

Matemática Numérica Paralela

Pedro H A Konzen

22 de janeiro de 2021

Licença

Este trabalho está licenciado sob a Licença Atribuição-CompartilhaIgual 4.0 Internacional Creative Commons. Para visualizar uma cópia desta licença, visite http://creativecommons.org/licenses/by-sa/4.0/deed.pt_BR ou mande uma carta para Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Prefácio

Nestas notas de aula são abordados tópicos sobre computação paralela aplicada a métodos numéricos. Como ferramentas computacionais de apoio, exploramos exemplos de códigos em C/C++.

Agradeço a todos e todas que de modo assíduo ou esporádico contribuem com correções, sugestões e críticas. :)

Pedro H A Konzen

Sumário

Capa	i
Licença	ii
Prefácio	iii
Sumário	iv
1 Introdução	1
2 Computação Paralela (MP)	4
2.1 Olá, Mundo!	4
Respostas dos Exercícios	5
Referências Bibliográficas	6

Capítulo 1

Introdução

A computação paralela e distribuída é uma realidade em todas as áreas de pesquisa aplicadas. À primeira vista, pode-se esperar que as aplicações se beneficiam diretamente do ganho em poder computacional. Afinal, se a carga (processo) computacional de uma aplicação for repartida e distribuída em $n_p > 1$ processadores (**instâncias de processamentos**, *threads* ou *cores*), a computação paralela deve ocorrer em um tempo menor do que se a aplicação fosse computada em um único processador (em serial). Entretanto, a tarefa de repartir e distribuir (**alocação de tarefas**) o processo computacional de uma aplicação é, em muitos casos, bastante desafiadora e pode, em vários casos, levar a códigos computacionais menos eficientes que suas versões seriais.

Repartir e distribuir o processo computacional de uma aplicação sempre é possível, mas nem sempre é possível a computação paralela de cada uma das partes. Por exemplo, vamos considerar a [iteração de ponto fixo](#)

$$x(n) = f(x(n-1)), \quad n \geq 1, \quad (1.1)$$

$$x(0) = x_0, \quad (1.2)$$

onde $f : x \mapsto f(x)$ é uma função dada e x_0 é o ponto inicial da iteração. Para computar $x(100)$ devemos processar 100 vezes a iteração (1.1). Se tivéssemos a disposição $n_p = 2$ processadores, poderíamos repartir a carga de processamento em dois, distribuindo o processamento das 50 primeiras iterações para o primeiro processador (o processador 0) e as demais 50 para o segundo processador (o processador 1). Entretanto, pela característica do processo iterativa, o processador 1 ficaria ocioso, aguardando o processador 0 computar $x(50)$. Se ambas instâncias de processamento compartilharem

a mesma memória computacional (**memória compartilhada**), então, logo que o processador 0 computar $x(50)$ ele ficará ocioso, enquanto que o processador 1 computará as últimas 50 iterações. Ou seja, esta abordagem não permite a computação em paralelo, mesmo que reparta e distribua o processo computacional entre duas instâncias de processamento.

Ainda sobre a abordagem acima, caso as instâncias de processamento sejam de **memória distribuída** (não compartilhem a mesma memória), então o processador 0 e o processador 1 terão de se comunicar, isto é, o processador 0 deverá enviar $x(50)$ para a instância de processamento 1 e esta instância deverá receber $x(50)$ para, então, iniciar suas computações. A **comunicação** entre as instâncias de processamento levanta outro desafio que é necessidade ou não da **sincronização** () eventual entre elas. No caso de nosso exemplo, é a necessidade de sincronização na computação de $x(50)$ que está minando a computação paralela.

Em resumo, o design de métodos numéricos paralelos deve levar em consideração a **alocação de tarefas**, a **comunicação** e a **sincronização** entre as instâncias de processamentos. Vamos voltar ao caso da iteração (1.1). Agora, vamos supor que $x = (x_0, x_1)$, $f : x \mapsto (f_0(x), f_1(x))$ e a condição inicial $x(0) = (x_0(0), x_1(0))$ é dada. No caso de termos duas instâncias de processamentos disponíveis, podemos computar as iterações em paralelo da seguinte forma. Iniciamos distribuindo x às duas instâncias de processamento 0 e 1. Em paralelo, a instância 0 computa $x_0(1) = f_0(x)$ e a instância 1 computa $x_1(1) = f_1(x)$. Para computar a nova iterada $x(2)$, a instância 0 precisa ter acesso a $x_1(1)$ e a instância 1 necessita de $x_0(1)$. Isto implica na sincronização das instâncias de processamentos, pois uma instância só consegue seguir a computação após a outra instância ter terminado a computação da mesma iteração. Agora, a comunicação entre as instâncias de processamento, depende da arquitetura do máquina. Se as instâncias de processamento compartilham a mesma memória (memória compartilhada), cada uma tem acesso direto ao resultado da outra. No caso de uma arquitetura de memória distribuída, ainda há a necessidade de instruções de comunicação entre as instâncias, i.e. a instância 0 precisa enviar $x_0(1)$ à instância 1, a qual precisa receber o valor enviado. A instância 1 precisa enviar $x_1(1)$ à instância 0, a qual precisa receber o valor enviado. O processo segue análogo para cada iteração até a computação de $x(100)$.

A primeira parte destas notas de aula, restringe-se a implementação de métodos numéricos paralelos em uma arquitetura de memória compartilhada. Os exemplos computacionais são apresentados em linguagem C/C++ com a

interface [OpenMP](#). A segunda parte, dedica-se a implementação paralela em arquitetura de memória distribuída. Os códigos C/C++ são, então, construídos com a interface [OpenMPI](#).

Capítulo 2

Computação Paralela (MP)

2.1 Olá, Mundo!

```
1 | #include <iostream>
2 | #include <omp.h>
3 |
4 | using namespace std;
5 |
6 | int main(int argc, char *argv[]) {
7 |
8 | #pragma omp parallel
9 |     {
10 |         int id = omp_get_thread_num();
11 |         cout << "Processo" << id << ": Olá, Mundo!\n";
12 |     }
13 |
14 |     return 0;
15 | }
```

Em construção ...

Resposta dos Exercícios

Referências Bibliográficas

- [1] D.P. Dimitri and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.
- [2] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2. edition, 2003.