Parallel Programming
Theoretical Speedup Laws

Radu Nicolescu
Department of Computer Science
University of Auckland

4 June 2019

1. Speedup concepts
2. Amdahl's formula
3. Gustafson's formula
4. Reconciling
5. Maximum
6. Challenge
7. Guenther

Outline

Basic speedup concepts

```
1 Speedup = sequential-time / parallel-time
```

Assume first an ideal case:

- A program has an intrinsic sequential part, S
- And a parallel part, P, which can be parallelized on any number of processing elements (processors, cores)
- We first ignore the very real parallel overhead, created by extra code, cache architectures, bandwidth limitations, synchronization requirements, ...

Basic speedup concepts

```
1 Speedup = sequential-time / parallel-time
```

Assume first an ideal case:

- A program has an intrinsic sequential part, S
- And a parallel part, P, which can be parallelized on any number of processing elements (processors, cores)
- We first ignore the very real parallel overhead, created by extra code, cache architectures, bandwidth limitations, synchronization requirements, ...

Basic speedup concepts

```
1 Speedup = sequential-time / parallel-time
```

Assume first an ideal case:

- A program has an intrinsic sequential part, S
- And a parallel part, P, which can be parallelized on any number of processing elements (processors, cores)
- We first ignore the very real parallel overhead, created by extra code, cache architectures, bandwidth limitations, synchronization requirements, ...

Basic speedup concepts

```
1 Speedup = sequential-time / parallel-time
```

Assume first an ideal case:

- A program has an intrinsic sequential part, S
- And a parallel part, P, which can be parallelized on any number of processing elements (processors, cores)
- We first ignore the very real parallel overhead, created by extra code, cache architectures, bandwidth limitations, synchronization requirements, ...

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Basic speedup concepts

The parallel part can be dived into chunks, each of which can run on its own thread or processor

Caveat:

- *if* the chunks are too small (too much fragmentation)
- *if* the chunks frequently try to access their shared data (too much race)
- *if* the chunks exchange too many small messages (too chatty)
- *then* the ratio overhead / business can become prohibitive
- *and* the parallel version can in practice run slower than the sequential

As mentioned, for the moment, we will assume an ideal case, when *none* of these happens

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

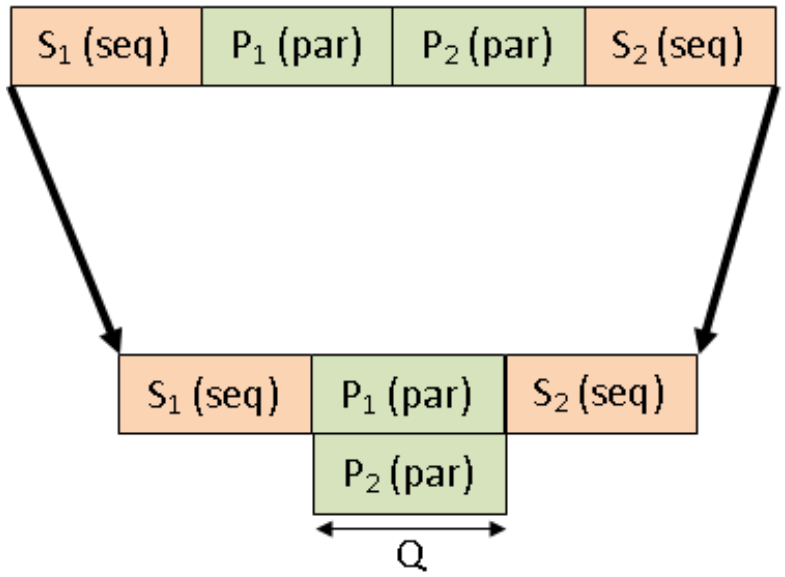
Speedup concepts

- How much speedup can you gain, assuming an unlimited supply of processing elements, N?
- Amdahl and Gustafson have proposed two similar formulas
- However, their formulas indicate different speedup bounds !?
- Amdahl ("the pessimist") indicates a bounded speedup ☹
- Gustafson ("the optimist") indicates an unlimited speedup ☺
- How to reconcile these formulas?
- At least one seems to be lying...
- Or, are they both lying?
- Or, could they both be correct?

Outline

Amdahl's formula

Assume a fixed program, π, as indicated by the following diagram.

