# Modelos Realísticos de Computação Paralela

AMAURY ANTÔNIO DE CASTRO JUNIOR[1]
CLÁUDIA YOSHIE NASU
EDSON NORBERTO CÁCERES
HENRIQUE MONGELLI

UFMS – Universidade Federal de Mato Grosso do Sul
CCET – Centro de Ciências Exatas e Tecnologia
DCT – Departamento de Computação e Estatística
Cx. Postal – 579 – CEP – 79070-900 – Campo Grande (MS)
{amaury,cnasu,edson,mongelli}@dct.ufms.br

**Resumo:** A maioria das aplicações desenvolvidas para máquinas paralelas consideram apenas o problema da paralelização com baixos requerimentos de comunicação. Entretanto, em máquinas reais, o tempo de comunicação é, em geral muito maior que o tempo de computação. Conseqüentemente, muitos algoritmos paralelos teoricamente eficientes, para o modelo PRAM, não produzem o desempenho esperado quando implementados em máquinas paralelas reais. Este trabalho apresenta os modelos de computação paralela, ditos realísticos, que buscam uma maior proximidade entre o desempenho teórico e prático dos algoritmos desenvolvidos. Esses modelos incorporam algumas características intrínsecas à computação paralela e assim, refletem a realidade com mais exatidão.

**Palavras-chave:** paralelismo, modelos de computação paralela, algoritmos paralelos, algoritmos distribuídos, computação paralela.

## 1. Introdução

A computação paralela tem como objetivo usar simultaneamente um conjunto de processadores interligados, de modo a resolver um problema conjuntamente mais rápido que o processamento seqüencial usando somente um processador [9].

Um *computador paralelo* é simplesmente uma coleção de processadores, tipicamente do mesmo tipo, interconectados de maneira a permitir a coordenação de suas atividades e a troca de dados [6]. O uso de computadores paralelos se faz necessário em diversos problemas onde o volume de dados e cálculos é grande e precisa-se de rapidez na obtenção das respostas.

Com a computação paralela a criação de algoritmos modifica-se sensivelmente. As estratégias utilizadas em algoritmos seqüenciais para resolver um problema não servem totalmente para a criação de algoritmos paralelos para o mesmo problema [7].

Além disso, é preciso considerar a natureza dos problemas, alguns são inerentemente paralelizáveis; outros não apresentam um paralelismo tão transparente; e existem aqueles em que o grau de paralelismo é muito baixo ou inexistente. No desenvolvimento dos algoritmos temos que considerar também a arquitetura paralela a ser utilizada, pois isto afeta o desempenho dos algoritmos.

O modelo PRAM, que poderia ser um padrão único para a computação paralela, não consegue capturar com exatidão a noção de paralelismo por ser completamente teórico. As características não incorporadas ao modelo, tais como, custo adicional para referência a memória não-local e latência, têm grande impacto no desempenho dos algoritmos.

Os modelos de rede, por sua vez, são felizes na solução da inviabilidade de acesso em tempo constante a uma memória global. Mas os algoritmos, por outro lado, tendem a ser específicos para uma determinada topologia. E ainda, a suposição de que a comunicação entre vizinhos é feita em

[1] Professor do Curso de Engenharia da Computação da UCDB – Universidade Católica Dom Bosco em Campo Grande/MS.

tempo constante não reflete a realidade dos computadores paralelos.

A busca de um modelo adequado necessita da compreensão e da incorporação, em tal modelo, de características intrínsecas à computação paralela, bem como ignorar aquelas características secundárias superáveis através da tecnologia. Os modelos apresentados a seguir, denominados *realísticos*, buscam estabelecer padrões amplamente aceitos que reflitam as dificuldades inerentes do próprio paralelismo.

## 2. Modelos de Computação Paralela

No modelo de computação seqüencial de Von Neumann (RAM), que assume a existência de uma única unidade central de processamento e uma memória de acesso aleatório, é possível estabelecer uma relação entre os desempenhos das implementações e dos seus respectivos algoritmos através das medidas de complexidade de tempo baseadas em análises assintóticas. Neste modelo, estas medidas são capazes de refletir corretamente o desempenho dos algoritmos seqüenciais, servindo como referência para as implementações. No entanto, na computação paralela, a mesma relação entre algoritmos e implementações ainda não encontrou um modelo apropriado. A presença de diversos elementos de processamento torna a definição de um modelo de computação paralela consideravelmente mais complexa. Nesta seção, apresentaremos os modelos paralelos mais conhecidos e utilizados para o desenvolvimento e a análise de algoritmos paralelos, começando pelo modelo PRAM.

