Parallel System Performance

Jan 6, 2020

Course outline (Pacheco; GGKK; Quinn)

- Motivation (1;1;1)
- **How to quantify performance improvement (2.6; 5; 7)**
- Parallel hardware architecture (2.2-2.3; 2,4; 2)
- Parallel programming frameworks
  - Pthreads for shared memory (4; 7; -)
  - OpenMP for shared memory (5; 7.10; 17)
  - MPI for distributed memory (3; 6; 4)
  - CUDA/OpenCL for GPU,
  - Hadoop/Spark/Mapreduce for distributed systems
- Parallel program verification
- Parallel algorithm design
- Some case studies

Why is performance analysis important?

- Being able to accurately predict the performance of a parallel algorithm
  - can help decide whether to actually go to the trouble of coding and debugging it.
- Being able to analyze the execution time exhibited by a parallel program
  - Can help understand barriers to higher performance
  - Can help predict how much improvement can be realized by increasing number of processors

Well-known performance prediction formulas

- Amdahl's Law
  - Help decide whether a program merits parallelization
- Gustafson-Barsi's Law
  - Way to evaluate performance of a parallel program
- Karp-Flatt metric
  - Decide whether the principal barrier to speedup is the amount of inherently sequential code or parallel overhead
- Iso-efficiency metric
  - Way to evaluate the scalability of a parallel algorithm executing on a parallel computer. Help choose the design that will achieve higher performance when the number of processors increase.

Operations performed by a parallel algorithm

- Computations that must be performed sequentially $\sigma(n)$
- Computations that can be performed in parallel $\varphi(n)$
- Parallel overhead (communication operations and redundant computations) $\kappa(n,p)$

Operations performed by a parallel algorithm

- Computations that must be performed sequentially σ(n)
- Computations that can be performed in parallel φ(n)
- Parallel overhead (communication operations and redundant computations) κ(n,p)

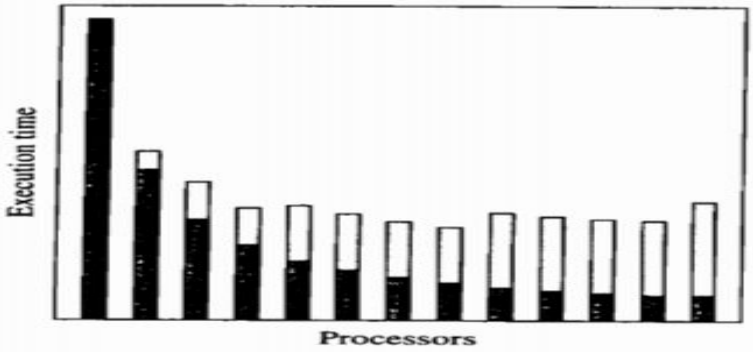

Figure 7.1 Nontrivial parallel algorithms have a computation component (black bars) that is a decreasing function of the number of processors used and a communication component (gray bars) that is an increasing function of the number of processors. For any fixed problem size there is an optimum number of processors that minimizes overall execution time.

Speedup and efficiency

We design and implement parallel programs in the hope that they will run faster than their sequential counterparts.

- Speedup = (Sequential execution time)/(Parallel execution time)

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

Speedup and efficiency

We design and implement parallel programs in the hope that they will run faster than their sequential counterparts.

- Speedup = (Sequential execution time)/(Parallel execution time)

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

- Efficiency = Speedup/(Processors used)

$$\varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{p\left(\sigma(n) + \varphi(n)/p + \kappa(n, p)\right)}$$

$$\Rightarrow \varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{p\sigma(n) + \varphi(n) + p\kappa(n, p)}$$

Since all terms are greater than or equal to zero, $0 \leq \varepsilon(n, p) \leq 1$.

Amdahl's Law

Consider the expression for speedup we have just derived.

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

Since $\kappa(n, p) > 0$,

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)} \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

Amdahl's Law

Consider the expression for speedup we have just derived.

