Capítulo 1

Análise de Desempenho

O objetivo deste capítulo é estudar o desempenho de implementações paralelas em termos de tempo de execução, speedup, eficiência e overhead paralelo.

1.1 Tempo de execução, Speedup e eficiência

Na análise de um algoritmo sequencial podem ser investigadas características como o número de vezes que cada parte do algoritmo deve ser executada e a quantidade de memória necessária.

Para medir o custo de execução de um algoritmo sequencial é comum definir uma função de custo ou função de complexidade *T*. Assim, *T(n)* é a medida do tempo necessário para executar um algoritmo para um problema de tamanho *n*.

A função de complexidade de tempo *T(n)* na realidade não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.

O tempo de execução sequencial pode ser impactado pelos seguintes fatores:

- tamanho e outras características da entrada
- hardware que está sendo usado
- linguagem de programação e compilador

Na análise de desempenho de programas paralelos, o tempo de execução depende de duas variáveis: tamanho da entrada (n) e número de processos ou threads (p). Assim, a função que denota desempenho é *T(n,p)*. O tempo de execução é o tempo decorrido desde o momento que o primeiro processo ou thread começa a execução até quando o último processo completa a execução do seu último comando.

Essa definição implica que se múltiplos processos estão executando em um único elemento de processamento físico, o tempo de execução poderá ser substancialmente maior do que se os processos estão executando em elementos de processamento físicos separados. Em geral, assume-se que cada processo está executando em um elemento de processamento físico separado.

Quando o desempenho de um programa paralelo é discutido, o objetivo é comparar o comportamento do programa paralelo em relação ao sequencial. Assim, as medidas mais usadas são speedup e eficiência.

Speedup é a razão entre o tempo de execução de uma solução sequencial e o tempo de execução paralela de um programa.

$$\boxed{Speedup = \frac{tempo\ \ de\ \ execucao\ \ sequencial}{tempo\ \ de\ \ execucao\ \ paralela}} \tag{1.1}$$

Assim, o speedup alcançado por um algoritmo paralelo executando sobre p elementos de processamento é a taxa entre o tempo gasto pelo computador paralelo para executar o algoritmo para 1 elemento de processamento e o tempo gasto pelo mesmo computador paralelo executar o correspondente algoritmo paralelo usando p elementos de processamento.

As operações realizadas por um algoritmo paralelo podem ser classificadas em 3 categorias:

- Computações que podem ser realizadas sequencialmente;
- Computações que podem ser realizadas em paralelo;
- *Overhead* paralelo (operações de comunicação e computações redundantes).

Com essas categorias em mente, podemos produzir um modelo de speedup. Seja $S(n,p)$ o speedup alcançado para resolver um problema de tamanho n em p elementos de processamento, $Ts(n)$ denota a porção sequencial da computação, $Tsp(n)$ denota a porção da computação que pode ser executada em paralelo e $Tov(n,p)$ denota o tempo requerido pelo overhead paralelo.

Um programa sequencial, executando em um único elemento de processamento, pode realizar uma única computação de cada vez. Então, o programa gastará o tempo $Ts(n) + Tsp(n)$ para executar as computações necessárias. Um programa sequencial não requer comunicações, então a expressão de tempo de execução sequencial não tem o termo $Tov(n,p)$.

$$\boxed{Tempo\ sequencial = Ts(n) + Tsp(n)} \tag{1.2}$$

Vamos considerar o melhor tempo de execução paralelo possível. A porção sequencial de computação não pode beneficiar de paralelização. Ela contribuiu com $Ts(n)$ do tempo de execução do programa paralelo, independente de quantos elementos de processamento estejam disponíveis. No melhor caso, a porção de computação que pode ser executada em paralelo é dividida igualmente entre os p elementos de processamento. Neste caso, o tempo necessário para realizar essas operações é $\frac{Tsp(n)}{p}$. Finalmente, deve-se adicionar o tempo $Tov(n,p)$ para comunicação entre elementos de processamento requerida pelo programa paralelo.

$$\boxed{Tempo\ paralelo = Ts(n) + \frac{Tsp(n)}{p} + Tov(n,p)} \tag{1.3}$$

Numa situação otimista, a porção da computação pode ser dividida perfeitamente entre os elementos de processamento, mas nem sempre isto ocorre. Então, a expressão completa para speedup é:

$$\boxed{S(n,p) \leq \frac{Ts(n) + Tsp(n)}{Ts(n) + \frac{Tsp(n)}{p} + Tov(n,p)}} \tag{1.4}$$

Adicionar elementos de processamento diminui o tempo de computação (pela divisão do trabalho entre os elementos de processamento), mas aumenta o tempo de comunicação. Em um mesmo ponto, o aumento do tempo de comunicação é maior do que o decréscimo do tempo de computação. Neste ponto, o tempo de execução começa a aumentar. Já que o speedup é inversamente proporcional ao tempo de execução, a curva de speedup começa a declinar.

