5: "Flavors" of Scalability in Manycore Processors

Seminars in *Scalable Computing*

Vittorio Scarano

Università di Salerno

ISISLab

Dottorato di Ricerca in Informatica

Plan

Plan

Scalability

- **Scalability**: the property of a solution to a problem to mantain its *efficiency* as the *dimension* grows
- Some keywords to be addressed in the context of parallel programming:
  - Efficiency: speedup over the "corresponding" sequential solution
  - Dimension: processors number, type or interconnection; problem size (memory)
- Big-Oh notation for algorithms: scalability, but only in principle
  - what happens when you fill the current level of the memory hierarchy you are using
  - what happens when number of processors grows to infinity
  - ...

Plan

A very "old" law: 1967

- Described (only informally!) by Gene Amdahl (IBM) in a 3 page papers of 1967 "*Validity of the single processor approach to achieving large scale computing capabilities*"
- The "validity of single processor" is described against the supportes of the parallel organization of computers (with parallel memories, connected by a bus or point-to-point, with parallel execution streams)
- The basic idea is that:
  1. the "*data management housekeeping*", that is inherently sequential, cannot be parallelized (and therefore improved) on a parallel computer (no matter how many resources it has)
  2. any "geometrically related" problem, given the irregularity of shapes/regions/etc. cannot be mapped onto a regular geometry of components

Speedup according to Amdahl

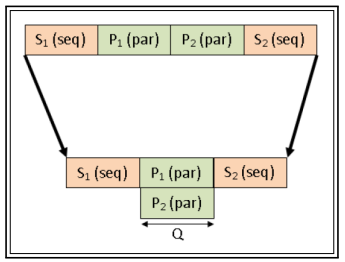
- The*speedup* S of X is the ratio between the sequential time to execute X and the parallel time (n processors) to execute X
- Let P be the part of X that can be parallelized
  - with n processors the parallel part takes time $\frac{P}{n}$ while the sequential takes time S
- Then the speedup is:

Amdahl's law

By executing X on n processors, the speedup is:

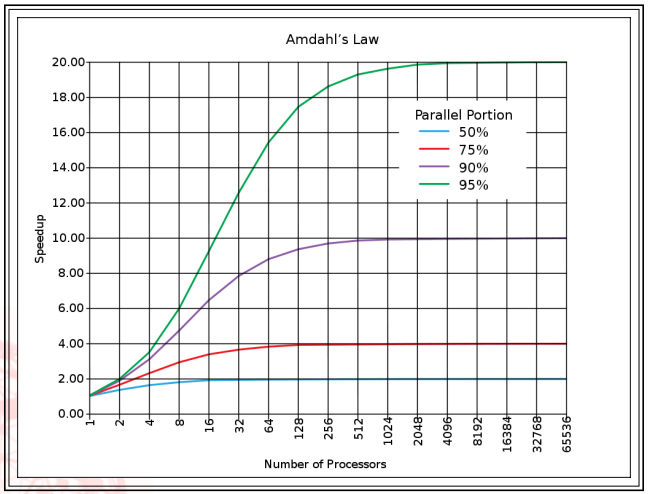
$$\text{Speedup}_A = \frac{S+P}{S+\frac{P}{n}}$$

An example



- Assume a *fixed program* X with a sequential S and a parallelizable part P
- $n = 2$, $S = S_1 + S_2$, $P = P_1 + P_2$, $S_i = P_i$ fixed
- $\text{Speedup}_A(X) = \frac{(S+P)}{S+P/2} = 4/3$

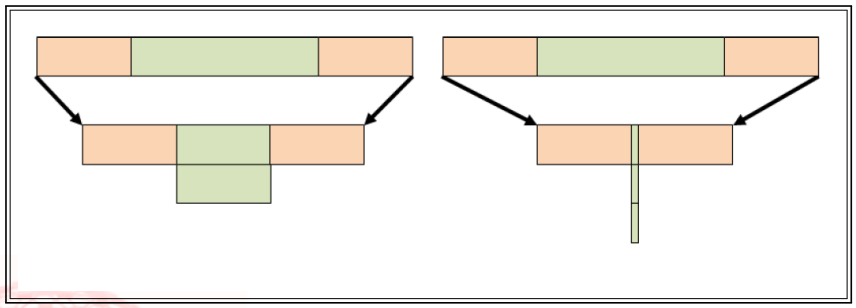
A diagram of the law



Interpretation and comments

- Amdahl's law indicates that the sequential part of a program will slow down any speedup that we can hope to get from parallelization
- $\lim_{n\to\infty} \text{Speedup}_A = \lim_{n\to\infty} \frac{S+P}{S+\frac{P}{n}} = \frac{S+P}{S}$
- If we set the Amdahl's coefficient $\alpha = S/(S+P)$, speedup is bounded by $1/\alpha$
- *Law of diminishing return* on investment
- $\Rightarrow$ it is not enough to buy/invest in new hardware, but the sequential part must be negligible with respect to the parallel part (*good for us, Computer Scientists!*)

The situation at the limit



Plan

Laws and reality

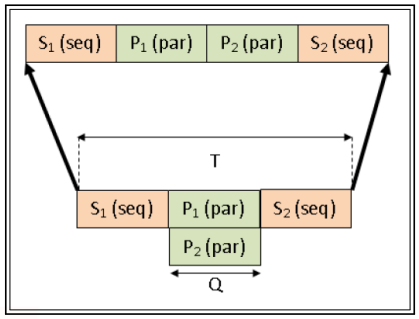
- 1988: Gustafson writes his (and his team) experience:

"Reevaluating Amdahl's Law" (Comm. of ACM, 1988)

The steepness of Amdahl's law graph when $S \to 0$ for $N = 1024$ implies that very few problems will experience even a 100-fold speedup. Yet, for 3 applications ($S = 0.4 - 0.8$ percent) we experience speedups between 1016 and 1021.

