirigidos ao desenvolvimento de novos modelos de computação paralela. Inicialmente foram propostas diversas extensões do modelo PRAM. Mais recentemente foram desenvolvidos dois novos modelos, que *não* são extensões do PRAM. São eles o modelo BSP [Val90] e LogP [CKP+93]. Estes dois modelos tem encontrado significativa receptividade por parte de pesquisadores na área de algoritmos paralelos. O objetivo deste artigo é descrever sucintamente estes novos modelos, comparando-os e contrastando-os, em particular quanto a aspectos relativos ao projeto e análise de algoritmos paralelos.

O artigo está estruturado da seguinte forma: na seção 2 descrevemos brevemente o modelo PRAM, apontando seus defeitos e mencionando algumas extensões. Na seção 3 descrevemos o modelo BSP e na seguinte o modelo LogP. Na seção 5 apresentamos a crítica comparativa dos três modelos, incluindo-se aí algumas especulações sobre a viabilidade dos modelos tendo em vista o panorama atual do processamento paralelo. Uma conclusão sugerindo linhas de pesquisa termina o artigo.

2 O modelo PRAM

O modelo PRAM supõe que os processadores se comunicam por memória compartilhada. Cada célula de memória pode ser acessada em tempo constante por qualquer processador. Submodelos estipulam regras para acesso concorrente. Os processadores devem trabalhar de forma síncrona, isto é, a cada passo do programa cada processador pode realizar uma computação local, ou ler, ou escrever na memória, e o passo seguinte só se inicia depois que todos os processadores tiverem terminado seus respectivos passos correntes.

Como se pode perceber por esta curta descrição, este modelo ignora diversos aspectos de máquinas paralelas reais. A latência de memória e o assincronismo de muitas dessas máquinas são dois dos mais importantes aspectos que não são levados em conta pelo PRAM; e estes aspectos têm efeitos profundos no desempenho de programas paralelos.

Além dos problemas do modelo em si, é preciso também levar em conta o uso que os projetistas de algoritmos fazem dele. Algoritmos desenvolvidos para o PRAM em geral supõem que o número de processadores disponível é proporcional ao tamanho do problema que se quer resolver. Isto é obviamente falso nas máquinas reais, que têm um número fixo de processadores. A teoria do modelo PRAM não se preocupa com este fato devido ao teorema de Brent [Bre74], que afirma essencialmente que uma algoritmo paralelo para n processadores e tempo de execução $T(n)$ pode em geral ser executado por $k < n$ processadores com acréscimo em $T(n)$ proporcional a n/k. Este teorema no entanto supõe que não há problema em *escalonar* os k processadores para fazer a tarefa dos n processadores originais, o que nem sempre é verdade. Ademais, os custos desse escalonamento não são levados em conta.

A despreocupação com o escalonamento de processadores foi levada ainda mais adiante com o esquema Trabalho-Tempo de apresentação de algoritmos [J92]. Nesse esquema os algoritmos paralelos são formulados apenas apresentando as operações que podem ser executadas em paralelo, sem o respectivo escalonamento de processadores. Isto por um lado simplifica

ainda mais o trabalho do projetista de algoritmos, mas por outro lado joga para um passo intermediário (que pode ser executado "mecanicamente") o problema do escalonamento e com isso esconde seus custos.

A despreocupação com diversos problemas práticos das máquinas paralelas contrasta com a preocupação com otimalidade. Grande parte da pesquisa em algoritmos PRAM têm procurado encontrar algoritmos cujo *custo*, isto é, o produto $p \times T(n)$ (onde p é o número de processadores) é o mesmo do mais rápido algoritmo seqüencial conhecido. Essa preocupação faz com que sejam desenvolvidos algoritmos bastante complexos para realizar tarefas básicas, tais como o cômputo da posição global de um elemento numa lista ligada (*list ranking*). Os algoritmos resultantes escondem constantes altas em seus tempos de execução expressos através da notação O. Considerando que esses mesmos algoritmos são depois usados como subrotinas em outros algoritmos para problemas mais complexos, chega-se a um resultado final que é, em muitos casos, absolutamente inimplementável.