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

Since $\kappa(n, p) > 0$,

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)} \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

Let f denote the inherently sequential portion of the computation. In other words, $f = \sigma(n)/(\sigma(n) + \varphi(n))$. Then

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

$$\Rightarrow \psi(n, p) \leq \frac{\sigma(n)/f}{\sigma(n) + \sigma(n)(1/f - 1)/p}$$

$$\Rightarrow \psi(n, p) \leq \frac{1/f}{1 + (1/f - 1)/p}$$

$$\Rightarrow \psi(n, p) \leq \frac{1}{f + (1 - f)/p}$$

Numerical Examples

Amdahl's Law [2]
Let f be the fraction of operations in a computation that must be performed sequentially, where $0 \leq f \leq 1$. The maximum speedup ψ achievable by a parallel computer with p processors performing the computation is

$$\psi \leq \frac{1}{f + (1 - f)/p}$$

EXAMPLE 1

Suppose we are trying to determine whether it is worthwhile to develop a parallel version of a program solving a particular problem. Benchmarking reveals that 90 percent of the execution time is spent inside functions that we believe we can execute in parallel. The remaining 10 percent of the execution time is spent in functions that must be executed on a single processor. What is the maximum speedup that we could expect from a parallel version of the program executing on eight processors?

■ Solution
By Amdahl's Law

$$\psi \leq \frac{1}{0.1 + (1 - 0.1)/8} \approx 4.7$$

We should expect a speedup of 4.7 or less.

Numerical Examples

EXAMPLE 3

Suppose we have implemented a parallel version of a sequential program with time complexity $\Theta(n^2)$, where n is the size of the dataset. Assume the time needed to input the dataset and output the result is

$$(18000 + n)\ \mu\text{sec}$$

This constitutes the sequential portion of the program. The computational portion of the program can be executed in parallel; it has execution time

$$(n^2/100)\ \mu\text{sec}$$

What is the maximum speedup achievable by this parallel program on a problem of size 10,000?

■ Solution

By Amdahl's Law

$$\psi \leq \frac{(28{,}000 + 1{,}000{,}000)\ \mu\text{sec}}{(28{,}000 + 1{,}000{,}000/p)\ \mu\text{sec}}$$

Limitations of Amdahl's Law

Amdahl's Law ignores overhead associated with the introduction of parallelism. Let's return to our previous example. Suppose the parallel version of the program has $\lceil \log n \rceil$ communication points. At each of these points, the communication time is

$$10{,}000\lceil \log p \rceil + (n/10)\ \mu\text{sec}$$

For a problem of size 10,000, the total communication time is

$$14(10{,}000\lceil \log p \rceil + 1{,}000)\ \mu\text{sec}$$

Now we have taken into account all of the factors included in our formula for speedup: $\sigma(n)$, $\varphi(n)$, and $\kappa(n, p)$. Our prediction for the speedup achievable by the parallel program solving a problem of size 10,000 on p processors is

$$\psi \leq \frac{(28{,}000 + 1{,}000{,}000)\ \mu\text{sec}}{(42{,}000 + 1{,}000{,}000/p + 140{,}000\lceil \log p \rceil)\ \mu\text{sec}}$$

The solid line in Figure 7.2 plots a new upper bound on speedup predicted by this more comprehensive formula. Taking communication time into account gives us a more realistic prediction of the parallel program's performance.

