# Otimização e Processamento Paralelo de Alto Desempenho

**Celso Carneiro Ribeiro e Noemi de la Rocque Rodriguez**

## INTRODUÇÃO

A utilização de métodos quantitativos de otimização e de técnicas da pesquisa operacional vem se revelando cada vez mais como uma ferramenta de grande importância na solução de problemas de engenharia e de planejamento. Tais ferramentas encontram aplicação crescente em problemas nos setores energético e petrolífero, no projeto e na exploração de redes de comunicação, em problemas de planejamento e escalonamento da produção, de roteamento de veículos e de gestão de frotas de empresas aéreas (composição de frotas, rodízio de tripulações), entre outras áreas.

O advento das técnicas e ferramentas de programação paralela, portáveis e de fácil uso, e a disponibilidade de máquinas paralelas, com desempenho e confiabilidade crescentes e custos cada vez menores, vem contribuindo sobremaneira para o sucesso da aplicação de métodos de otimização como uma solução prática, computacionalmente viável. Implementações paralelas são tão mais eficientes quanto a aceleração que proporcionam aproxima-se do número de processadores utilizados. Neste trabalho, apresenta-se um rápido panorama das técnicas de programação paralela. São apresentadas duas aplicações reais, em que o paralelismo permitiu tratar problemas de otimização computacionalmente difíceis com uma grande aceleração, cuja resolução por máquinas seqüenciais era praticamente inviável. Finalmente, apresenta-se algumas tendências na área de *cluster computing*, que representa a plataforma de hardware mais promissora, ilustrando-se ainda outras aplicações industriais na área bancária e de serviços de Internet.

***Celso Ribeiro e Noemi Rodriguez***
*Depto. de Informática, PUC-Rio*

## COMUNICAÇÃO ENTRE PROCESSOS

Uma aplicação paralela ou multiprocessada é formada por um conjunto de processos que cooperam na obtenção de um resultado. Tipicamente, esses processos precisam se comunicar para trocar dados e pedidos entre si. Como cada processo progride de forma independente dos demais, muitas vezes é necessário definir pontos de sincronização entre eles, de forma a garantir a coerência da computação global.

Nos últimos anos, vários sistemas operacionais passaram a oferecer suporte a um tipo de processos denominados *threads*, ou processos de peso leve, que podem se comunicar através de variáveis globais compartilhadas. Do ponto de vista do programador paralelo, *threads* são importantes em máquinas com vários processadores, onde eles permitem o desenvolvimento de aplicações paralelas cujas partes se comunicam por

memória compartilhada e, além disso, muitas vezes executam de forma mais eficiente do que processos tradicionais. A comunicação por memória compartilhada requer um cuidado grande com o problema de sincronização. Para garantir uma relação de precedência entre comandos de processos distintos, o programador deve utilizar comandos explícitos de sincronização, que devem ser providos pela linguagem ou sistema operacional. Com *threads*, o mecanismo de sincronização mais freqüentemente oferecido é o de variáveis de condição, que correspondem a um serviço que permite que um processo permaneça bloqueado até que uma condição seja verdadeira.

O desenvolvimento de aplicações baseadas em variáveis compartilhadas e condições pode ser bastante complexo. Os bloqueios e sinalizações associados a variáveis de condição podem estar espalhados por um código extenso, e qualquer acesso a uma variável global pode representar uma necessidade de sincronização, tornando árdua a tarefa de depuração do programa paralelo. Muitos autores acreditam que a forma alternativa de comunicação entre processos, por troca de mensagens, representa uma base mais segura para o desenvolvimento de aplicações paralelas.

De qualquer maneira, a troca de mensagens é a forma natural de comunicação entre processos que não compartilham memória (por exemplo, entre processos que executam nos vários nós de um *cluster* de computadores). Em sua forma mais básica, um mecanismo de troca de mensagens fornece uma primitiva para envio de dados para um processo específico (*send*) e de recebimento de dados de um processo específico ou de qualquer processo que tenha enviado alguma mensagem (*receive*). Muitas abstrações e expansões podem ser oferecidas em cima desse modelo básico. Por exemplo, pode-se citar bibliotecas de programação paralela que, além de fornecerem diversas variantes de *send* e *receive*, muitas vezes oferecem mecanismos para comunicação e sincronização de grupos [11,12].

## ALGUMAS FERRAMENTAS DE PROGRAMAÇÃO PARALELA

Um número crescente de ferramentas (bibliotecas, linguagens) para programação paralela estão disponíveis para o desenvolvimento de aplicações paralelas. Duas destas ferramentas, utilizadas recentemente em projetos desenvolvidos em cooperação com duas grandes empresas brasileiras, são descritas a seguir.

