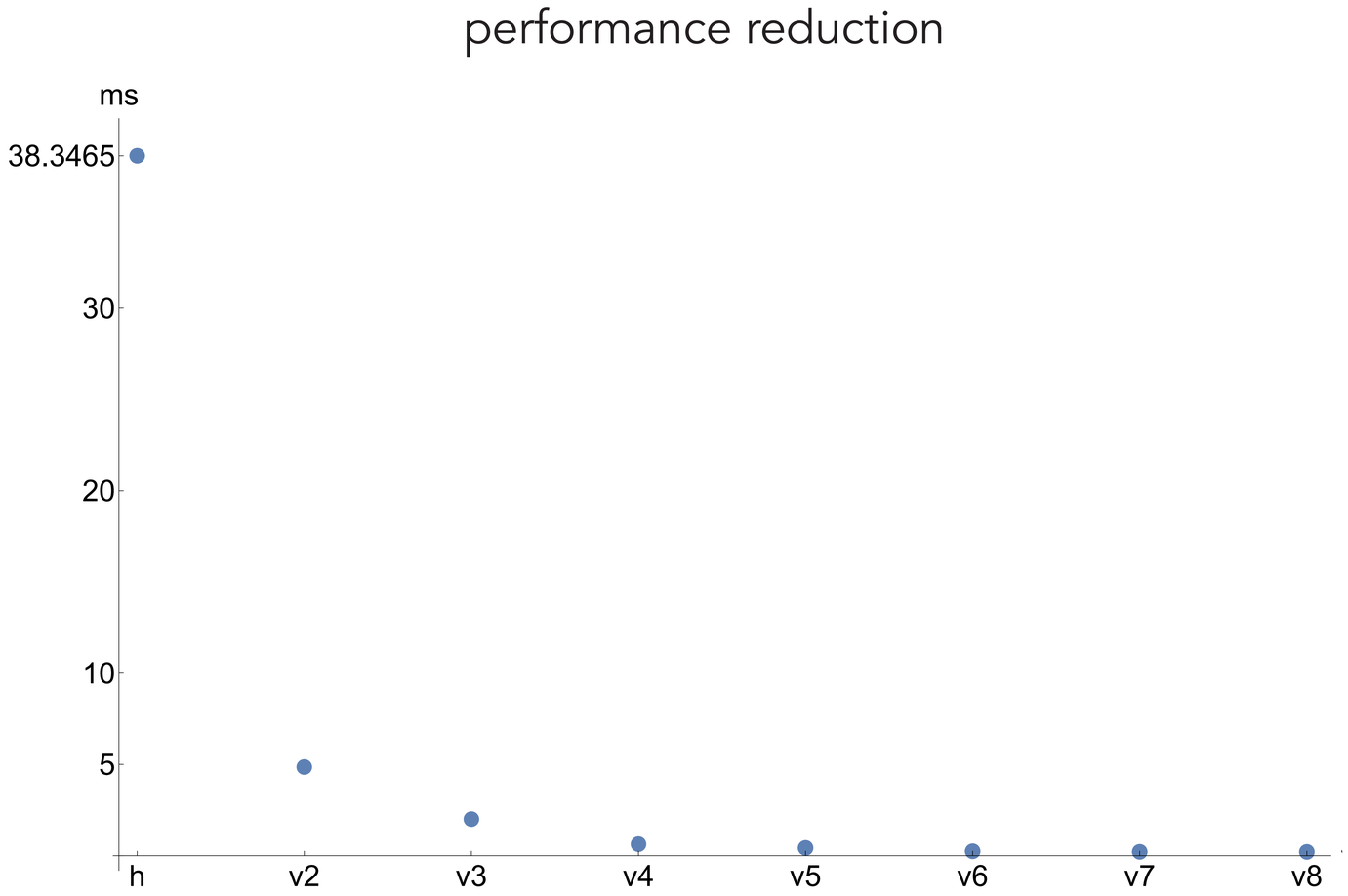
GPU Programming

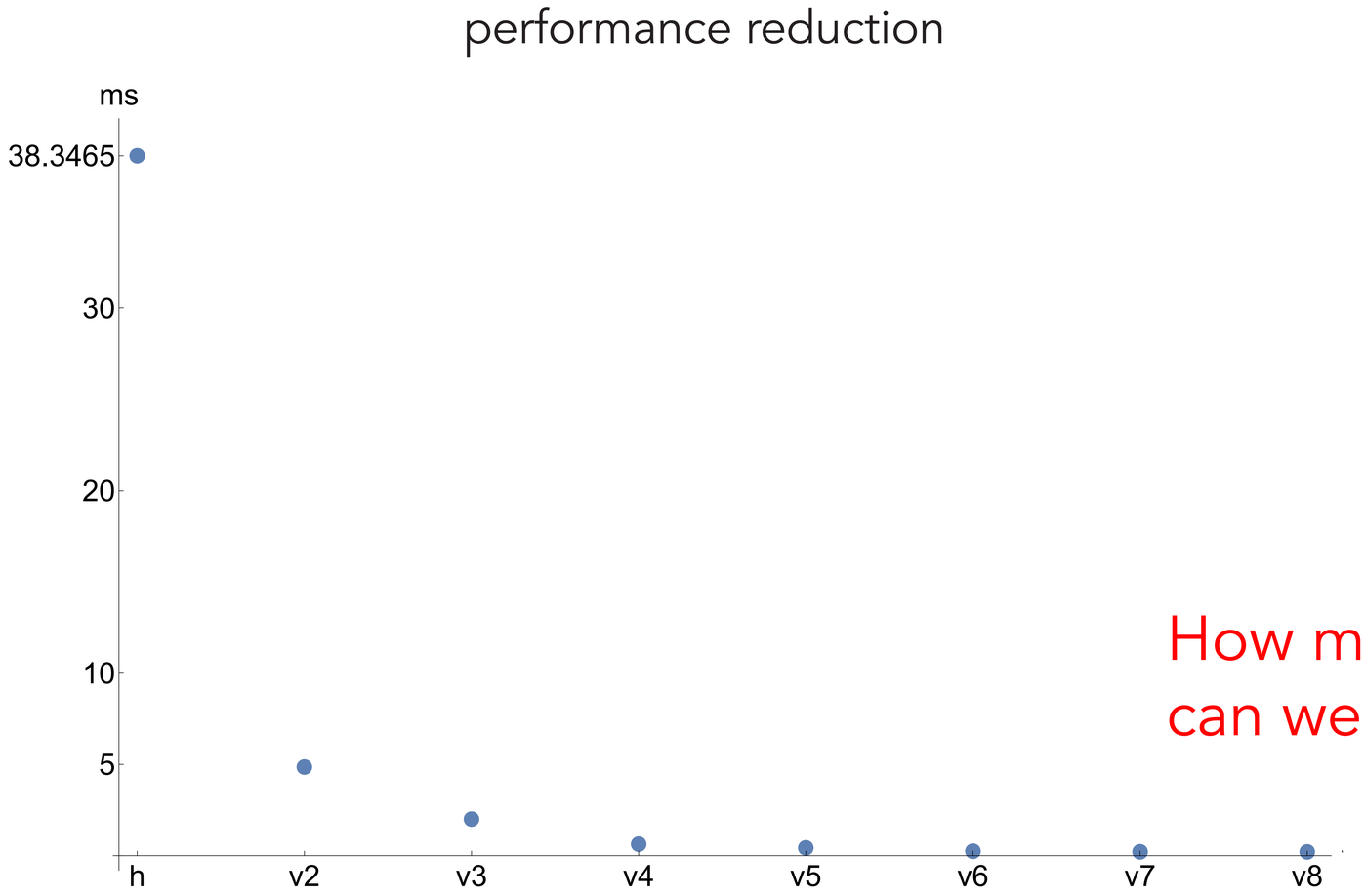
How fast can we get (revisited)?

