Whitepaper

Is the free lunch really over? Scalability in Many-core Systems: Part 1 in a Series

The traditional approach to application performance was to simply wait for the next generation of processor; most software developers did not need to invest in performance tuning, and enjoyed a "free lunch" from hardware improvements. With the industry shift to multiple core systems, the situation has changed: performance has to be realized through concurrency, with application designed to scale as the number of cores increases. In this opening paper of a series, we look at aspects of predicting scalability, and pose the question: *is parallel scalability the new free lunch?*

Contents

1 Introduction....3

2 Predicting Scalability....4

5 Conclusion....8

6 References....9

Figures

1-1. Amdahl's Law curves for 8 cores, varying degrees of parallelism....5

1-2. Amdahl's Law curves for 2^{16} cores, varying degrees of parallelism....6

§

1 Introduction

During the "Serial Coding Era", that historic period in computer evolution when hardware platforms were built around one single-core processor, the more astute developers noticed that performance improvements to their work came more or less automatically, with release of the next generation processor. Blogger Joel Spolsky put it this way:

As a programmer, thanks to plummeting memory prices, and CPU speeds doubling every year, you had a choice. You could spend six months rewriting your inner loops in Assembler, or take six months off to play drums in a rock and roll band, and in either case, your program would run faster. Assembler programmers don't have groupies.

So, we don't care about performance or optimization much anymore.

While anachronistically posted (in autumn of 2007, when the Serial Coding Era was well and truly over), the message accurately describes choices once available; performance improvements were free -- just wait a bit for new hardware.

Herb Sutter, C++ and now concurrency guru for Dr Dobbs, famously alerted us to the end of all that in his 2005 (!) article, The Free Lunch is Over. For many sound reasons, including semiconductor physics, chip design has moved to multiple cores; single-core performance gains will be substantially less dramatic than before. There are gains to be realized, but only with effort: software must now be designed and implemented for concurrency.

As Sutter and many others point out, concurrency is hard. Few developers are yet comfortable with it, traditional programming models ignore it, and the currently favored implementation mechanism – threads – is fraught with peril (race conditions). Among the challenges is scalability: how does the application perform as additional cores are added? We will examine aspects of scalability in this series of papers.

2 Predicting Scalability

The term scalability is nicely defined by blogger Werner Vogels:

A service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added.

This concept is broadly applicable. It is a key concern for data centers, for example, in predicting the impact of adding more servers, or of implementing virtualization. We focus here more narrowly, on CPU cores as the resource under consideration, and confine the discussion to threads as the implementation model.

We are reminded (Sutter, again) that “cores” are both an execution and a memory resource, thanks to local cache memory inherent in all current designs; we will address scaling considerations for both aspects of cores, in Chapter 3.

2.1 Things look grim (Amdahl was an optimist)

Scalability discussions nearly always start with “Amdahl’s law”, the clever-looking equation which states that serial code is not parallel code – or rather, you cannot speed up serial code by adding more cores. A common expression of this is:

$$Speedup(p) = \frac{1}{s + (1 - s) / p}$$

Where *Speedup*, as a function of the number of processing units *p*, depends upon the values of the serial portion of code *s*, and the parallelizable portion *(1-s)*. For example, consider a code which is 90% parallelizable (and thus *s* = 0.1), running on 8 cores (thus *p* = 8); the resulting *Speedup* is 4.7. This represents the maximum possible speedup realizable in this case. Figure 1 shows this result graphically, along with results for codes with larger serial components. Perfect scaling, where *s* = 0, is represented by the blue diagonal line; we are falling rather short of perfection, in these cases.

The results in Figure 1-1 show scaling predictions for modest systems, up to 8 cores. What happens in a many core system, or with a more favorable fraction of the code parallelizable? This is shown in Figure 1-2 (from Wikipedia), and the picture is not encouraging: even a code which is 95% perfectly parallelized will not speed up more than a factor of 20, regardless of the number of cores available.

Figure 1-1. Amdahl's Law curves for 8 cores, varying degrees of parallelism.