### 2.1 Modelo PRAM

O modelo PRAM [6] (*Parallel Random Access Machine*) é uma extensão do modelo seqüencial RAM e o mais conhecido modelo de computação paralela. Pode ser descrito como sendo um conjunto de processadores operando de modo síncrono, sob o controle de um relógio comum. Cada processador é identificado por um índice único e possui uma memória local própria, podendo comunicar-se com os demais processadores através de uma memória global compartilhada. Essa forma de comunicação possibilita o acesso (leitura ou escrita) simultâneo de vários processadores a uma mesma posição da memória global. Desse modo, surgem três variações do modelo PRAM baseadas na permissão, ou não, do acesso concorrente:

- **EREW** (*Exclusive Read, Exclusive Write*) - Este modelo não permite qualquer acesso simultâneo a uma mesma posição da memória por mais de um processador, seja para leitura ou escrita;
- **CREW** (*Concurrent Read, Exclusive Write*) - Este modelo permite somente leitura simultânea de uma mesma posição da memória por mais de um processador, não sendo a mesma operação permitida para a escrita simultânea;
- **CRCW** (*Concurrent Read, Concurrent Write*) - Este modelo permite tanto a leitura, como a escrita concorrente em uma mesma posição da memória por mais de um processador. Para evitar os possíveis conflitos do acesso simultâneo, temos os seguintes critérios:
  - **CRCW comum:** Na escrita simultânea, todos os processadores devem escrever o mesmo valor;
  - **CRCW prioritário:** Os valores escritos simultaneamente podem ser diferentes. Ficará armazenado na posição da memória o valor escrito pelo processador de maior prioridade (assumimos como sendo o processador com menor índice);
  - **CRCW arbitrário:** : Os valores escritos simultaneamente podem ser diferentes e apenas um, entre todos os processadores, poderá escrever, não importando qual deles.

O modelo PRAM, apesar de sua importância conceitual e teórica, não consegue capturar com exatidão a noção de paralelismo. As características não incorporadas ao modelo durante o desenvolvimento dos algoritmos, tais como custo adicional para referência a memória global e latência, têm grande impacto no desempenho das implementações. Como alternativas para este problema surgem os chamados modelos realísticos, como o BSP e o CGM, descritos em seguida, que incorporam algumas dessas características.

### 2.2 Modelo BSP

O modelo BSP (*Bulk Synchronous Parallel Model*) - Modelo Paralelo Síncrono Volumoso - foi proposto por Valiant [11], em 1990. Além de ser um dos mais importantes modelos realísticos, foi um dos primeiros a considerar os custos de comunicação e a abstrair as características de uma máquina paralela em um pequeno número de parâmetros. O objetivo principal deste modelo é servir de modelo ponte entre as necessidades de hardware e software na computação paralela. Segundo Valiant, este é um dos fatores responsáveis pelo sucesso do modelo seqüencial de Von Neumann.

Uma máquina BSP consiste de um conjunto de $p$ processadores com memória local, comunicando-se através de algum meio de interconexão, gerenciados por um *roteador* e com facilidade de sincronização global. Um algoritmo BSP consiste de uma seqüência de *superpassos* separados por *barreiras de sincronização*, como mostra a Figura 1. Em um superpasso, a cada processador é atribuído um conjunto de operações independentes, consistindo de uma

combinação de passos de computação, usando dados disponibilizados localmente no início do superpasso, e passos de comunicação, através de instruções de envio e recebimento de mensagens. Neste modelo, uma *h*-relação em um superpasso corresponde ao envio e/ou recebimento de, no máximo, *h* mensagens em cada processador. Uma mensagem enviada em um superpasso será recebida somente no próximo superpasso.

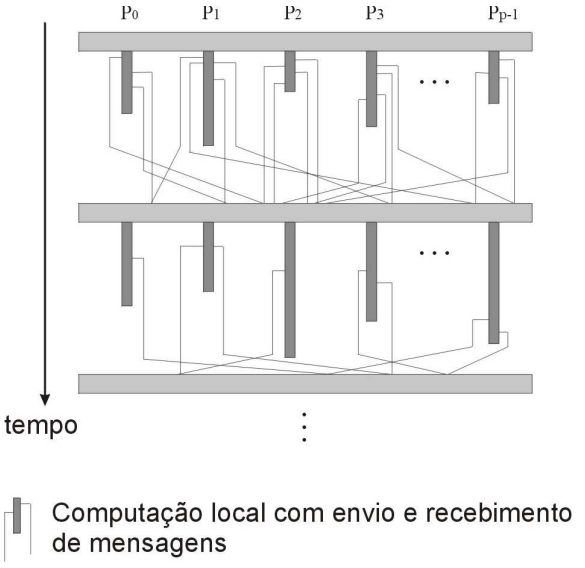


**Figura 1 –** O modelo BSP[5]

Os parâmetros do modelo BSP são os seguintes:

