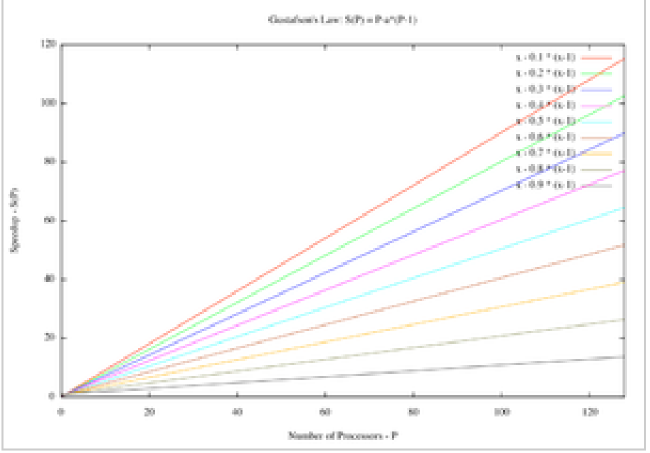
Gustafson's law

From Wikipedia, the free encyclopedia

Gustafson's Law (also known as **Gustafson–Barsis' law**[1]) is a law in computer science which says that computations involving arbitrarily large data sets can be efficiently parallelized. Gustafson's Law provides a counterpoint to Amdahl's law, which describes a limit on the speed-up that parallelization can provide, given a fixed data set size. Gustafson's law was first described [2] by John L. Gustafson and his colleague Edwin H. Barsis:



Gustafson's Law

$$S(P) = P - \alpha \cdot (P - 1)$$

where P is the number of processors, S is the speedup, and α is the largest non-parallelizable fraction of any parallel process.

Gustafson's law addresses the shortcomings of Amdahl's law, which does not fully exploit the computing power that becomes available as the number of machines increases. Gustafson's Law instead proposes that programmers tend to set the size of problems to use the available equipment to solve problems within a practical fixed time. Therefore, if faster (more parallel) equipment is available, larger problems can be solved in the same time.

Accordingly, Gustafson called his metric *scaled speedup*, because in the above expression $S(P)$ is the ratio of the total, single-process execution time to the per-process parallel execution time; the former scales with P, while the latter is assumed fixed or nearly so. This is in contrast to Amdahl's Law, which takes the single-process execution time to be the fixed quantity, and compares it to a shrinking per-process parallel execution time. Thus, Amdahl's law is based on the assumption of a fixed problem size: it assumes the overall workload of a program does not change with respect to machine size (i.e., the number of processors). Both laws assume the parallelizable part is evenly distributed over P processors.

The impact of Gustafson's law was to shift research goals to select or reformulate problems so that solving a larger problem in the same amount of time would be possible. In a way the law redefines efficiency, due to the possibility that limitations imposed by the sequential part of a program may be countered by increasing the total amount of computation.

Contents

Derivation of Gustafson's law

The execution time of the program on a parallel computer is decomposed into:

$$(a+b)$$

where a is the sequential time and b is the parallel time, on any of the *P* processors. (Overhead is ignored.)

The key assumption of Gustafson and Barsis is that the total amount of work to be done in parallel *varies linearly with the number of processors*. Then practical implication is of the single processor being more capable than the single processing assignment to be executed in parallel with (typically similar) other assignments. This implies that *b*, the per-process parallel time, should be held fixed as *P* is varied. The corresponding time for sequential processing is

$$a + P \cdot b.$$

Speedup is accordingly:

$$(a + P \cdot b)/(a+b).$$

Defining

$$\alpha = a/(a+b)$$

to be the sequential fraction of the parallel execution time, we have

$$S(P) = \alpha + P \cdot (1-\alpha) = P - \alpha \cdot (P-1).$$

Thus, if α is small, the speedup is approximately *P*, as desired. It may even be the case that α diminishes as *P* (together with the problem size) increases; if that holds true, then *S* approaches *P* monotonically with the growth of *P*.

Thus Gustafson's law seems to rescue parallel processing from Amdahl's law. It is based on the idea that if the problem size is allowed to grow monotonically with *P*, then the sequential fraction of the workload would not ultimately come to dominate. This is enabled by means of having most of the assignments individually containable within a single processor's scope of processing; thus a single processor may provide for multiple assignments, while a single assignment shouldn't span more than a single processor. This is also the rule for relating projects with work-sites, having multiple projects per site, but only one site per project.