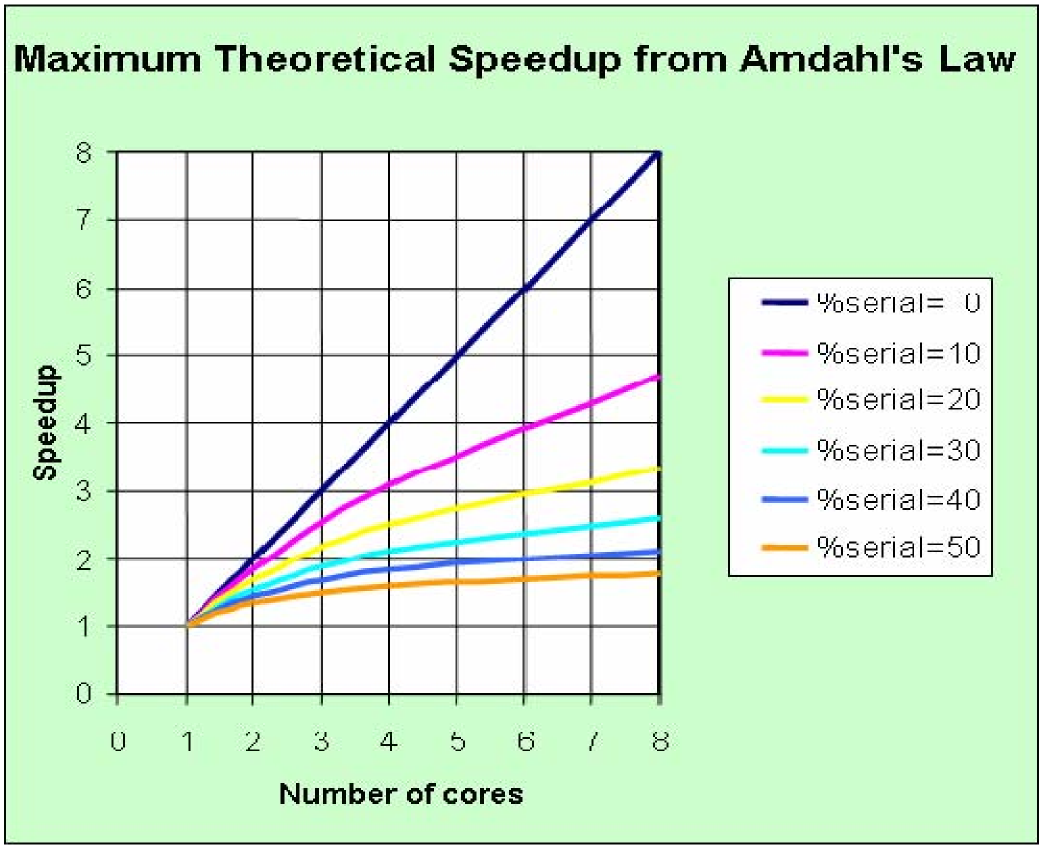
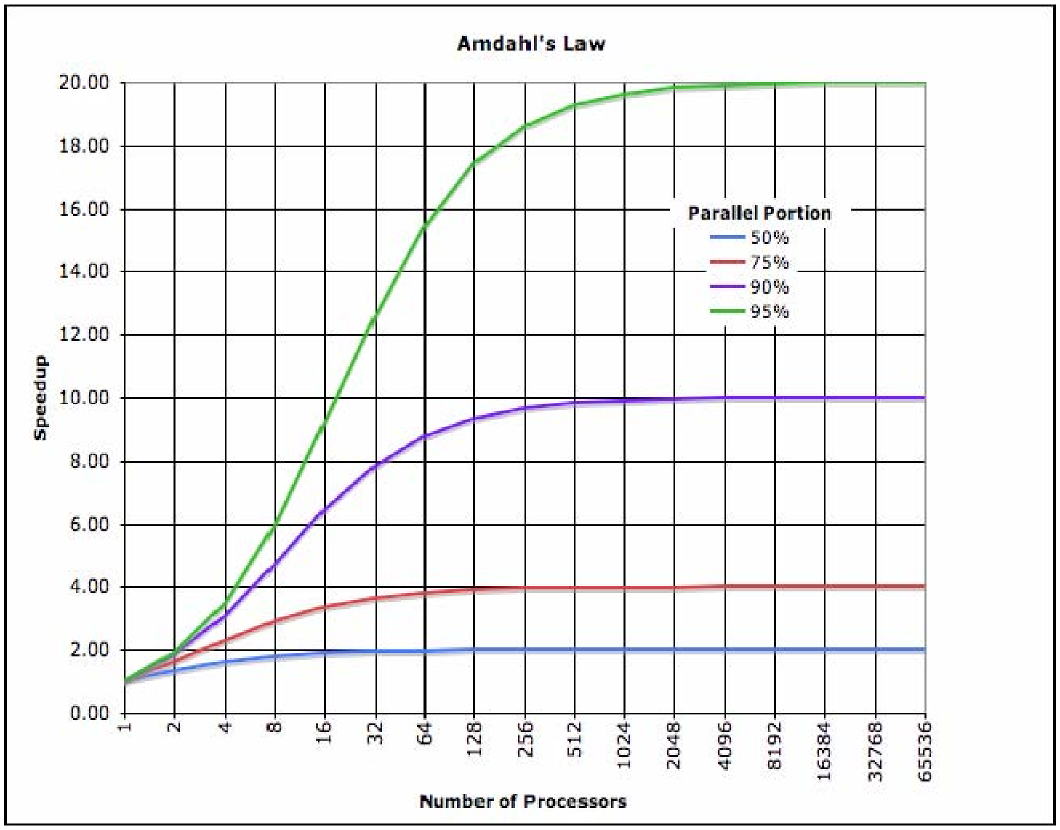