Apesar dessas críticas, há trabalhos que mostram que o modelo PRAM não é tão ruim quanto parece. Hsu, Ramachandran e Dean [HRD93] apresentaram implementações paralelas de diversos algoritmos PRAM para grafos numa máquina MasPar com 16K processadores. Seus resultados foram bastante satisfatórios, mostrando que a performance dos algoritmos prevista pelo modelo foi bem próxima daquela observada. No entanto, nesse trabalho os autores se limitaram a resolver problemas cujo tamanho não excedia o número de processadores disponível, evitando assim a questão do escalonamento. Ademais, os autores implementaram apenas algoritmos PRAM simples, que em geral não são ótimos.

Vishkin [Vis92, Vis94] é outro defensor do modelo PRAM, afirmando, por exemplo, que algoritmos PRAM quando executados seqüencialmente podem em alguns casos serem mais rápidos do que os melhores algoritmos seqüenciais conhecidos. A razão disto é, de um lado, o fato de que algoritmos PRAM devem necessariamente saber a cada passo quais os endereços de memória necessários para a execução do passo; de outro lado, é a existência, nas máquinas modernas, de mecanismos de *pipeline* e *prefetch* de posições de memória. A combinação destes dois fatores produziria implementações seqüenciais bastante eficientes a partir de algoritmos PRAM. Vishkin mostra como evidência um estudo experimental [Vis94] onde se comparam os desempenhos seqüenciais de duas implementações para o problema da construção de árvores de sufixos. A implementação mais rápida foi aquela baseada no algoritmo PRAM. Independente deste aspecto, Vishkin acredita que não se deve ignorar o modelo PRAM pois novos desenvolvimentos tecnológicos podem vir a torná-lo viável na prática.

Conforme mencionado acima, muitas extensões do modelo PRAM foram propostas. Dentre estas destacamos as seguintes: APRAM [CZ89] que incorpora assincronismo; BPRAM [ACS89], que utiliza transferência de dados em blocos, objetivando modelar e minimizar a latência; e QRQW PRAM [GMR94], que modela com mais realismo o acesso concorrente a posições de memória. O problema dessas extensões é que a principal preocupação continua sendo a obtenção de algoritmos eficientes/ótimos (ou de cotas inferiores) segundo as diretrizes básicas do modelo PRAM, e os algoritmos resultantes em muitos casos continuam sendo pouco relevantes para a prática do processamento paralelo. Esse não é o caso dos modelos que passamos a descrever.

3 O Modelo BSP

O modelo BSP foi proposto por Valiant [Val90] e é um modelo de memória distribuída. Ele consiste das seguintes partes:

- Um número p de processadores e módulos de memória chamados indistintamente de *componentes*.
- Um *roteador* que transporta mensagens, tipicamente operações de leitura e escrita, ponto-a-ponto entre os componentes.
- Mecanismos para sincronização dos componentes a cada passo.

Apesar de síncrono o modelo supõe que a sincronização ocorre em períodos de L unidades de tempo. L é um parâmetro do modelo, e seu valor pode variar de uma máquina para outra. Supõe-se entretanto que L deve ser no mínimo igual ao tempo requerido para acessos não-locais (isto é, o tempo de latência do sistema).

A execução de um programa em um computador BSP consiste de um conjunto de *superpassos*. Em cada superpasso os componentes podem enviar e receber mensagens e realizar processamento local. Ao final de cada período de L unidades de tempo é executada uma sincronização global (barreira de sincronização) para verificar se todos os componentes encerraram suas tarefas. Se este não for o caso, outras L unidades de tempo são alocadas até que o superpasso termine. L portanto é um limite inferior para a duração de um superpasso.

Outro parâmetro do modelo é g, definido como sendo a razão entre o número total de operações locais realizadas por todos os componentes numa unidade de tempo e o número de mensagens transmitidas pelo roteador numa unidade de tempo. Esta definição equivale essencialmente ao inverso de *bandwidth*. O valor de g deve ser tal que num superpasso seja possível que um componente envie ou receba um total de h mensagens a um custo em número de operações locais (ou tempo) de gh, para h suficientemente grande. Se dentro de um período de L unidades de tempo o número de mensagens enviadas/recebidas (digamos h_0) for tal que $gh_0 < L$ o custo em tempo do envio/recebimento dessas mensagens permanece L.