- *n*: tamanho do problema;
- *p*: número de processadores disponíveis, cada qual com sua memória local;
- *L*: o tempo mínimo entre dois passos de sincronização. Também chamado de parâmetro de periodicidade ou latência de um superpasso;
- *g*: é a capacidade computacional dividida pela capacidade de comunicação de todo o sistema, ou seja, a razão entre o número de operações de computação realizadas em uma unidade de tempo e o número de operações de envio e recebimento de mensagens. Este parâmetro descreve a taxa de eficiência de computação e comunicação do sistema.

Os dois últimos parâmetros, *L* e *g*, são usados para computar os custos de comunicação no tempo de execução de um algoritmo BSP. O parâmetro *L* representa o custo de sincronização, de tal forma que cada operação de sincronização contribui com *L* unidades de tempo para o tempo total de execução. *L* também pode ser visto como a latência de comunicação, para que os dados recebidos possam ser acessados somente no próximo superpasso. A capacidade de comunicação de uma rede de computadores está relacionada ao parâmetro *g*. Através deste parâmetro podemos estimar o tempo tomado pela troca de dados entre processadores. Se o número máximo de mensagens enviadas por algum processador durante uma troca simples é *h*, então seriam necessárias até *gh* unidades de tempo para a conclusão da troca. Na prática, o valor de *g* é determinado empiricamente, para cada máquina paralela, através da execução de benchmarks apropriados [8].

De modo mais formal, o tempo total de execução de um superpasso de um algoritmo BSP é igual a $w_i + gh_i + L$, onde $w_i = \max\{L; t_1 ;...; t_p\}$ e $h_i = \max\{L; c_1 ;...; c_p\}$, $t_j$ e $c_j$ são, respectivamente, o número de operações de computação executadas e o número de mensagens recebidas e/ou enviadas pelo processador *j* no superpasso *i*. O custo total do algoritmo é dado pela soma dos custos de cada um dos superpassos. Vamos assumir que o algoritmo BSP possui um total de *T* superpassos e, sejam

a soma de todos os valores de *w* e *h* de cada superpasso.

$$W = \sum_{i=0}^{T} w_i \qquad \text{e} \qquad H = \sum_{i=0}^{T} h_i$$

Então, o custo total de um algoritmo BSP é $W + gH + LT$. O valor de *W* representa o total de computações locais e o valor de *H* representa o volume total de comunicação.

### 2.3 Modelo CGM

O modelo CGM (*Coarse Grained Multicomputer*) - Multicomputador com Granularidade Grossa - foi proposto por Dehne *et al* [4], em 1993. É semelhante ao modelo BSP, no entanto, é definido em apenas dois parâmetros:

- *n*: tamanho do problema;
- *p*: número de processadores disponíveis, cada um com uma memória local de tamanho *O(n/p)*.

O termo "granularidade grossa" (*coarse grained*) vem do fato de que o tamanho do problema é consideravelmente maior que o número de processadores, ou seja, $n/p \geq p$. Um algoritmo CGM consiste de uma seqüência de rodadas (*rounds*), alternando fases bem definidas de computação local e comunicação global, como mostra a Figura 2. Normalmente, durante uma rodada de computação utilizamos o melhor algoritmo seqüencial para o processamento dos dados disponibilizados localmente.

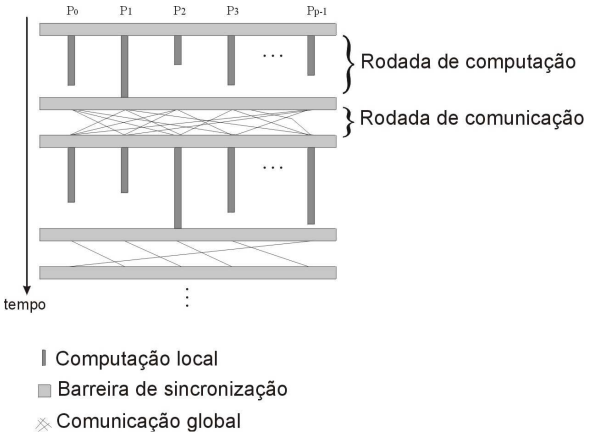


**Figura 2 –** O modelo CGM[5]

Em uma rodada de comunicação, uma *h*-relação (com $h = O(n/p)$) é roteada, isto é, cada processador envia $O(n/p)$ dados e recebe $O(n/p)$ dados. No modelo CGM, o custo de comunicação é modelado pelo número total de rodadas de comunicação.