Christian Lessig

Performance



Performance

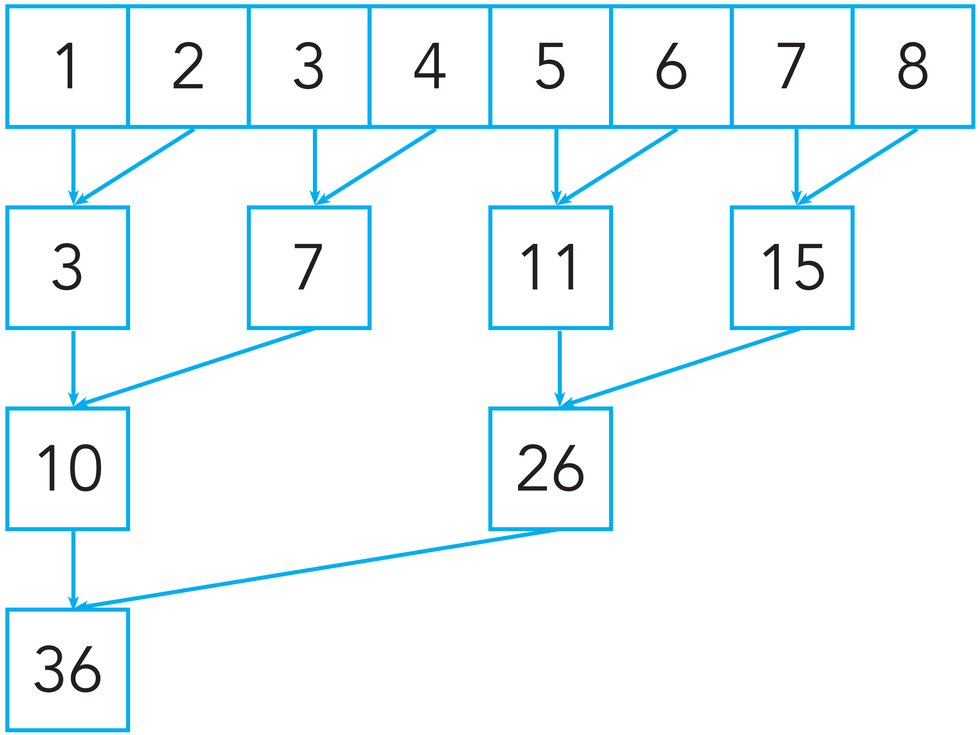


How much faster can we get?

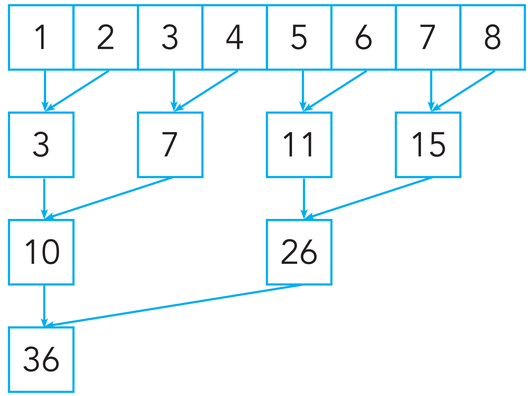
How to bound attainable performance?

Running example

Reduction:



Running example



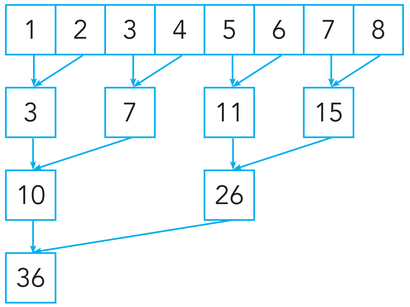
Reduction:

- Parallel summation (multiplication, ...)
- We assume 2^k elements
  - k = 23 in our numerical experiments
- Work (serial): $W_s = 2^k - 1$

Performance

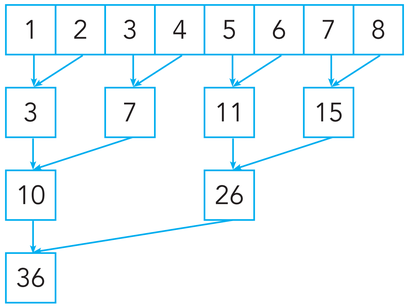
$$\text{speedup} = \frac{\text{execution time for serial execution}}{\text{execution time for parallel execution}}$$

Performance



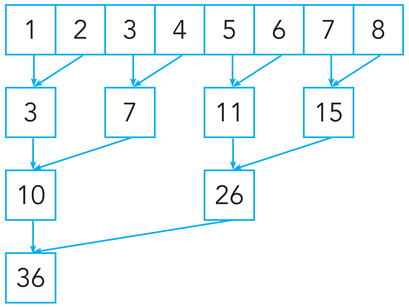
$$\text{speedup}_M = \frac{\text{measured time for serial program}}{\text{measured time for optimized parallel program}}$$

Performance



$$\text{speedup}_M = \frac{195.76\ \text{ms}}{0.197\ \text{ms}}$$

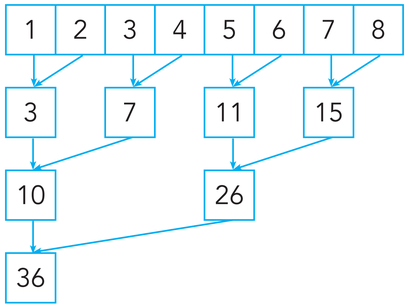
Performance



single-threaded Cuda...

