On the Performance and Scalability of Parallel Architectures

Principles of Scalable Performance

Tópico 4

Performance and Scalability

- The performance of parallel computers relies on a design that balances software and hardware (balancing what is required or available with what can be realised or used).
- Just as performance is affected by a number of parameters, so too is the scalability of parallel systems.
- Thus, system architects and programmers must exploit parallelism in a balanced approach.

What measures can be used to determine the performance and effectiveness of parallel systems?

Degree of Parallelism (DOP)

The *Degree of Parallelism* reflects the extent to which software parallelism matches hardware parallelism. The DOP, with respect to some time period t, is simply the number of processors that can be **used** in that time period t.

Note, in practice, a program may use a different number of processors at different times during its execution.

The DOP of a program could be larger than that of the machine upon which the program is running, i.e. DOP is defined under the assumption that we have an unbounded amount of hardware resources.

Execution Rate

The *computing capacity* Δ of a single processor is approximated by the execution rate (such as MIPS or MFLOPS) ignoring memory access, communication latency or system overhead penalties.

Work

The *work* done by a program within a time period of length $t_2 - t_1$ is:

$$\begin{aligned} W &= \int_{t_1}^{t_2} \Delta \cdot DOP(t)\, dt \\ &= \sum_{i=1}^{m} \Delta \cdot DOP(i) \cdot t_i \;=\; \Delta \sum_{i=1}^{m} i \cdot t_i \end{aligned}$$

where m is the maximum parallelism and t_i is the total amount of time that the degree of parallelism $= i$.

Average Parallelism

The average parallelism of a program

$$= \frac{\text{“}Total\text{”}\ Parallelism}{Total\ Elapsed\ Time} = \frac{\frac{Work}{\Delta}}{Total\ Elapsed\ Time}$$

$$= \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} DOP(t)\, dt = \frac{\sum_{i=1}^{m} i \cdot t_i}{\sum_{i=1}^{m} t_i}$$

This doesn't seem to tell us much??

Remember we need to balance architectural parallelism with program parallelism. So how much program parallelism is really available?
– Various claims.

Speedup

Parallel processing speedup comes in various guises. Let us use the following definition:

> The *speedup* achieved by a parallel algorithm running on n processors is the ratio between the time taken by that parallel computer executing the **fastest** serial algorithm and the corresponding parallel algorithm using n processors.

$$\text{Speedup } S = \frac{\textit{Execution time on a } 1\text{--}\textit{processor machine}}{\textit{Execution time on a } n\text{--}\textit{processor machine}}$$

Speedup$_{(cont)}$

$$S \;=\; \frac{\frac{W}{\Delta} \;=\; \sum_{i=1}^{m} DOP(i) \cdot t_i \;=\; \sum_{i=1}^{m} i \cdot t_i}{Execution\ time\ on\ a\ n - processor\ machine}$$

- What is the fastest possible execution time on a n-processor machine?
  - Assume $n = max_i\{\mathrm{DOP}(i)\}$, i.e. assume we have all the processors we can use.
  - Also assume there are no overheads such as communication latency.

Speedup$_{(cont)}$

- Clearly, the fastest time is

$$= \sum_{i=1}^{m} \frac{W_i}{\Delta \cdot i} = \sum_{i=1}^{m} \frac{i \cdot t_i}{i} = \sum_{i=1}^{m} t_i$$

- So, the **best** speed-up (under ideal conditions - see the assumptions):

$$S = \frac{\sum_{i=1}^{m} i \cdot t_i}{\sum_{i=1}^{m} t_i}$$

This coincides with the *Average Parallelism*. Thus, average parallelism is an upper bound on the (asymptotic) speedup.

Lessons

- Speed-up can be formulated in many different ways, with different assumptions.
- The idea is

$$Speedup \; = \; \frac{Best\ sequential\ time}{Best\ parallel\ time}$$

- The question is how to define "Best"?

Be careful when you hear speed-up claims. For example, some people define the speedup of a parallel algorithm running on n processors to be time taken by the parallel algorithm on one processors divided by the time taken by the parallel algorithm using n processors. This exaggerates the speedup because the overheads of parallelisation are ignored.