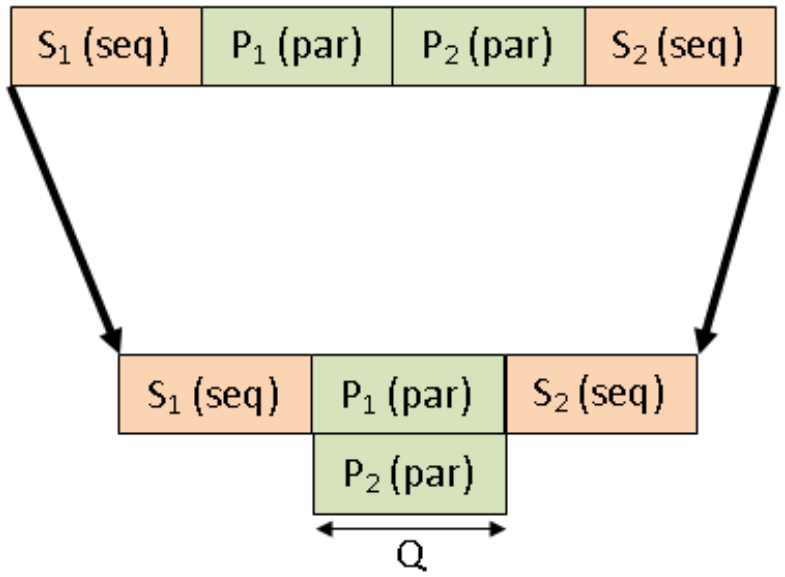


More specifically, in this example: $N = 2$, $S = S_1 + S_2$, $P = P_1 + P_2$, $S_i = P_i = Q$ fixed.

```
1 Speedup_A(π) = (S+P)/(S+P/2) = 4/3 = 1.33
```

Amdahl's formula

Assume a fixed program, π, as indicated by the following diagram.



More specifically, in this example: $N = 2$, $S = S_1 + S_2$, $P = P_1 + P_2$, $S_i = P_i = Q$ fixed.

```
1 Speedup_A(π) = (S+P)/(S+P/2) = 4/3 = 1.33
```

Amdahl's formula

More general, for arbitrary N:

$$\text{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

Setting $\alpha = S/(S+P)$ (Amdahl's coefficient), we obtain an equivalent version of Amdahl's formula:

$$\text{Speedup}_A(\pi) = 1 \,/\, (\alpha + (1-\alpha)/N) \nearrow 1/\alpha$$

Upper-bound for speedup: $1+P/S = 1/\alpha$! ☹☹☹

Amdahl's formula

More general, for arbitrary N:

$$\text{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

Setting $\alpha = S/(S+P)$ (Amdahl's coefficient), we obtain an equivalent version of Amdahl's formula:

$$\text{Speedup}_A(\pi) = 1 / (\alpha + (1-\alpha)/N) \nearrow 1/\alpha$$

Upper-bound for speedup: $1+P/S = 1/\alpha$! ☹☹☹

Amdahl's formula

More general, for arbitrary N:

$$\text{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

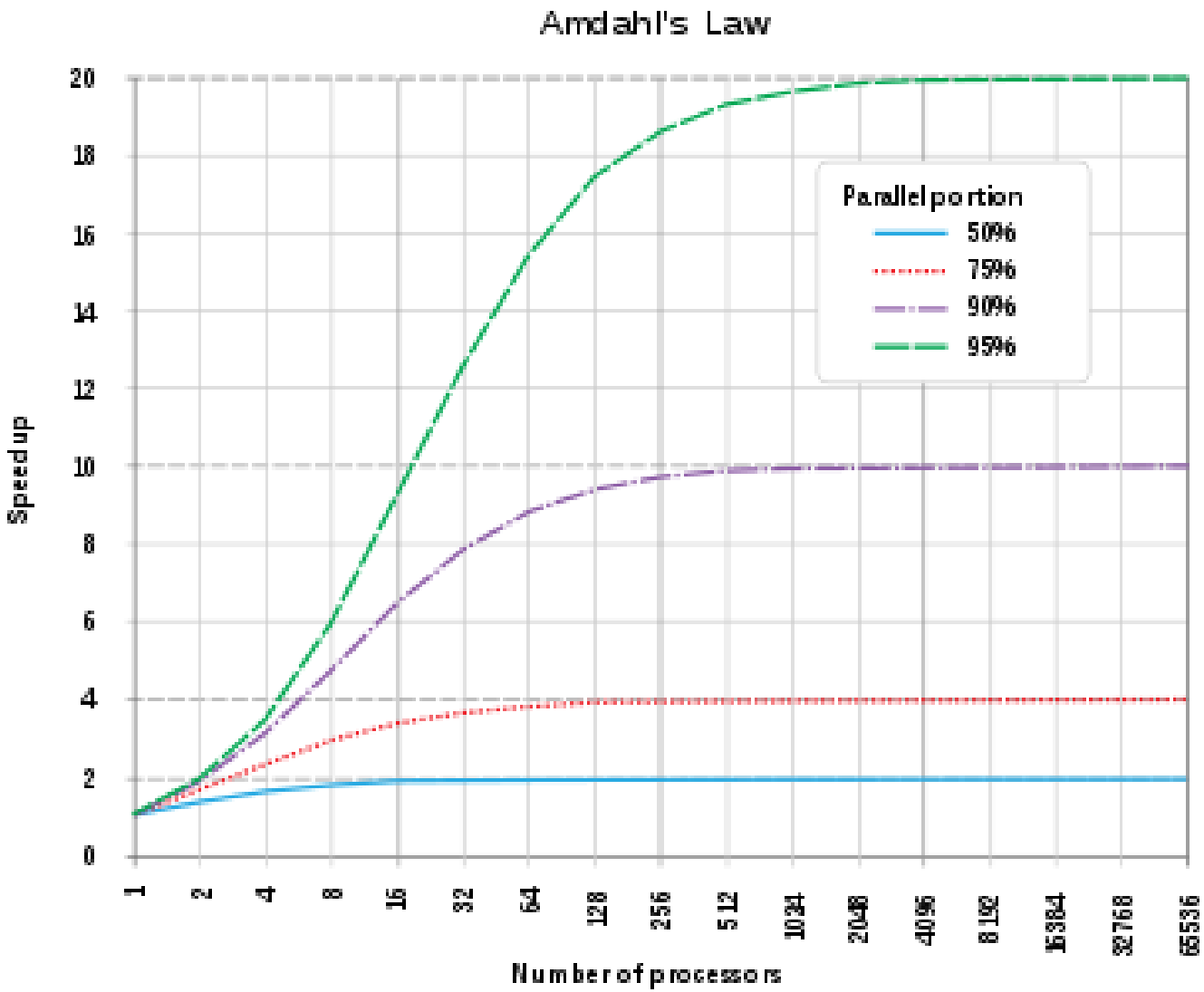
Setting $\alpha = S/(S+P)$ (Amdahl's coefficient), we obtain an equivalent version of Amdahl's formula:

$$\text{Speedup}_A(\pi) = 1 / (\alpha + (1-\alpha)/N) \nearrow 1/\alpha$$

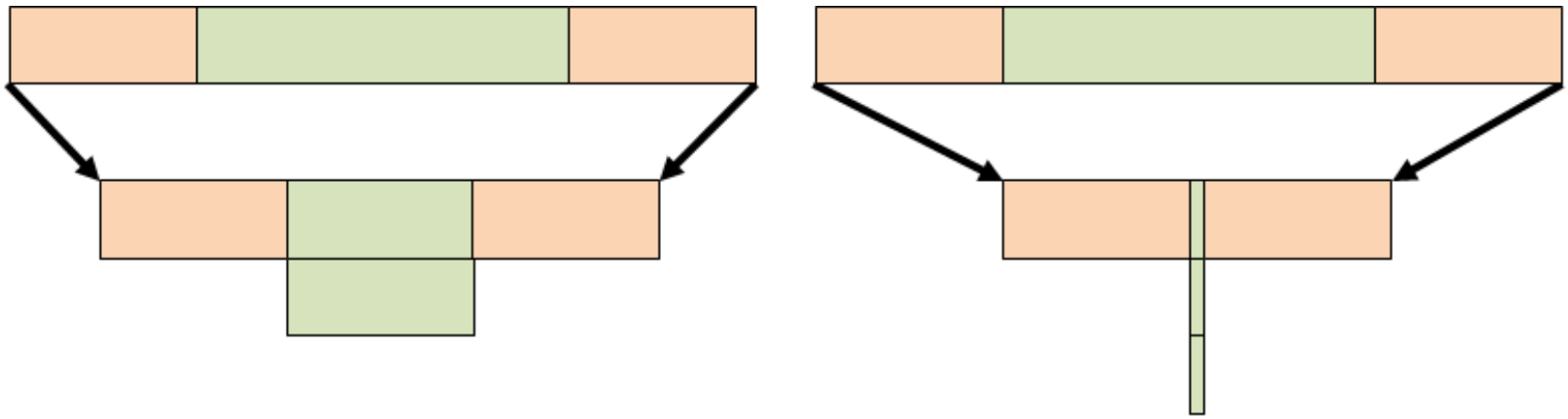
Upper-bound for speedup: $1+P/S = 1/\alpha$! ☹☹☹

Amdahl's formula

Typical speedup graph (from wikipedia):



Amdahl's formula



You can squeeze the parallel part as much as you like, by throwing in more processors, but you cannot squeeze the sequential part!

Outline

Gustafson's formula

Assume a family of similar programs, Π, which have a fixed size sequential part (S) and a variable size parallel part (P).

Given any fixed time-frame, $T = S + Q$ (thus Q is also fixed),

consider a sequence of programs $\{\pi_N\}_n \subset \Pi$, where their parallel part, P_N, is multiple of Q, $P_N = NQ$.

Clearly, given N processors, program π_N takes
$S + P_N/N = S + NQ/N = S + Q = T$ time
(i.e. fits into the fixed time-frame T).

Gustafson's formula

Assume a family of similar programs, Π, which have a fixed size sequential part (S) and a variable size parallel part (P).

Given any fixed time-frame, $T = S + Q$ (thus Q is also fixed),

consider a sequence of programs $\{\pi_N\}_n \subset \Pi$, where their parallel part, P_N, is multiple of Q, $P_N = NQ$.

Clearly, given N processors, program π_N takes
$S + P_N/N = S + NQ/N = S + Q = T$ time
(i.e. fits into the fixed time-frame T).

Gustafson's formula

Assume a family of similar programs, Π, which have a fixed size sequential part (S) and a variable size parallel part (P).

Given any fixed time-frame, $T = S + Q$ (thus Q is also fixed),

consider a sequence of programs $\{\pi_N\}_n \subset \Pi$, where their parallel part, P_N, is multiple of Q, $P_N = NQ$.