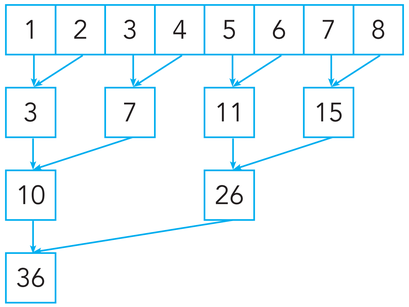
$$\text{speedup}_M = \frac{195.76 \text{ ms}}{0.197 \text{ ms}}$$

Performance



$$\text{speedup}_M = \frac{195.76\ \text{ms}}{0.197\ \text{ms}} = 993.71$$

Performance



$$\text{speedup}_M = \frac{195.76\ \text{ms}}{0.197\ \text{ms}} = 993.71$$

can we predict this number?

Amdahl's law

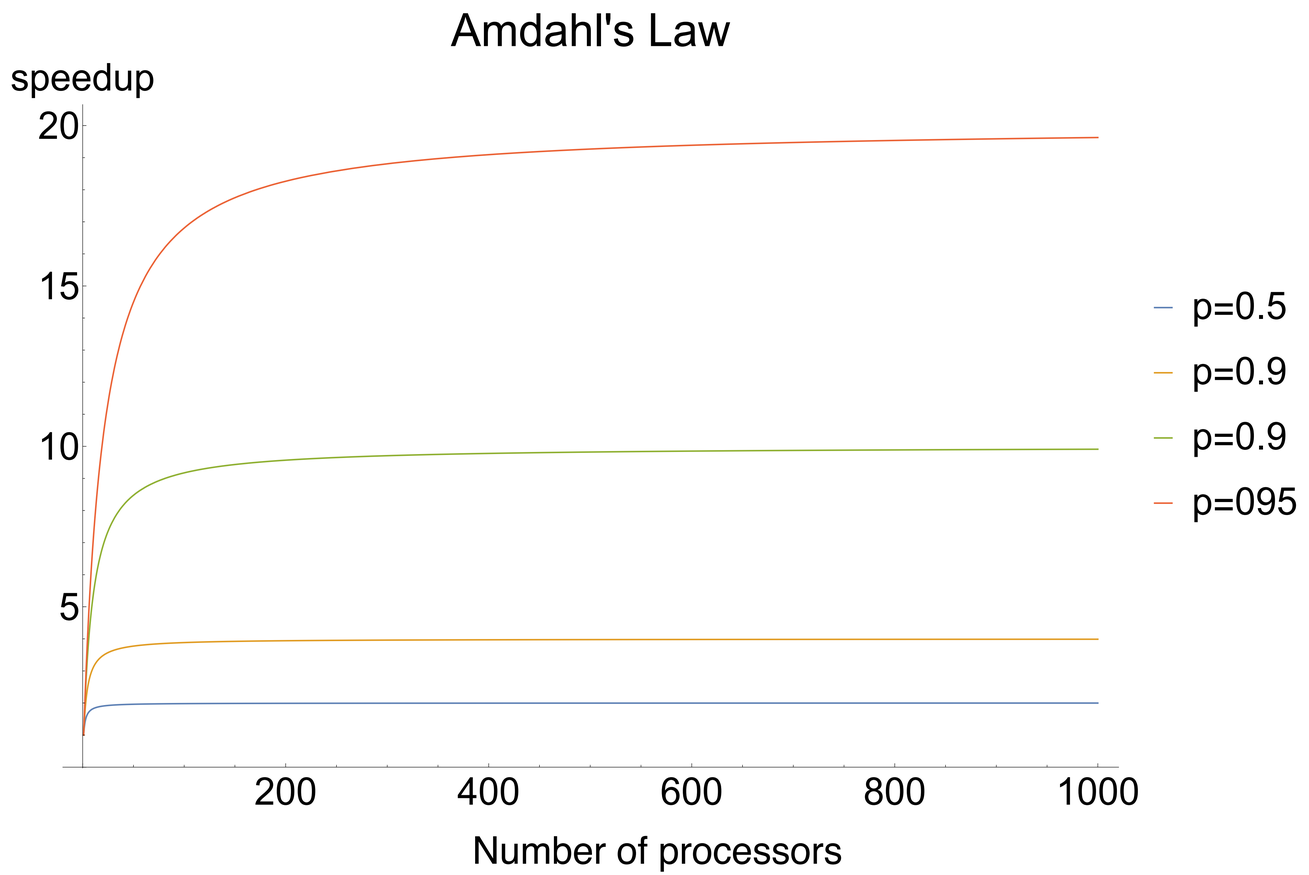
$$\text{speedup} = \frac{\text{execution time for serial execution}}{\text{execution time for parallel execution } \textbf{\textit{when}} \text{ possible}}$$

J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach, fourth edition, Morgan Kaufmann, 2007; p. 40.