Limitations of Amdahl's Law

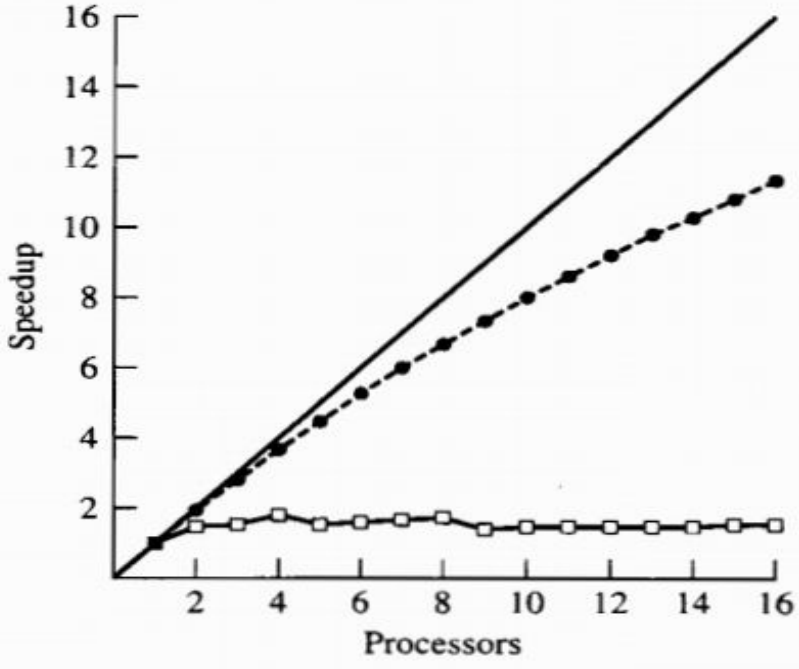

Figure 7.2 Speedup predicted by Amdahl's Law (dashed line) is higher than speedup prediction that takes communication overhead into account (solid line).

Amdahl Effect

Typically, $\kappa(n, p)$ has lower complexity than $\varphi(n)$. That is the case with the hypothetical problem we have been considering: $\kappa(n, p) = \Theta(n \log n + n \log p)$, while $\varphi(n) = \Theta(n^2)$. Increasing the size of the problem increases the computation time faster than it increases the communication time. Hence for a fixed number of processors, speedup is usually an increasing function of the problem size. This is called the **Amdahl effect** [42]. Figure 7.3 illustrates the Amdahl effect by plotting expected speedup for our hypothetical problem. As problem size n increases, so does the height of the speedup curve.

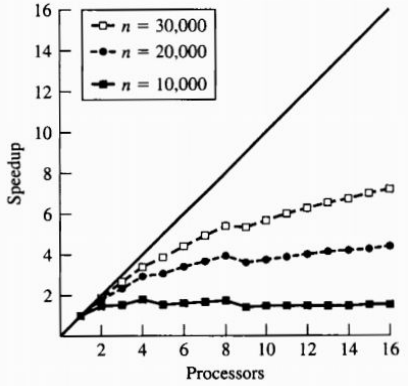

Figure 7.3 For any fixed number of processors, speedup is usually an increasing function of the problem size. This is called the **Amdahl effect.**

Gustafson-Barsi's Law

What happens if we treat time as a constant and let the problem size increase with the number of processors? The inherently sequential fraction of a computation typically decreases as problem size increases (the Amdahl effect). Increasing the number of processors enables us to increase the problem size, decreasing the inherently sequential fraction of a computation, and increasing the quotient between serial execution time and parallel execution time (speedup).

Consider the expression for speedup we have derived. Since $\kappa(n, p) \geq 0$,

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

Let s denote the fraction of time spent in the *parallel* computation performing inherently sequential operations. The fraction of time spent in the parallel computation performing parallel operations is what remains, or $(1 - s)$. Mathematically,

$$s = \frac{\sigma(n)}{\sigma(n) + \varphi(n)/p}$$

$$(1 - s) = \frac{\varphi(n)/p}{\sigma(n) + \varphi(n)/p}$$

Hence

$$\sigma(n) = \big(\sigma(n) + \varphi(n)/p\big)s$$

$$\varphi(n) = \big(\sigma(n) + \varphi(n)/p\big)(1 - s)p$$

Gustafson-Barsi's Law

Therefore

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

$$\Rightarrow \psi(n, p) \leq \frac{\left(\sigma(n) + \varphi(n)/p\right)\left(s + (1 - s)p\right)}{\sigma(n) + \varphi(n)/p}$$

$$\Rightarrow \psi(n, p) \leq s + (1 - s)p$$

$$\Rightarrow \psi(n, p) \leq p + (1 - p)s$$

Numerical Examples

Gustafson-Barsis's Law [46]
Given a parallel program solving a problem of size n using p processors, let s denote the fraction of total execution time spent in serial code. The maximum speedup ψ achievable by this program is

$$\psi \leq p + (1 - p)s$$

Vicki plans to justify her purchase of a $30 million Gadzooks supercomputer by demonstrating its 16,384 processors can achieve a scaled speedup of 15,000 on a problem of great importance to her employer. What is the maximum fraction of the parallel execution time that can be devoted to inherently sequential operations if her application is to achieve this goal?