- The criticism: "One does not take a *fixed-size* problem and run it on various numbers of processors (except in academic research)"
- You should assume run time constant and not the problem size

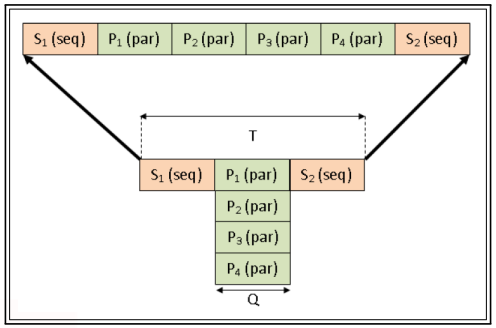
The idea - 1



- Consider programs with a sequential part S, fixed, and a fixed time frame, $T = S + Q$.
- The speedup obtained by X is

$$\text{Speedup}_G(X) = \frac{S + 2Q}{S + Q} = 4/3$$

But, you can get more "mileage"...



- The speedup obtained by X is

$$\text{Speedup}_G(X) = \frac{S + 4Q}{S + Q} = 6/3$$

The formula

- Consider programs, with a sequential part S, fixed, and a fixed time frame, $T = S + Q$.
- Then the speedup by using n processors according to Gustafson (and Barsis) is:

Gustafson-Barsis' law

By executing X on n processors, the speedup is:

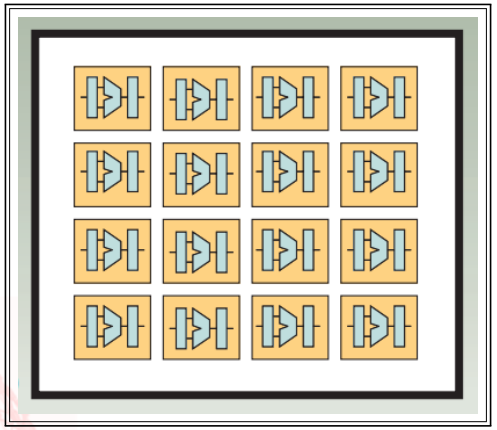
$$\text{Speedup}_G = \frac{S + nQ}{S + Q}$$

- With $n \to \infty$ the speedup is unbounded!

Designer dilemma

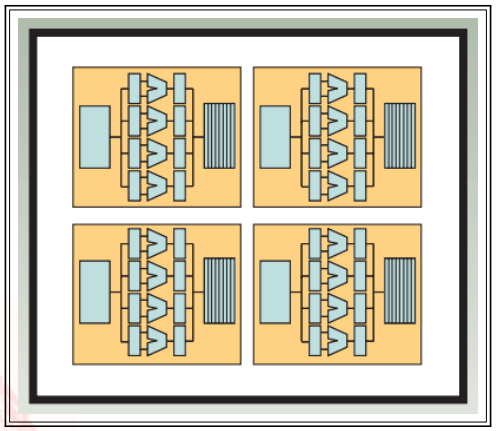
- Assume that technological constraints bound the number of transistors on a multicore chip
- The dilemma: how to organize them? Many cores of small capacity or fewer cores of large capacity?
- The model: assume that on a chip you can place at most n *Base Core Equivalents* (BCE) of computational power 1
- Area of r BCEs can be used to obtain a processor with performances $perf(r)$
- In general, $perf(r)$ is sublinear; usually $perf(r) = \sqrt{r}$ (Pollack rule)
- "*Amdahl's Law in the Multicore Era*, M.D. Hill, M.R. Marty, IEEE Computer 41(7), Nov. 2008.

Symmetric Multicore Highly Parallel



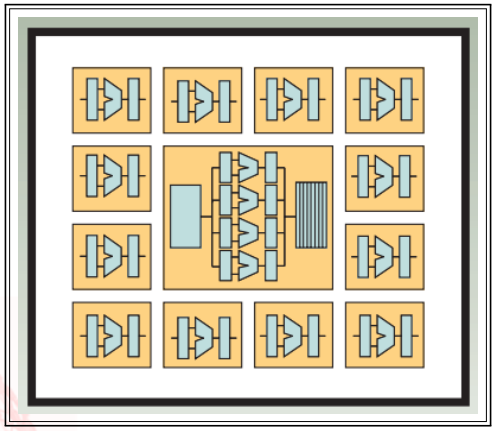
PicoChip, Connex Machine, Tilera (TILE64)