Parameters for Evaluating Parallel Computation

Speed-up, which represents how much faster an application can execute, can be simply summarised by $S(n) = \frac{T(1)}{T(n)}$ for n processors.

Other performance metrics besides speedup include:

- *Efficiency* which indicates the actual degree of speedup achieved – $E(n) = \frac{S(n)}{n}$

  The lowest efficiency corresponds to sequential execution, the highest when all n processors are utilised.

  Note: $1 \leq S(n) \leq n$ and $\frac{1}{n} \leq E(n) \leq 1$

Parameters$_{(cont)}$

Efficiency could be viewed as any of the following:

- average speed-up per processor;
- ratio of the actual speed-up to MAX speed-up;
- ratio of sequential to parallel work done (comparative effectiveness) $\frac{T_1(=W_1)}{n \cdot T_n}$ where any overhead (delay) is included in the parallel work done.

Parameters$_{(cont)}$

- *Redundancy* signifies the extent of matching between software parallelism and hardware parallelism:

$$R(n) \quad = \quad \frac{Work\ done\ by\ n-processor\ program}{Work\ done\ by\ sequential\ program}$$

(Again, "work" is basically the total number of operations.)

Redundancy can be view as how many "overhead" operations are needed to coordinate the parallel tasks.

Parameters$_{(cont)}$

- *Utilisation* is the percentage of resources kept busy:

$$
\begin{aligned}
U(n) &= \frac{Work\ done\ by\ n-processor\ program}{n \cdot T(n)} \\
&= R(n) \cdot E(n)
\end{aligned}
$$

The utilisation (of processors collectively) effectively measures how many processors are active and how many were de-activated ("masked").

$$\frac{1}{n} \leq E(n) \leq U(n) \leq 1 \leq R(n) \leq \frac{1}{E(n)} \leq n$$

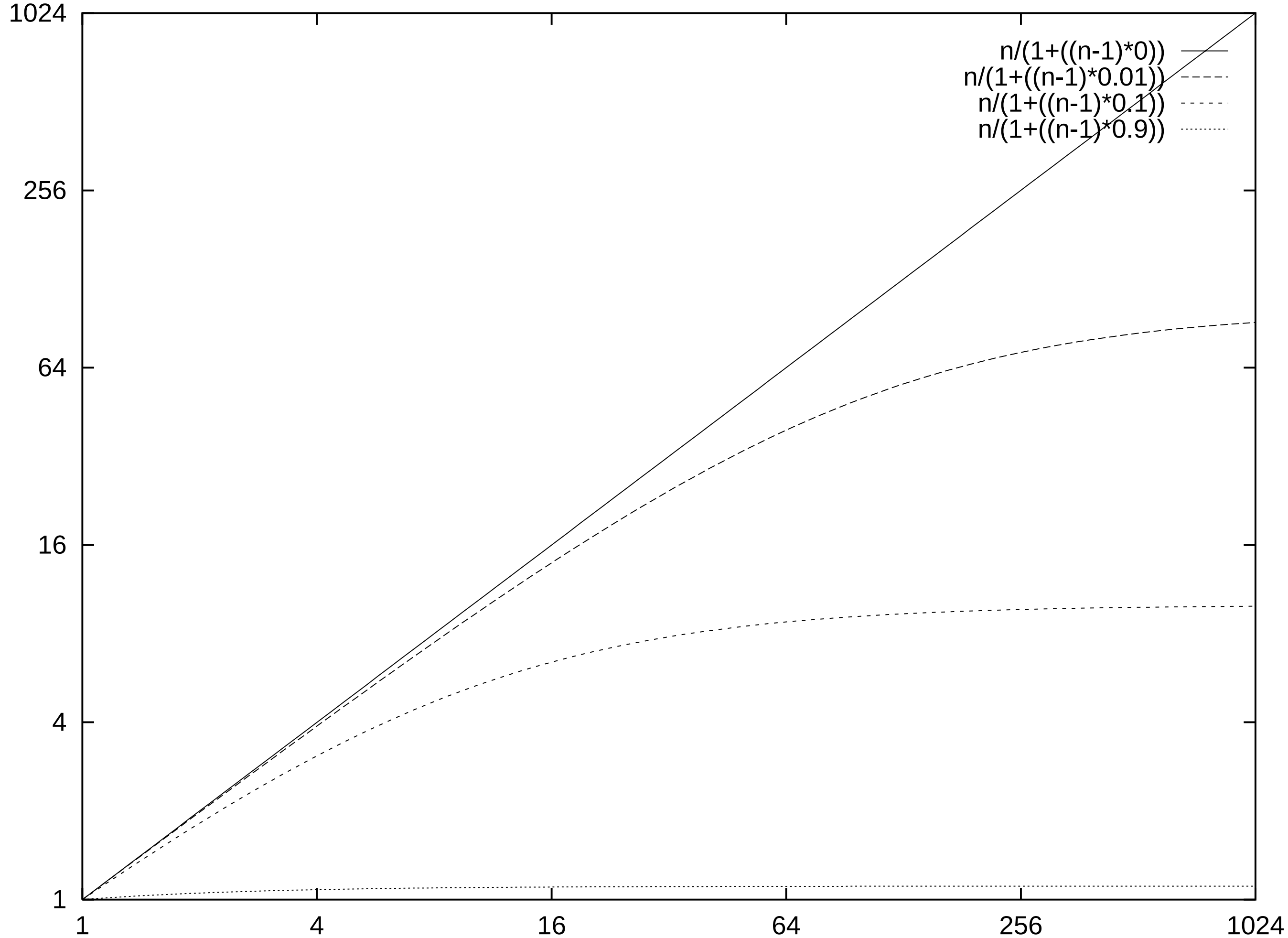
Amdahl's Law (1967)

