São Carlos, 7 de julho de 2022

SSC0903 - Computação de Alto Desempenho

Grupo G07

Nome	NUSP
Che Fan Pan	11200421
Eduardo Cavalari Valença	11234381
Marcos Vinícius Firmino Pietrucci	10770072

Descrição do projeto do algoritmo paralelo - Método Iterativo de Jacobi-Richardson

1 - Introdução

A realização do projeto de algoritmo paralelo utilizando o Método Iterativo de Jacobi-Richardson para encontrar as soluções das N equações de uma matriz quadrada, será descrita pela metodologia PCAM, proposta por Ian Foster.

São 4 etapas que determinam como o problema será resolvido utilizando computação paralela. A primeira chama-se Particionamento, em que entendemos o problema e atribuímos tarefas concorrentes para resolvê-lo. A seguir é a Comunicação, que descreve como essas tarefas irão se comunicar durante o algoritmo. Após isso, é considerado na Aglomeração qual será a arquitetura da máquina utilizada para a execução do código, além da atribuição das tarefas a processos ou threads. Por fim, no Mapeamento, é definido como os processos serão distribuídos pelos elementos de processamento da máquina alvo.

2 - Funcionamento do método

2.1 - Transformação linear

O método funciona primeiramente transformando uma sistema linear $Ax = b$ em $x = Cx' + g$. Podemos tomar como exemplo a matriz abaixo:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Separando cada x_i utilizando a diagonal principal e assumindo todos os $a_{ii} != 0$.

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}\left(b_1 - \left(a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n\right)\right) \\ x_2 = \dfrac{1}{a_{22}}\left(b_2 - \left(a_{21}x_1 + a_{23}x_3 + \ldots + a_{2n}x_n\right)\right) \\ \quad . \qquad\quad . \qquad\qquad . \\ \quad . \qquad\quad . \qquad\qquad . \\ x_n = \dfrac{1}{a_{nn}}\left(b_n - \left(a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{n,n-1}x_{n-1}\right)\right) \end{cases}$$

Agora podemos definir C e g de forma a corresponder a equação acima.

$$C = \begin{pmatrix} 0 & -\dfrac{a_{12}}{a_{11}} & -\dfrac{a_{13}}{a_{11}} & \ldots & -\dfrac{a_{1n}}{a_{11}} \\ -\dfrac{a_{21}}{a_{22}} & 0 & -\dfrac{a_{23}}{a_{22}} & \ldots & -\dfrac{a_{2n}}{a_{22}} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ -\dfrac{a_{n1}}{a_{nn}} & -\dfrac{a_{n2}}{a_{nn}} & -\dfrac{a_{n3}}{a_{nn}} & \ldots & 0 \end{pmatrix} \qquad g = \begin{pmatrix} \dfrac{b_1}{a_{11}} \\ \dfrac{b_2}{a_{22}} \\ . \\ . \\ \dfrac{b_n}{a_{nn}} \end{pmatrix}$$

Vamos começar a atribuir valores aos x_i. Dado uma aproximação inicial para os x_i denominado $x^{(0)}{}_i$, podemos obter $x^{(1)}$, … , $x^{(k)}$ por meio da relação recursiva $x^{(k+1)} = Cx^{(k)} + g$.

$$\begin{cases} x^{(k+1)}{}_1 = \dfrac{1}{a_{11}}\left(b_1 - \left(a_{12}x^{(k)}{}_2 + a_{13}x^{(k)}{}_3 + \ldots + a_{1n}x^{(k)}{}_n\right)\right) \\ x^{(k+1)}{}_2 = \dfrac{1}{a_{22}}\left(b_2 - \left(a_{21}x^{(k)}{}_1 + a_{23}x^{(k)}{}_3 + \ldots + a_{2n}x^{(k)}{}_n\right)\right) \\ \quad . \qquad\quad . \qquad\qquad . \\ \quad . \qquad\quad . \qquad\qquad . \\ x^{(k+1)}{}_n = \dfrac{1}{a_{nn}}\left(b_n - \left(a_{n1}x^{(k)}{}_1 + a_{n2}x^{(k)}{}_2 + \ldots + a_{n,n-1}x^{(k)}{}_{n-1}\right)\right) \end{cases}$$

2.2 - Critério de convergência

Como dito anteriormente, uma das etapas da transformação linear requer dividir o termo $(b_n - \sum_{i=1}^{n} [a_{ni}\ x_i])$ por a_{ii}, com um n fixado. Dessa forma, um dos critérios necessários é que nenhum dos elementos da diagonal principal seja nulo.