A Figura 1.1 apresenta um algoritmo paralelo não trivial que tem uma componente de computação (barra preta) que é uma função decrescente de um número de elementos de processamento usados e uma componente de comunicação (barra branca) que é uma função crescente do número de elementos de processamento. Para qualquer problema de tamanho fixo há um número ótimo de elementos de processamento que minimiza o tempo de execução global. A eficiência de um programa paralelo é uma medida de utilização do elemento de processamento. É

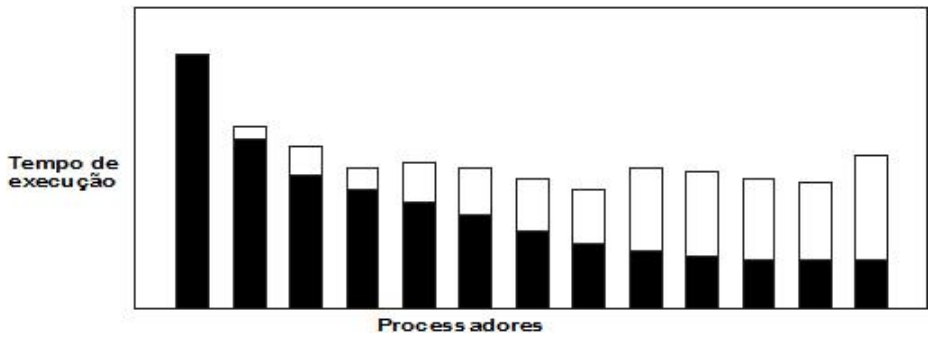

Figura 1.1: Gráfico de tempo de execução x elementos de processamento

definida como a razão entre o speedup e o número de elementos de processamento usados:

$$Eficiencia = \frac{Ts(n)}{p * Tp}$$

$$Eficiencia = \frac{Speedup}{p} = \frac{S(n,p)}{p}$$

De uma maneira mais formal, seja $E(n,p)$ a eficiência de uma computação paralela para resolver um problema de tamanho n em p elementos de processamento.

$$\boxed{E(n,p) = \frac{S(n,p)}{p}} \tag{1.5}$$

Construir uma definição em função do speedup (equação 1.2) ficaria:

$$E(n,p) \leq \frac{Ts(n) + Tsp(n)}{p * (Ts(n) + \frac{Tsp(n)}{p} + Tov(n,p))}$$

$$E(n,p) \leq \frac{Ts(n) + Tsp(n)}{p * Ts(n) + Tsp(n) + p * Tov(n,p)}$$

Já que todos os termos são maiores ou iguais a zero, $0 \leq E(n,p) \leq 1$. Exemplo: Se o algoritmo sequencial conhecido para resolver um problema executa em 8 s., enquanto um algoritmo paralelo resolvendo o mesmo problema executa em 2 s., quando 5 elementos de processamento são usados, então dizemos que o algoritmo paralelo exibe um speedup de 4 com 5 elementos de processamento e tem uma eficiência de 0,8 com 5 elementos de processamento.

$$Speedup = \frac{tempo\ de\ execucao\ sequencial}{tempo\ deexecucao\ paralelo}$$

$$Speedup = \frac{8}{2} = 4$$
$$Eficiencia = \frac{Speedup}{5} = \frac{4}{5} = 0,8$$

1.1.1 Speedup superlinear

Para um dado problema, pode haver mais de um algoritmo sequencial disponível, mas nem todos serem igualmente portáveis para paralelização.

Quando um computador sequencial é utilizado, é natural utilizar o algoritmo sequencial que resolve o problema na menor quantidade de tempo.

Dado um algoritmo paralelo, é justo julgar seu desempenho com relação ao mais rápido algoritmo sequencial que resolva o mesmo problema num único elemento de processamento. Teoricamente, o speedup nunca pode exceder o número de elementos de processamento, p. Se o melhor algoritmo sequencial leva Ts unidades de tempo para resolver um dado problema num único elemento de processamento, então um speedup de p pode ser obtido sobre p elementos de processamento se nenhum dos elementos de processamento gasta mais do que $\frac{Ts}{p}$ unidades de tempo.

Um speedup maior do que p (speedup superlinear) é observado quando o trabalho realizado por um algoritmo sequencial é maior do que na sua formulação paralela ou devido às características de hardware que colocam a implementação sequencial em desvantagem. Por exemplo, a quantidade de dados para um problema pode ser muito grande para caber numa cache de um único elemento de processamento, degradando o seu desempenho. Quando particionado entre diversos elementos de processamento, as partições de dados individuais seriam pequenas o suficiente para caber nas caches dos elementos de processamento.

Exercício: Considere as Figuras 1.2(a) e 1.2(b).

Nas Figuras 1.2(a) e 1.2(b), temos uma árvore de busca, onde o nó colorido representa a solução do problema.

1. Se uma busca sequencial nesta árvore é realizada usando o algoritmo de busca em profundidade, contando somente a visita ao nó na descida na árvore, quanto tempo leva para encontrar a solução se visitar cada nó da árvore leva uma unidade de tempo?

   Como a execução é sequencial, a visita a cada nó ocorre, um após a outra. Assim, do nó raiz até o nó solução serão visitados 12 nós.

2. Assumindo que a árvore foi particionada entre 2 processos, como mostrado na Figura 1.2(b), se ambos os processos realizarem busca em profundidade

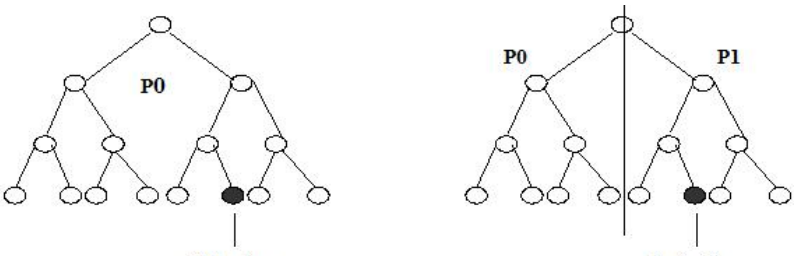

Figura 1.2: Árvores referentes ao algoritmo de busca em profundidade

sobre suas respectivas metades da árvore, quanto tempo leva-se para encontrar a solução? Qual é o speedup? Há um speedup superlinear? Por que? Os processos P0 e P1 começam a execução no nó raiz e visita cada nós da subárvore. Como somente P1 chega à solução, o tempo execução é dado por esse processo, que corresponde ao número de nós visitados até chegar à solução. Assim, serão necessárias 5 unidades de tempo. O speedup é a razão entre o tempo de execução sequencial e o tempo de execução paralela, ou seja, 12/5 =2,5. Esse speedup não é superlinear, pois é maior do que 2. Para haver speedup superlinear é necessário que o speedup seja inferior ao número de processos.