Em 1995, o IEEE definiu um padrão de interface de programação com *threads*, conhecido como *Posix threads* ou *Pthreads* [5]. Desde então, diversos sistemas operacionais vêm incluindo suporte a esse padrão, também implementado por uma biblioteca disponível publicamente [8]. No modelo *Pthreads*, vários *threads* podem ser definidos dentro de um único processo. O processo funciona como o espaço de endereços de memória que uma determinada aplicação pode utilizar. Os diversos *threads* associados a um processo compartilham memória: quando um *thread* realiza escritas na memória (por exemplo, atribuições a variáveis), outro *thread* pode ler o resultado. O sistema operacional gerencia a alocação de processadores para a execução dos *threads* como se eles fossem processos independentes. Em uma máquina com múltiplos processadores, diferentes *threads* de uma mesma aplicação podem ser simultaneamente executados em diferentes CPUs. A interface Pthreads define basicamente os seguintes grupos de rotinas: rotinas de controle, que permitem criar, eliminar e suspender a execução de *threads*, rotinas de sincronização, entre as quais as que manipulam as variáveis de condição descritas na seção anterior, e rotinas para controle de escalonamento, que permitem alterar as prioridades relativas dos diversos *threads* de uma aplicação.

Com o aumento da banda passante oferecida pelas redes locais, passou a ser comum o uso de redes de estações de trabalho como se fossem máquinas multiprocessadoras. Surgiram então desde o final dos anos 80 uma série de bibliotecas de programação paralela para este tipo de ambiente [6]. Entre elas, provavelmente a que obteve a maior divulgação e difusão de uso foi o PVM (Parallel Virtual Machine). PVM foi inicialmente implementada para máquinas Unix que se comunicassem por TCP/IP, aproveitando assim o impulso de crescimento da Internet. Atualmente existem versões para diversos outros sistemas. A biblioteca PVM é disponível publicamente, através da Internet, e amplamente documentada, em especial através de um livro [4] também disponível por rede.

Em 1995, autores e usuários de bibliotecas de troca de mensagens fomaram um grupo [7] com o intuito de definir uma interface de programação paralela comum, de modo a evitar a proliferação de programas escritos em diferentes "dialetos". Esse grupo envolveu aproximadamente 40 pessoas de 60 instituições. O resultado desse esforço foi a definição do MPI (Message Passing Interface) [12], que inclui a definição de interfaces de um conjunto de rotinas para comunicação por troca de mensagens. Desde a publicação do MPI, em 1995, diversas implementacões já foram desenvolvidas, sendo várias delas disponíveis publicamente [7]. O desenvolvimento dessas várias implementacões totalmente compatíveis com a definição original vem fazendo do MPI um padrão ad hoc para bibliotecas de troca de mensagens. O núcleo do MPI é formado por rotinas de comunicação entre pares de processos (tipo *send* e *receive*). Além desse núcleo, o MPI define várias outras rotinas, entre as quais se destacam rotinas de comunicação e sincronização em grupo. A instalação das bibliotecas das diferentes versões de MPI costuma ser simples. Embora exija uma certa intimidade com os conceitos de programação paralela, a programação com MPI não apresenta maiores dificuldades.

## APLICAÇÃO NO SETOR PETROLÍFERO

A otimização da exploração de reservatórios de petróleo requer muitas avaliações de possíveis combinações das variáveis de

decisão envolvidas, tais como as propriedades dos reservatórios, a localização dos poços e os parâmetros de escalonamento da produção, de modo a obter a melhor estratégia em termos econômicos. Executar uma simulação para tal número de avaliações pode não ser viável em termos práticos, devido aos elevados tempos de computação envolvidos. Uma das ferramentas eficientes desenvolvidas para resolver esta aplicação foi um algoritmo genético híbrido [2], cuja estratégia de otimização integra análise econômica, simulação dos reservatórios e otimização do projeto.

Embora tal ferramenta tenha apresentado um ótimo desempenho na prática, encontrando soluções economicamente muito boas, os tempos de computação observados são extremamente elevados. Isto decorre da necessidade de executar muitas simulações, uma para cada configuração avaliada. Desta forma, o paralelismo oferece uma excelente alternativa para este tipo de aplicação, permitindo que diferentes configurações sejam avaliadas simultaneamente por diferentes processadores. Através de um projeto contratado em 1998 pela Gerência de Tecnologia e Simulação de Reservatórios da Petrobras, foi desenvolvida uma implementação paralela utilizando MPI deste algoritmo genético híbrido. MPI foi escolhida em função de sua portabilidade e das facilidades que oferece para programação do controle de falhas nos processadores atuando de forma concorrente.