Amdahl's law

$$S = \frac{1}{(1-p) - p/N}$$

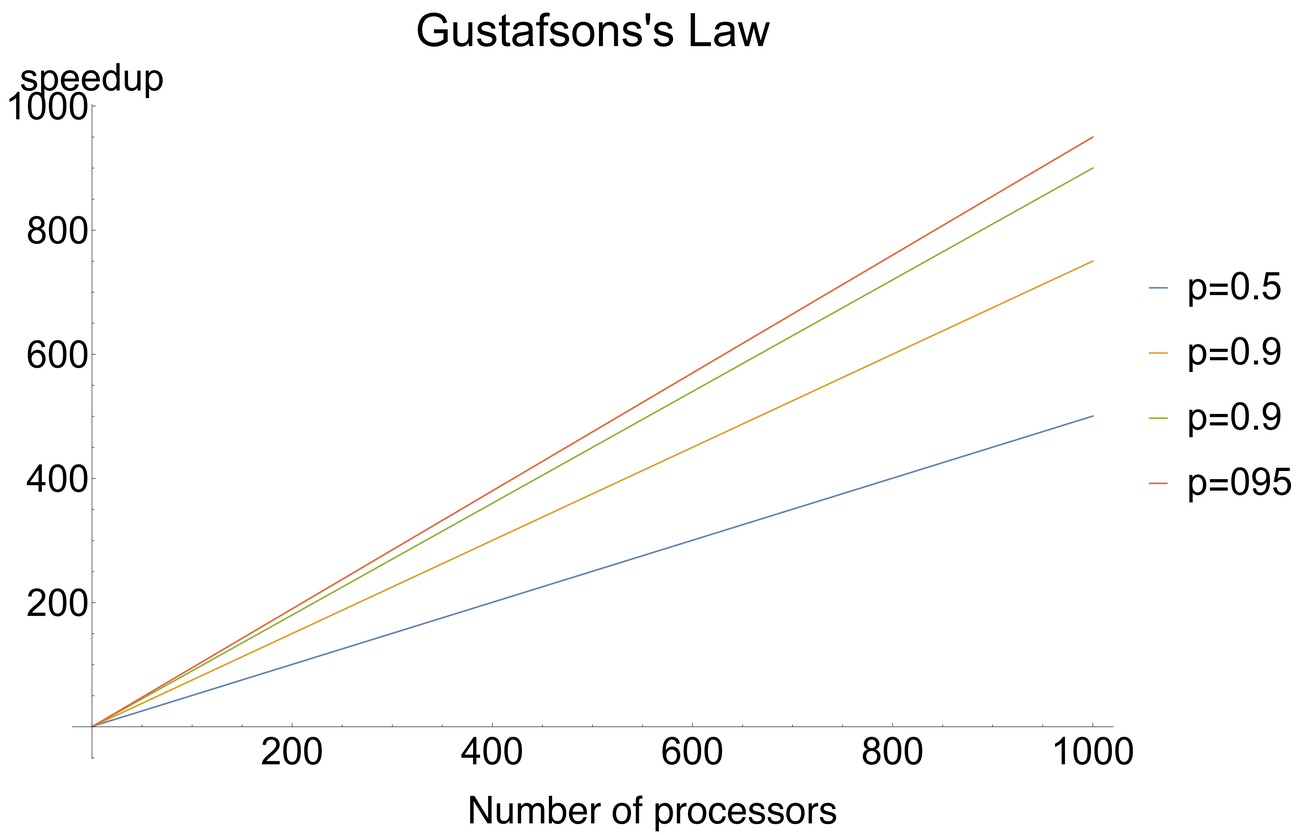
Amdahl’s law



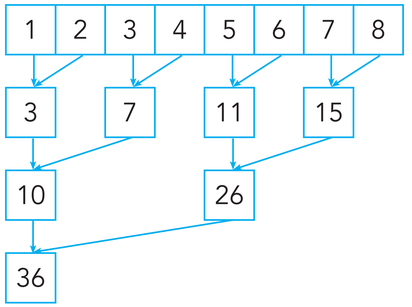
Gustafson's law

$$S_W = (1 - p) + p\,N$$

Gustafson’s law



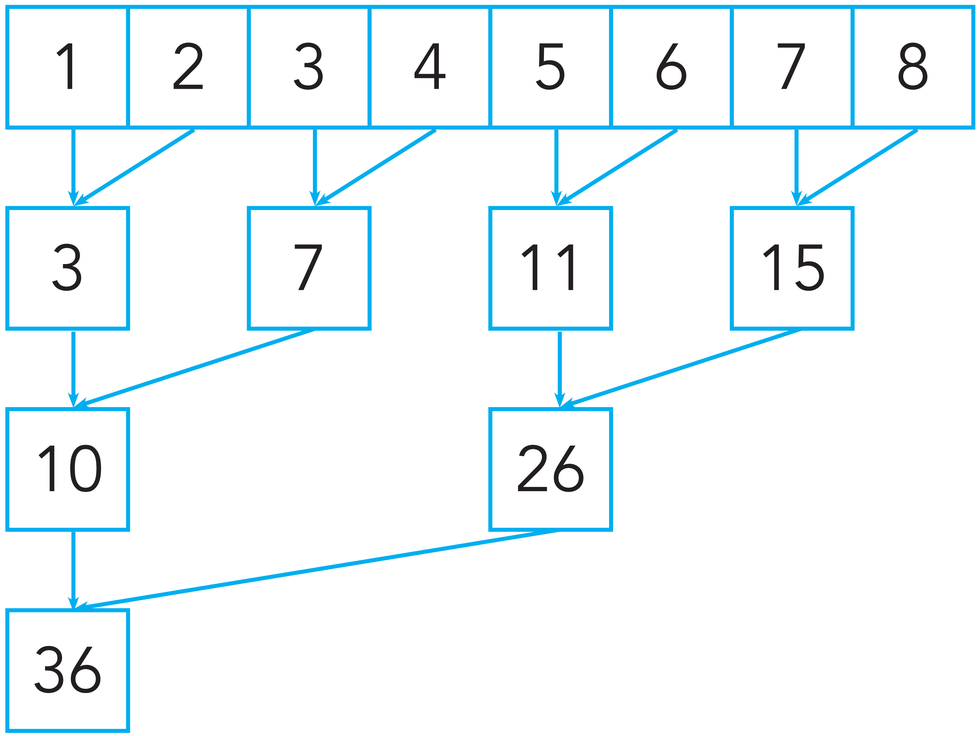
Amdahl's law



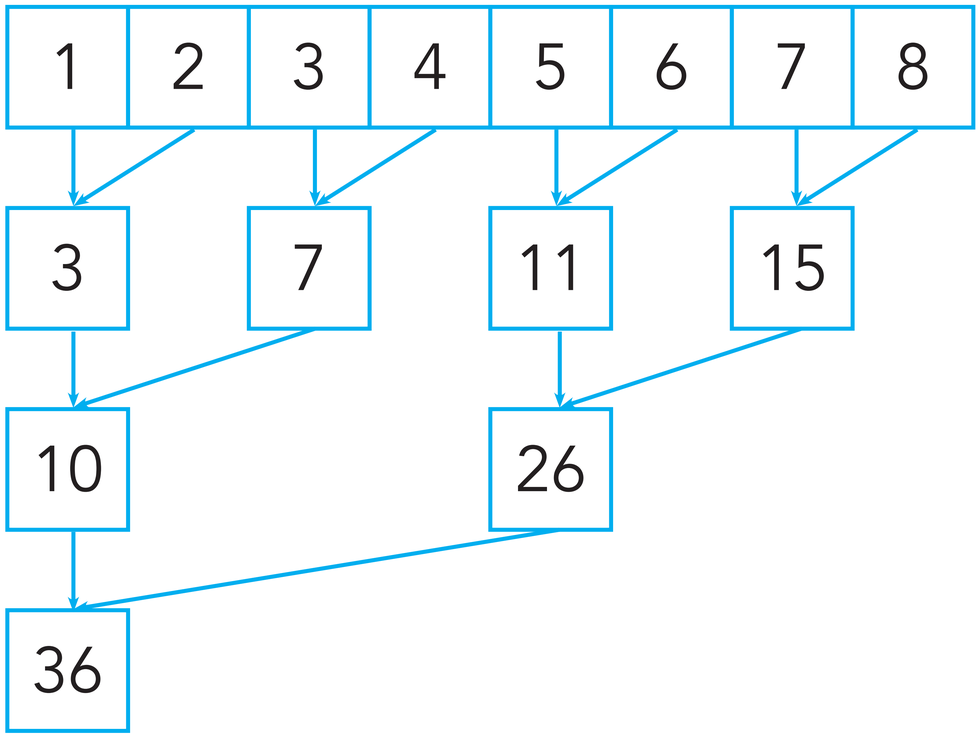
$$S = \frac{1}{(1-p) - p/N}$$

fraction of
parallelizable work

Amdahl's law

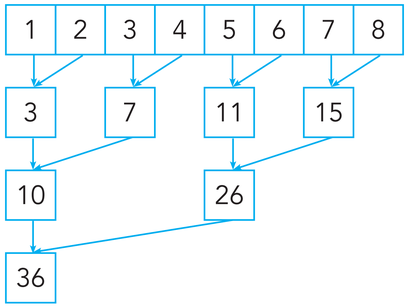


Amdahl's law



not parallel-izable

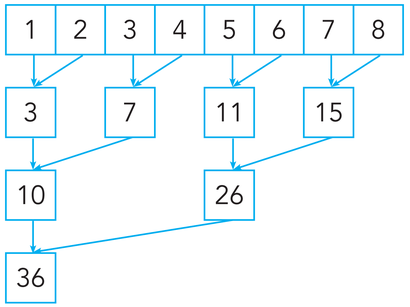
Amdahl's law



$$S = \frac{1}{(1-p) - p/N}$$

fraction of
parallelizable work

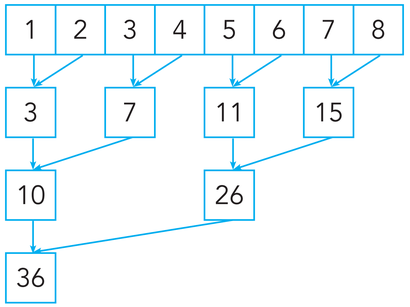
Amdahl's law



$$S = \frac{1}{(1-p) - p/N}$$

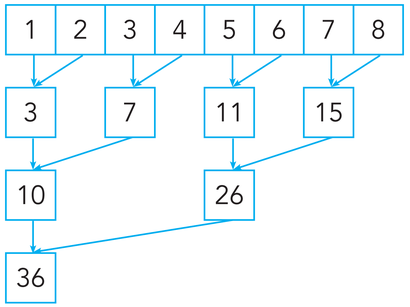
$$\frac{2^k - 2}{2^k - 1}$$

Amdahl's law



$$S = \frac{(2^k - 1)N}{(2^k - 2) + N}$$

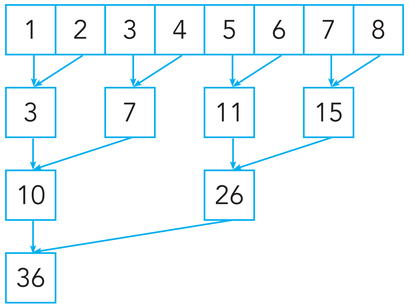
Amdahl's law



$$S = \frac{(2^{k} - 1)N}{(2^{k} - 2) + N}$$

k = 23 in experiments

Amdahl's law



$$S = \frac{(2^k - 1)N}{(2^k - 2) + N}$$

k = 23 in experiments

How big is N?

How big should we chose N?

- Number N_C of threads specified in Cuda

How big should we chose N?

- Number N_C of threads specified in Cuda