A driving metaphor

Amdahl's Law approximately suggests:

> “ Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph. No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city. Since it has already taken you 1 hour and you only have a distance of 60 miles total; going infinitely fast you would only achieve 60 mph. ”

Gustafson's Law approximately states:

> “ Suppose a car has already been traveling for some time at less than 90mph. Given enough time and distance to travel, the car's average speed can always eventually reach 90mph, no matter how long or how slowly it has already traveled. For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour, and so on. ”

Applications

Application in research

Amdahl's law presupposes that the computing requirements will stay the same, given increased processing power. In other words, an analysis of the same data will take less time given more computing power.

Gustafson, on the other hand, argues that more computing power will cause the data to be more carefully and fully analyzed: pixel by pixel or unit by unit, rather than on a larger scale. Where it would not have been possible or practical to simulate the impact of nuclear detonation on every building, car, and their contents (including furniture, structure strength, etc.) because such a calculation would have taken more time than was available to provide an answer, the increase in computing power will prompt researchers to add more data to more fully simulate more variables, giving a more accurate result.

Application in everyday computer systems

Amdahl's law reveals a limitation in, for example, the ability of multiple cores to reduce the time it takes for a computer to boot to its operating system and be ready for use. Assuming the boot process was mostly parallel, quadrupling computing power on a system that took one minute to load might reduce the boot time to just over fifteen seconds. But greater and greater parallelization would eventually fail to make bootup go any faster, if any part of the boot process were inherently sequential.

Gustafson's law argues that a fourfold increase in computing power would instead lead to a similar increase in expectations of what the system will be capable of. If the one-minute load time is acceptable to most users, then that is a starting point from which to increase the features and functions of the system. The time taken to boot to the operating system will be the same, i.e. one minute, but the new system would include more graphical or user-friendly features.

Limitations

Some problems do not have fundamentally larger datasets. As example, processing one data point per world citizen gets larger at only a few percent per year. The principal point of Gustafson's law is that such problems are not likely to be the most fruitful applications of parallelism.

Algorithms with nonlinear runtimes may find it hard to take advantage of parallelism "exposed" by Gustafson's law. Snyder[3] points out an $O(N^3)$ algorithm means that double the concurrency gives only about a 26% increase in problem size. Thus, while it may be possible to occupy vast concurrency, doing so may bring little advantage over the original, less concurrent solution—however in practice there have been massive improvements.

Hill and Marty[4] emphasize also that methods of speeding sequential execution are still needed, even for multicore machines. They point out that locally inefficient methods can be globally efficient when they reduce the sequential phase. Furthermore, Woo and Lee[5] studied the implication of energy and power on future many-core processors based on Amdahl's Law, showing that an asymmetric many-core processor can achieve the best possible energy efficiency by activating an optimal number of cores given the amount of parallelism is known prior to execution.

See also

- Scalable parallelism

References

1. Michael McCool; James Reinders; Arch Robison (2013). *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier. p. 61.
2. Reevaluating Amdahl's Law (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.6348), John L. Gustafson, Communications of the ACM 31(5), 1988. pp. 532-533. Also as a web page here (http://www.johngustafson.net/pubs/pub13/amdahl.htm)
3. Type Architectures, Shared Memory, and The Corollary of Modest Potential (http://www.cs.washington.edu/homes/snyder/TypeArchitectures.pdf), Lawrence Snyder, Ann. Rev. Comput. Sci. 1986. 1:289-317.
4. Amdahl's Law in the Multicore Era (http://www.cs.wisc.edu/multifacet/amdahl/), Mark D. Hill and Michael R. Marty, IEEE Computer, vol. 41, pp. 33–38, July 2008. Also UW CS-TR-2007-1593, April 2007.
5. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era (http://www.computer.org/portal/web/csdl/doi/10.1109/MC.2008.530), Dong Hyuk Woo and Hsien-Hsin S. Lee, IEEE Computer, vol. 41, No. 12, pp.24-31, December 2008.

External links

Retrieved from "https://en.wikipedia.org/w/index.php?title=Gustafson%27s_law&oldid=674896300"

Categories: Analysis of parallel algorithms | Programming rules of thumb