3.1 Acessos concorrentes

Todo modelo de computação paralela deve se ocupar com a questão dos acessos concorrentes a posições ou módulos de memória. No caso do BSP, não há nenhum mecanismo explícito para o controle dos acessos concorrentes; esses acessos são simplesmente tratados um de cada vez. Por esse motivo a latência de um acesso concorrente será linearmente proporcional ao número de processadores que participam deste acesso. Isto provoca um atraso, que pode ser inaceitável caso o número de processadores seja relativamente grande.

Umas das soluções existentes para implementação de acessos concorrentes, em tempo não linearmente proporcional ao número de processadores, é o uso de redes de combinação (*combining networks*) [GGK+83, Ran87], que são mecanismos de *hardware* que podem combinar mensagens concorrentes ou replicar mensagens para diversos processadores-destino.

Uma rede de combinação é realizada através de ligações físicas entre os componentes. O problema deste tipo de solução é que, em geral, não conhecemos o padrão de comunicação

entre os componentes, e desta forma não podemos usar uma rede de combinação fixa com poucas ligações. Sendo assim, temos que fazer uso de mais ligações para permitir uma maior flexibilidade de comunicação. Tal flexibilidade, porém, somente é alcançada a um custo muito alto devido ao grande número de ligações necessárias.

O autor do modelo BSP, ciente desse problema, procura explorar as possibilidades de tratamento de acessos concorrentes via software [Val92]. Nessa proposta, os acessos concorrentes são combinados em duas fases. Na primeira fase, todos os acessos são espalhados, através de uma função de *hashing*, para processadores intermediários. Estes devem combinar as mensagens recebidas e que são para um mesmo módulo de memória numa só mensagem, que na segunda fase é enviada ao módulo de memória apropriado. No trabalho citado são apresentados resultados que mostram que com grande probabilidade o atraso final, após essas duas fases, é pequeno.

3.2 Folga Paralela

Outra questão abordada pelo modelo BSP é o fato do número p de processadores de qualquer máquina ser fixo. Dado um algoritmo PRAM que exige v processadores (chamados aqui de processadores *virtuais*), seu desempenho no modelo BSP com $p < v$ processadores deverá ser ótimo, a menos de fatores constantes. Isso somente será conseguido em geral se a diferença entre v e p for suficientemente grande. A essa diferença dá-se o nome de *folga paralela* (*parallel slackness*).

A exploração da folga paralela se dá da seguinte forma. Suponha a execução de uma operação de leitura de posição de memória não local, que na prática chega a custar até 10 vezes o tempo de uma operação local. Se $p = v$, o mapeamento dos processadores virtuais para os efetivos terá uma relação de 1:1, logo caso um processador necessite executar uma leitura ele ficará esperando até que o dado esteja disponível. Se $p < v$, cada processador efetivo responderá pela execução de $\lceil \frac{v}{p} \rceil$ processadores virtuais. Neste caso, quando um processador executa uma operação de longa latência, ele pode usar o tempo da latência para executar operações escalonadas para outros processadores virtuais, escondendo desta maneira a latência da leitura de memória. O modelo BSP supoõe que a exploração da folga paralela é tarefa automática atribuída a um compilador.

A questão que surge agora é: qual deve ser a diferença mínima entre v e p para que se consiga otimalidade no desempenho dos algoritmos? A resposta depende do valor do parâmetro L e da forma como são tratados os acessos concorrentes. Valiant [Val90] mostra que se é sabido de antemão que se no máximo h acessos concorrentes irão ocorrer, é possível manter a otimalidade da simulação de v processadores por p processadores se $v = \Omega(hp \log p)$.

Notamos aqui que o modelo BSP procura ser um bom simulador de algoritmos PRAM, e como tal deixa de levar em conta certos aspectos práticos. Um deles é que a exploração da folga paralela implica em freqüentes mudanças de contexto (*context switchings*), com um custo não desprezível. Outro aspecto é que as simulações, mesmo ótimas, trazem embutidas constantes multiplicativas que podem ser significativas.