1.2 Lei de Amdahl

É uma forma de expressar um speedup máximo como uma função da quantidade de paralelismo e da fração da computação que é inerentemente sequencial. Considere a expressão que foi derivada anteriormente (1.2):

$$S(n,p) \leq \frac{Ts(n)+Tsp(n)}{Ts(n)+\frac{Tsp(n)}{p}+Tov(n,p)}$$

Como $Tov(n,p) > 0$,

$$S(n,p) \leq \frac{Ts(n)+Tsp(n)}{Ts(n)+\frac{Tsp(n)}{p}+Tov(n,p)} \leq \frac{Ts(n)+Tsp(n)}{Ts(n)+\frac{Tsp(n)}{p}}$$

Seja f porção sequencial da computação.

Em outras palavras, $f = \frac{Ts(n)}{(Ts(n)+Tsp(n))}$.

Então,

$$S(n,p) \leq \frac{Ts(n)+Tsp(n)}{Ts(n)+\frac{Tsp(n)}{p}}$$

$$S(n,p) \leq \frac{\frac{Ts(n)}{f}}{Ts(n)+Ts(n)\frac{(\frac{1}{f}-1)}{p}}$$

$$S(n,p) \leq \frac{\frac{1}{f}}{1+\frac{(\frac{1}{f-1})}{p}}$$

$$S(n,p) \leq \frac{1}{f+\frac{(1-f)}{p}}$$

Assim a lei de Amdahl fica enunciada da seguinte forma:

Seja f a porção sequencial da computação de operações em uma computação que deve ser realizada sequencialmente, onde $0 \leq f \leq 1$.

O speedup máximo alcançável por um computador paralelo com p elementos de processamento realizando computação é

$$\boxed{S \leq \frac{1}{f+\frac{(1-f)}{p}}} \tag{1.6}$$

A lei de Amdahl é baseada na hipótese que tentamos resolver um problema de tamanho fixo tão rápido quanto possível. Ele provê um limite superior do speedup alcançável aplicando um certo número de elementos de processamento para resolver o problema em paralelo. Também pode ser usado para determinar speedup assintótico alcançável quando o número de elementos de processamento aumenta.

Exemplo 1: Suponha que estamos tentando determinar se é viável desenvolver uma versão paralela de um programa para resolver um problema particular. Um benchmarking revela que 90% do tempo de execução é gasto dentro de funções que acreditamos que podem ser executadas em paralelo. O restante dos 10% do tempo execução é gasto em funções que devem ser executadas em um único elemento de processamento. Qual é o speedup máximo que pode ser esperado de uma versão paralela do programa executando em 8 elementos de processamento?

Solução:

Pela lei de Amdahl

$$S \leq \frac{1}{0,1 + \frac{(1-0,1)}{8}} \approx 4,7$$

Devemos esperar um speedup de no máximo 4,7.

Exemplo 2: Se 25% das operações de um programa paralelo devem ser realizadas sequencialmente, qual é o speedup máximo alcançável?

Solução: O speedup máximo alcançável é

$$lim_{p \to \infty} \frac{1}{0,25 + \frac{(1-0,25)}{p}} = 4$$

Exemplo 3: Suponha que seja implementada uma versão paralela de um programa sequencial com complexidade de tempo $\Theta(n^2)$, onde n é o tamanho do conjunto de dados. Assuma que o tempo necessário para entrar com o conjunto de dados e gerar o resultado é $(18000+n)\mu s$. Este constitui a porção de tempo sequencial do programa. A porção computacional do programa pode ser executada em paralelo; seu tempo de execução é $(n^2/100)\mu s$. Qual é o speedup máximo alcançável por este programa paralelo para um problema de tamanho 10.000?

Solução: Pela lei de Amdahl,

$$S \leq \frac{(28.000 + 1.000.000)\mu s}{(28.000 + \frac{1.000.000}{p})}\mu s$$

$$lim_{p \to \infty} \frac{(28.000 + 1.000.000)}{(28.000 + \frac{1.000.000}{p})} = 36,71$$

O speedup máximo alcançável por este programa paralelo para n=10.000 será de 36,71.

1.2.1 Limitações da lei de Amdahl

A Lei de Amdahl ignora o overhead associado com a introdução de paralelismo. Vamos retornar ao exemplo anterior. Suponha que a versão paralela do programa tenha $\lceil log\ n \rceil$ pontos de comunicação. Em cada um desses pontos, o tempo de comunicação é $10.000 * \lceil log\ p \rceil + (n/10)\mu s$.

Para um problema de tamanho 10.000, o tempo de comunicação total é

$14 * (10.000 * \lceil log\ p \rceil + 1.000)\mu s$.

Deve-se levar em conta todos esses fatores incluídos na fórmula para speedup: $Ts(n)$, $\frac{Tsp(n)}{p}$ e $Tov(n,p)$. Nossa precisão para o speedup alcançável pela solução

do programa paralelo de um problema de tamanho 10.000 em p elementos de processamento é

$$S \leq \frac{(28.000 + 1.000.000)\mu s}{(42.000 + 1.000.000/p + 140.000\lceil logp \rceil)\mu s}$$

A linha sólida marcada com quadradinhos na Figura 1.3 é o limite superior de speedup da Lei de Amdahl.