- Let W be the number of operations in a parallel program ("work").
- Each operation takes some fixed amount of time t.
- Assume that a fraction α of the program **cannot** execute in parallel (thus 1 - α of the program **can**).
- Assume the machine is "bi-modal" (can run in both sequential and parallel mode).
- In parallel mode, the machine executes at a rate that is n times that of sequential mode (for "machine size n").

Amdahl's Law$_{(cont)}$

$$
\begin{aligned}
S(n) &= \frac{T(1)}{T(n)} \\
T(1) &= W \cdot t \\
T(n) &= \alpha \cdot W \cdot t + \frac{(1-\alpha) \cdot W \cdot t}{n} \\
Thus\ S(n) &= \frac{1}{\alpha + \frac{1-\alpha}{n}}
\end{aligned}
$$

Amdahl's Law$_{(cont)}$



Amdahl's Law$_{(cont)}$

Note that as n grows, $S(n) \to \frac{1}{\alpha}$. The more sequentialism there is, the worse the speedup.

α is sometimes known as the *sequential bottleneck* in programs [c.f. von Neumman bottleneck] and this bottleneck cannot be avoided by simply increasing the number of processors.

This result had two effects on the parallel community:

1. Discouraged the development of massively parallel processing (MPP) systems; and
2. Increased the attention paid to parallelising compilers, in order to reduce the value of α.

Amdahl's Law$_{(cont)}$

- Recall, W_i = amount of work executed with degree of parallelism i in a program.
- $\alpha = W_1$ and (1 - α) = W_n i.e. the previous slides discussed a normalised situation where $W_1 + W_n = \alpha$ + (1 - α) = 1

$$S(n) = \frac{W_1+W_n}{W_1+\frac{W_n}{n}}$$

Amdahl's Law is now known as *fixed load speedup*.

Amdahl's Law$_{(cont)}$

The problem of a sequential bottleneck cannot be solved just by increasing the number of processors in a system. The real problem lies in the existence of a sequential fraction of program code. This property imposed a very pessimistic view on parallel processing for two decades.

There has been a lot of argument over Amdahl's Law, specifically over the assumptions upon which it is based. Generally it is a valid law if you accept that the program will not grow as machine parallelism grows i.e. the number of operations and their sequential/parallel proportion stay fixed (e.g. the input data size stays the same).

Amdahl's Law$_{(cont)}$

- Assumes workload cannot scale to match the available computing power as the machine size is increased. Thus the fixed load model prevents scalability in performance.

- But isn't it logical to use additional parallelism to solve larger problems (larger input sets)?