Clearly, given N processors, program π_N takes
$S + P_N/N = S + NQ/N = S + Q = T$ time
(i.e. fits into the fixed time-frame T).

Gustafson's formula

Assume a family of similar programs, Π, which have a fixed size sequential part (S) and a variable size parallel part (P).

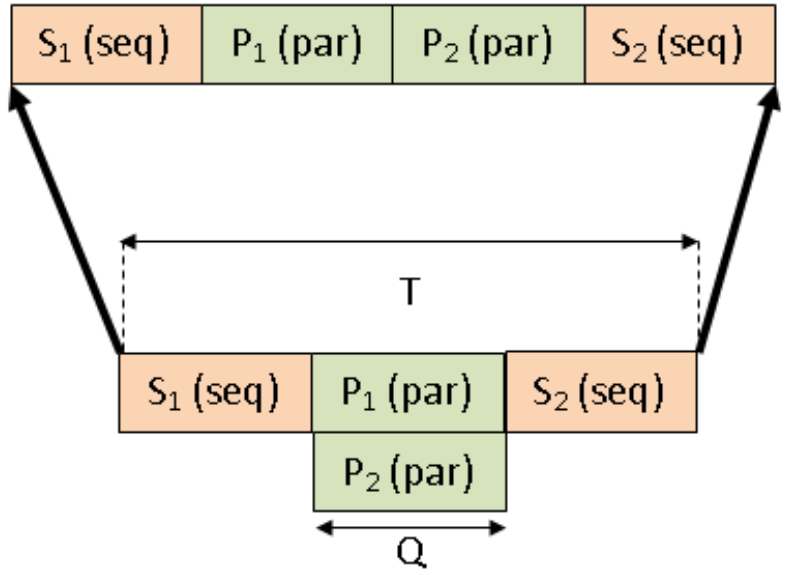
Given any fixed time-frame, $T = S + Q$ (thus Q is also fixed),

consider a sequence of programs $\{\pi_N\}_n \subset \Pi$, where their parallel part, P_N, is multiple of Q, $P_N = NQ$.

Clearly, given N processors, program π_N takes
$S + P_N/N = S + NQ/N = S + Q = T$ time
(i.e. fits into the fixed time-frame T).

Gustafson's formula

For example, consider a program $\pi_2 \in \Pi$, where $P_2 = 2Q$ is evenly divided between two processors, as indicated by the following diagram:

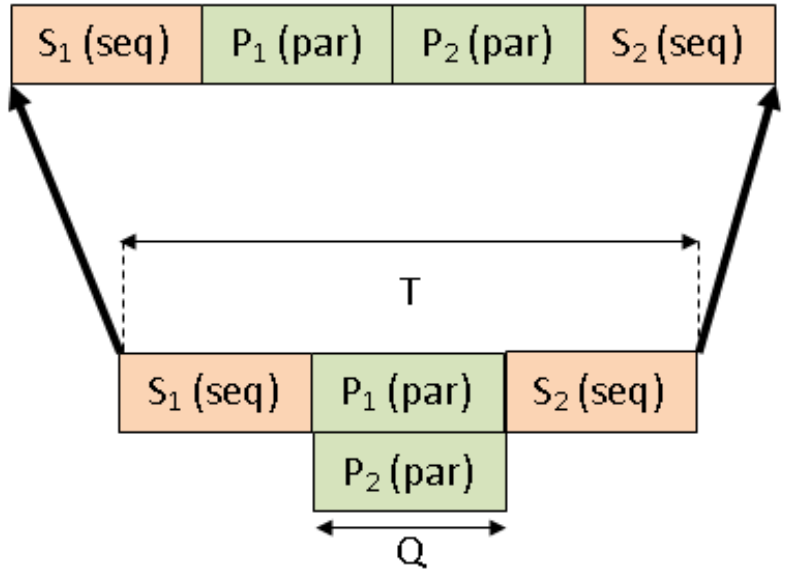


For π_2, we obtain the same speedup as indicated by Amdahl's law:

```
1  Speedup_G(π2) = (S+2Q)/(S+Q) = 4/3 = 1.33
```

Gustafson's formula

For example, consider a program $\pi_2 \in \Pi$, where $P_2 = 2Q$ is evenly divided between two processors, as indicated by the following diagram:



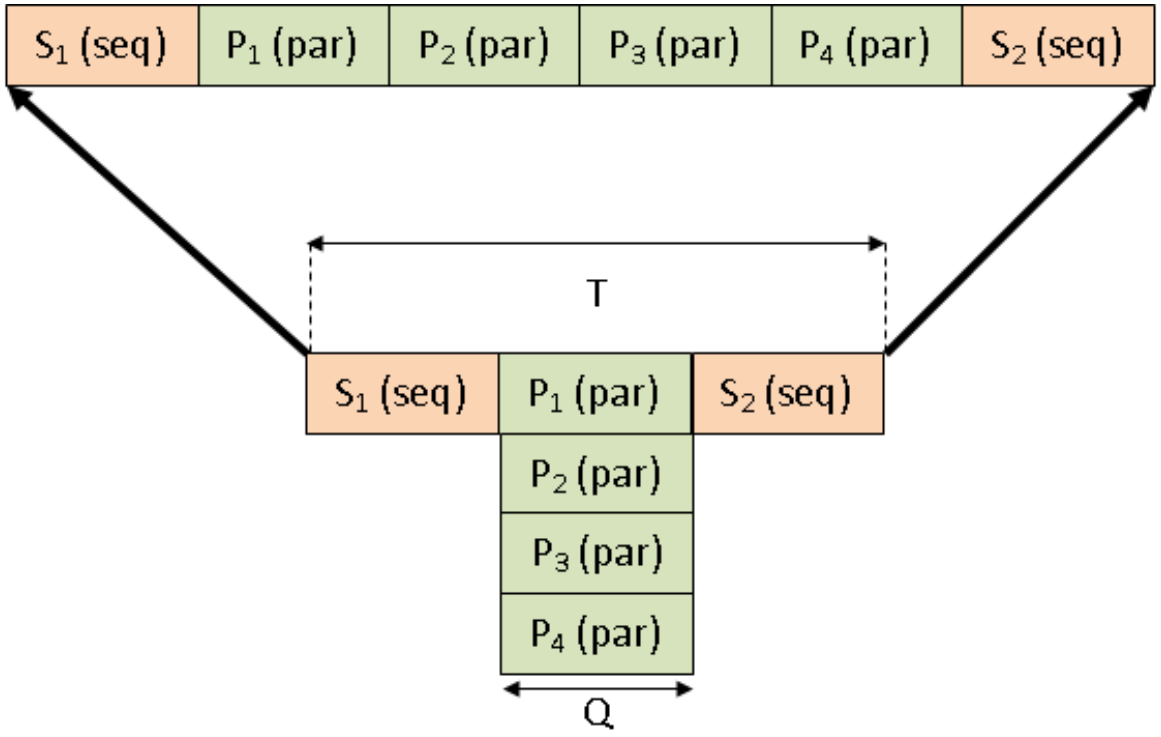
For π_2, we obtain the same speedup as indicated by Amdahl's law:

1 $$\boxed{\mathrm{Speedup}_G(\pi_2) = (S+2Q)/(S+Q) = 4/3 = 1.33}$$

Gustafson's formula

However, other programs from the same family can be much more substantially accelerated.

For example, consider a program $\pi_4 \in \Pi$, as indicated by the following diagram:



Gustafson's formula

Program π_4 shows a higher speedup:

$$\text{Speedup}_G(\pi_4) = (S+4Q)/(S+Q) = 6/3 = 2.00$$

Gustafson's formula

More general, the family $\{\pi_N\}_N$ shows unbounded speedup:

1 $$\mathrm{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Setting $\gamma = S/(S+Q)$ (Gustafson's coefficient), we obtain an equivalent version of Gustafson's formula:

1 $$\mathrm{Speedup}_G(\pi_N) = \gamma + (1-\gamma)N \nearrow \infty$$

There is no upper-bound for speeding up! ☺☺☺

Gustafson's formula

More general, the family $\{\pi_N\}_N$ shows unbounded speedup:

$$\text{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Setting $\gamma = S/(S+Q)$ (Gustafson's coefficient), we obtain an equivalent version of Gustafson's formula:

$$\text{Speedup}_G(\pi_N) = \gamma + (1-\gamma)N \nearrow \infty$$

There is no upper-bound for speeding up! ☺☺☺

Gustafson's formula

More general, the family $\{\pi_N\}_N$ shows unbounded speedup:

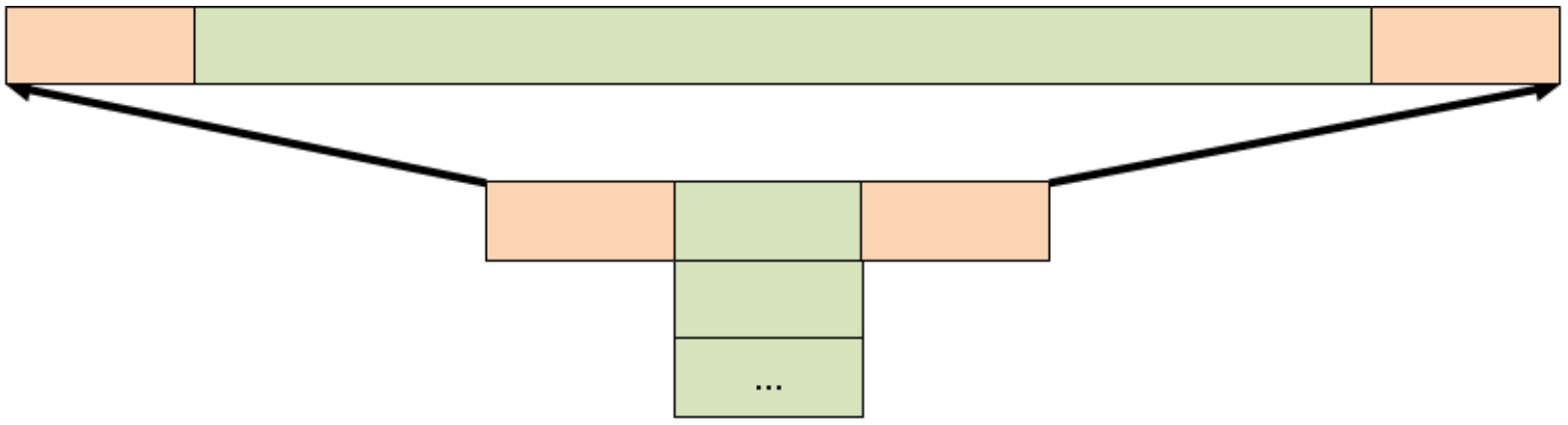
$$\text{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Setting $\gamma = S/(S+Q)$ (Gustafson's coefficient), we obtain an equivalent version of Gustafson's formula:

$$\text{Speedup}_G(\pi_N) = \gamma + (1-\gamma)N \nearrow \infty$$

There is no upper-bound for speeding up! ☺☺☺

Gustafson's formula



Gustafson's linear formula shows that, in any given time-frame, T, you can run arbitrarily large problems (from such a family of programs)!

Outline

Reconciling Amdahl and Gustafson

- Consider a given fixed-size problem
  - Amdahl shows that you cannot speed it up arbitrarily
  - regardless how many processing elements you put to the task
  - the speedup is bounded from above by $(S+P)/S = 1/\alpha$
- Consider a given fixed time-frame, T
  - Gustafson shows that in this time-frame you can run arbitrarily large problems
  - as long as you have an unlimited supply of processing elements
  - his speedup formula considers related problems of different parallel size

Reconciling Amdahl and Gustafson

- Consider a given fixed-size problem
  - Amdahl shows that you cannot speed it up arbitrarily
  - regardless how many processing elements you put to the task
  - the speedup is bounded from above by $(S+P)/S = 1/\alpha$
- Consider a given fixed time-frame, T
  - Gustafson shows that in this time-frame you can run arbitrarily large problems
  - as long as you have an unlimited supply of processing elements
  - his speedup formula considers related problems of different parallel size

Essentially, same formula, with different fixed params!

- Amdahl: fixed S and P, i.e. fixed-size problem = S+P

$$\mathrm{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

- Conversions

$$P = NQ$$
$$Q = P/N$$

- Gustafson: fixed S and Q, i.e. fixed time frame = S+Q

$$\mathrm{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Essentially, same formula, with different fixed params!

- Amdahl: fixed S and P, i.e. fixed-size problem = S+P

$$\text{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

- Conversions

$$P = NQ$$
$$Q = P/N$$

- Gustafson: fixed S and Q, i.e. fixed time frame = S+Q

$$\text{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Essentially, same formula, with different fixed params!

- Amdahl: fixed S and P, i.e. fixed-size problem = S+P

$$\mathrm{Speedup}_A(\pi) = (S+P)/(S+P/N) \nearrow (S+P)/S$$

- Conversions

$$P = NQ$$
$$Q = P/N$$

- Gustafson: fixed S and Q, i.e. fixed time frame = S+Q

$$\mathrm{Speedup}_G(\pi_N) = (S+NQ)/(S+Q) \nearrow \infty$$

Critics

Both laws consider ideal cases which ignore that:

- Typically, the coefficients depend on the number of processors, N, and on the problem size.
- In particular, the size of the sequential part, S, may also grow with the problem size (although not necessarily as fast as the size of the parallel part, P).
- There are many important but ignored factors in the parallel overhead and in the underlying system, software and hardware (e.g., racing, memory/caching, interconnection architecture)

On the other side, exploiting such factors, is there possible to beat Amdahl, or, at least, to "cheat" him (to some extent)?

Critics

Both laws consider ideal cases which ignore that:

- Typically, the coefficients depend on the number of processors, N, and on the problem size.
- In particular, the size of the sequential part, S, may also grow with the problem size (although not necessarily as fast as the size of the parallel part, P).
- There are many important but ignored factors in the parallel overhead and in the underlying system, software and hardware (e.g., racing, memory/caching, interconnection architecture)

On the other side, exploiting such factors, is there possible to beat Amdahl, or, at least, to "cheat" him (to some extent)?

Critics

Both laws consider ideal cases which ignore that:

- Typically, the coefficients depend on the number of processors, N, and on the problem size.
- In particular, the size of the sequential part, S, may also grow with the problem size (although not necessarily as fast as the size of the parallel part, P).
- There are many important but ignored factors in the parallel overhead and in the underlying system, software and hardware (e.g., racing, memory/caching, interconnection architecture)

On the other side, exploiting such factors, is there possible to beat Amdahl, or, at least, to "cheat" him (to some extent)?

Critics

Both laws consider ideal cases which ignore that:

- Typically, the coefficients depend on the number of processors, N, and on the problem size.
- In particular, the size of the sequential part, S, may also grow with the problem size (although not necessarily as fast as the size of the parallel part, P).
- There are many important but ignored factors in the parallel overhead and in the underlying system, software and hardware (e.g., racing, memory/caching, interconnection architecture)

On the other side, exploiting such factors, is there possible to beat Amdahl, or, at least, to "cheat" him (to some extent)?