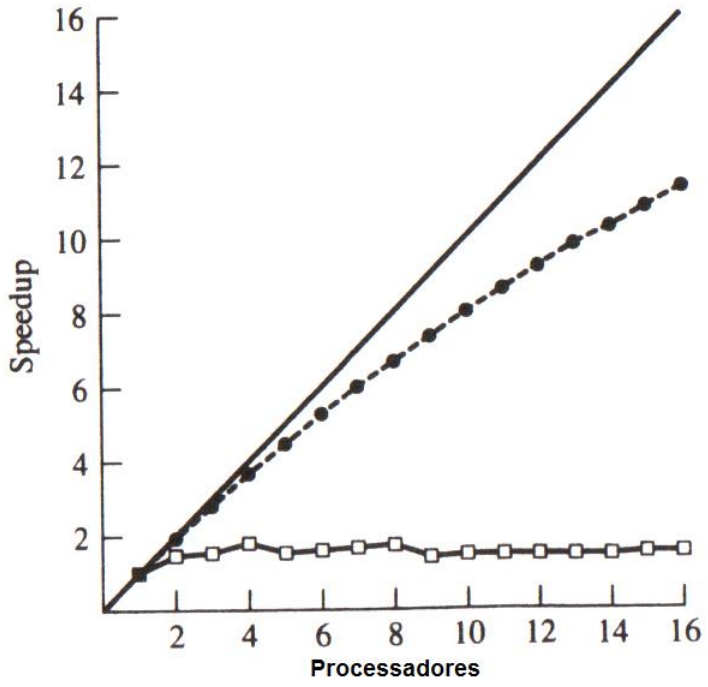

Figura 1.3: Speedup previsto pela lei de Amdahl (linha pontilhada) é maior do que o speedup previsto que tem overhead de comunicação adicionado (linha sólida marcada com quadradinhos).

1.2.2 Efeito Amdahl

Tipicamente, $Tov(n,p)$ tem complexidade menor do que $Tsp(n)$. Este é o caso com o problema hipotético, que foi considerado que $Tov(n,p) = \Theta(n * log\ n + n * log\ p)$, enquanto $Tsp(n) = \Theta(n^2)$. Aumentar o tamanho do problema aumento o tempo de computação mais rapidamente do que aumenta o tempo de comuni-

cação. Então para um número fixo de elementos de processamento, o speedup é geralmente um aumento em função do tamanho do problema. Este efeito é denominado efeito de Amdahl. A Figura 1.4 ilustra o efeito de Amdahl plotando o speedup esperado para o problema hipotético. Quando o tamanho do problema n cresce, a altura da curva de speedup também cresce.

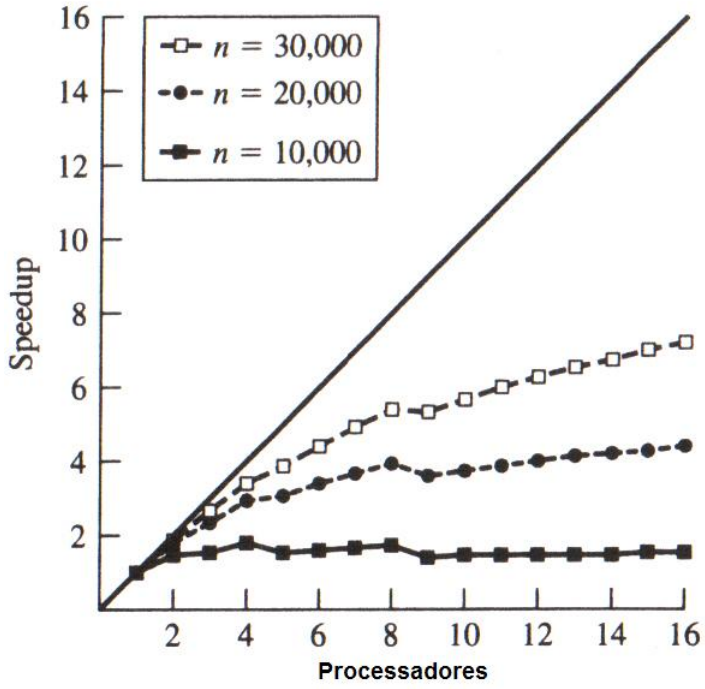

Figura 1.4: Para qualquer número fixo de elementos de processamento, o speedup está sofrendo um aumento em função do tamanho do problema.

1.3 Fontes de overhead paralelo em programas paralelos

Ao utilizar o dobro de recursos computacionais, espera-se que um programa execute duas vezes mais rápido. Porém, em programas paralelos isto raramente ocorre devido aos overheads associados ao paralelismo. Uma quantificação precisa destes overheads é fundamental para entender o desempenho de programas paralelos.

Um profile de execução de um programa paralelo é ilustrado na Figura 1.5. Nesta figura, o profile indica o tempo gasto realizando computação essencial (isto

é, computação que deveria ser realizada pelo programa sequencial para resolver a mesma instância de problema) juntamente com a computação excedente (computação não realizada na formulação sequencial), a parcela de comunicação e de ociosidade.

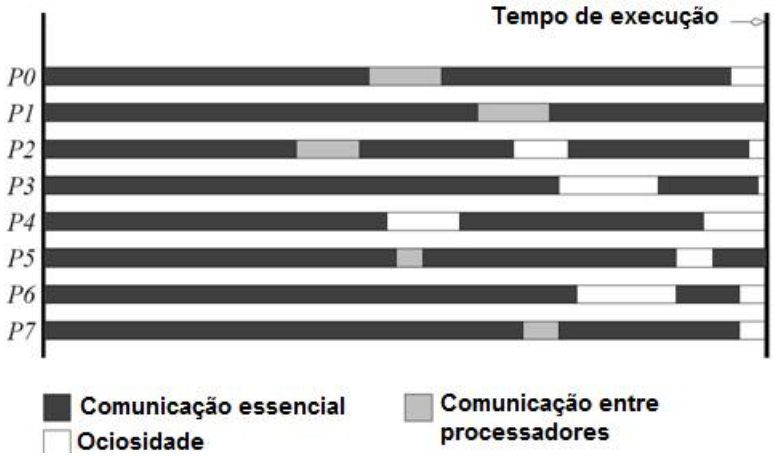

Figura 1.5: Profile de execução de programa paralelo hipotético executando em 8 elementos de processamento.

