Implementação de algoritmo paralelo para simulação de condução de calor em objeto bidimensional utilizando o método de diferenças finitas.

Flávio Luiz dos Santos de Souza
Prof. MSc. Rodrigo Daniel Malara
Centro Universitário de Araraquara – UNIARA
Departamento das Ciências da Administração e Tecnologia
Curso Engenharia de Computação

RESUMO

A computação de alto desempenho tem sido utilizada como recurso para resolver problemas de grande complexidade em que na maioria das vezes não se tem um poder computacional disponível. Apesar de se tornarem áreas distintas de um mesmo seguimento, são difíceis de separar no conceito ou na analise de computação paralela hardware e software. Contudo, as super maquinas apresentam dificuldade para resolver vários quebra-cabeças da física e da matemática, como a previsão climática em uma escala inferior a 50 km^2, em um tempo hábil. Com o objetivo de utilizar programação concorrente, este trabalho propõe estudar os paradigmas de sistemas distribuídos, que permite utilizar toda gama de potencial da maquina disponível, exigindo assim o domínio das diferentes configurações de hardware existente no mercado e o conceito que envolve a distribuição de tarefas e a comunicação interprocessos a fim de simular um problema físico. Realizando uma simulação de condução de calor em uma chapa metálica, através de cálculos matemáticos, deve-se gerar um speedup linear e positivo com programação seqüencial e outra concorrente. De forma que o resultado apresente um aumento da performance que seja diretamente proporcional ao aumento do recurso computacional empregado na execução.

***Palavras chaves**: alto desempenho, concorrente, distribuída*

ABSTRACT

Techniques for high performance computing and concurrent programming are used to solve problems of great complexity and therefore high computational cost. Existing computers are not able to produce results in time to various problems of physics and mathematics, for example, climate prediction on scales smaller than 50 km2.
Uses the principles of concurrent programming involves knowledge about programming that involves the distribution of tasks and interprocess communication and this paper is to study the different hardware configurations on the market for high performance computing, to study the paradigms of concurrent programming allowing the developer to use the computational power available for such equipment and produce a proof of concept using the knowledge acquired. This prototype needs to get linear speedup positive and so the performance increase is directly proportional to the increase of computational resources used for its implementation.

Keywords: High performance; Concurrent.

1 - INTRODUÇÃO

Técnicas de computação de alto desempenho e programação concorrente são utilizadas para resolver problemas de grande complexidade e conseqüentemente alto custo computacional. Os computadores atuais não são capazes de produzir resultados em tempo hábil para vários problemas da física e da matemática, por exemplo, a previsão climática em escalas menores que 50 km^2.

As utilizações de princípios de programação concorrente envolvem conhecimentos adicionais sobre programação que envolve a distribuição de tarefas e a comunicação interprocessos e este trabalho se propõe a estudar as diferentes configurações de hardware existentes no mercado para computação de alto desempenho, estudar os paradigmas de programação concorrente que permitam ao desenvolvedor utilizar o poder computacional disponibilizado por tais equipamentos e produzir uma prova de conceito utilizando o conhecimento adquirido.

Tal protótipo precisa obter speedup positivo e linear de forma que o aumento de performance seja diretamente proporcional ao aumento de recursos computacionais empregados para a sua execução.

2 - OBJETIVO

- Produzir uma revisão bibliográfica acerca do hardware e técnicas de programação e frameworks disponíveis para elaborar programas concorrentes adequados para computadores de alto desempenho.

- Implementar um programa paralelo usando um framework de paralelização para simular o fenômeno físico *"Condução de calor"* em uma chapa metálica utilizando o método numérico de diferenças finitas.

- Criar representações gráficas da chapa através de imagens antes e após a aplicação do algoritmo paralelo.

- Calcular o speedup da solução paralela.