3.3 Um exemplo de Algoritmo

Para tornar mais concreta a descrição do modelo apresentamos abaixo um algoritmo simples tal como ele seria projetado no modelo BSP. Nesse algoritmo é usada a operação `send(valor,destino)`, que manda `valor` para o processador `destino`.

O problema resolvido pelo algoritmo é o da difusão (*broadcast*) de um dado a partir de um processador-origem para n posições de memória espalhadas uniformemente por p processadores. A idéia é que seja criada uma *árvore de difusão*, em que o processador-origem ocupa a raiz. No início do algoritmo o processador-origem envia o dado para $d-1$ outros processadores. No passo seguinte todos os processadores que já reberam o dado realizam, cada um, o envio do dado a $d-1$ novos processadores. Ao final, cada processador tem uma cópia do dado a qual deve ser copiada em (n/p) posições locais de memória. A descrição do algoritmo resulta a seguinte:

Entrada: Um processador, P_1, contendo o dado a ser enviado para os outros processadores, o grau d da árvore a ser usada e as n posições de memória onde deve ser colocado o dado. O vetor `A` é um vetor local a cada processador.

Saída: n cópias do dado nos p vetores locais A.

```
(1)              for i := 1 to p do in parallel
(2)                 { primeira fase }
(3)                 k := 1
(4)                 while k <= p/d do
(5)                    if (i <= k) then
(6)                       for j:=1 to d-1 do
(7)                          send(dado,P_(i+(d-j)k))
(8)                    k := k * d
(9)                 { segunda fase }
(10)                for j := 1 to n/p do
(11)                   A[j] := dado
```

Na primeira fase do algoritmo acima, cada processador realiza a cada passo $d-1$ `send`s, a um custo de $(d-1)g$ unidades de tempo. A malha da primeira fase é é executada $\log_d p$ vezes, já que esta simula uma árvore d-ária. A segunda fase gasta n/p unidades de tempo. O tempo total pode ser expresso então por $T(n) = O(dg \log_d p + n/p)$.

Até aqui não há nada essencial que diferencie esse algoritmo de um algoritmo PRAM. A diferença aparece quando levamos em conta os parâmetros do modelo BSP. Os `send`s da linha (7) já indicam que é preciso que $L = O(dg)$, para que a sincronização cause atrasos limitados por fatores constantes. Além disso, a questão principal a ser resolvida é: quais são as condições em que é possível atingir o tempo ótimo $T_o(n) = O(n/p)$? Ou seja, qual o valor de d (a menos dos demais fatores constantes) que faz $O(dg \log_d p + n/p) = O(n/p)$? A resposta é

$$d = O((n/(pg \log p)) \log(n/(pg \log p))).$$

Esse resultado imediatamente mostra que só conseguiremos otimalidade se $n = \Omega(pg \log p)$.

O exemplo dado é bastante simples, mas ilustra o modo como o modelo BSP deve ser usado para analisar um algoritmo. No caso de outros problemas mais complexos, o projeto e análise de seus respectivos algoritmos são bem mais sofisticados, particularmente quando se insiste em otimalidade. Um exemplo disso no caso do problema da ordenação pode ser encontrado em [GV93].

4 O Modelo LogP

O modelo LogP foi proposto por Culler *et al.* [CKP+93] e pretende ser um aperfeiçoamento do modelo BSP levando em conta diversas máquinas paralelas reais desenvolvidas nos últimos 5 anos. Seus autores acreditam que já houve uma convergência nas arquiteturas de máquinas paralelas e propuseram um modelo que leva em conta as características consideradas fundamentais dessa convergência. O resultado é um modelo semelhante ao BSP mas de mais baixo nível, como se verá adiante.