Interação entre processos: qualquer sistema paralelo não trivial requer que seus elementos de processamento tenham interação e comuniquem dados (por exemplo, dados intermediários). O tempo gasto comunicando dados entre elementos de processamento é geralmente a fonte mais significativa de overhead de processamento paralelo.

Ociosidade: elementos de processamento em um sistema paralelo podem tornar-se ociosos devidos a muitas razões, tais como desbalanceamento de carga, sincronização e presença de componentes sequenciais em um programa paralelo.

Em muitas aplicações paralelas (por exemplo, quando a geração de tarefas é dinâmica), é impossível (ou no mínimo difícil) de prever o tamanho das subtarefas atribuídas a vários elementos de processamento. Entretanto, o problema não pode ser subdividido estaticamente entre os elementos de processamento enquanto mantém uma carga uniforme. Se diferentes elementos de processamento têm diferentes cargas de trabalho, alguns elementos de processamento podem estar ociosos durante uma parte do tempo que outros estão trabalhando no problema. Em alguns programas paralelos, elementos de processamento devem sincronizar certos pontos durante a execução do programa paralelo. Se todos os elementos de processamento não estão prontos para sincronizar ao mesmo tempo, então aqueles

que estão prontos ficarão ociosos esperando o restante ficar pronto. Partes de um algoritmo podem não ser paralelizáveis, permitindo somente um elemento de processamento trabalhar sobre ele. Enquanto um elemento de processamento trabalho na parte sequencial, todos os outros elementos de processamento devem esperar.

Computação excedente:O mais rápido algoritmo sequencial conhecido para um dado problema pode ser difícil ou impossível de paralelizar, forçando ao programador a usar um algoritmo paralelo baseado em um algoritmo sequencial mais simples, que seja facilmente paralelizável (isto é, com um grau de concorrência mais alta).

Um algoritmo paralelo baseado no melhor algoritmo sequencial pode ainda realizar mais computação agregada do que o algoritmo sequencial. Um exemplo de tal computação é o algoritmo de Transformada Rápida de Fourier. Em sua versão sequencial, os resultados de certas computações podem ser reutilizadas. Entretanto, na versão paralela, estes resultados não podem ser reutilizados porque eles são gerados por diferentes elementos de processamento. Entretanto, algumas computações são realizadas múltiplas vezes em elementos de processamento diferentes.

1.4 Métrica de Karp-Flatt

A Lei de Amdahl e a Lei de Gustafson-Barsis ignoram o overhead paralelo, gerando um speedup superestimado ou scaled speedup. Karp-Flatt propuseram outra métrica, chamada de fração sequencial determinada experimentalmente, que pode prover percepções valiosas de desempenho.

Utilizando as equações 1.4

Considera o overhead paralelo $Tov(n,p)$

Tempo sequencial: $T(n,1) = Ts(n) + Tsp(n)$

Tempo paralelo: $T(n,p) = Ts(n) + \frac{Tsp(n)}{p} + Tov(n,p)$

Fração sequencial determinada experimentalmente

$e = \frac{(Ts(n) + Tov(n,p))}{T(n,1)}$

$Ts(n) + Tov(n,p) = T(n,1)e$

$T(n,p) = T(n,1)e + T(n,1)\frac{(1-e)}{p}$

$S = \frac{T(n,1)}{T(n,p)}$

$T(n,1) = T(n,p)S$

$T(n,p) = T(n,p)Se + T(n,p)S\frac{(1-e)}{p}$

$$1 = Se + S\frac{(1-e)}{p}$$
$$\frac{1}{S} = e + \frac{(1-e)}{p}$$
$$\frac{1}{S} = e + \frac{1}{p} - \frac{e}{p}$$
$$\frac{1}{S} = e(1 - \frac{1}{p}) + \frac{1}{p}$$

$$\boxed{e = \frac{1/S - 1/p}{1 - 1/p}} \tag{1.7}$$

Exemplo 1:

O benchmarking de um programa paralelo em 1, 2, ..., 8 elementos de processamento produz os seguintes resultados de speedup (Tabela 1.1):

p	2	3	4	5	6	7	8
S	1,82	2,50	3,08	3,57	4,00	4,38	4,71

Tabela 1.1: Speedups exemplo 1

Qual é a razão primária para o programa paralelo alcançar um speedup de somente 4,71 com 8 elementos de processamento?

Solução: Usando a métrica de Karp-Flatt (expressão 1.4), podemos calcular a fração sequencial determinada experimentalmente correspondente a cada número de elementos de processamento, como apresentado na Tabela 1.2.

p	2	3	4	5	6	7	8
S	1,82	2,50	3,08	3,57	4,00	4,38	4,71
e	0.10	0.10	0.10	0.10	0.10	0.10	0.10

Tabela 1.2: Cálculo de e referente aos speedups da Tabela 1.1

Já que ***e*** não aumenta com o número de elementos de processamento, o motivo é a oportunidade limitada para paralelismo (a fração grande da computação que é inerentemente sequencial).