  - Conceptual parallelism (in software)

How big should we chose N?

- Number N_C of threads specified in Cuda
  - Conceptual parallelism (in software)
- Number N_P of threads running in parallel

How big should we chose N?

- Number N_C of threads specified in Cuda
  - Conceptual parallelism (in software)
- Number N_P of threads running in parallel
  - This is the real concurrency on the device

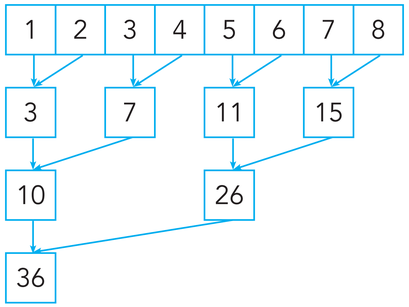
How big should we chose N?

- Number N_C of threads specified in Cuda
  - Conceptual parallelism (in software)
- Number N_P of threads running in parallel
  - This is the real concurrency on the device
- Number N_F of threads in flight on the device

How big should we chose N?

- Number N_C of threads specified in Cuda
  - Conceptual parallelism (in software)
- Number N_P of threads running in parallel
  - This is the real concurrency on the device
- Number N_F of threads in flight on the device
  - We need more than N_P threads to hide latency

Amdahl's law

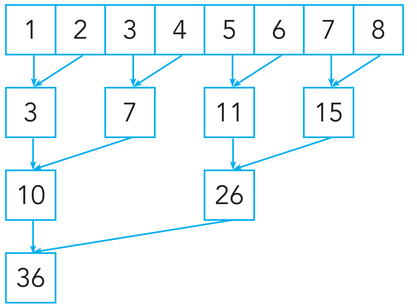


$$S(k, N) = \frac{(2^k - 1)N}{(2^k - 2) + N}$$

k = 23 in experiments

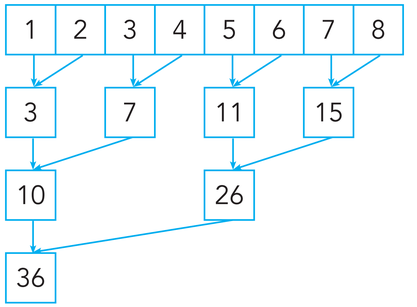
How big is N?

