4
Computação Paralela

4.1.
Introdução

Nos últimos anos observa-se uma tendência cada vez maior do aumento da demanda computacional na resolução de grandes problemas. Exemplos de aplicações que exigem alto poder de computação são vistos em estudos de seqüenciamento genético (modelagem do DNA), previsão de movimentos de corpos celestes, renderização de imagens tridimensionais, estudos sísmicos e meteorológicos. Diversas possibilidades computacionais foram criadas para a resolução destes problemas, tais como a criação de supercomputadores (*CRAY*, por exemplo), estruturas de *clusters* de computadores, computadores pessoais que apresentam mais de um núcleo de processamento e até a utilização de unidades de processamento gráfico (GPU´s, do inglês *Graphics Processing Units*) para processamento paralelo de processos.

Arquiteturas de computadores baseadas em multiprocessadores e redes de comunicação de alta velocidade e baixo custo vêm substituindo rapidamente os dispendiosos supercomputadores, tais como os de arquitetura *CRAY* na solução dos mais variados problemas científicos. Computação paralela aplicada de forma eficiente nestes conjuntos de computadores multi-processados pode resultar em ganhos consideráveis em termos de tempos de execução, tornando possível, por exemplo, aceleração de aplicativos essenciais para a monitoração de diversos tipos de sistemas em tempo-real. No entanto, como a maioria dos algoritmos utilizados para tais tarefas foram inicialmente desenvolvidos com arquiteturas computacionais seqüenciais, ou seja, com um único processador, estes não fazem uso completo destes novos ambientes paralelos de computação. Desta forma, há a necessidade de se adaptar antigas metodologias ou desenvolver novas técnicas, que sejam capazes de aproveitar a capacidade computacional

extra, advinda dos recentes e economicamente viáveis, sistemas paralelos, ou distribuídos, de computação de alto desempenho.

O papel do paralelismo na aceleração de componentes computacionais fora reconhecido por décadas (GRAMA *et al*., 2003). No passado, contudo, tendências no desenvolvimento de *hardware* não eram claras e as interfaces foram desenvolvidas de forma dedicada a aplicações e sistemas específicos, sem nenhuma padronização. Nestas circunstâncias, portanto, o desenvolvimento de programas capazes de executar tarefas em paralelo significava altíssimos custos, tanto em termos de *hardware* quanto em equipe de desenvolvimento de *software* altamente qualificada.

Fator importante a ser destacado é que o aumento na velocidade dos processadores encontra barreiras cada vez maiores, à medida que limites físicos, tais como a velocidade da luz, são atingidos. A freqüência de operação dos processadores passa a aumentar a um passo bem mais reduzido do quê o experimentado durante os anos 90. Desta forma, a tendência é de que os requerimentos de capacidade computacional não serão preenchidos por processadores mais rápidos, mas pelo aumento do número de operações executadas simultaneamente num determinado período de tempo, através de processadores trabalhando paralelamente (FOSTER, 1995).

Aliadas à velocidade dos processadores, a latência e a largura de banda para a comunicação com os sistemas de armazenamento de dados, ou memórias, representam fatores cruciais para a manutenção de níveis razoáveis de desempenho computacional. Normalmente, o número de acessos aos elementos de memória tende a superar o número de operações de ponto flutuante (flops, do inglês *floating point operations*), fato este que impõe restrições agudas ao desempenho de certos programas paralelos. O emprego de memórias dispostas de forma hierárquica de acordo com suas velocidades de acesso (também chamadas *caches*) ajuda a minimizar este problema, mas não de forma definitiva. Plataformas de computação paralela, no entanto, apresentam sistemas de memória com melhor desempenho, dado à maior quantidade de memória e banda de comunicação com as mesmas disponíveis aos processadores (FOSTER, 1995; BERTSEKAS, 1997; GRAMA *et al*., 2003). Adicionalmente, ao passo que problemas aumentam tanto em tamanho como complexidade, bancos de dados

cada vez maiores são também necessários, o quê pode inviabilizar a concentração das mesmas em uma só localização física. Em tais situações, computação paralela surge com a única opção. Além disto, uma motivação extra ao emprego de computação paralela ocorre em casos que não permitam a construção de centrais de bancos de dados devido a requerimentos técnicos anormais e/ou razões de segurança e confidencialidade dos dados.