A convergência de arquiteturas mencionada se baseia nas seguintes observações. Diversos fatores tecnológicos estão fazendo com que as máquinas paralelas de grande porte atuais sejam constituídas de centenas ou, no máximo, de milhares de nós, cada nó contendo um processador poderoso (centenas de megaflops), não customizado, e com bastante memória local (centenas de megabytes). A tecnologia de interconexão dos nós, apesar de ter evoluído bastante, não consegue acompanhar as crescentes velocidades dos processadores e de memória, o que faz prever que a taxa de transferência de dados (*bandwidth*) continuará baixa em relação à velocidade dos processadores, assim como a latência continuará sendo relativamente alta. Por outro lado, modernas técnicas de roteamento indicam que a particular topologia de interconexão não será um fator importante. Exemplos de máquinas que se enquadram nas características acima são Connection Machine 5, Cray T3D, Intel Paragon e IBM SP-2.

Levando em conta essas observações, Culler *et al.* propuseram o modelo LogP, que é um modelo de memória distribuída, com um número P de módulos, cada um composto de um processador e sua memória local. Além de P, os demais parâmetros que definem o modelo são os seguintes:

- L, um limite superior para latência.
- o, custo adicional de comunicação (*overhead*) incorrido por um processador para transmitir ou receber uma mensagem.
- g (*gap*), que define o intervalo mínimo entre transmissões consecutivas ou recepções consecutivas de mensagens.

Aqui cabe observar que o nome do modelo não é uma referência à função logaritmo, mas simplesmemente uma aglutinação dos nomes dos parâmetros.

Como podemos verificar pelos parâmetros do modelo, ele não supõe a existência de nenhuma topologia particular para a rede de interconexão (tal como no modelo BSP). Por outro lado, o modelo supõe que a rede de interconexão tem uma capacidade finita, isto é, no máximo um certo número de mensagens podem estar circulando na rede num dado instante. Esse número é $\lceil \frac{L}{g} \rceil$ para garantir que todas mensagens poderão ser processadas mesmo que

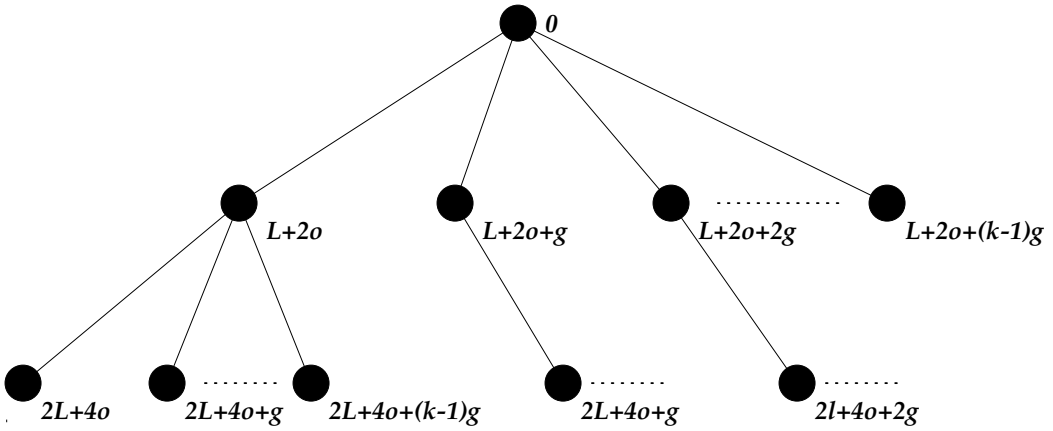

Figura 1: Árvore para difusão de um único dado

destinadas a um único processador. As previsões de desempenho no modelo LogP só são válidas se este limite imposto à rede de interconexão for respeitado.

Os proponentes do modelo LogP acreditam que este conjunto de parâmetros dificilmente poderia ser menor caso realmente desejemos uma boa expressão da realidade. O uso de todos os parâmetros está sujeito a cada aplicação e a cada máquina. Em alguns casos alguns parâmetros podem ser desprezados pois não influenciam no comportamento total do algoritmo.

A diferença principal em relação ao modelo BSP é a ausência de um mecanismo de sincronização. Culler *et al.* justificam essa ausência por constatarem que em muitas máquinas paralelas atuais tais mecanismos não existem e a sincronização, quando necessária, deve ser feita através de troca de mensagens, tal como no modelo LogP.