O outro critério para que seja possível convergir com o método de Gauss-Jacobi é se, para toda linha da matriz **A**, houver a seguinte condição:

$$\alpha_k = (\sum_{j=1,\, j \neq k}^{n} |a_{kj}|)/a_{kk}$$
$$\alpha_k < 1$$

Ou seja, a soma dos valores absolutos dos coeficientes de uma linha da matriz **A** devem ser menores do que o coeficiente do elemento que pertence à diagonal principal dessa mesma linha.

3 - Metodologia PCAM

3.1 - Estudando o problema

Antes de aplicar a metodologia PCAM é necessário entender o funcionamento do algoritmo proposto e todas suas etapas. Isso permitirá que analisemos com maior eficácia durante o particionamento.

1. Criação da matriz **A** e vetor **B**, ambos pseudo-aleatórios de forma que o método de Jacobi-Richardson convirja.
2. Criação do vetor solução inicializado com uma aproximação inicial.
3. Realizar a soma dos elementos de uma linha multiplicada pelo vetor solução, excluindo o elemento da diagonal principal.
4. Aplicar a equação do método para cada elemento do vetor solução.
5. Encontrar a maior diferença entre o valor da iteração passada de qualquer elemento do vetor solução com a nova iteração.
6. Encontrar o maior valor do vetor solução da nova iteração para poder calcular, junto com a maior diferença, a condição de parada do método.
7. Verificar se a condição de parada é inferior a uma margem pré-estabelecida, caso positivo, o método é concluído, caso contrário retorna ao passo 3.
8. Calcular a diferença do valor encontrado (multiplicação da matriz **A** pelo vetor solução) e os valores do elementos do vetor **B**.

3.2 - Particionamento

Vamos imaginar o menor grão possível para cada um dos pontos mencionados no estudo do problema.

Para o ponto 1, podemos considerar a criação de um valor aleatório atribuída a uma tarefa. Como é necessário realizar essa tarefa para cada elemento da matriz, a qual possui dimensões NxN, então se totalizam N^2 tarefas.

Analisando o ponto 2, é necessário realizar uma atribuição do valor inicial estipulado para cada elemento do vetor, considerando um vetor de N elemento, logo, serão geradas N tarefas.

Para o ponto 3, podemos aplicar uma tarefa para cada um dos elementos da matriz. Para cada linha, será realizada uma operação de redução de soma, lembrando de desconsiderar a operação caso o elemento pertença à diagonal principal. Por consequência, serão gerados um total de N^2 tarefas, considerando uma matriz quadrada NxN.

Já no ponto 4 cada tarefa seria responsável por um elemento do vetor solução, ou seja, um cálculo da equação de Jacobi. Esta tarefa manipula apenas um elemento da diagonal principal, um elemento do vetor **B** e o elemento do vetor solução de posição correspondente, respeitando o método.

Agora pensando na etapa 5 temos um vetor de diferença em módulo entre a iteração atual e a iteração passada, uma para cada elemento. Assim, cada tarefa fica responsável por cada elemento desse vetor (N tarefas), a qual aplica uma operação de redução de máximo de ordem logarítmica para encontrar a maior mudança da iteração.

O particionamento da etapa 6 é praticamente idêntico ao da etapa 5. Cada tarefa fica responsável por cada elemento desse vetor solução (N tarefas), a qual aplica uma operação de redução de máximo de ordem logarítmica para encontrar o maior valor do vetor.

Para a etapa 7, como o algoritmo já dispõe dos valores máximos encontrados nas etapas 5 e 6, será necessário apenas uma tarefa, a qual realizará a comparação desse valor com a tolerância exigida.

Por fim, na etapa 8 é necessário realizar uma multiplicação para cada valor da linha da matriz com seu respectivo valor do vetor solução, totalizando N tarefas.

Considerando a descrição do problema anterior e particionamento realizado, consideramos como passível de paralelização os pontos 3, 4 , 5 e 6.