4.2. *Clusters* de Computadores

Nas últimas décadas, avanços na tecnologia de hardware tornaram os computadores pessoais (PCs) uma alternativa de alto custo-benefício, em comparação aos tradicionais supercomputadores, na solução de problemas científicos de grande escala. No início dos anos 90, engenheiros da NASA sugeriram que a utilização de conjuntos computadores pessoais de baixo-custo interligados através redes de alta velocidade poderiam duplicar a capacidade computacional dos supercomputadores. Principalmente, se um dado aplicativo pudesse ser executado por um número alto de processos concorrentes. De acordo com seus estudos, eles observaram que o custo-benefício de tais sistemas de computadores iria mais do que compensar o tempo adicional necessário para trocar informações entre os diferentes processos pertinentes a uma mesma tarefa.

Eventualmente, este conceito tornou-se conhecido como *cluster Beowulf*, cuja estrutura genérica é ilustrada na Figura 23 (ROCHA, 2003).

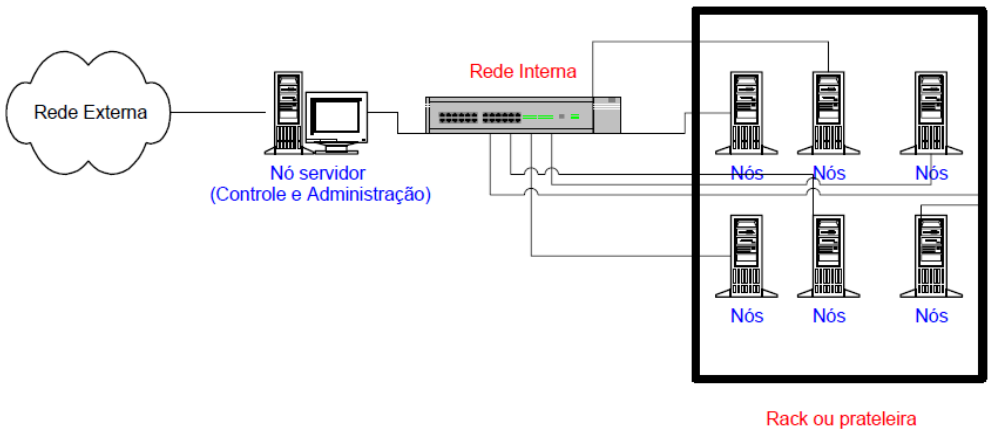

Figura 23: Cluster de computadores tipo *Beowulf* genérico.

Outra vantagem deste conjunto de PCs sobre os supercomputadores é a maior facilidade de programação e manutenção. Supercomputadores eram muitas vezes construídos de forma a suprir objetivos específicos, que, não raramente, também exigia técnicas especiais de programação compatíveis com determinados modelos. No caso dos conjuntos de PCs, as arquiteturas de processadores utilizadas são bem difundidas, permitindo reuso de código e, conseqüentemente, reduzindo significativamente o tempo de desenvolvimento e implantação de programas paralelos. No tocante a redes de comunicação, muitos são os tipos disponíveis no mercado, com distintos princípios de funcionamento. No entanto, interfaces padronizadas de programação paralela, tais como a *Message Passing Interface* (MPI) e a *Parallel Virtual Machine* (PVM), contribuem decisivamente para desvincular a programação da parte física (*hardware)*, incentivando a utilização dos *clusters*.

4.3. Métricas de Desempenho em Sistemas Paralelos

Ao se analisar o desempenho de programas paralelos, com intuito de comparar algoritmos distintos, algumas métricas são normalmente utilizadas, tais como o fator de aceleração e eficiência (FOSTER, 1995; GRAMA *et al*., 2003; WILKINSON e ALLEN, 2005).