Amdahl's law



$$S\left(k = 23, N_C = 2^{23}/2\right) = 2796203$$

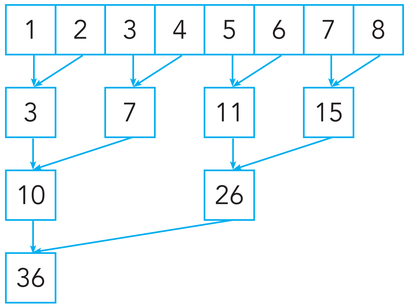
Amdahl's law



$$S\left(k = 23, N_C = 2^{23}/2\right) = 2796203$$

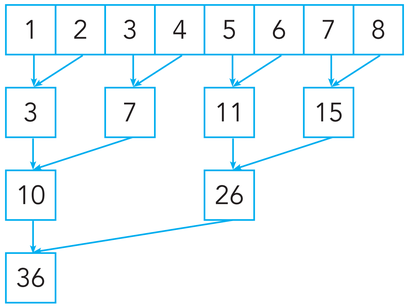
far from
empirical speedup
0.993×10^3

Amdahl's law



$$S(k = 23, N_P = 2048) = 2048$$

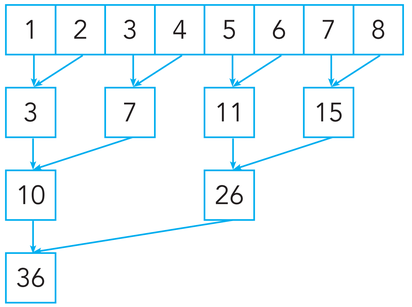
Amdahl's law



$$S(k = 23, N_P = 2048) = 2048$$

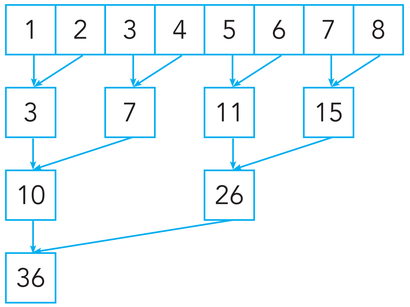
quite close to
empirical speedup

Amdahl's law



$$S(k = 23, N_F = 2^{15}) = 32641$$

Amdahl's law

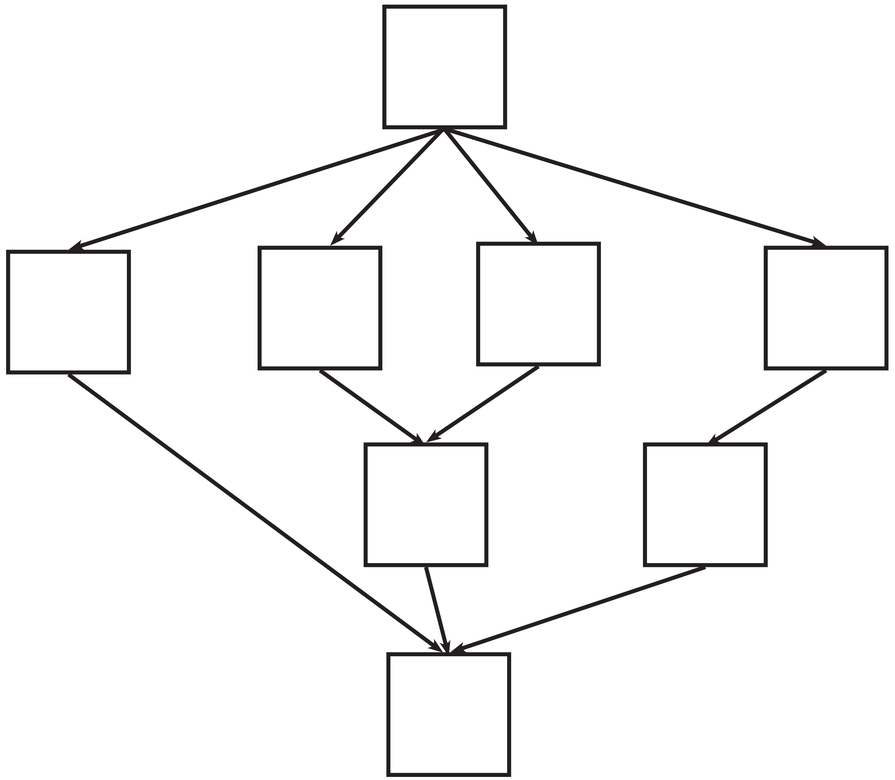


$$S(k = 23, N_F = 2^{15}) = 32641$$

quite far from
empirical speedup

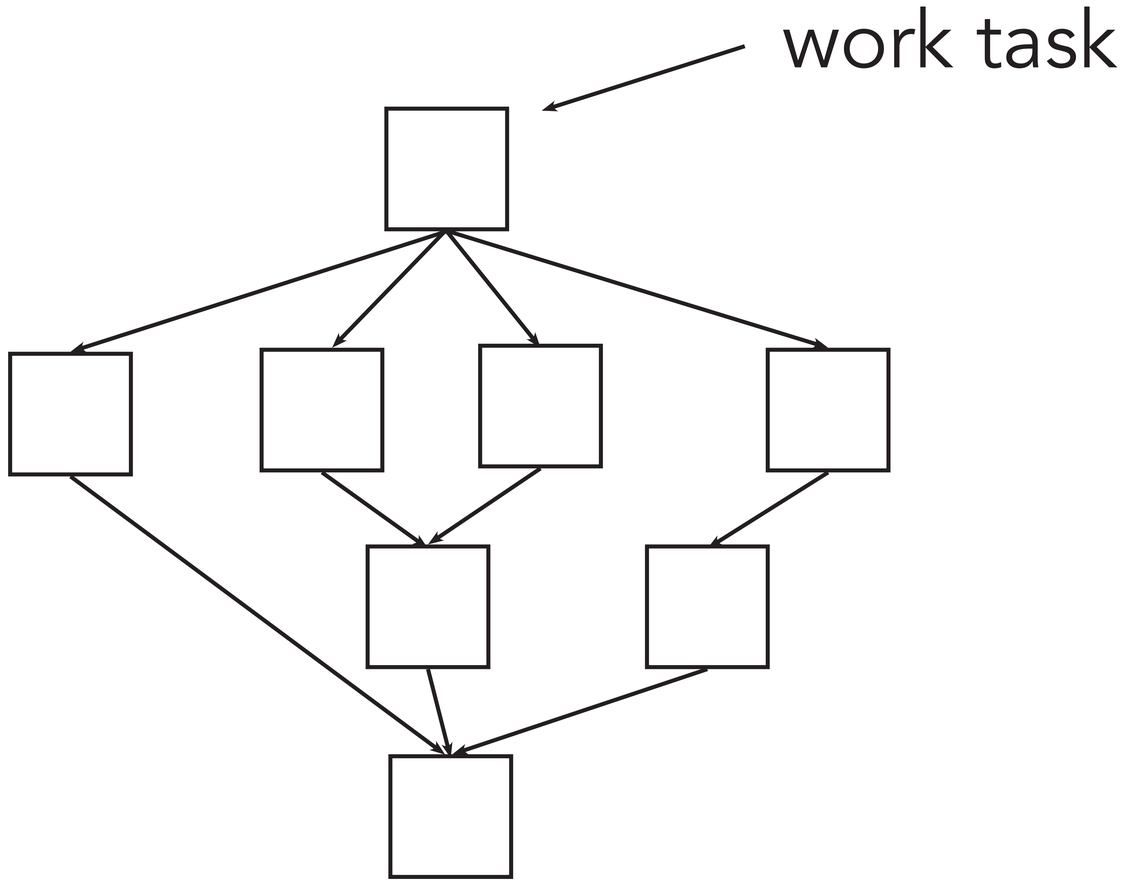
Can we do better?