Symmetric Multicore Lowly Parallel



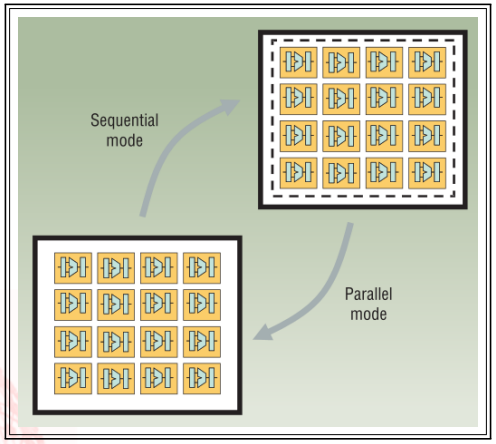
Intel, AMD

Asymmetric Multicore



IBM/Sony Cell, Intel IXP

Dynamic Multicore



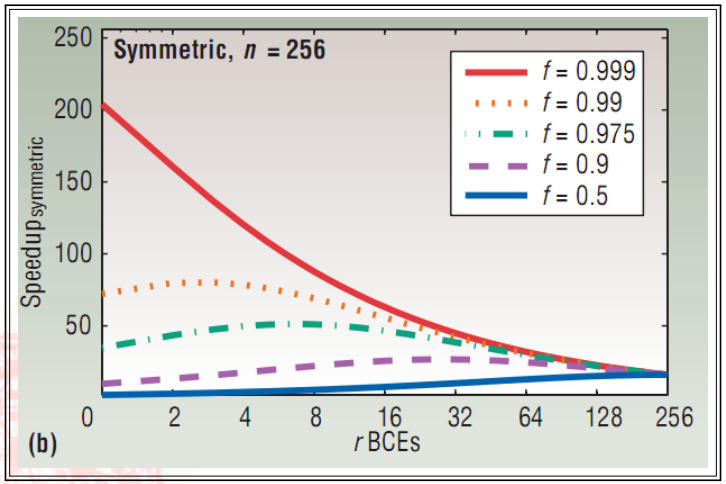
Plan

Amdahl's law for symmetric multicore

- Speedup depends on the parallelizable part of the program, f, by resources n on chip (in BCEs) and by resources dedicated to each core (r BCE)
- There are n/r core, each with performance $perf(r) = \sqrt{r}$
- Amdahl's law, in this case, is

$$\text{Speedup}_{symm}(f, n, r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r) \cdot n/r}}$$

Performances: Symmetric Multicore

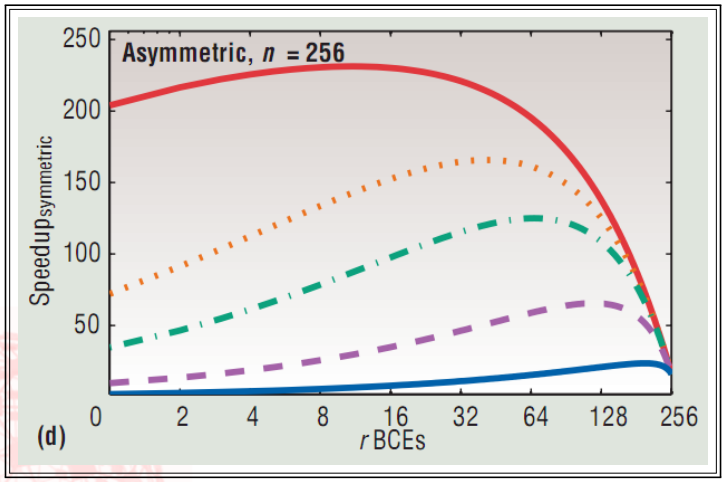


Amdahl's law for asymmetric multicore

- Speedup depends on the parallelizable part of the program, f, by resources n on chip (in BCEs) and by resources dedicated to each core (r BCE)
- For the asymmetric multicore, a processor has more resources (r) and there are $n - r$ core with 1 BCE each
- In total $1 + n - r$ core, with different performances
- In the sequential part of the program, we can use the largest (r BCEs) core.
- In the parallel part, we can use all the cores (each with its performance)
- Amdahl's law, in this case, is:

$$\text{Speedup}_{asymm}(f, n, r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)+n-r}}$$

Performances: Asymmetric Multicore

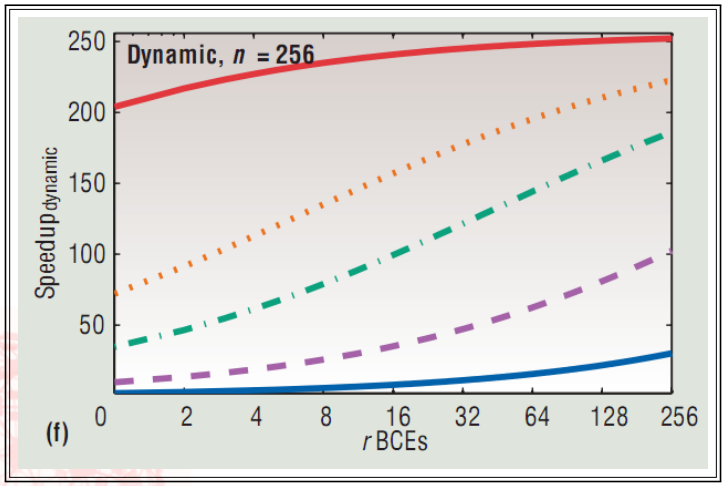


Amdahl's law in the dynamic case

- Speedup depends on the parallelizable part of the program, f, by resources n on chip (in BCEs) and by resources dedicated to each core (r BCE)
- If it is possible to exploit each core (with multithread, for example) then each processor can be both a single processor (with r BCEs), in sequential, and n processors with 1 BCE of processing power
- In the sequential part, we can use the "largest" core (r BCE)
- In the parallel part, we can use all the n cores
- Amdahl's law, in this case, is:

$$\text{Speedup}_{asymm}(f, n, r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{n}}$$

Performances: Dynamic Multicore



Comments

- Software is not infinitely parallel/sequential
- Data movements and tasks add overhead
- Scheduling on asymmetric/dynamic can be more costly than on symmetric
- "Learning curve" for programmers
  - More costly to develop parallel software than sequential software
  - With asymmetric, double (at least) the number of platform to develop software on

Plan

The conclusions of Hill-Marthy paper

Pessimists will bemoan our model's simplicity and lament that much of the design space we explore can't be built with known techniques. We charge you, the reader, to develop better models, and, more importantly, to invent new software and hardware designs that realize the speedup potentials this article displays. Moreover, research leaders should temper the current pendulum swing from the past's underemphasis on parallel research to a future with too little sequential research. To help you get started, we provide slides

The answer

Reevaluating Amdahl's law in the multicore era

Xian-He Sun*, Yong Chen
Computer Science Department, Illinois Institute of Technology, United States

ARTICLE INFO

Article history:
Received 5 December 2008
Received in revised form
22 March 2009
Accepted 9 May 2009
Available online 22 May 2009

Keywords:
Multicore architecture
Scalability
Scalable computing
Memory wall

ABSTRACT

Microprocessor architecture has entered the multicore era. Recently, Hill and Marty presented a pessimistic view of multicore scalability. Their analysis was based on Amdahl's law (i.e. fixed-workload condition) and challenged readers to develop better models. In this study, we analyze multicore scalability under fixed-time and memory-bound conditions and from the data access (memory wall) perspective. We use the same hardware cost model of multicore chips used by Hill and Marty, but achieve very different and more optimistic performance models. These models show that there is no inherent, immovable upper bound on the scalability of multicore architectures. These results complement existing studies and demonstrate that multicore architectures are capable of extensive scalability.