4.3.1. Fator de Aceleração

O fator de aceleração é uma medida que relaciona a aceleração obtida por um algoritmo paralelo com o melhor algoritmo seqüencial disponível para a execução da uma mesma tarefa.

Matematicamente, o fator de aceleração *S* (do inglês, *speedup*), corresponde à razão entre o tempo gasto na computação de um dado problema utilizando-se de apenas um processador, t_s, e o tempo gasto por um programa que utiliza *p* processadores de forma paralela, t_p.

$$S(p) = \frac{t_s}{t_p} \tag{4.1}$$

4.3.2. Fator de Aceleração Máximo

Diversos fatores limitam o aumento do fator de aceleração, tais como períodos de ociosidade e comunicação entre processadores, bem como o fato de nem todo o código poder ser paralelizado.

Em um sistema de computação paralelo ideal, tanto o processamento seqüencial necessário pelo programa como o tempo de comunicação são nulos, enquanto o trabalho a ser executado em paralelo pode ser caracterizado pela distribuição equilibrada do tempo seqüencial original t_s entre os p processadores. Neste caso, o tempo necessário para resolver um dado problema em paralelo seria t_s/p, levando a um fator de aceleração de p vezes.

Porém, considerando que a comunicação entre processos seja insignificante, e que a parcela de trabalho executado em paralelo tenda a zero à medida que o número de processadores p aumenta, o fator de aceleração S obtido por tal programa paralelo estaria limitado a $1/f_{ps}$, onde f_{ps}, representa a fração de tempo gasto na parte não-paralelizável. Tal característica dos programas paralelos foi primeiramente observada em (AMDAHL, 1967) e representa o fator de acelaração máximo.

4.3.3. Eficiência

Eficiência, por sua vez, mede a fração de tempo no qual cada unidade de processamento é efetivamente utilizada (GRAMA *et al*., 2003). A representação matemática da eficiência computacional de um programa paralelo é dada pela razão entre o fator de aceleração e o número de processadores utilizados, como definido abaixo.

$$E = \frac{S(p)}{p} = \frac{t_s}{t_p \times p} \tag{4.2}$$

Pode ser observado que a eficiência de um sistema de computação paralela ideal iguala-se a unidade. Na prática, no entanto, a comunicação entre processos tende a aumentar com o número de processadores, levando a eficiências normalmente inferiores a um.

4.4. Interfaces de Programação Paralela

Muitos são os modelos computacionais utilizados em computação paralela, tais como paralelismo de dados, memória compartilhada, troca de mensagens, operações em memória remota, processos, e também, combinação dos anteriores. Tais modelos se diferenciam em vários aspectos, como por exemplo, se a memória disponível é localmente compartilhada ou geograficamente distribuída e, volume de comunicação tanto em *hardware* como em *software* (GROPP *et al*., 1999).

A aplicação de tais modelos, portanto, depende fortemente do problema a ser resolvido e a arquitetura de computação paralela alvo. Esta situação fez com que vários fabricantes de sistemas de computação paralela desenvolvessem suas próprias bibliotecas de interface, focando em característica, muitas vezes, inexistentes em outras arquiteturas. Desta forma, o código desenvolvido com bases nestas bibliotecas proprietárias não era portátil. Em face desta dificuldade, impulsionada pelos avanços na tecnologia de sistemas paralelos de computação, foi reconhecida a necessidade de interfaces de programação que fossem eficientes, funcionais e portáteis nas mais variadas arquiteturas disponíveis no mercado. Neste contexto, a *Message Passing Interface* (MPI) e a *Open Multi-Processing Application Program Interface* (OpenMP) foram concebidos.

4.4.1. *Message Passing Interface* (MPI)

Na medida em que sistemas computacionais distribuídos tornaram-se mais populares e problemas de portabilidade na área de programação paralela mais latente, um esforço conjunto de vários fabricantes e usuários de sistemas paralelos criaram, em 1994, a interface padrão MPI para programação paralela por meio de troca de mensagens.