Performance Analysis

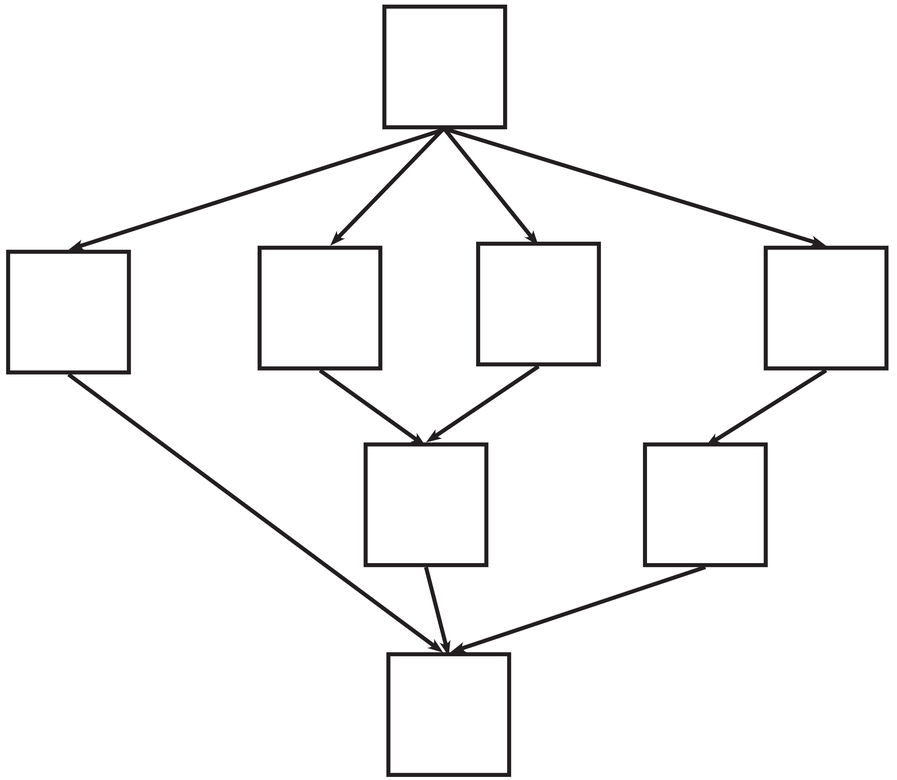


Performance Analysis

DAG models program



Performance Analysis



serial

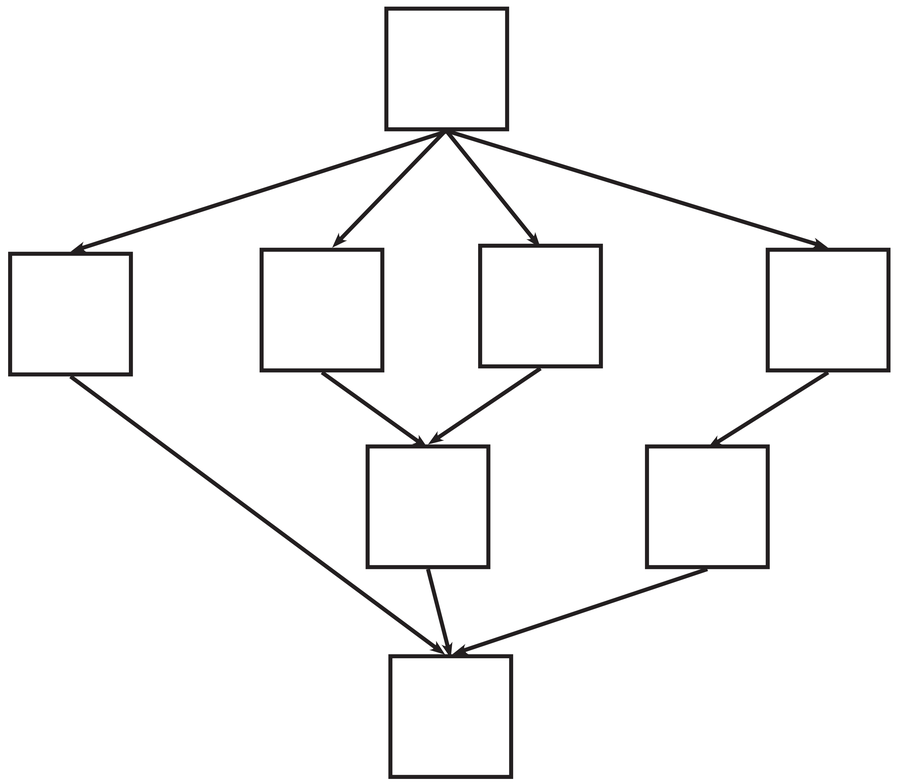
parallel

serial

Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$



serial

parallel

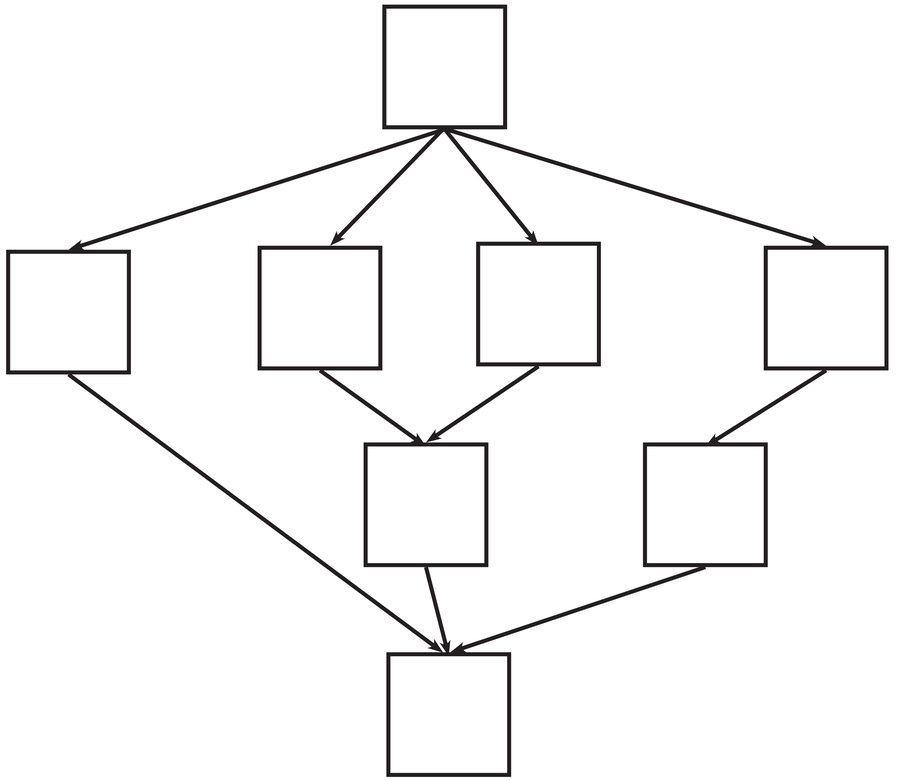
serial

Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$

But there are dependencies!