4.1 Um Algoritmo

Apesar da semelhança com o modelo BSP, o modelo LogP é usado de maneira muito diferente no projeto e análise de algoritmos paralelos. O objetivo é que ele seja usado como um *guia* e não como um modelo formal ao qual poderíamos, por exemplo, associar uma linguagem de programação. O uso do modelo LogP para resolver um determinado problema vai indicar como os dados do problema devem ser particionados entre os processadores e em que momentos mensagens devem ser trocadas. Porém esse particionamento e o tempo de envio de mensagens estarão parametrizados por L, o e g. Somente a atribuição de valores concretos para esses parâmetros permitirá a expressão de um algoritmo propriamente dito. O exemplo desta seção tornará estas idéias mais claras.

Usaremos nesta seção também o problema da difusão de dados. Tal como no caso do modelo BSP teremos aqui uma árvore de difusão. Vamos supor que o dado a ser difundido já se encontra no processador P_0 no tempo 0. A partir deste instante o processador P_0 já está apto a enviar o dado para outro processador e pode reenviar o dado a outros processadores a cada intervalo de g unidades de tempo. Conseqüentemente P_0 enviará um dado nos tempos 0, g, $2g$, $3g$, e assim por diante. Quando o dado sai de P_0 no tempo 0 ele estará disponível no seu destino no tempo $L + 2o$, correspondendo a um custo extra de comunicação, o, para o envio; uma latência, L, correspondendo ao tempo que o dado leva do nó fonte para o

nó destino; e finalmente outro custo extra de comunicação para o recebimento do dado. Podemos generalizar estes tempos exemplificados para o processador P_0 para qualquer outro processador. Suponha que um processador recebeu o dado a ser difundido no tempo a, então este mesmo processador poderá enviar este dado para outros processadores nos tempos a, $a+g$, $a+2g$ e assim sucessivamente. Similarmente, se um processador envia um dado no tempo b ele estará disponível em seu destino no tempo $b+L+2o$. A figura 1 ilustra o método.

Note que a abordagem descrita no parágrafo anterior não pode ser escrita como um pseudo-código tal como no modelo BSP. Por exemplo, só poderemos saber se P_0 deve ou não enviar uma mensagem a (digamos) P_4, se os valores dos parâmetros L, o, e g forem tais que essa mensagem chega antes do que uma mensagem enviada por um outro processador mais abaixo na árvore.

Outro problema que ilustra a utilização do modelo LogP é o problema da soma de n números. Culler *et al.* mostram que a soma de n números é essencialmente o inverso do problema da difusão. Novamente aqui temos uma árvore, mas o fluxo de mensagens vai das folhas para a raiz. Todos os processadores realizam somas locais, cujos resultados (menos no caso da raiz) são enviados a processadores mais acima na árvore. Nós intermediários da árvore recebem mensagens com somas parciais, somam esses resultados às suas próprias somas parciais e passam o novo resultado adiante. A raiz da árvore se encarrega de fazer a soma final. Tal como no caso da difusão, somente com valores para os parâmetros L, o e g é que se pode obter uma árvore concreta para a soma de n números. Na verdade, o processo é mais complexo. Ele envolve a determinação da árvore ótima de soma (ou de difusão) para os parâmetros P, L, o e g, e somente a partir desta é que se pode chegar na distribuição dos dados pelos processadores e no escalonamento de mensagens que permite realizar a soma em tempo mínimo. Detalhes podem ser encontrados em [CKP+93]. Para os propósitos deste artigo basta notar que um problema simples como "soma de n números" exige solução aparentemente complexa, particularmente quando comparada com a solução via PRAM.

O baixo nível do modelo LogP torna desnecessária a preocupação com folga paralela, existente no caso do BSP. O modelo LogP é um modelo que se preocupa com fatores constantes, pois seus autores consideram que em muitas situações os fatores multiplicativos das simulações do modelo PRAM, seja através do modelo BSP seja por outros mecanismos, são inaceitáveis para o desempenho na prática.