Critics

Both laws consider ideal cases which ignore that:

- Typically, the coefficients depend on the number of processors, N, and on the problem size.
- In particular, the size of the sequential part, S, may also grow with the problem size (although not necessarily as fast as the size of the parallel part, P).
- There are many important but ignored factors in the parallel overhead and in the underlying system, software and hardware (e.g., racing, memory/caching, interconnection architecture)

On the other side, exploiting such factors, is there possible to beat Amdahl, or, at least, to "cheat" him (to some extent)?

Can we do better?

Interesting readings:

- Herb Sutter: *Break Amdahl's Law!*, Dr. Dobb's, January 17, 2008. "To paraphrase Churchill: We shall cheat him (Amdahl) in the data sizes, we shall cheat him in the loops, we shall cheat him in the pipelines, we shall cheat him in the unimagined new features. We shall never surrender!"
- Yuan Shi: *Reevaluating Amdahl's Law and Gustafson's Law*, Temple University, October 1996.
- Andrzej Karbowski: *Amdahl and Gustafson-Barsis Laws Revisited*, Warsaw University of Technology, September 2008.

Outline

Maximum speedup

What is the maximum speedup, for a given problem and number of processors, N?

- According to Amdahl

$$\text{Speedup}_A = (S+P)/(S+P/N) \leq N$$

- According to Gustafson

$$\text{Speedup}_G = (S+QN)/(S+Q) \leq N$$

As expected, both formulas agree in this case; the speedup is maximum linear in N:

$$\text{Speedup} \leq N$$

Maximum speedup

What is the maximum speedup, for a given problem and number of processors, N?

- According to Amdahl

$$\text{Speedup}_A = (S+P)/(S+P/N) \leq N$$

- According to Gustafson

$$\text{Speedup}_G = (S+QN)/(S+Q) \leq N$$

As expected, both formulas agree in this case; the speedup is maximum linear in N:

$$\text{Speedup} \leq N$$

Maximum speedup

What is the maximum speedup, for a given problem and number of processors, N?

- According to Amdahl

$$\text{Speedup}_A = (S+P)/(S+P/N) \leq N$$

- According to Gustafson

$$\text{Speedup}_G = (S+QN)/(S+Q) \leq N$$

As expected, both formulas agree in this case; the speedup is maximum linear in N:

$$\text{Speedup} \leq N$$

Maximum speedup

What is the maximum speedup, for a given problem and number of processors, N?

- According to Amdahl

$$\text{Speedup}_A = (S+P)/(S+P/N) \leq N$$

- According to Gustafson

$$\text{Speedup}_G = (S+QN)/(S+Q) \leq N$$

As expected, both formulas agree in this case; the speedup is maximum linear in N:

$$\text{Speedup} \leq N$$

Outline

Super-linear speedup?

The quest for the Holy Grail in Parallel Computing...

Consider a suboptimal selection sort, with sequential runtime k^2 steps (where k=a.Length):

```
1  void SelectionSort(int[] a) {
2      for (int i = 0; i < a.Length - 1; i++) {
3          int min = i;
4          for (int j = i + 1; j < a.Length; j++) {
5              if (a[j] < a[min]) min = j;
6          }
7          Swap(ref a[i], ref a[min]);
8      }
9  }
```

Super-linear speedup?

The quest for the Holy Grail in Parallel Computing...

Consider a suboptimal selection sort, with sequential runtime k^2 steps (where k=a.Length):

```
1  void SelectionSort(int[] a) {
2      for (int i = 0; i < a.Length - 1; i++) {
3          int min = i;
4          for (int j = i + 1; j < a.Length; j++) {
5              if (a[j] < a[min]) min = j;
6          }
7          Swap(ref a[i], ref a[min]);
8      }
9  }
```

Super-linear speedup?

The quest for the Holy Grail in Parallel Computing...

Consider a suboptimal selection sort, with sequential runtime k^2 steps (where k=a.Length):

```
1  void SelectionSort(int[] a) {
2      for (int i = 0; i < a.Length - 1; i++) {
3          int min = i;
4          for (int j = i + 1; j < a.Length; j++) {
5              if (a[j] < a[min]) min = j;
6          }
7          Swap(ref a[i], ref a[min]);
8      }
9  }
```

Super-linear speedup?

Consider a parallel version with 2 tasks running on 2 cores (2 parallel tasks)

- Task 1, selection sort on 1st half of a : $(k/2)^2 = k^2/4$ steps
- Task 2, selection sort on 2nd half of a : $(k/2)^2 = k^2/4$ steps
- Join tasks 1 and 2 and sequentially merge results (linear time, negligible for large k)

Total parallel time $= k^2/4$ steps, twice faster than maximum expected, $k^2/2$ steps!

Super-linear speedup?

Consider a parallel version with 2 tasks running on 2 cores (2 parallel tasks)

- Task 1, selection sort on 1st half of a : $(k/2)^2 = k^2/4$ steps
- Task 2, selection sort on 2nd half of a : $(k/2)^2 = k^2/4$ steps
- Join tasks 1 and 2 and sequentially merge results (linear time, negligible for large k)

Total parallel time $= k^2/4$ steps, twice faster than maximum expected, $k^2/2$ steps!