History seems repeating itself

- Worried of the pessimistic implications of Amdahl's law, computer vendors were concentrated thirty years ago on machines with few powerful processors
- After Gustafson-Barsi's law, many vendors built succesfully large parallel machines (nCube, Thinking Machines) ...
- Actually, parallel machines do enroll tens of thousands of processors
- The view of the authors: the Hill-Marthy analysis of manycore processors has the same pitfalls of Amdahl's law (fixed size vs. fixed time) and, more important, there is the *memory wall* problem

Speedup on fixed-time - 1

- Amdahl's law assumes the the problem size (the *workload*) is fixed when run on the unehnanced (sequential system) (fixed-size)
- Gustafson-Barsi's law argues that powerful machines are designer for larger problems and the workload should be scaled up (fixed-time)
- $$\text{Speedup}_{FT} = \frac{\text{Seq time of solving Scaled Workload}}{\text{Par time of solving Scaled Workload}}$$
- Let w = original workload and let w' = the scaled workload be such that they finish in the same time with 1 processor (sequential) and with m processors

Speedup on fixed-time - 2

- Assume scaling only occurs on the parallel part
- Then:
$$w' = (1-f)w + fmw$$
- $$\text{Speedup}_{FT} = \frac{\text{Seq time of solving } w'}{\text{Par time of solving } w'}$$
- $$\text{Speedup}_{FT} = \frac{\text{Seq time of solving } w'}{\text{Seq time of solving } w}$$
- $$\text{Speedup}_{FT} = \frac{w'}{w} = \frac{(1-f)w + fmw}{w} = (1-f) + mf$$
- ... that is the Gustafson-Barsi's law: assuming that the workload can increase, building large scale parallel systems is beneficial

The memory wall - 1

- Not always the workload can increase without bounds: the physical constraint of memory is often "hit"
- Let $w*$ be the scaled workload under memory space constraints
- $$\text{Speedup}_{MB} = \frac{\text{Seq time of solving } w^*}{\text{Par time of solving } w^*}$$
- Assume each node is a processor-memory pair: each node has a memory of size M (L1-cache)
- Let $g(x)$ be the increase of the parallel part of the workload as the memory capacity increases by x
- That is: $w = g(M)$ and the parallel part is $g(m \cdot M) = g(m \cdot g^{-1}(w))$

The memory wall - 2

- Let w = original workload and let w^* = scaled workload be such that they finish in the same time resp. with 1 processor (sequential) and with m processors
- $$\text{Speedup}_{MB} = \frac{\text{Seq time of solving } w^*}{\text{Par time of solving } w^*}$$
$$\text{Speedup}_{MB} = \frac{(1-f)w + f \cdot g(m \cdot g^{-1}(w))}{(1-f)w + \frac{f \cdot g(m \cdot g^{-1}(w))}{m}}$$
- If $g(x)$ is a power function $g(x) = ax^b$ we can write:
$$g(mx) = a(mx)^b = m^b \cdot ax^b = m^b g(x) = \overline{g}(m)g(x)$$
where $\overline{g}(m)$ is the power function with coefficient 1.
- By assuming $g(x)$ polinomial, approximated by its highest degree term, we have:

The memory wall - 3

- We have:
$$\text{Speedup}_{MB} = \frac{(1-f)w + f \cdot \overline{g}(m)w}{(1-f)w + \frac{f \cdot \overline{g}(m)w}{m}}$$
- By simplifying the w we get the final version:
$$\text{Speedup}_{MB} = \frac{(1-f) + f \cdot \overline{g}(m)}{(1-f) + \frac{f \cdot \overline{g}(m)}{m}}$$
- This is called the *Sun-Ni's law* for memory bounded scalability.
- When $\overline{g}(m) = 1$ it becomes the Amdahl's law ...
- When $\overline{g}(m) = m$ it becomes the Gustafson-Barsi law

An example: matrix multiplications

- Two matrices $N \times N$ to multiply
- Computation needed: $2N^3$
- Memory needed: $x = 3N^2 \Rightarrow x/3 = N^2$
- $$g(x) = 2\left(\sqrt{x/3}\right)3 = cx^{3/2}$$
$\Rightarrow \overline{g}(x) = x^{3/2}$
- The speedup for matrix multiplication is:

$$\text{Speedup}_{MB} = \frac{(1-f) + f \cdot \overline{g}(m)}{(1-f) + \frac{f \cdot g(m)}{m}} = \frac{(1-f) + f \cdot m^{3/2}}{(1-f) + f \cdot m^{1/2}}$$

Plan

A view of Hill-Marthy analysis - 1

- Fixed-size workload assumption (like Amdahl's law)
- The speedup is the ratio of the performances (enhanced over the original)
  - since the performances are the reciprocal of execution time

$$\text{Speedup} = \frac{T_{original}}{T_{enhanced}}$$

- If w is the workload, the $T_{original} = w/perf(1) = w$
- The performances over an n-BCE multicore is:

$$T_{enhanced} = \frac{(1-f)w}{perf(r)} + \frac{fw}{n/r \cdot perf(r)}$$

A view of Hill-Marthy analysis - 2

- Then the speedup is:

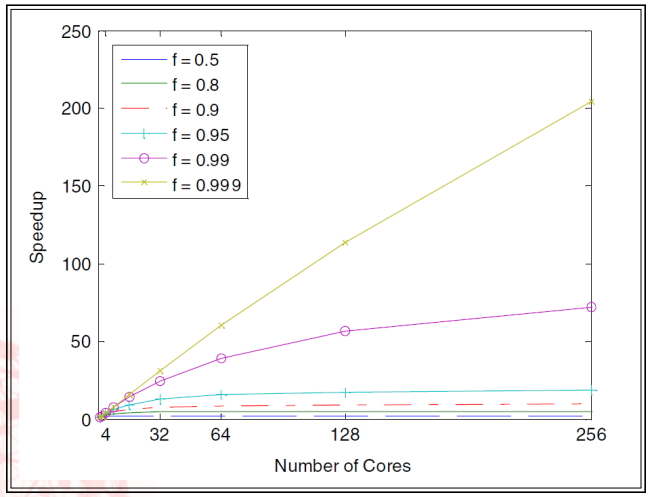
$$\text{Speedup} = \frac{T_{original}}{T_{enhanced}} = \frac{w}{\frac{(1-f)w}{perf(r)} + \frac{fw}{n/r \cdot perf(r)}}$$

$$\text{Speedup} = \frac{1}{\frac{(1-f)}{perf(r)} + \frac{f}{n/r \cdot perf(r)}}$$

- Since $perf(r)$ is a constant c (given a fixed design) and $m = n/r$ we can write "Amdahl's style"

$$\text{Speedup} = \frac{c}{(1-f) + \frac{f}{m}}$$

The result of fixed-size scalability



Plan

Hill-Marthy analysis in fixed-time - 1

- Assume symmetric architecture, each core with its L1 cache
- The Hill-Marthy law is:

$$\text{Speedup} = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot n}}$$

- Now, to measure the fixed-time scalability, we need to fix the *initial* point (let $n = r$) and the *scaled* number of cores (let $n = mr$)
  - then ask that the sequential time at initial point be equal to the the parallel time at the scaled point

Hill-Marthy analysis in fixed-time - 2

- $$\frac{(1-f)w}{perf(r)} + \frac{fw}{perf(r)} = \frac{(1-f)w}{perf(r)} + \frac{fw'}{perf(r) \cdot m}$$
- Hence, we obtain that $w' = mw$

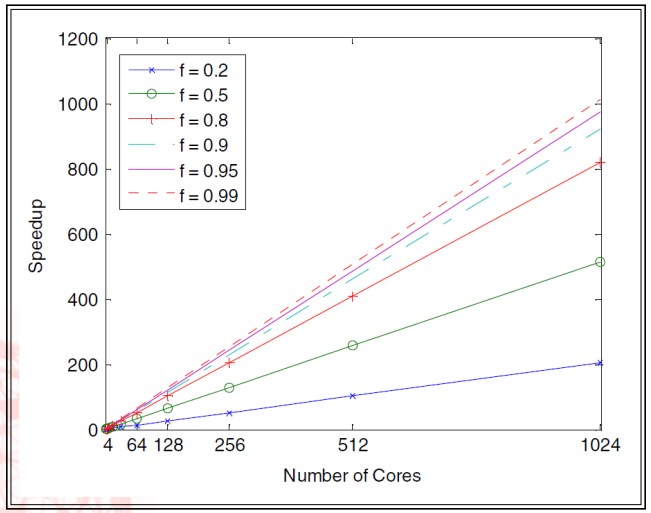
$$\text{Speedup}_{FT} = \frac{\text{Seq time of solving } w'}{\text{Par time of solving } w'} = \frac{\text{Seq time of solving } w'}{\text{Seq time of solving } w}$$

$$\text{Speedup}_{FT} = \frac{\frac{(1-f)w}{perf(r)} + \frac{fw'}{perf(r)}}{w/perf(r))} =$$

$$= \frac{(1-f)w + f(mw)}{w} = (1-f) + fm$$

- That is, multicore are scalable

The result of fixed-time scalability



Plan

Memory bound scalability

- Let w^* be the scaled workload under memory capacity constraint
- Assume we scale from r cores to mr cores
-

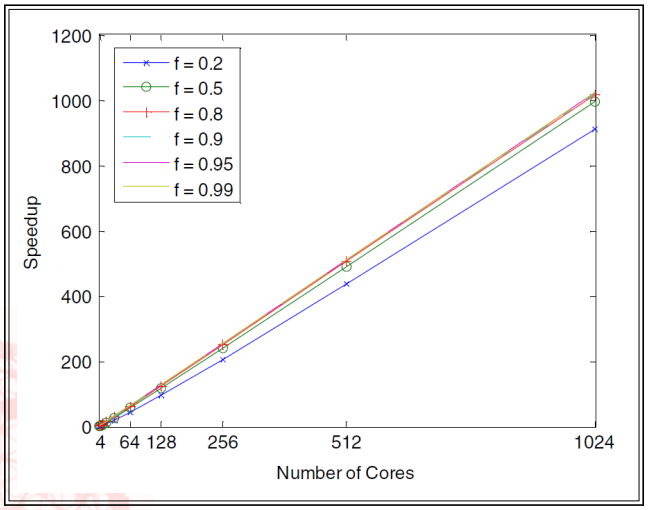
$$\text{Speedup}_{MB} = \frac{\text{Seq time of solving } w'}{\text{Par time of solving } w^*}$$

$$= \frac{\frac{(1-f)w}{perf(r)} + \frac{fw^*}{perf(r)}}{\frac{(1-f)w}{perf(r)} + \frac{fw^*}{m \cdot perf(r)}} =$$