■ Solution
Using Gustafson-Barsis's Law:

$$\begin{aligned} 15{,}000 &= 16{,}384 - 16{,}383s \\ \Rightarrow \quad s &= 1{,}384/16{,}383 \\ \Rightarrow \quad s &= 0.084 \end{aligned}$$

The Karp-Flatt Metric

Because Amdahl's Law and Gustafson-Barsis's Law ignore $\kappa(n, p)$, the parallel overhead term, they can overestimate speedup or scaled speedup. Karp and Flatt have proposed another metric, called the experimentally determined serial fraction, which can provide valuable performance insights [59].

Recall that we have represented the execution time of a parallel program executing on p processors as

$$T(n, p) = \sigma(n) + \varphi(n)/p + \kappa(n, p)$$

where $\sigma(n)$ is the inherently serial component of the computation, $\varphi(n)$ is the portion of the computation that may be executed in parallel, and $\kappa(n, p)$ is overhead resulting from processor communication and synchronization, and redundant computations. The serial program does not have any interprocessor communication or synchronization overhead, so its execution time is

$$T(n, 1) = \sigma(n) + \varphi(n)$$

We define the **experimentally determined serial fraction** e of the parallel computation to be

$$e = \big(\sigma(n) + \kappa(n, p)\big)/T(n, 1)$$

The Karp-Flatt Metric

Hence

$$\sigma(n) + \kappa(n, p) = T(n, 1)e$$

We may now rewrite the parallel execution time as

$$T(n, p) = T(n, 1)e + T(n, 1)(1 - e)/p$$

Let's use ψ as a shorthand for $\psi(n, p)$. Since speedup $\psi = T(n, 1)/T(n, p)$, we have $T(n, 1) = T(n, p)\psi$. Hence

$$\begin{aligned}
& T(n, p) = T(n, p)\psi e + T(n, p)\psi(1 - e)/p \\
\Rightarrow \quad & 1 = \psi e + \psi(1 - e)/p \\
\Rightarrow \quad & 1/\psi = e + (1 - e)/p \\
\Rightarrow \quad & 1/\psi = e + 1/p - e/p \\
\Rightarrow \quad & 1/\psi = e(1 - 1/p) + 1/p \\
\Rightarrow \quad & e = \frac{1/\psi - 1/p}{1 - 1/p}
\end{aligned}$$

The Karp-Flatt Metric [59]

Given a parallel computation exhibiting speedup ψ on p processors, where $p > 1$, the experimentally determined serial fraction e is defined to be

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

Numerical Examples

Benchmarking a parallel program on 1, 2, . . . , 8 processors produces the following speed-up results:

p	2	3	4	5	6	7	8
ψ	1.82	2.50	3.08	3.57	4.00	4.38	4.71

What is the primary reason for the parallel program achieving a speedup of only 4.71 on eight processors?

■ Solution

Using the formula we have developed, we can compute the experimentally determined serial fraction e corresponding to each data point:

p	2	3	4	5	6	7	8
ψ	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

Since the experimentally determined serial fraction is not increasing with the number of processors, the primary reason for the poor speedup is the limited opportunity for parallelism—that is, the large fraction of the computation that is inherently sequential.

Numerical Examples

EXAMPLE 2

Benchmarking a parallel program on 1, 2, . . . , 8 processors produces the following speed-up results:

p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71

What is the primary reason for the parallel program achieving a speedup of only 4.71 on eight processors?

■ Solution

We begin by computing the experimentally determined serial fraction for each of these program runs:

p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	0.070	0.075	0.080	0.085	0.090	0.095	0.1

Since the experimentally determined serial fraction is steadily increasing as the number of processors increases, the principal reason for the poor speedup is parallel overhead. This could be time spent in process startup, communication, or synchronization, or it could be an architectural constraint.