3 - METODOLOGIA

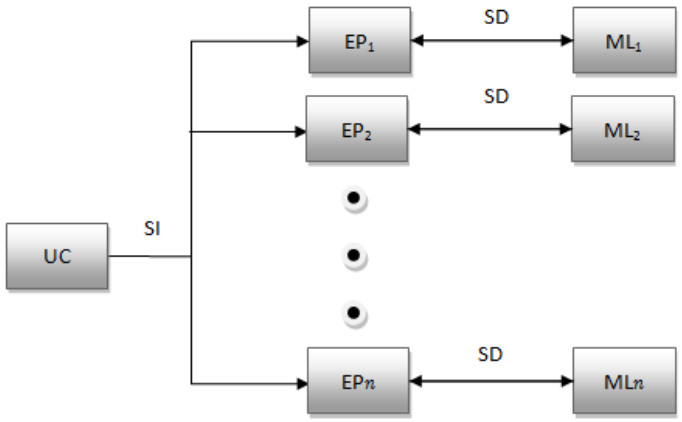
A metodologia desenvolvida tem como objetivo o estudo de computação de alto desempenho através da simulação de transferência de calor por diferenças finitas. Para a simulação é importante conhecer a *Lei de Fourier*, arquitetura SIMD (Single Instruction Mutiple Data) e em um nível de abstração maior SPMD (Single Program Multiple Data).

A taxonomia de Flynn abrange quatro modelos de arquitetura SISD, SIMD, MISD e MIMD o qual todos são baseados nos fluxos de instruções e dados. A proposta de Flynn é ainda a forma mais comum de classificar sistemas de processamento paralelo.

Flynn's Taxonomy	**Single Instruction**	**Multiple Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

SIMD *(Single Instruction Multiple Data)*

Esta classificação corresponde ao processamento de vários dados sob o comando de apenas uma instrução. Cada elemento de processamento tem uma memória de dados associada. Nesta classe estão os processadores vetoriais e matriciais.



Outros modelos de programação

Os principais modelos de parametrização são o *Message passing, DataParallelism* e *Shared Memory*, cada um desse paradigma tem sua flexibilidade e mecanismos de integração de tarefa.

- *Message passaing*: certamente é o modelo mais utilizado hoje. Cria-se varias tarefas com nome único, cada uma encapsula seus dados locais. As tarefas enviam e recebem mensagens de outras tarefas, fazendo assim a comunicação entre si.

- *DataParallelism*: o modelo de paralelismo é atingido na execução em cada processador de uma parte de um único processo distribuído, definido assim uma arquitetura SIMD.

- *Shared Memory:* compartilhando de um mesmo espaço de memória onde é possível ler e escrever de forma assíncrona. A grande vantagem deste modelo é a agilidade de manipulação de dados, entretanto o seu gerenciamento se tornar algo custoso e difícil.

Neste trabalho utiliza o paralelismo de dados e para isso usamos o OpenMP.

Parametrização com OpenMP

Apesar de se tornarem áreas distintas de um mesmo seguimento, hardware e software são difíceis de separar no conceito ou na analise de computação paralela de alto desempenho. O excelente desempenho do paralelismo depende de ambos, não ajudará em nada ter uma boa arquitetura paralela se não tiver um bom proveito do algoritmo e vice-versa!

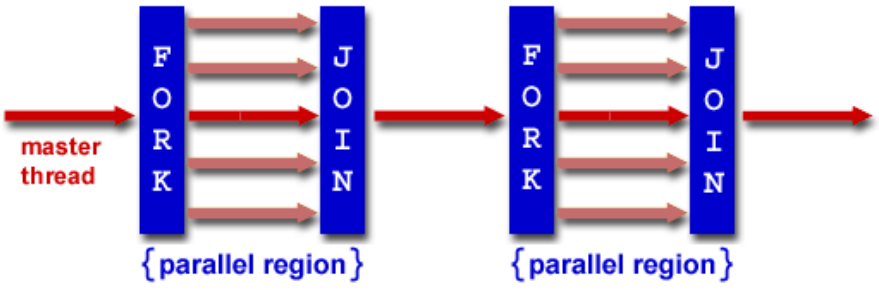
Neste projeto utilizaremos *OpenMP*, que é uma API de parametrização de memória compartilhada (todas tarefas dividem o mesmo endereçamento de memória, sendo executa de forma assíncrona).