No artigo que introduziu o modelo LogP [CKP+93] há descrições da abordagem de outros problemas tais como transformada rápida de Fourier, e decomposição LU. Recentemente um trabalho sobre ordenação no modelo LogP foi publicado [Dus94]. Nesses trabalhos reportam-se bons resultados de implementações obtidas a partir do modelo.

5 Uma análise crítica dos modelos

Pela exposição acima deve ter ficado claro que os três modelos descritos se situam em níveis diferentes de abstração, sendo o modelo PRAM mais abstrato, o modelo BSP intermediário, e o modelo LogP o mais concreto. Obviamente, quanto mais abstrato o modelo mais fácil será o projeto de algoritmos. Por outro lado, maiores serão os fatores que dificultarão a obtenção de uma implementação com bom desempenho. A questão que queremos responder nesta seção é se os detalhes a mais introduzidos pelos modelos BSP e LogP os tornarão modelos de uso

corrente.

Incialmente é preciso ressaltar que o modelo BSP tem a grande vantagem de poder ser usado em conjunto com o modelo PRAM e suas extensões através de simulações eficientes. Isso permite que projetistas de algoritmos continuem usando o modelo PRAM e que os algoritmos já desenvolvidos para esse modelo possam, em princípo, ser aproveitados em implementações. No entanto, as exigências de sincronização por hardware do modelo BSP podem torná-lo inviável. Valiant argumenta que trata-se de um pequeno custo a ser pago para obter boas condições de programabilidade, mas não está claro que os fabricantes de máquinas paralelas estão dispostos a incluir mais esse custo em seus projetos. O modelo LogP, ao se adaptar a uma realidade existente, parece ter mais condições de viabilidade.

Pela discussão do parágrafo anterior, fica claro que na análise dos 3 modelos é preciso levar em conta a situação do processamento paralelo na atualidade. O objetivo último do processamento paralelo é se tornar de "propósito geral", possivelmente suplantando o processamento seqüencial como forma dominante e preferencial de utilização dos computadores [McC93]. Entretanto é sabido que processamento paralelo ainda sofre percalços para se estabelecer de forma sólida no mercado. A razão predominante parece ser o contínuo aperfeiçoamento dos processadores seqüenciais, tornando as grandes máquinas paralelas desvantajosas em termos de custo-benefício. Um reflexo disto são as recentes falências ou concordatas de grandes firmas fabricantes de multiprocessadores. Se persistirem essas condições, as redes de *workstations* talvez se tornem a única arquitetura paralela viável no futuro, conforme sugeriu Patterson [Pat93].

Levando essa análise em conta a conclusão básica que se pode tirar é que um modelo de computação paralela só terá sucesso se modelar adequadamente essas redes de *workstations*, sem perdas significativas de desempenho em implementações. Sob esta ótica, somente o modelo LogP preeenche esses requisitos, dentre os aqui apresentados. Mesmo assim, o modelo supõe processadores relativamente homogêneos. No caso de redes heterogêneas os parâmetros genéricos do modelo tenderiam a penalizar as máquinas mais velozes em função das mais lentas. De qualquer modo, essas observações sugerem que uma linha de pesquisa a ser investigada é a verificação do comportamento de implementações derivadas a partir do modelo LogP em redes homogêneas de *workstations*. Isto se deve ao fato de que, tanto quanto sabemos, os proponentes do modelo têm procurado até aqui validar sua proposta apenas em multiprocessadores tais como a Connection Machine 5.

Não estamos querendo dizer com estes comentários que multiprocessadores do tipo CM-5 estão com os dias contados. Certamente um grande número de aplicações só poderão ser executadas em máquinas desse tipo. Porém essas aplicações, num curto prazo, ainda serão *nichos* se comparadas à vasta gama de utilização dos processadores seqüenciais. Essa situação faz com que o desempenho seja o critério supremo da utilidade de uma particular máquina, e assim os fatores constantes, desprezados pelo PRAM e relegados a um segundo plano pelo BSP, continuarão tendo por algum tempo mais importância do que programabilidade ou portabilidade. Este aspecto reforça a viabilidade do modelo LogP.