Exemplo 2:

O benchmarking de um programa paralelo em 1, 2, ..., 8 elementos de processamento produz os seguintes resultados de speedup:

p	2	3	4	5	6	7	8
S	1,87	2,61	3,23	3,73	4,14	4,46	4,71

Tabela 1.3: Speedups do exemplo 2

Qual é a razão primária para o programa paralelo alcançar um speedup de somente 4,71 com 8 elementos de processamento?
Solução:
Usando a métrica de Karp-Flatt (expressão 1.4), podemos calcular a fração sequencial determinada experimentalmente correspondente a cada número de elementos de processamento, como apresentado na Tabela 1.3. Já que a fração sequencial

p	2	3	4	5	6	7	8
S	1,82	2,50	3,08	3,57	4,00	4,38	4,71
e	0.070	0.075	0.080	0.085	0.090	0.095	0.1

Tabela 1.4: Cálculo de *e* referente aos speedups da Tabela 1.3

determinada experimentalmente está aumentando gradativamente com o número de elementos de processamento, a razão principal para o baixo desempenho é o overhead paralelo. Isto pode ocorrer devido ao tempo gasto no startup de processos, ou sincronização, ou por restrições arquiteturais.

1.5 Escalabilidade

Escalabilidade é a medida da habilidade do sistema paralelo aumentar o desempenho quando o número de elementos de processamento aumenta. Se o sistema paralelo sofre um aumento no número de máquinas a seu serviço, deseja-se saber quanto desse aumento é realmente refletido no poder de processamento. A avaliação da escalabilidade permite entender o comportamento do crescimento do sistema paralelo e prever o seu desempenho com uma quantidade maior de processadores, mas não está no escopo deste texto.

1.6 Medindo o tempo de execução

O tempo de execução (*elapsed time*) mede o tempo gasto com uso de CPU, acesso a disco, a memória, I/O, etc. O tempo de CPU é o tempo que a CPU consome realizando cálculos, não incluindo o tempo de espera para entrada e saída ou para

executar outros programas. Pode se subdividir em tempo de CPU do sistema e tempo de CPU do usuário.

Para avaliar o tempo de execução de um programa paralelo, usa-se o tempo de CPU do usuário, pois este é o tempo gasto executando as linhas de código que estão no programa paralelo.

Em arquitetura de computadores, o tempo de execução é igual ao valor do contador de instruções multiplicado pelo número de ciclos por instrução e pelo tempo do ciclo de instrução.

Em programas multithread o número de ciclos por instrução e o número de instruções podem variar a cada execução. Esse efeito pode aumentar de acordo com o aumento do número de núcleos de processamento.

No sistema operacional Linux, o comando **time**, digitado no terminal, obtém o tempo em segundos, com 3 componentes:

- *real* - retorna o tempo decorrido entre a invocação e o término do processo
- *user* – retorna o tempo de CPU que o processo gastou (modo usuário). É a soma dos tempos *tms_utime* e *tms_cutime* presentes na *struct tms* retornada por times(2).
- *sys* – tempo de CPU do sistema (modo kernel). É a soma dos tempos *tms_stime* e *tms_cstime* presente na *struct tms* retornada por times(2)

Existem também bibliotecas do sistema operacional que permitem medir o tempo dentro programa. No Linux, time.h é bastante usada. Vamos apresentar o uso de **clock()** e **gettimeofday()**.

clock_t clock(void) retorna o número de *ticks* de *clock* decorridos desde o início da execução do programa.

Exemplo 1: Programa que calcula o tempo para realizar a soma de valores de 0 a (10000000 -1) usando a função *clock()*.

```
#include <time.h>
#include <iostream>
#include <unistd.h>
using namespace std;
int main(){
   clock_t start, end;
   float total;
   double soma=0;
   start = clock();
   for (double i=0;i< 10000000;i++)
   {
```

```
      soma+=i;
   }
   end = clock();
   total = (float) (end - start)/CLOCKS_PER_SEC;
   cout << "Soma = " << soma <<endl;
   cout  << "Tempo total gasto pela CPU: " << total;
   return 0;
}
```

Após compilar o programa, uma execução pode gerar a seguinte saída, já que o tempo pode variar de execução para execução e de máquina para máquina:

```
Soma = 5e+13
Tempo total gasto pela CPU: 0.079827
```

Executando no terminal com o comando time, tem-se:

```
$time ./time-exemplo
Soma = 5e+13
Tempo total gasto pela CPU: 0.079827
real 0m0,104s
user 0m0,083s
sys 0m0,000s
```

Pode-se perceber que o tempo medido dentro do programa foi igual ao tempo de usuário obtido pelo comando time, pois o programa é um processo que foi criado no espaço do usuário.

Outra função que pode ser utilizada para medir tempo dentro de um programa é a gettimeofday, que obtém segundos e microssegundos:

```
int gettimeofday(struct timeval *tv, struct timezone *tz);

 struct timeval {
    time_t tv_sec; /* seconds */
    suseconds_t tv_usec; /* microseconds */
 };
```

Exemplo 2: Programa que calcula o tempo para realizar a soma de valores de 0 a (10000000 -1) usando a função *gettimeofday()*.

```
#include <time.h>
#include <sys/time.h>
#include <iostream>
using namespace std;
```

```
int main()
{
    int total=0;
    struct timeval start, end;
    gettimeofday(&start, NULL);
    for (double i=0;i< 10000000;i++)
    {
      total+=i;
    }
    gettimeofday(&end, NULL);
    cout << "Tempo decorrido: " <<
    ((end.tv_sec * 1000000 + end.tv_usec) -
    (start.tv_sec * 1000000 + start.tv_usec)) << endl;
    return 0;
}
```

Ao executar o programa para medir o tempo dentro do programa, obtém-se, por exemplo, o tempo abaixo:

```
Tempo decorrido: 130457
```

O tempo obtido está em microssegundos.
Se executar medindo o tempo com *time*, pode-se obter:

```
$time ./time-exemplo
Tempo decorrido: 120557
real 0m0,125s
user 0m0,125s
sys 0m0,000s
```

O tempo decorrido 120557 está em microssegundos e está próximo ao tempo do usuário que foi 0,125 segundos.