Iso-efficiency metric

Let's refer to a parallel program executing on a parallel computer as a **parallel system.** The **scalability** of a parallel system is a measure of its ability to increase performance as the number of processors increases.

As we have already seen, speedup (and hence efficiency) is typically an increasing function of the problem size, because the communication complexity is usually lower than the computational complexity. We call this the Amdahl effect. In order to maintain the same level of efficiency when processors are added, we can increase the problem size.

These ideas are formalized by the **isoefficiency relation.** To derive the iso-efficiency relation, we return to our original definition of speedup:

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

$$\Rightarrow \psi(n, p) \leq \frac{p\big(\sigma(n) + \varphi(n)\big)}{p\sigma(n) + \phi(n) + p\kappa(n, p)}$$

$$\Rightarrow \psi(n, p) \leq \frac{p\big(\sigma(n) + \varphi(n)\big)}{\sigma(n) + \phi(n) + (p-1)\sigma(n) + p\kappa(n, p)}$$

Iso-efficiency metric

Let's refer to a parallel program executing on a parallel computer as a **parallel system.** The **scalability** of a parallel system is a measure of its ability to increase performance as the number of processors increases.

As we have already seen, speedup (and hence efficiency) is typically an increasing function of the problem size, because the communication complexity is usually lower than the computational complexity. We call this the Amdahl effect. In order to maintain the same level of efficiency when processors are added, we can increase the problem size.

These ideas are formalized by the **isoefficiency relation.** To derive the isoefficiency relation, we return to our original definition of speedup:

$$\psi(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n, p)}$$

$$\Rightarrow \psi(n, p) \leq \frac{p\big(\sigma(n) + \varphi(n)\big)}{p\sigma(n) + \phi(n) + p\kappa(n, p)}$$

$$\Rightarrow \psi(n, p) \leq \frac{p\big(\sigma(n) + \varphi(n)\big)}{\sigma(n) + \phi(n) + (p-1)\sigma(n) + p\kappa(n, p)}$$

We define $T_o(n, p)$ to be the total amount of time spent by all processes doing work not done by the sequential algorithm. One component of this time is the time $p - 1$ processes spend executing inherently sequential code. The other component of this time is the time all p processes spend performing interprocessor communications and redundant computations. Hence $T_o(n, p) = (p-1)\sigma(n) + p\kappa(n, p)$. Substituting $T_o(n, p)$ into our previous equation, we get:

$$\Rightarrow \psi(n, p) \leq \frac{p\big(\sigma(n) + \varphi(n)\big)}{\sigma(n) + \phi(n) + T_o(n, p)}$$

Iso-efficiency metric

Since efficiency equals speedup divided by p:

$$\varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \phi(n) + T_o(n, p)}$$

$$\Rightarrow \varepsilon(n, p) \leq \frac{1}{1 + \frac{T_o(n,p)}{\sigma(n)+\varphi(n)}}$$

Recalling that $T(n, 1)$ represents sequential execution time:

$$\Rightarrow \varepsilon(n, p) \leq \frac{1}{1 + T_o(n, p)/T(n, 1)}$$

$$\Rightarrow \frac{T_o(n, p)}{T(n, 1)} \leq \frac{1 - \varepsilon(n, p)}{\varepsilon(n, p)}$$

$$\Rightarrow T(n, 1) \geq \frac{\varepsilon(n, p)}{1 - \varepsilon(n, p)} T_o(n, p)$$

Course outline (Pacheco; GGKK; Quinn)

- Motivation (1;1;1)
- **How to quantify performance improvement (2.6; 5; 7)**
- Parallel hardware architecture (2.2-2.3; 2,4; 2)
- Parallel programming frameworks
  - Pthreads for shared memory (4; 7; -)
  - **OpenMP for shared memory (5; 7.10; 17)**
  - MPI for distributed memory (3; 6; 4)
  - CUDA/OpenCL for GPU,
  - Hadoop/Spark/Mapreduce for distributed systems
- Parallel program verification
- Parallel algorithm design
- Some case studies