A arquitetura do OpenMP em SPMD, cada processo roda exatamente o mesmo programa (embora algumas partes do programa sejam executadas somente pelo processo mestre), mas com um conjunto diferente de dados. Dessa forma, os dados são divididos pelo processo mestre e distribuídos entre os processos escravos, cada processo receberá informações diferentes.

Formada por um conjunto de diretivas compiláveis interpretadas em C/C++ e Fortran, permitindo total controle da parametrização, o framework utiliza o modelo de execução fork-join:

FORK: O thread mestre cria um time de segmentos paralelos. As instruções que estão dentro da construção da região paralela são executadas em paralelo pelas diversos threads do time

JOIN*:* No termino do processo paralelo é sincronizado todas as informações compartilhadas deixando-as somente ativa o *master thread.*



Toda a programação OpenMP é interpretado por diretivas de compilação, o prefixo das diretivos em Fortran são diferentes das C/C++ e para este caso deve-se conter *#pragma omp*, que indica ao compilador que o troche de código seguinte deve ser gerenciado pela API.

- **#pragma omp master**: identifica uma seção que deve ser executado apenas por um thread mestre.
- **#pragma omp parallel**: instrui explicitamente para o compilador o código que deverá ser parametrizado.
- **#pragma omp parallel for**: informa ao compilador que deve-se distribuir as iterações por diferentes threads

- **shared**: todos os dados , assinalados como shared, serão compartilhado. Deixando-o visível e acessível por todos os threads.

- **private**: cada dado, assinalado com private, terá uma copia local e será usado temporariamente em cada região paralel, ate o fim da thread em que a pertence.

- **schedule**: Muito utilizado em loops, são atribuídos de acordo com um dos 3 tipos agendamento (static, dynamic e guided)

    - Static: As iterações serão divididas igualmente entre os threads, mas é necessário definir um padrão de de loop agendados.
    - Dynamic: Uma vez que o thread termina uma iteração uma nova é atribuída a ele.
    - Guided: Assim como a Dynamic, uma nota iteração é atribuída a um thread, mas uma quantidade mínima deve ser especificada.

O principal objetivo da parametrização é diminuir o tempo de processamento comparado com uma versão serial do algoritmo. A métrica mais utilizada é o fator *speed-up*, representa o ganho de velocidade de processamento que uma aplicação apresenta diante a *n* processadores. A equação que define a métrica é:

$$Sn = \frac{Ts}{Tn} \qquad \begin{matrix} Ts - Temp\ Serial \\ Tn - Tempo\ Paralelo \end{matrix}$$

Porem nem todo o código são parametrizados, é possível de se encontrar trecho que são executados por um único thread para este caso é utilizado a *Lei de Amdahl*.

4 - DESENVOLVIMENTO

Uma chapa metálica em temperatura ambiente transmite o calor de uma fonte constante. Qual seria a temperatura em um ponto específico num dado tempo?

Partindo do questionamento descrito, dividiremos a placa em uma matriz 5x5, assim é possível determinar o valor da temperatura em uma dada célula dessa matriz. Quanto maior for à dimensão da matriz maior será a precisão do resultado logo exigirá um poder computacional maior.

Considerando que somente a primeira linha da matriz, que esta em contato direto com a fonte de calor, tem uma temperatura inicial de 100 graus e o restante a 0 grau. Adotando também que a cada iteração é o tempo necessário para aquecer a placa.

A nova temperatura de uma dada célula sofre influencia do calor de todas ao seu redor, portanto a somatória da temperatura dos elementos com a temperatura dela mesma, dividido pela quantidade de células participante dará a temperatura atual da célula.