Super-linear speedup?

Consider a parallel version with 2 tasks running on 2 cores (2 parallel tasks)

- Task 1, selection sort on 1st half of a : $(k/2)^2 = k^2/4$ steps
- Task 2, selection sort on 2nd half of a : $(k/2)^2 = k^2/4$ steps
- Join tasks 1 and 2 and sequentially merge results (linear time, negligible for large k)

Total parallel time $= k^2/4$ steps, twice faster than maximum expected, $k^2/2$ steps!

Super-linear speedup?

- This parallel version seems to show a super-linear $4\times$ speedup, while running only on a 2-core machine!
- Could this be true, or is it a mirage? Explain!
- Hint: Re-run the parallel version sequentially!

Super-linear speedup?

- This parallel version seems to show a super-linear $4\times$ speedup, while running only on a 2-core machine!
- Could this be true, or is it a mirage? Explain!
- Hint: Re-run the parallel version sequentially!

Super-linear speedup?

- This parallel version seems to show a super-linear $4\times$ speedup, while running only on a 2-core machine!
- Could this be true, or is it a mirage? Explain!
- Hint: Re-run the parallel version sequentially!

Super-linear speedup?

- Hint: Re-run the parallel version sequentially!
- You will get another sequential version, which runs in $k^2/4 + k^2/4 = k^2/2$ steps, i.e. twice faster than the original sequential version!
- Conclusion: Your parallel version uses a different algorithm than the original sequential version.
- In fact, it is the first step towards a better (optimal) algorithm: merge sort, which runs in $k \log k$ steps!
- A parallel version of merge sort will show only linear speedup (not super-linear).

Super-linear speedup?

- Hint: Re-run the parallel version sequentially!
- You will get another sequential version, which runs in $k^2/4 + k^2/4 = k^2/2$ steps, i.e. twice faster than the original sequential version!
- Conclusion: Your parallel version uses a different algorithm than the original sequential version.
- In fact, it is the first step towards a better (optimal) algorithm: merge sort, which runs in $k \log k$ steps!
- A parallel version of merge sort will show only linear speedup (not super-linear).

Super-linear speedup?

- Hint: Re-run the parallel version sequentially!
- You will get another sequential version, which runs in $k^2/4 + k^2/4 = k^2/2$ steps, i.e. twice faster than the original sequential version!
- Conclusion: Your parallel version uses a different algorithm than the original sequential version.
- In fact, it is the first step towards a better (optimal) algorithm: merge sort, which runs in $k \log k$ steps!
- A parallel version of merge sort will show only linear speedup (not super-linear).

Super-linear speedup?

- Hint: Re-run the parallel version sequentially!
- You will get another sequential version, which runs in $k^2/4 + k^2/4 = k^2/2$ steps, i.e. twice faster than the original sequential version!
- Conclusion: Your parallel version uses a different algorithm than the original sequential version.
- In fact, it is the first step towards a better (optimal) algorithm: merge sort, which runs in $k \log k$ steps!
- A parallel version of merge sort will show only linear speedup (not super-linear).

Super-linear speedup?

- Hint: Re-run the parallel version sequentially!
- You will get another sequential version, which runs in $k^2/4 + k^2/4 = k^2/2$ steps, i.e. twice faster than the original sequential version!
- Conclusion: Your parallel version uses a different algorithm than the original sequential version.
- In fact, it is the first step towards a better (optimal) algorithm: merge sort, which runs in $k \log k$ steps!
- A parallel version of merge sort will show only linear speedup (not super-linear).

Outline

Guenther – Universal Scalability Law (USL)

- A general formula, also applicable to assess the parallel speedup, under the same conditions as postulated by Amdahl

$$\text{Speedup}_U = N \;/\; (1 + \alpha(N-1) + \beta N(N-1))$$

- More realistic: peak, then slow down when $N \nearrow \infty$
- Amdahl's law is a corollary for $\beta = 0$, isn't it?

Guenther – Universal Scalability Law (USL)

- A general formula, also applicable to assess the parallel speedup, under the same conditions as postulated by Amdahl

$$\text{Speedup}_U = N \;/\; (1 + \alpha(N-1) + \beta N(N-1))$$

- More realistic: peak, then slow down when $N \nearrow \infty$
- Amdahl's law is a corollary for $\beta = 0$, isn't it?

Guenther – Universal Scalability Law (USL)

- A general formula, also applicable to assess the parallel speedup, under the same conditions as postulated by Amdahl

$$\text{Speedup}_U = N \;/\; (1 + \alpha(N-1) + \beta N(N-1))$$

- More realistic: peak, then slow down when $N \nearrow \infty$
- Amdahl's law is a corollary for $\beta = 0$, isn't it?

Guenther – Universal Scalability Law (USL)

https://blogs.msdn.microsoft.com/ddperf/2009/04/29/parallel-scalability-isnt-childs-play-part-2-amdahls-law-v#9576239