O padrão MPI era uma especificação dos procedimentos de comunicação entre processos e, não uma implementação. Os processos, por sua vez, poderiam ser locais (*e.g.,* processadores de vários núcleos) ou remotos (*e.g.*, sistemas de distribuídos). De posse de tal padronização, tanto fabricantes como usuários foram capazes de produzir implementações da interface MPI de forma eficiente, aproveitando as vantagens específicas de cada arquitetura computacional. Além disto, programas paralelos, baseados na MPI, tornaram-se portáteis e poderiam ser compilados com qualquer implementação MPI disponível.

Atualmente, inúmeras implementações do padrão MPI são publicamente disponibilizadas, tais como a MPICH (GROPP *et al*., 1996), e Open MPI (GABRIEL *et al*., 2004) e suportam redes homogêneas e heterogêneas, sistemas de memória compartilhada e distribuída.

4.4.2. O*pen Multi-Processing Application Program Interface* (OpenMP)

Em 1997, quando processadores com vários núcleos de processamentos começaram a se tornar prevalentes, a OpenMP foi criada para direcionar, explicitamente, paralelismo em ambientes computacionais paralelos com memória compartilhada. De forma análoga à padronização da MPI, OpenMP foi definida também por um esforço conjunto de fabricantes de *hardware* e desenvolvedores de software. O principal objetivo na definição do OpenMP foi a especificação de um modelo portátil que provesse programadores com uma interface simples e flexível para a programação paralela em sistemas de memória compartilhada.

O padrão OpenMP consiste em uma série de diretivas de compilação, biblioteca de funções e um conjunto variáveis de ambiente que influenciam a execução de programas paralelos. Tal interface recebe versões em C/C++ e Fortran em todas as arquiteturas disponíveis no mercado, incluindo os sistemas operacionais baseados em Unix e Windows.

4.5. Proposta de Paralelização do Problema de Planejamento Energético

No presente trabalho, para a avaliação da redução do tempo computacional na solução do problema de planejamento energético, propõe-se a utilização de um código com partes que utilizam processamento paralelo.

Uma característica importante da Programação Dinâmica está no fato do cálculo do custo ótimo de cada estado operativo de um estágio poder ser realizado de forma independente dos outros estados do mesmo estágio, o que possibilita a paralelização destes processos.

Desta forma, o fluxograma apresentado anteriormente na Figura 19 pode ser reformulado utilizando processamento paralelo. A Figura 24 mostra o fluxograma da proposta a ser avaliada, onde observa-se que, no Bloco 1, a obtenção dos custos dos estados operativos é realizada utilizando processos paralelos.

A estrutura utilizada é a SPMD (do inglês *"Single Program Multiple Data"*), onde os processadores executam o mesmo código, variando o dado de entrada processado por cada um deles.

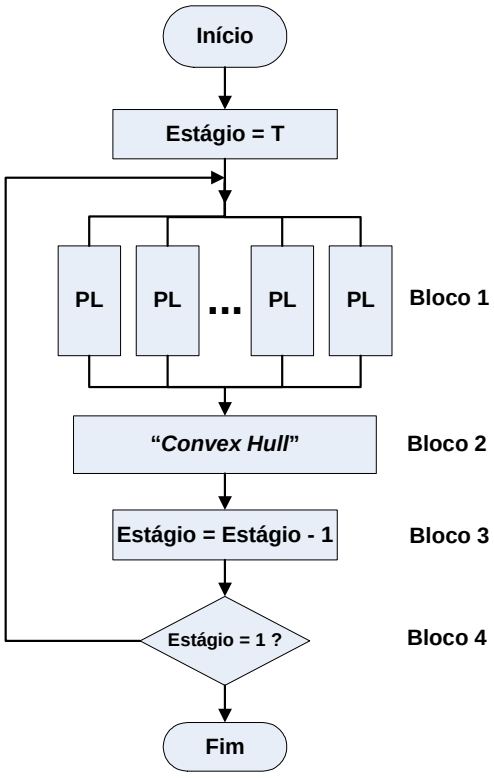

Figura 24: Algoritmo da obtenção das funções de custo futuro utilizando fechos convexos e processamento paralelo