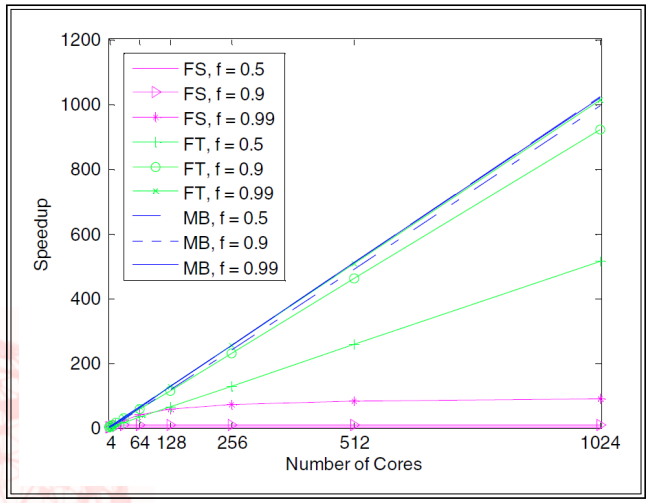
$$= \frac{(1-f)w + fw^*}{(1-f)w + \frac{fw^*}{m}}$$

$$= \frac{(1-f)w + f \cdot \overline{g}(m)w}{(1-f)w + \frac{f \cdot \overline{g}(m)w}{m}} = \frac{(1-f) + f \cdot \overline{g}(m)}{(1-f) + \frac{f \cdot \overline{g}(m)}{m}}$$

The result of memory-bound scalability



Summary of scalability



Conclusions of the authors

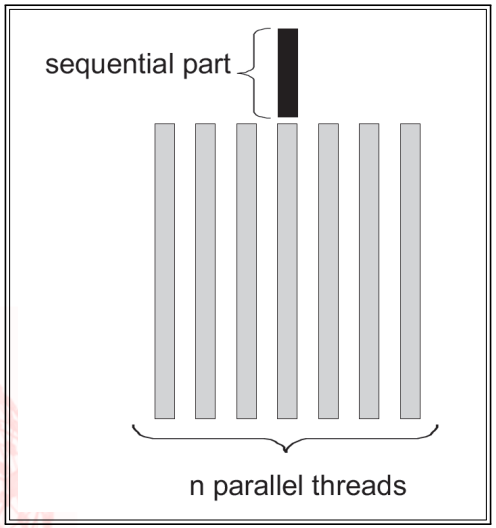
- Hill-Marthy's work is a corollary of Amdahl's law and applies only if users don't increase their computing demands when given more computing power
- Multicore processors **are** scalable, with fixed-time and memory-bound scalability models

Plan

Motivations

- As we have seen, Amdahl's law (and its derivations) have a significant impact on the design of multicore processors
  - "*How many/large should be the core(s) in a multicore?*"
- The software model in Amdahl's law is simple: either sequential or completely parallel code
- Something is missing in this view: the cost of synchronization through critical sections in the parallel part of the code
- By adding this factor, the analysis will show that the parallelism is also fundamentally limited by the synchronization

Amdahl's law software model

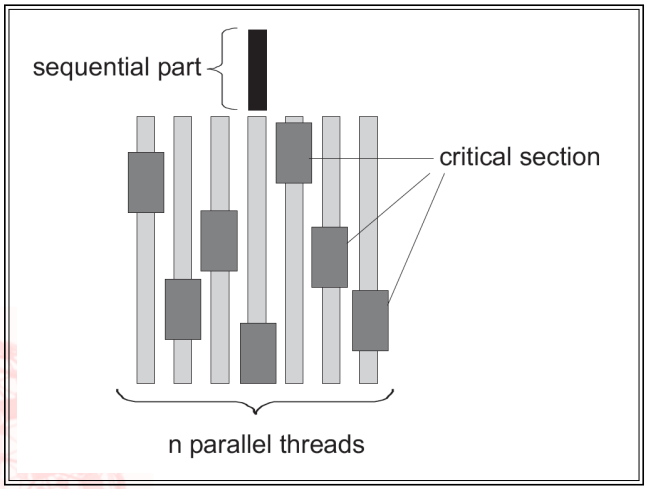


Plan

Modeling critical sections

- Let us split the program execution in three parts:
  1. f_{seq} is the sequential part: it cannot be parallelized (thread spawning, data distribution, merging data from threads, etc.)
  2. $f_{par,ncs}$ is the parallelizable part: infinitely parallelizable since it does not need synchronization (i.e. outside critical sections)
  3. $f_{par,cs}$ is the parallelizable part that requires sunchronization: if two or more threads compete for entering, their execution is serialized
- Of course, $f_{seq} + f_{par,ncs} + f_{par,cs} = 1$

Including the critical sections



Execution time: two cases

- The evaluation is distinguished in two cases:

1. the total time can be approximated by the average per-thread execution time
   - the threads all execute equally long: the average makes "sense" as a measure
2. the total time is determined by a slowest thread
   - one of the thread is consistently slower than others

(1) Avg Thread: the model - 1

- The sequential part does not scale with parallelism: $T_{seq} \propto f_{seq}$
- The parallel fraction that lies outside the critical section is parallelizable: $T_{par,ncs} \propto \frac{f_{par,ncs}}{n}$
- We need to compute $T_{par,cs}$
- Define the *probability for a critical section* during the parallel execution P_{cs}

$$P_{cs} = Pr\{\text{critical section}|\text{parallel}\} = \frac{f_{par,cs}}{f_{par,ncs} + f_{par,cs}}$$

(1) Avg Thread: the model - 2

- The probability for i threads out of n threads to be in CS is binomially distributed (random entering into CS) hence:

$$Pr\{\text{i of n threads in critical section}\} = \binom{n}{i} P_{cs}^{i}(1-P_{cs})^{n-i}$$