- What happens when you allow the problem size to scale? $\rightarrow$ Gustafson's (change to Amdahl's) Law.

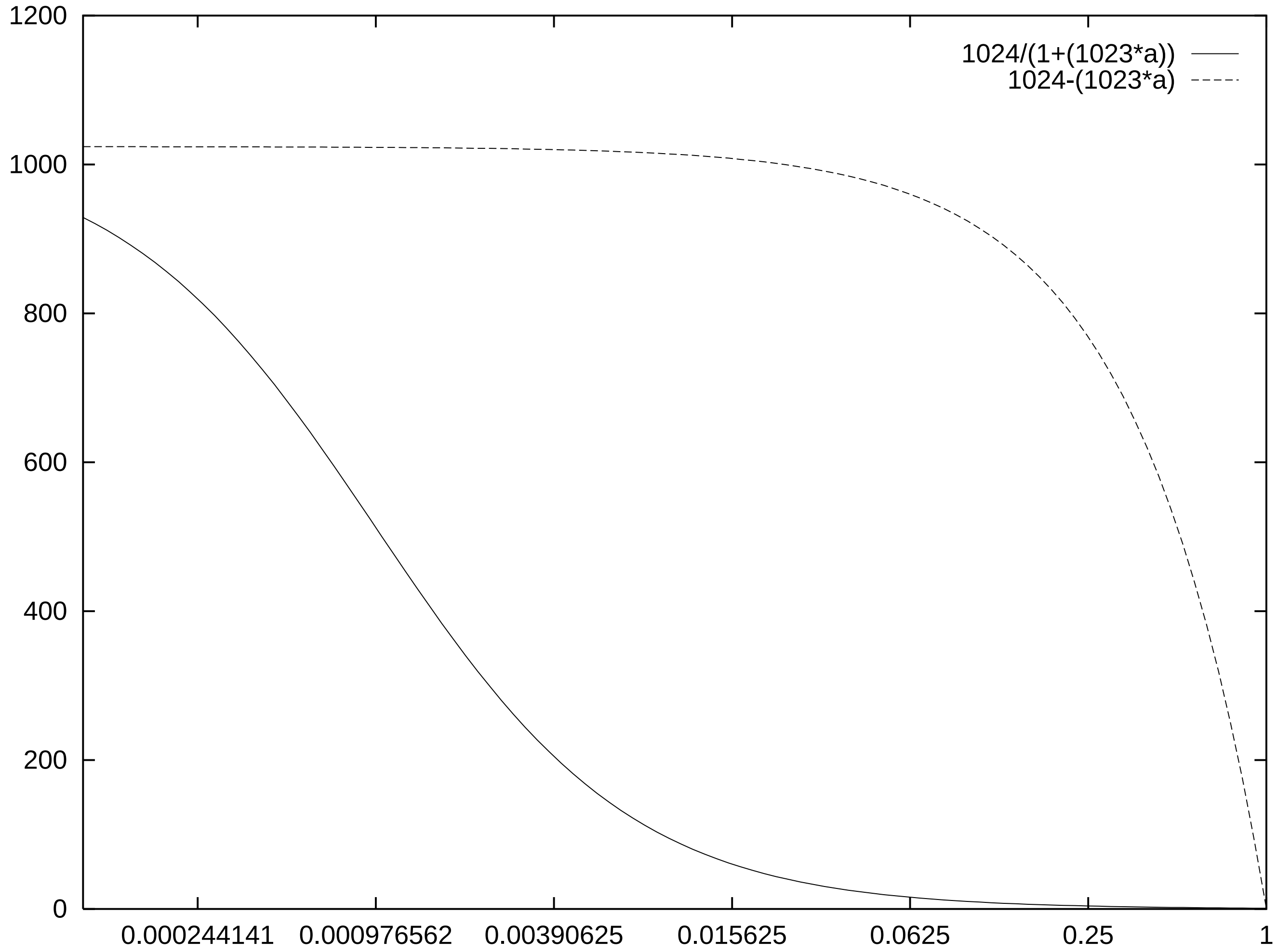
Gustafson's Law

The fixed load model is appropriate for time critical applications e.g. real time systems. However, many scientific applications require greater accuracy rather than faster execution times. This motivated Gustafson (1987) to propose the *fixed time speedup* model.

$$
\begin{aligned}
S'(n) &= \frac{W_1 + n \cdot W_n}{W_1 + W_n} \\
&= n - \alpha(n-1)
\end{aligned}
$$

In effect, Gustafson says you can maintain speedup in many instances if you increase the problem size along with the machine parallelism. When the problem can scale to match the available computing power, the sequential fraction is no longer a bottleneck.