100	100	100	100	100
25	5	α	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

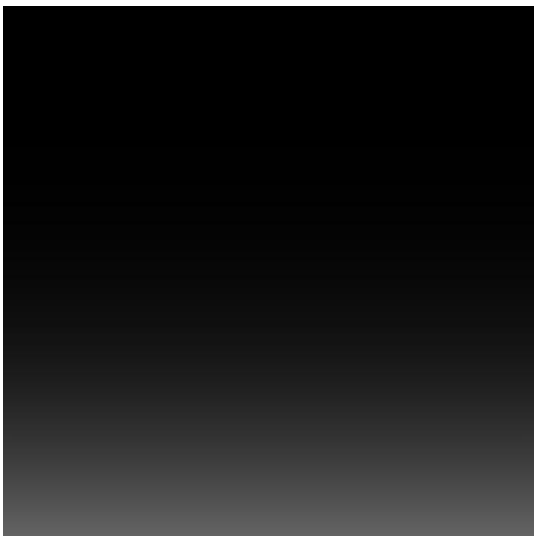
Definiremos a temperatura da célula α, todas as células sinaladas em não vermelho entra no calculo, sabendo que a temperatura atual de α é de 0°. Logo temos:

$$\alpha = \frac{100 + 5 + \alpha + 0 + 0}{5} = \frac{100 + 5 + 0 + 0 + 0}{5} = 21$$

Tanto para a codificação seqüencial quanto para paralela, o cenário de simulação é o mesmo uma matriz de 700 x 700 com iterações 100.000. O resultado esperado é de que ambas apresentem a mesma resposta (ambas as matrizes com a mesma temperatura nas mesmas células), mas com tempo de processamento diferente (parallel timer < seqüencial timer), pois as tarefas de calculo serão distribuídas em Threads na simulação paralela e na seqüencial será um único thread.

5 - RESULTADOS

Para comprovar o resultado são criadas imagens PNG que representaram as matrizes geradas em cada cenário. O valor numérico de cada célula da matriz representa uma tonalidade entre as cores pretas e brancas. Quanto menor a temperatura mais próxima do branco ela estará e quanto maior a temperatura mais próxima do preto ela ficará.



Se executar a simulação em processadores mais atuais como o Core i3, Core i5 ou Core i7, o *speedup* apresentado será menor, pois o número de threads é maior do que usado na simulação concorrente, a configuração utilizada é equivalente a quantidade de núcleos existentes no processador em que a simulação ocorre (processador do ambiente Core Duo).

6 - CONSIDERAÇÔES FINAIS

Apesar de se tornarem áreas distintas de um mesmo seguimento, hardware e software são difíceis de separar no conceito ou na analise de computação paralela de alto desempenho.

O excelente desempenho de do paralelismo depende de ambos, não ajudará em nada ter uma boa arquitetura paralela se não tiver um bom proveito do algoritmo e vice-versa. Contudo, os resultados obtidos foram positivos e agraveis, entretanto deve-se enfatizar que nem sempre é possível obter um *speed-up* perfeito. Pois o mesmo depende diretamente da relação de computação e comunicação, deve-se ter um trabalho computacional de maior tempo em relação ao gasto com comunicação entre processadores.

7 - BIBLIOGRAFIA

FOSTER, I. *Designing and Building Parallel Programs*. Addison-Wesley, Inc. 1995.

TANENBAUM, A.S. - *Organização Estruturada de Computadores*. Prentice Hall, Inc.2007

STALLINGS, W. – *Arquitetura e Organização de Computadores. Prentice Hall*, Inc.2002

TANENBAUM, A.S. - *Sistemas Operacionais Modernos, Prentice Hall*, São Paulo , 2003, 2ª. ed.

COULOURIS, G., DOLLIMORE, J., KINBERG. *Sistemas Distribuídos: Conceitos e Projetos*. 1a. edição. Bookmann, 2007

TANENBAUM, A. VAN STEEN, *M. Sistemas Distribuídos: Princípios e Paradigmas*. 2ª. Edição. Prentice-Hall, 2007