Os resultados obtidos foram particularmente animadores, indicando a aderência do paralelismo como solução para este tipo de problema. Em razão das tarefas paralelizadas (a simulação de cada configuração) serem completamente independentes e do tempo de processamento de cada simulação ser bastante elevado (da ordem de algumas horas, em certos casos), a estratégia de paralelização consegue atingir rendimento máximo, obtendo-se acelerações da ordem do número de processadores utilizados. Deve-se ainda mencionar a flexibilidade do MPI, já que esta aplicação vem sendo processada em redes completamente heterogêneas, formadas por estações de trabalho de pelo menos três fabricantes diferentes.

## APLICAÇÃO NO SETOR ELÉTRICO

Problemas de programação linear com um número muito grande de variáveis e restrições aparecem em muitas aplicações em diferentes áreas, tais como planejamento e operação de sistemas de produção e transmissão de energia, projeto de redes de telecomunicações, localização e dimensionamento de plataformas de petróleo, distribuição de produtos, roteamento de veículos, localização de centros de serviços e exploração de recursos naturais, entre outras.

Um problema importante no setor elétrico, em especial dentro do atual contexto econômico, consiste na alocação de custos de serviços de transmissão entre os diversos agentes (empresas) que participam do sistema elétrico interligado, cada um controlando um certo número de usinas e devendo atender uma certa demanda em sua área de responsabilidade. Através de projeto contratado em 1997 pelo Laboratório de Análises Técnico-Econômicas de Sistemas de Potência do Centro de Pesquisas em Energia Elétrica (CEPEL), desenvolveu-se e implementou-se uma metodologia para resolver este problema de programação linear com um número exponencial de restrições, baseado em uma formulação através da teoria dos jogos cooperativos [3]. A dificuldade computacional associada à resolução deste problema decorre do número extremamente elevado de subproblemas de planejamento da expansão da transmissão que devem ser resolvidos a cada iteração de um procedimento de geração de restrições, cada um deles constituindo por si só um problema de programação inteira.

A solução inovadora, proposta na primeira fase deste projeto, consistiu na utilização de meta-heurísticas de última geração (algoritmos genéticos, busca tabu e GRASP [9]) para acelerar o procedimento de geração de restrições, minimizando-se a chamada de procedimentos exatos, mas computacionalmente mais custosos. A metodologia desenvolvida foi aplicada a um sistema realista de grande porte, formado por parte da rede brasileira, definido por 19 agentes (empresas), 79 barras e 283 circuitos. O sistema computacional desenvolvido na linguagem C implementando a metodologia proposta foi capaz de resolver de forma exata este problema [10], utilizando quase 42 horas de processamento em um computador Sun UltraSparc com 256 Mbytes de memória. Embora o tempo de processamento seja elevado, isto decorre da dificuldade intrínseca do problema, que até então não havia sido resolvido numericamente.

Em uma segunda fase, procurou-se utilizar o processamento paralelo como forma de aceleração da metodologia proposta. A estratégia de paralelização adotada baseia-se em buscas cooperativas paralelas, cada uma delas implementando uma heurística diferente para acelerar a geração de restrições [1]. No caso desta aplicação, foi utilizada uma máquina paralela Sun Starfire ENT10000 do CENAPAD MG-GO, dispondo de até 32 processadores UltraSparc de 250 MHz. A implementação paralela em C utiliza Posix *threads*, em razão da arquitetura desta máquina ser baseada em memória compartilhada (vide seção 2). Os resultados obtidos através do processamento paralelo demonstram claramente a eficácia do emprego de paralelismo na solução desta classe bastante importante de problemas de otimização. A implementação da estratégia adotada, usando uma configuração com apenas um processador mestre e quatro escravos (dentre os 32 disponíveis), permitiu reduzir o tempo de processamento para apenas 5.4 horas, ou seja, para cerca de apenas 12% do tempo observado originalmente.

## TENDÊNCIAS

O desenvolvimento de ferramentas de software portáveis, eficientes e simples de utilizar, associado à disponibilidade de

uma gama de máquinas paralelas com grande capacidade de processamento e custos decrescentes, vem tornando o processamento paralelo uma alternativa viável para o desenvolvimento de técnicas computacionalmente eficientes para a solução de problemas práticos reais de simulação, otimização, *data mining* e processamento de imagens, entre outras aplicações.

Dentre as atuais tendências em processamento paralelo, destaca-se a idéia de *cluster computing*. Basicamente, um *cluster* é uma coleção de PCs ou estações de trabalho conectados por uma rede, montados com componentes comerciais padrão. Os supercomputadores até recentemente oferecidos como a principal alternativa para o processamento paralelo de alto desempenho (tais como computadores Cray ou Connection Machine) apresentavam preços muito elevados (na casa de alguns milhões de dólares) e taxas de obsolescência extremamente altas, agravadas pelo rápido desaparecimento do mercado de alguns fabricantes. O significativo aumento do desempenho e da confiabilidade de PCs, redes e estações de trabalho viabilizou o uso de *clusters* como uma alternativa de baixo custo para processamento paralelo em aplicações que exigem recursos computacionais de forma intensiva.