- We define the *Contention probability*, P_{ctn} as the probability that 2 (or more) critical sections will contend the same resource (lock, transaction, etc.)
- Now, assuming that the CSs contend randomly, the probability that j threads contend for the same CS is binomially distributed:

$$Pr\{\text{j of i threads contend for the same CS}\} = \binom{i}{j} P_{ctn}^{j}(1-P_{ctn})^{i-j}$$

(1) Avg Thread: the model - 3

- We need to compute the average time spent in CS
- Let one thread T be in the critical section, the probability of other i out of remaining $n-1$ to be in CS is given by:

$$\binom{n-1}{i} P_{cs}^{i}(1-P_{cs})^{n-1-i}$$

- The probability that j out of these i will contend is given by:

$$\binom{i}{j} P_{ctn}^{j}(1-P_{ctn})^{i-j}$$

- If other j thread contend the CS with T, $j+1$ thread will be serialized ...
- ... and it will take $\frac{(j+1)f_{par,cs}}{n}$ to execute the critical section
  - $\frac{jf_{par,cs}}{n}$ to execute the j critical sections (and T is waiting)
  - $\frac{f_{par,cs}}{n}$ to execute the CS of T.

(1) Avg Thread: the model - 4

In summary, the average time spent in a critical section is:

- by averaging over all possible i
- of the probability that i other threads are in a CS ...
- times the probability that j of the i threads contend with T
- times the time spent in to execute the (delayed because of synchronization) Critical Section

$$T_{par,cs} \propto \sum_{i=0}^{n-1} \binom{n-1}{i} P_{cs}^{i}(1-P_{cs})^{n-1-i} \cdot \sum_{j=0}^{i} \binom{i}{j} P_{ctn}^{j}(1-P_{ctn})^{i-j} \cdot \frac{j+1}{n} f_{par,cs}$$

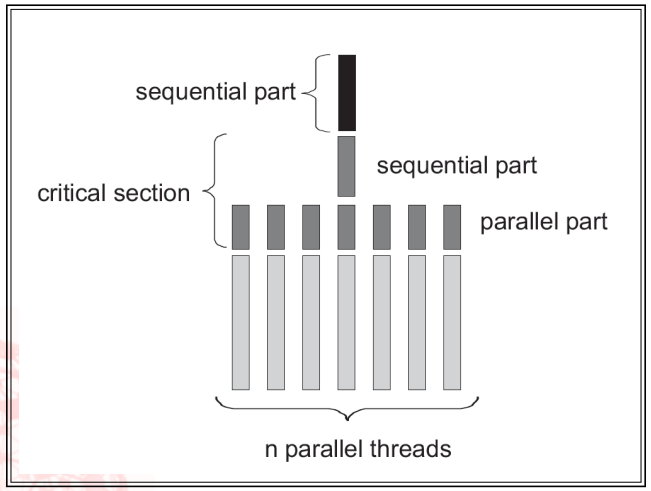
(1) Avg Thread: the model - 5

$$T_{par,cs} \propto \sum_{i=0}^{n-1} \binom{n-1}{i} P_{cs}^{i}(1-P_{cs})^{n-1-i}. \cdot \sum_{j=0}^{i} \binom{i}{j} P_{ctn}^{j}(1-P_{ctn})^{i-j} \cdot \frac{j+1}{n} f_{par,cs}$$

- By exploiting the property that $\sum_{i=0}^{n} \binom{n}{i} P^{i}(1+P)^{n-i} = nP$ (avg of a binomial distribution), we obtain:

$$T_{par,cs} \propto f_{par,cs} \cdot \left(P_{cs}P_{ctn} + (1 - P_{cs}P_{ctn}) \cdot \frac{1}{n}\right)$$

Modeling as a sequential plus a parallel part



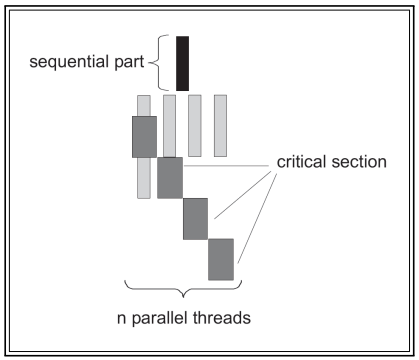
Execution time: two cases

- The evaluation is distinguished in two cases:

1. the total time can be approximated by the average per-thread execution time
   - the threads all execute equally long: the average makes "sense" as a measure
2. the total time is determined by a slowest thread
   - one of the thread is consistently slower than others

(2) Slowest Thread: the model - 1

- The assumption that the time of execution of parallel part is dominated by the average per-thread execution time does not always hold
- In case the contention is high, the total time of execution may be dominated by one slowest thread.



(2) Slowest Thread: the model - 2

- Assume that the execution of the slowest thread is determined by the serialized execution of all the contending critical sections
- The average time of the slowest thread is influenced by:
  1. sequential part: $T_{seq} \propto f_{seq}$
  2. time spent in contending: $T_{par,cs} \propto f_{par,cs} P_{ctn}$
  3. $T_{par} \propto \frac{f_{par,cs}(1-P_{ctn})+f_{par,ncs}}{2n}$
- Then, the average execution of the slowest thread is:

$$T \propto f_{seq} + f_{par,cs} P_{ctn} + \frac{f_{par,cs}(1 - P_{ctn}) + f_{par,ncs}}{2n}$$

Integration to Amdahl's law

- Since the time of a parallel program is:

$$T \propto f_{seq} + \max\Big\{ f_{par,cs} P_{cs} P_{ctn} + \frac{f_{par,cs}(1 - P_{cs}P_{ctn}) + f_{par,ncs}}{n}, \; f_{par,cs} P_{ctn} + \frac{f_{par,cs}(1 - P_{ctn}) + f_{par,ncs}}{2n} \Big\}$$

- Then, the Amdahl's law can be, asymptotically, reduced to:

$$\lim_{n\to\infty} S = \frac{1}{f_{seq} + \max\{f_{par,cs}P_{cs}P_{ctn}, f_{par,cs}P_{ctn}\}} = \frac{1}{f_{seq} + f_{par,cs}P_{ctn}}$$

Plan

Asymmetric multicore - 1

- One of the arguments in favour of asymmetric multicore processors is that the sequential part gets quickly executed by the (only) large core ...
- ... while the parallel part is spedup by the many (small) cores
- Critical sections do limit the potential benefit of a single large core: when contention occurs, sequential execution due to queuing on a resource is performed on the small cores, thereby inefficiently.