3.3 - Comunicação

As comunicações entre as tarefas são responsáveis por fornecer os dados de entrada para tarefas e por recuperar os resultados das computações feitas. Logo, se faz necessário a existência de uma tarefa encarregada de organizar o recebimento e envio de mensagens, além de ser encarregado do andamento do algoritmo.

Esta tarefa ficará responsável por fazer uma comunicação ponto a ponto para cada uma das demais. Além disso, será também enviado ponto a ponto seus respectivos valores do vetor **B**. Por fim, será realizado o *broadcast* do vetor solução, já que esse precisa estar presente por completo em todas as tarefas.

Detalhando um pouco mais, cada tarefa recebe o seu respectivo elemento de **A** de maneira independente, por meio de uma comunicação ponto a ponto com uma tarefa arbitrariamente eleita para tal função. Em seguida, haverá uma sequência de comunicações locais ponto a ponto entre as tarefas, de forma a gerar a redução das etapas 3, 5 e 6.

A cada iteração do método, as tarefas deverão realizar uma operação de redução total, com seus maiores valores das etapas 5 e 6 encontrados localmente e também realizarão o envio e recebimento dos valores do vetor solução para sua atualização.

3.4 - Aglomeração

A nossa plataforma alvo é um cluster de computadores da USP, ou seja, um ambiente multiprocessado e de memória distribuída. Além disso, serão utilizadas várias threads por nó de processamento do cluster, fazendo o uso, também, de memória compartilhada. Nesse contexto, deve-se tentar amenizar ao máximo a interação entre nós diferentes de forma que a comunicação não se torne um gargalo no desempenho do sistema.

Para a etapa 3 podemos aglomerar as tarefas de cada elemento em uma única tarefa responsável por toda uma linha da matriz, resultando em N *threads* (uma para cada linha).

Cada *thread* deve realizar uma varredura sequencial da linha (favorecendo o uso de cache), somando os valores dos elementos multiplicados pelo respectivo valor do vetor solução, excluindo o elemento da diagonal principal.

Para a etapa 4, utilizaremos a mesma lógica da etapa 3, de forma que cada thread aplique a equação do método para a(s) linha(s) a(s) qual(is) é responsável. Como estamos utilizando em menor escala a memória compartilhada de um mesmo nó, os riscos de falso compartilhamento diminuem.

Para a etapa 5 e 6 podemos seguir conforme descrito no particionamento e gerar N *threads* também aglomerando as *threads* de cada elemento para cada uma das N linhas da matriz.

3.5 - Mapeamento

Vamos agora iniciar a atribuição de *threads* de cada uma das etapas. As máquinas do cluster possuem um processador multi-core, o que nos dá boa flexibilidade.

Para a etapa 3, podemos dividir toda a carga em blocos de *threads* de forma simétrica para cada um dos nós do cluster (C), um grão elevado para compensar a alta carga de comunicação. Cada um dos nós ficará responsável por N/C blocos de *threads*, sendo N o número de linhas da matriz. Esses blocos serão distribuído em uma razão de $N/(C * Threads)$. Com a construção do código, testes poderão ser efetuados a fim de verificar quantas atribuições por máquinas é ideal, podendo ser dois, quatro ou oito *threads* para cada máquina *quad-core.*

A etapa 4 será inserida no mesmo mapeamento da etapa 3, impedindo assim o gasto de recursos para criação de novas *threads* dentro de cada nó do cluster.

As etapas 5 e 6 devem ser executadas simultaneamente para evitar novas criações de *threads*. O processo de mapeamento aqui será análogo à etapa 3, dividindo o número de

threads por nós e atribuindo a cada processador um número de 2 a 8 blocos de *threads* (será variado para testes).

4 - Discussões

Durante a confecção do trabalho foi discutido sobre a paralelização da geração da matriz **A**. Por meio de testes de mesa se constatou que essa paralelização só traria algum benefício ao código para valores de N superiores a 10^4, em que começamos a observar a ocorrência da Lei de Amdahl. A partir disso, o tempo de execução do programa paralelo se torna limitado por sua parte sequencial.

Entretanto, o grupo discutiu e chegou à conclusão de que o foco do problema não seria na construção da matriz. Neste trabalho os valores dos coeficientes são gerados de forma pseudo-aleatória, portanto, não representam uma situação real.