*Clusters* podem ser montados rapidamente, utilizando versões de domínio público de Linux como sistema operacional; redes Ethernet, SCI ou Myrinet; e ambientes populares de software como PVM ou MPI. Alguns sistemas baseados em *clusters* encontram-se entre as máquinas apresentando melhores índices de custo/desempenho. Recentemente foi noticiada a montagem de um *cluster* formado por 1000 computadores Pentium II 350 MHz padrão e um computador hospedeiro. O laboratório Sandia disponibilizou em agosto de 1999 um *cluster* com cerca de 1400 processadores Compaq Alpha que atinge 1.1 TFlop/s de desempenho de pico, que o coloca entre os 10 sistemas mais rápidos atualmente disponíveis.

Outra grande vantagem destes sistemas decorre de serem facilmente escaláveis (adição de novos processadores) e atualizáveis (*upgrade* dos processadores, componentes, redes e software). Finalmente, *clusters* oferecem uma importante solução não apenas para aplicações científicas, mas também para aplicações comerciais exigindo altas taxas de disponibilidade (por exemplo, bancos e operadoras de cartões de crédito com processamento de transações on-line). Neste caso, o sistema pode rapidamente identificar e isolar um elemento do *cluster* que entre em pane, transferindo as tarefas por ele desempenhadas para outro processador. Provedores de serviços livres de correio eletrônico (e.g. hotmail) e mecanismos de busca na Internet (e.g. hotbot) são motorizados por *clusters*, de modo a prover altas taxas de acesso simultâneo a um grande número de usuários.

O Laboratório de Paralelismo da PUC-Rio conta atualmente com um *cluster* formado por 32 processadores Pentium II 400 MHz IBM PC 300GL, cada um com unidade de disco própria, conectados por uma rede Ethernet através de um *switch* IBM 8274 modelo W93 na placa ESM-C-32W que permite 32 segmentos a 10 Mbps, funcionando sob o sistema operacional Linux. Este *cluster* entrou em operação recentemente e foi um importante aporte para nossas instalações e capacidade de desenvolvimento de pesquisas e projetos. Entre os projetos atualmente em desenvolvimento neste *cluster*, deve-se citar a cooperação estabelecida com os laboratórios de pesquisa da AT&T nos Estados Unidos, visando o desenvolvimento de uma metodologia para solução de um problema de projeto de redes de acesso a serviços de telecomunicações. O objetivo consiste em desenvolver um software de otimização que permita balancear o lucro potencial que pode ser obtido provendo-se acesso a um novo serviço a determinados grupos de usuários, em relação ao custo de construir a rede que permita oferecer tais serviços.

## REFERÊNCIAS


[1] R.M. Aiex, C.C. Ribeiro e M.P. Aragão, "Parallel cut generation for service cost allocation in transmission systems", III Metaheuristics International Conference, 1-6, Angra dos Reis, 1999.

[2] A.C. Bittencourt e R.N. Home, "Reservoir Development and Design Optimization", Annual Technical Conference and Exhibition of the Society of Petroleum Engineering, 545-558, San Antonio, 1997.

[3] N. Campodónico e M.V. Pereira, "A methodology for transmission service cost allocation in the core of cooperative games", Relatório técnico, Power Systems Research Inc., Rio de Janeiro, 1996.

[4] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek e V. Sunderman, "PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing", The MIT Press, 1994. (disponível através do WWW em http://www.netlib.org/pvm3/book/pvmbook.html)

[5] Institute of Electric and Electronic Engineering, "Information Technology - Portable Operating Systems Interface (POSIX) - Part 1 - Amendment 2: Threads Extension", 1995.

[6] S. Martins, C.C. Ribeiro e N.R. Rodriguez, "Ferramentas para programação paralela em ambientes de memória distribuída", Investigación Operativa 5 (1996), 67-98.

[7] MPI Forum, http://www.unix.mcs.anl.gov/mpi/

[8] C. Provenzano, "Pthreads", http://www.mit.edu:8001/people/proven/pthreads

[9] C.R. Reeves, "Modern heuristic techniques for combinatorial problems", Blackwell, 1993.

[10] C.C. Ribeiro, M.P. Aragão, A.I. Tavares, R.M. Aiex e E.U. Barbosa, "Projeto LABTEC: Alocação de custos de serviços de transmissão", Relatório técnico, Departamento de Informática, PUC-Rio, Rio de Janeiro, 1997.

[11] V. Sunderam, "A framework for parallel distributed computing", Concurrency: Practice and Experience 2 (1990), 315-339.

[12] D. Walker, "The design of a standard message passing interface for distributed memory concurrent computers", Parallel Computing 20 (1994), 657-673.