O custo de um algoritmo CGM é a soma dos tempos obtidos em termos do número total de rodadas de computação local (análogo ao *W* do modelo BSP) e do número de superpassos (análogo ao *T* do modelo BSP), que equivale ao número total rodadas de comunicação.

### 2.4 Modelo BSP x Modelo CGM

Ambos os modelos, BSP e CGM, tentam reduzir os custos de comunicação de modo muito semelhante e buscam caracterizar uma máquina paralela através de um conjunto de parâmetros, sendo que dois destes se assemelham, o número de operações de computação local e o número de superpassos[2].

No entanto, existem algumas diferenças. Uma delas, segundo Götz[5], é que o modelo CGM simplifica o projeto e o desenvolvimento de algoritmos por ser um modelo mais simples e levemente mais poderoso que o modelo BSP.

Além disso, um algoritmo CGM pode ser transferido para o modelo BSP sem mudanças. Se um algoritmo CGM executa *W* operações de computação local, $T_{cp}$ superpassos de computação e $T_{cm}$ superpassos de comunicação, então o algoritmo BSP correspondente terá $O(W + ghT_{cm} + L(T_{cp} + T_{cm}))$ de custo [2].

Em resumo, os modelos BSP e CGM apresentam muitas similaridades. Entretanto, o modelo CGM simplifica os custos de comunicação, facilitando o projeto de algoritmos.

[2] No modelo CGM, denominados rodadas (*rounds*).

## 3. Conclusões

No final dos anos 80, o desenvolvimento de algoritmos paralelos para o modelo PRAM foi bastante grande. Infelizmente, os resultados teóricos obtidos não foram observados nas implementações nas máquinas existentes. Nos anos 90, surgem os modelos BSP e CGM. Os algoritmos desenvolvidos nesses modelos, quando implementados nas máquinas atualmente existentes, apresentam resultados de *speedup* (tempo do algoritmo seqüencial dividido pelo tempo do algoritmo paralelo com *p* processadores) semelhantes aos previstos nas análises teóricas.

Os modelos CGM e BSP trouxeram um progresso considerável a área de algoritmos paralelos, mas claramente o estado da arte necessita de futuras pesquisas. Esses modelos são um bom campo para que estudantes de pós-graduação possam desenvolver suas pesquisas [3].

## 4. Referências

[1] E. Cáceres, A. Chan, F. Dehne, & S. W. Song. “Coarse grained parallel graph planarity testing”. Proc. *International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'2000)*, Las Vegas, Nevada, 2000.

[2] E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro & S. W. Song. “Eficient parallel graph algorithms for coarse grained multicomputers and BSP”. *Lecture Notes in Computer Science*, 1256:390-400, 1997.

[3] F. Dehne, “Guest Editor's Introduction”, *Algorithmica Special Issue on “Coarse grained parallel algorithms”* , Vol. 24, No. 3/4, July/August 1999, pp. 173-176.

[4] F. Dehne, A. Fabri & A. Rau-Chaplin. “Scalable parallel computational geometry for coarse grained multicomputers”. *ACM Conference on Computational Geometry*, 1993.

[5] Silvia M. Götz. “Communication-Eficient Parallel Algorithms for Minimum Spanning Tree Computations”. *PhD thesis*, Department of Mathematics and Computer Science - University of Paderborn - Germany, Maio 1998.

[6] Joseph JáJá. “An Introduction to Parallel Algorithms”. Addison-Wesley, 1992.

[7] Henrique Mongelli. “Algoritmos CGM para Busca Uni e Bidimensional de Padrões com e sem Escala”. *Tese de Doutorado*, Instituto de Matemática e Estatística - USP - Brasil, Abril 2000.

[8] Isabelle Guérin Lassous & Jens Gustedt. “List Ranking on a Coarse Grained Muliprocessor”. Relatório Técnico, RR-3640 INRIA-Lorraine – Março, 1999.

[9] Marco Aurélio Stefanes. “Algoritmos e implementações paralelas para florestas geradoras mínimas”. *Tese de Mestrado*, Instituto de Matemática e Estatística – USP – Brasil, Dezembro 1997.

[10] Marco Aurélio Stefanes. “Algoritmos paralelos para modelo realísticos”. *Qualificação de Doutorado* - Instituto de Matemática e Estatística - USP - Brasil, Março 2000.

[11] L. G. Valiant. “A bridging model for parallel computation”. *Communications of the ACM*, 33:103-111, 1990.