As bibliotecas de programação paralela também apresentam funções que permitem medir tempo do programa.

Em OpenMP tem-se:

```
double omp_get_wtime(void);
```

Exemplo 3: Programa que calcula o tempo para realizar a soma paralela de valores de 0 a (10000000 -1) usando OpenMP e a função *omp_get_wtime()*.

```
#include <iostream>
#include <omp.h>
```

```
using namespace std;

int main (){
   double start = omp_get_wtime();
   double  total=0;
   int i;
   #pragma omp parallel private(i) shared(total)
   #pragma omp for reduction(+:total)
   for (i=0;i< 10000000;i++)
      total+=i;
    cout << "Time: "<< omp_get_wtime()-start << endl;
   return 0;
}
```

Após executar o programa obteve-se o tempo abaixo em microssegundos:

```
Time: 0.0151992
```

Ao medir o tempo com *time* obteve-se:

```
Time: 0.0264863
real 0m0,031s
user 0m0,156s
sys 0m0,000s
```

Na biblioteca de passagem de mensagens MPI (Message Passing Interface) tem-se:

```
double MPI_Wtime()
```

Exemplo 4: Programa que calcula o tempo para realizar a soma paralela de valores de 0 a (10000000 -1) usando OpenMP e a função *MPI_Wtime()*

```
#include <iostream>
#include <mpi.h>
using namespace std;

int main (){

   double starttime;
    starttime = MPI_Wtime();
   double  total=0;
   int i;
   #pragma omp parallel private(i) shared(total)
   #pragma omp for reduction(+:total)
   for (i=0;i< 10000000;i++)
      total+=i;
    cout << "Time: "<< MPI_Wtime()-starttime << endl;
   return 0;
```

```
}
```

O programa foi compilado com mpic++ time-exemplo4.cpp -o time-exemplo4.o e executado com mpirun -np 1 ./time-exemplo4.0, obtendo o tempo em microsse-gundos:

```
Time: 0.0251179
```

Após executar o programa com time obteve-se o tempo abaixo:

```
Time: 0.0258652
real 0m0,123s
user 0m0,057s
sys 0m0,032s
```

1.7 Exercícios

Exercício 1: O benchmarking de um programa sequencial revela que 95% do tempo de execução é gasto dentro de funções que podem ser paralelizadas. Qual é o speedup máximo que pode ser esperado da execução de uma versão paralela para este programa em 10 elementos de processamento?
Solução:
Aplicando a lei de Amdahl (expressão 1.6), tem-se:
f=0,05
p=10
S = 1/(0,05 + (1-0,05)/10)
S = 6,90

Exercício 2: Para um tamanho de problema de interesse, 6% das operações do programa paralelo estão dentro de funções de I/O que são executadas em um único elemento de processamento. Qual é o número mínimo de elementos de processamento necessários para o programa paralelo exibir um speedup de 10?
Solução:
Aplicando a lei de Amdahl (expressão 1.6), tem-se:
f = 0,06
p=?
S = 10
10 = 1/(0,06 + ((1 - 0,06)/p))
p = 23,5
Como p é número de elementos de processamento, arredondamos para p=24.

Exercício 3: Qual é a fração máxima de tempo de execução que pode ser gasta realizando operações sequenciais se uma aplicação paralela está alcançando um speedup de 50 sobre sua contrapartida sequencial?
Solução:
Aplicando a lei de Amdahl (expressão 1.6), tem-se:
f = ?
p=?
S = 50
Para alcançar o speedup máximo, consideramos que temos o número máximo de elementos de processamento, ou seja, $p \to \infty$.

$$lim_{p\to\infty} \frac{1}{f + \frac{(1-f)}{p}} = 50$$

f = 0,02

Exercício 4: O programa paralelo de Shauna alcança um speedup de 9 em 10 elementos de processamento. Qual é a fração máxima de computação que pode consistir de operações sequenciais?

Solução: Aplicando a lei de Amdahl (expressão 1.6), tem-se:

f = ?

p = 10

S = 9

$9 = 1/(f + (1 - f)/10)$

f = 0,01

Exercício 5: O programa paralelo de Brandon executa em 242 segundos em 16 elementos de processamento. Através do benchmarking ele determina que 9 segundos é gasto realizando inicializações e limpeza em um dos elementos de processamento. Durante os 233 segundos restantes todos os 16 elementos de processamento estão ativos. Qual é o speedup alcançado pelo programa de Brandon?

Solução:

Aplicando a expressão de speedup 1.4 sem o overhead paralelo, tem-se:

$$S(n,p) \leq \frac{Ts(n) + Tsp(n)}{Ts(n) + \frac{Tsp(n)}{p}}$$

$Ts(n) = 9$

$Ts(n) + \frac{Tsp(n)}{16} = 242$

$\frac{Tsp(n)}{16}$ = (242 - 9)

$Tsp(n) = 233 * 16$

$S(n,p) = \frac{9 + 233 * 16}{9 + 233}$

$S(n,p) = 15,44$

Exercício 6: Courtney executou um benchmark em seu programa paralelo executando em 40 elementos de processamento. Ela descobriu que ele gasta 99% do tempo dentro do código paralelo. Qual é o speedup do programa dela?

Solução:

Aplicando a lei de Amdahl (expressão 1.6), tem-se:

f = 0,01

p = 40

S = $\frac{1}{0,01 + (0,99/40)}$

S = 28,78

Exercício 7: Os tempos de execução de 6 programas paralelos, rotulados I-IV, são passados por um benchmark em 1, 2, ..., 8 elementos de processamento. A Tabela 1.5 apresenta os speedups alcançados por estes programas.