Gustafson's Law$_{(cont)}$



Is Superlinear Speedup Possible?

- It depends on the assumptions.
- Some people argue that speedup cannot be greater than linear based on the premise that a single processor can always emulate parallel processors.
- If a parallel algorithm A solves an instance of a problem in t_n units of time on a parallel computer with n processors. Then algorithm A can solve the same problem instance in $n \times t_n$ units of time on the same computer with one processor through time slicing. Because parallel algorithms usually have an associated overhead, it is likely that there exists a sequential algorithm that can solve the problem instance in less $n \times t_n$, which means the speedup would be less than n.

Is Superlinear Speedup Possible?$_{(cont)}$

- The previous assumptions may have two flaws (one algorithmic and one architectural).

  1. The "best" algorithm is allowed to change depending on the problem instance chosen. [Not realistic.]

  2. A single processor may not be able to "emulate" the memory and cache hierarchy of parallel processors, thus n processor may be able to execute a parallel algorithm more than n times faster than a single processor executing the best sequential algorithm.

Scalability

Exploiting parallelism for higher performance requires both scalable architectures and scalable algorithms.

The $<metric>$ of a computer system increases linearly with respect to the number of processors used.

The $<metric>$ is usually **performance**, but could also be **cost**, or **cost-performance** combination.

Should the $<metric> =$ speedup or efficiency?

Lets come up with a good general definition of these in the light of the Amdahl/Gustafson Laws.

Scalability$_{(cont)}$

Speedup (general, abstract), with respect to machine parallelism n, some program (or algorithm) and problem size s:

$$S(s,n) = \frac{T(s,1)}{T(s,n)+h(s,n)}$$

Where $h(s,n)$ is the overhead cost of parallelism (non-work $\rightarrow$ redundancy, communication, synchronisation, I/O etc).

Similarly, **Efficiency**

$$E(s,n) = \frac{S(s,n)}{n}$$

Scalability$_{(cont)}$

Anyway, the first idea of defining "scalable" is to say that the program or algorithm and machine combination is scalable if, for all s and n,

$$\begin{aligned} S(s,n) &= \mathcal{O}(n) \\ E(s,n) &= \mathcal{O}(1) \end{aligned}$$

However, in general, these definitions are too restrictive (i.e. not many scalable systems).

Could also be redefined to allow for certain degrees of inefficiency, e.g. "almost scalable" being defined as $S(s,n) = \mathcal{O}(\frac{n}{logn})$.

Objectives for Scalable Computers

1. Size Scalability - The system should be scalable in terms of resources and expect to achieve a linear increase in performance with an incremental expansion (for a given set of applications).

2. Generation Scalability - Both the hardware technology and the software must be scalable.

3. Problem Scalability - The system should perform well as the problem size increases. This is related to the program granularity.

Difficulties

With respect to architecture, there are a number of hard problems preventing scalability:

- Non-constant node degree in networks.
- Layout of networks in three dimensions.
- If you do have compact layout in 3D, cooling may be a problem.
- In general, communication (including memory access) and synchronisation costs rise, which adversely affects speedup and efficiency.

Difficulties$_{(cont)}$

- Complex engineering solutions to these problems (if achievable) may be very costly - economically unscalable.

- Programmability: shared memory is easier to program, but distributed memory is more scalable.

Summary

- *Speedup* and *Efficiency* are two important measures for parallel processing systems.

- Beware your speedup may not be what you expect.

- Amdahl's Corollary: a small number of sequential operations can seriously damage your speedup.

Summary$_{(cont)}$

- Scalability analysis determines whether the parallel processing of your application can offer the desired improvement in performance.

- It is clear that no single scalability metric is sufficient.