Figure 1-2. Amdahl's Law curves for 2^{16} cores, varying degrees of parallelism.



Beyond these standard textbook considerations, the situation may be even worse: Amdahl's idealized expression ignores performance degradations from thread creation overhead, synchronization, and communication, so a "perfect" implementation will not reach even the meager scaling shown in these graphs. What hope, then, is there for parallel applications, especially for efficient execution on large numbers of cores?

2.2 Reevaluating Amdahl's Law (beware hidden assumptions)

The key flaw in Amdahl's Law was pointed out as early as 1988, by John Gustafson: it assumes either a fixed computational workload, or a fixed scaling of serial and parallel portions as the problem size is increased.

The alternative, often referred to as Scaled Speedup, looks instead at increasing the problem size for a given amount of time, and may be expressed as follows:

$$Speedup(p) = p + (1 - p)s$$

That is, the speedup for a given number of processors p is that number p, plus that part of the parallel code (1-p) spent on serial operations s. There is a subtle difference, compared to Amdahl's law: this *s* describes serial work when the application is run on *p* processors.

It is revealing to examine the scaling numbers for specific cases. Suppose an application is run on 64 cores, with a measured elapsed time of 220 seconds, with 11 seconds determined (through profiling, for example) to be dedicated to serial execution. Inserting these values into the Scaled Speedup formula gives:

$$Speedup = 64 + (1 - 64)\frac{11}{222}$$

$$= 64 - 63 * 0.05$$

$$= 60.85$$

That is, speedup is nearly a factor of 61, on 64 cores. If the same value for serial time, 0.05, is used in Amdahl's Law, that formula predicts a speedup of only 15 on 64 cores.

While the Scaled Speedup model makes the same idealizations of Amdahl's law (no overhead, etc), it exposes a dramatically more optimistic view regarding scalability.

3 Conclusion

To this point, we have only begun to set the stage: Amdahl's Law does not tell the full story, and parallel scalability is possible, in many cases.

With a fully scalable application, a doubling in core number would result in a near-doubling, so the investment in scalability now will pay off in "free" performance improvements as the hardware core count increases. Then, the astute developers could simply take time off…know any rock bands needing a drummer?

In the next installment, we examine factors which inhibit scalability, and how to mitigate them.

4 References

Joel on Software, http://www.joelonsoftware.com/items/2007/09/18.html

Herb Sutter, *The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software*, Dr. Dobb's Journal, 30(3), March 2005; widely reprinted

Werner Vogels, *A Word on Scalability,* All Things Distributed, http://www.allthingsdistributed.com/2006/03/a_word_on_scalability.html

Herb Sutter, *Super Linearity and the Bigger Machine*, Dr. Dobb's Portal, Mar 12, 2008 http://www.ddj.com/architect/206903306

John Gustafson, *Reevaluating Amdahl's Law*, Communications of the ACM 31(5), 1988; reposted at http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html.

A.1.1 About the author

Michael Wrinn is a senior course architect in the Intel Software College, where he collaborates with universities to bring parallel computing to the mainstream of undergraduate education. Prior assignments include managing Intel's software engineering lab in Shanghai, and directing the human interface technology research lab. He was Intel's representative to the committee which produced the first OpenMP specification, and remains active in the parallel computing community. Before joining Intel, Michael worked at Accelrys (San Diego), implementing commercial and research simulation codes on a wide variety of parallel/HPC systems. He holds a Ph.D. (in quantum mechanics) and a B.Sc. (mathematics/chemistry/physics) from McGill University.

§

Blank Page Style (To be used when the text ends on an odd page). “Next Page” section break MUST be added prior to adding Appendices.

§