Procs	Speedup					
	I	II	III	IV	V	VI
1	1,00	1,00	1,00	1,00	1,00	1,00
2	1,67	1,89	1,89	1,96	1,74	1,94
3	2,14	2,63	2,68	2,88	2,30	2,82
4	2,50	3,23	3,39	3,67	2,74	3,65
5	2,78	3,68	4,03	4,46	3,09	4,42
6	3,00	4,00	4,62	5,22	3,38	5,15
7	3,18	4,22	5,15	5,93	3,62	5,84
8	3,33	4,35	5,63	6,25	3,81	6,50

Tabela 1.5: Speedups alcançados pelos programas de I a VI

Para cada um desses programas, escolha o estado que melhor descreve seu desempenho em 16 elementos de processamento:

(a) O speedup alcançado em 16 elementos de processamento provavelmente será no mínimo 40% acima do speedup alcançado em 8 elementos de processamento.

(b) O speedup alcançado em 16 elementos de processamento provavelmente será menor do que 40% acima do speedup alcançado em 8 elementos de processamento, devido à componente sequencial da computação.

(c) O speedup alcançado em 16 elementos de processamento provavelmente será menor do que 40% acima do speedup alcançado em 8 elementos de processamento, devido ao aumento do overhead quando são adicionados elementos de processamento.

Solução:
Aplicando a métrica de Karp-Flat (equação 1.4), obtem-se a Tabela 1.6.

Usando os valores de e obtidos para 8 procs., pode-se obter speedup estimado para 16 procs (Tabela 1.7).

(a) As aplicações III, IV e VI apresentarão aumento acima de 40%.

(b) As aplicações I e V apresentam e constante e speedup com aumento inferior a 40%. A aplicação V tem uma fração sequencial acima de 15% que não pode

Procs	Fração sequencial e					
	I	II	III	IV	V	VI
2	0,20	0,06	0,06	0,02	0,15	0,03
3	0,20	0,07	0,06	0,02	0,15	0,03
4	0,20	0,08	0,06	0,03	0,15	0,03
5	0,20	0,09	0,06	0,03	0,15	0,03
6	0,20	0,10	0,06	0,03	0,15	0,03
7	0,20	0,11	0,06	0,03	0,16	0,03
8	0,20	0,12	0,06	0,04	0,16	0,03

Tabela 1.6: Speedups alcançados pelos programas de I a VI

Procs	Speedup para 16 procs.					
	I	II	III	IV	V	VI
16	3,99	5,72	8,41	10,00	4,77	10,71
(% aumento)	(19,81%)	(31,49%)	(49,38%)	(60 %)	(25,20%)	(64,77%)

Tabela 1.7: Speedups alcançados pelos programas de I a VI

ser paralelizada e mesmo variando um pouco é muito impactante.

(c) A aplicação II tem uma fração sequencial e que está variando a cada número de elementos de processamento, mostrando que há uma overhead paralelo.

Exercício 8: A aplicação X foi paralelizada com a criação de processos e executada em uma máquina que tinha 8 núcleos de processamento. A aplicação foi executada 10 vezes para vezes para 4, 8 e 16 processos, respectivamente, e calculada a média. Foi plotado o gráfico do tempo médio de execução de uma aplicação X apresentado na Figura 1.6. Calcule o Speedup, a Eficiência e a Fração sequencial determinada experimentalmente (e) pela métrica de Karp-Flat.

Solução:

Para resolver este exercício precisamos relembrar como se calcula o Speedup, a Eficiência e a fração sequencial determinada experimentalmente definida por Karp-Flat.

Speedup = S = $\frac{tempo\ \ de\ \ execucao\ \ sequencial}{tempo\ \ de\ \ execucao\ \ paralela}$

Eficiencia = $Speedup/procs$

$e = \frac{1/S - 1/p}{1 - 1/p}$

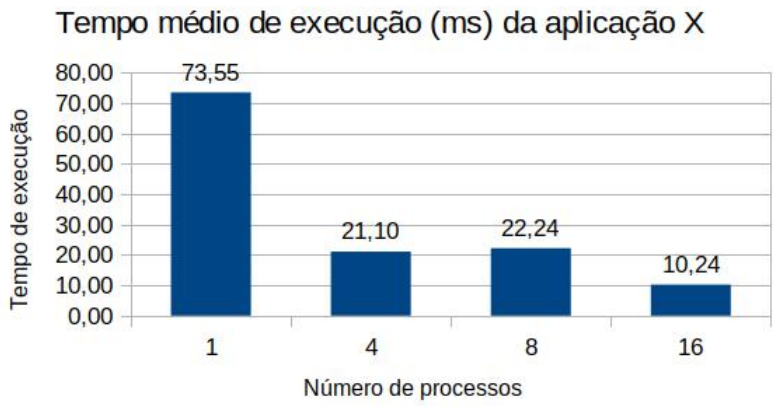

Figura 1.6: Gráfico de tempo de execução x número de processos

Realizando os cálculos, obtemos os resultados obtidos na Tabela 1.8. Pode-se observar que houve um redução expressiva na eficiência de utilização dos elementos de processamento, que é 0,15 para 16. Analisando o comportamento da fração sequencial determinada experimentalmente (e), percebe-se um aumento significando que há overhead paralelo que impacta o desempenho.

Procs.	4	8	16
Speedup	3,49	2,83	2,45
Eficiência	0,87	0,35	0,15
Fração e	0,05	0,26	0,37

Tabela 1.8: Speedup, eficiência e fração e do exercício 8

1.8 Referências

Quinn, M. J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill. 2004.

Grama, A., Gupta, A., Karypis, G. and Kumar, V. *Introduction to Parallel Computing*. Addison Wesley. Second Edition. 2003.

Pacheco, P. S. An Introduction to Parallel Programming. Morgan Kaufman. 2011.