6 Conclusão

Fizemos neste artigo uma breve análise de três modelos importantes de computação paralela. Uma discussão mais detalhada dos aspectos levantados aqui pode ser encontrada em [Men95].

A linha básica de pesquisa necessária para aclarar esta discussão é o teste dos modelos BSP e LogP em diversos tipos de multiprocessadores. Em particular, é de bastante interesse verificar a robustez do modelo LogP em redes homogêneas de *workstations*.

Referências

[ACS89] Alok Aggarwal, Ashok K. Chandra, e Marc Snir. On communication latency in PRAM computations (preliminary version). In *Proc. of ACM Symp. on Parallel Algorithms and Architectures*, pp. 11–21, 1989.

[Bre74] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, abril de 1974.

[CKP+93] David Culler, Richard Karp, David Patterson, Abhijit Sahay, KlausÊrik Schauser, Eunice Santos, Ramesh Subramonian, e Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. of 4th ACM PPOPP*, pp. 1– 12, Maio de 1993. Também como Technical Report UCB/CSD 92/713 - University of California, Berkeley - Computer Science Division (EECS).

[CZ89] Richard Cole e Ofer Zajicek. The APRAM: Incorporating asynchrony into the PRAM model. In *Proceedings of SPAA 89*, pp. 1–10, 1989.

[Dus94] Andrea Carol Dusseau. Modeling parallel sorts with LogP on the CM-5. Technical Report CSD-94-829, University of California at Berkeley, 1994.

[Fly72] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, Setembro de 1972.

[FW78] S. Fortune e J. Wyllie. Parallelism in random access machines. In *In Proceedings of 10th Annual ACM Siymposium on Theory of Computing*, pp. 114–118, San Diego, CA, 1978. ACM Press, New York.

[GGK+83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, e M. Snir. The NYU ultracomputer - designing an MIMD, shared-memory parallel machine. *IEEE Transactions on Computers*, 32:75–89, 1983.

[GMR94] Phillip B. Gibbons, Yossi Matias, e Vijaya Ramachandran. Efficient low-contention parallel algorithms. Technical report, AT&T, Setembro de 1994.

[GV93] Alexandros V. Gerbessiotis e Leslie G. Valiant. Direct bulk-syncronous parallel algorithms. *Proceedings of SWAT 92, in Lectures Notes in Computer Science*, 621:1–18, 1993. Também como Techinical Report TR-10-92 (Extended Version) - Harvard University - Center for Research in Computing Technology.

[HRD93] Tsan-Sheng Hsu, Vijaya Ramachandran, e Nathaniel Dean. Implementation of parallel graph algorithms on the MasPar. Technical Report TR-93-38, University of Texas at Austin, Julho de 1993.

[J92] Joseph Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, 1992.

[McC93] W. F. McColl. General Purpose Parallel Computing. In *Lectures in Parallel Computation*. Gibbons, P. (ed.), capítulo 13, pp. 337–391. Cambridge University Press, 1993.

[Men95] Ronaldo P. Menezes. Estudo de modelos de computação paralela e sua viabilidade como veículo de expressão algorítmica. Tese de mestrado, Universidade Estadual de Campinas - DCC, 1995. (em preparação).

[Pat93] David Patterson. Observations on massively parallel processors. In *DIMACS Workshop on Models, Architectures and Technologies for Parallel Computation*. 1993.

[Ran87] A. Ranade. How to emulate shared memory. In *Proc. 28th IEEE Symp. on Foundation of Computer Science*, pp. 185–194, 1987.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Communications of ACM*, 33(8):103–111, Agosto de 1990.

[Val92] Leslie G. Valiant. A combining mechanism for parallel computers. Technical Report TR-24-92, Harvard University - Center for Research in Computing Technology, Novembro de 1992.

[Vis92] Uzi Vishkin. A case for the PRAM as a standard programmer's model. Technical Report UMIACS-TR-92-130, University of Maryland, Novembro de 1992.

[Vis94] Uzi Vishkin. Can parallel algorithms enhance serial implementation? IPPS94, 1994.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the** DNA **Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br