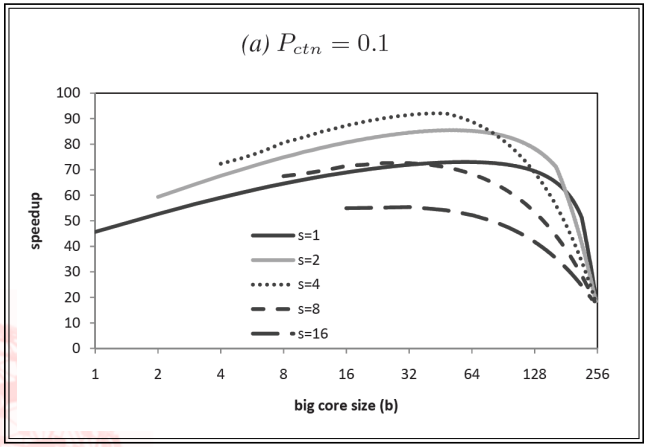
Asymmetric multicore - 2

- Assuming n cores with performances of 1, and one single core with performance pm the time is:

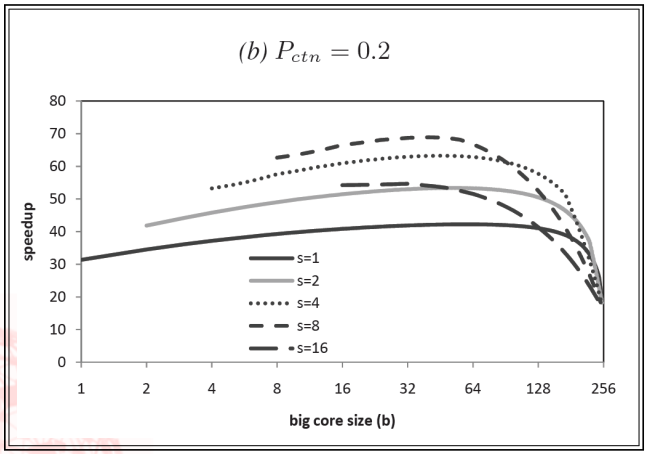
$$T \propto \frac{f_{seq}}{p} + \max\Big\{ f_{par,cs} P_{cs} P_{ctn} + \frac{f_{par,cs}(1 - P_{cs}P_{ctn}) + f_{par,ncs}}{n+p}, \; f_{par,cs} P_{ctn} + \frac{f_{par,cs}(1 - P_{ctn}) + f_{par,ncs}}{2(n+p)} \Big\}$$

- Then, small cores should be sufficiently large as not to impact negatively on the execution of the sequential part due to contention to CSs

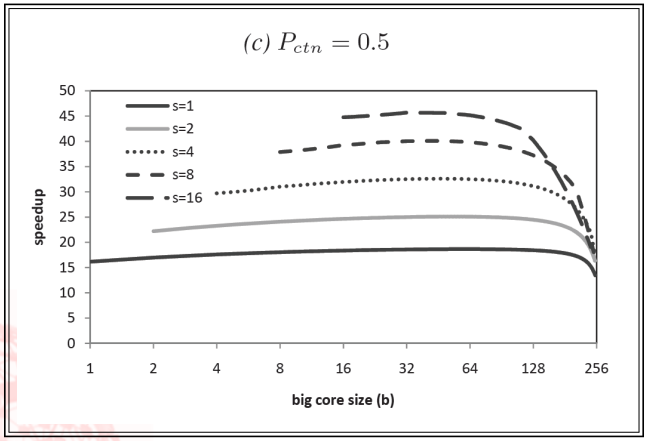
Big (b) vs. small (s) cores - 1



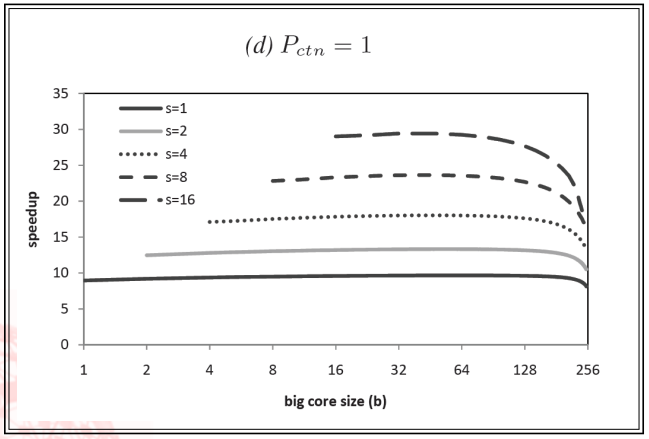
Big (b) vs. small (s) cores - 2



Big (b) vs. small (s) cores - 3



Big (b) vs. small (s) cores - 4



Plan

By considering concurrency ...

- ... asymmetric multicore offer smaller performance benefits than expected
  - contention is executed on one of the small cores
- ... many small cores in an asymmetric multicore may significatively bound the performances, when high contention rates are expected
- ... techniques for accelerating critical sections are needed (such as executing CSs on the big core, or predicting which CSs will compete, ...)
- Research still working on this ...