serial

parallel

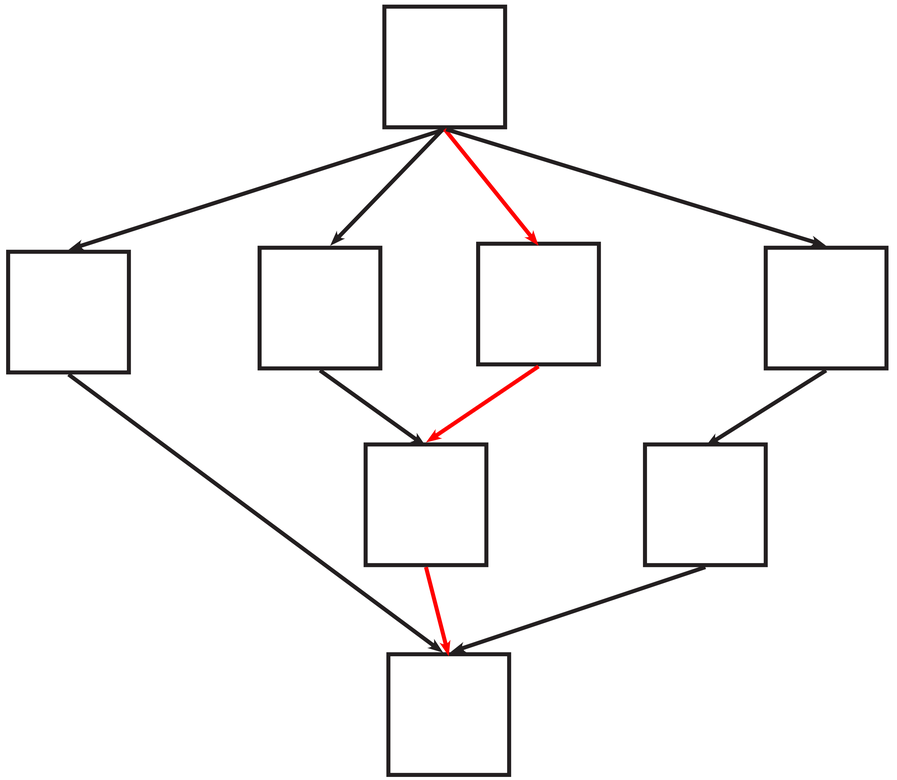
serial

Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$

But there are dependencies!



serial

parallel

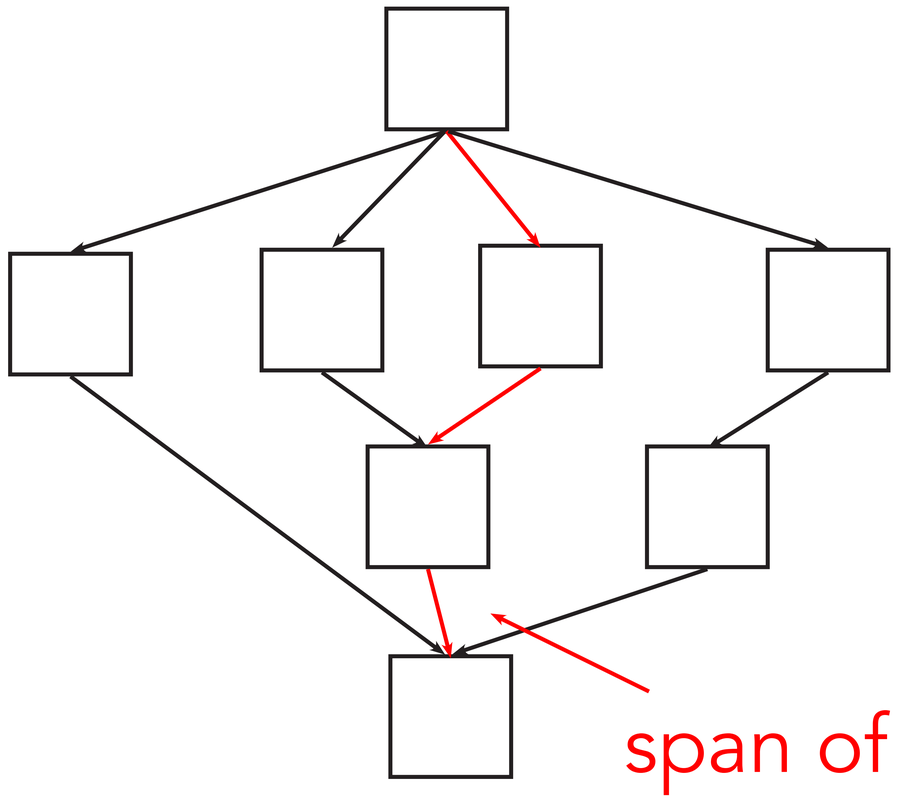
serial

Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$

But there are dependencies!



span of DAG

serial

parallel

serial

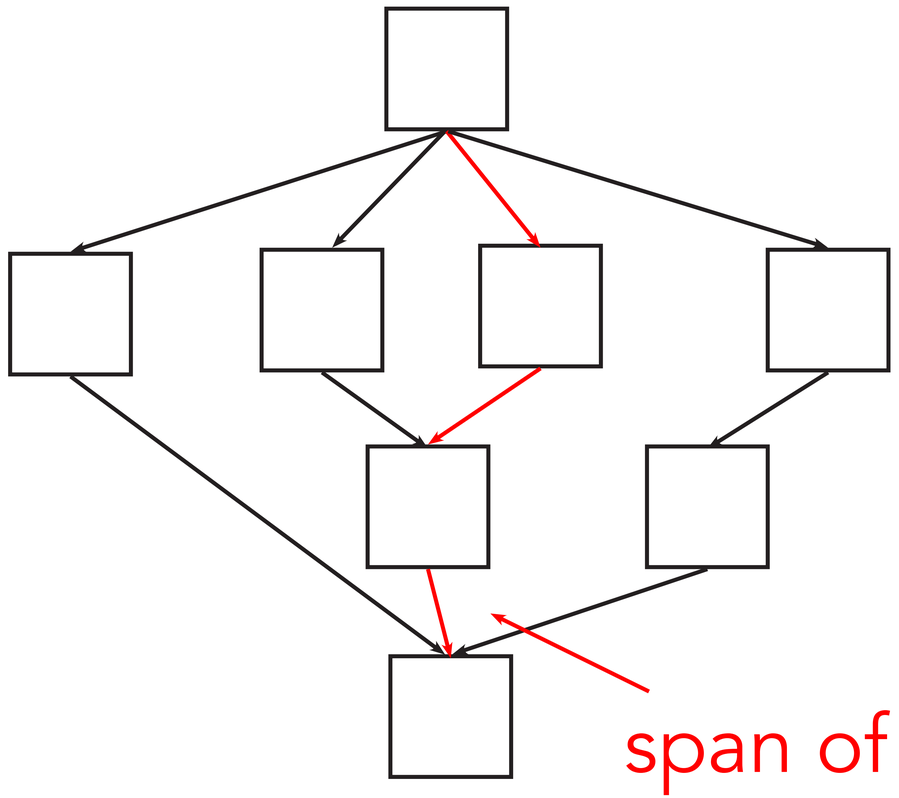
Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$

But there are dependencies!

$$T_{\min} = 4$$



serial

parallel

serial

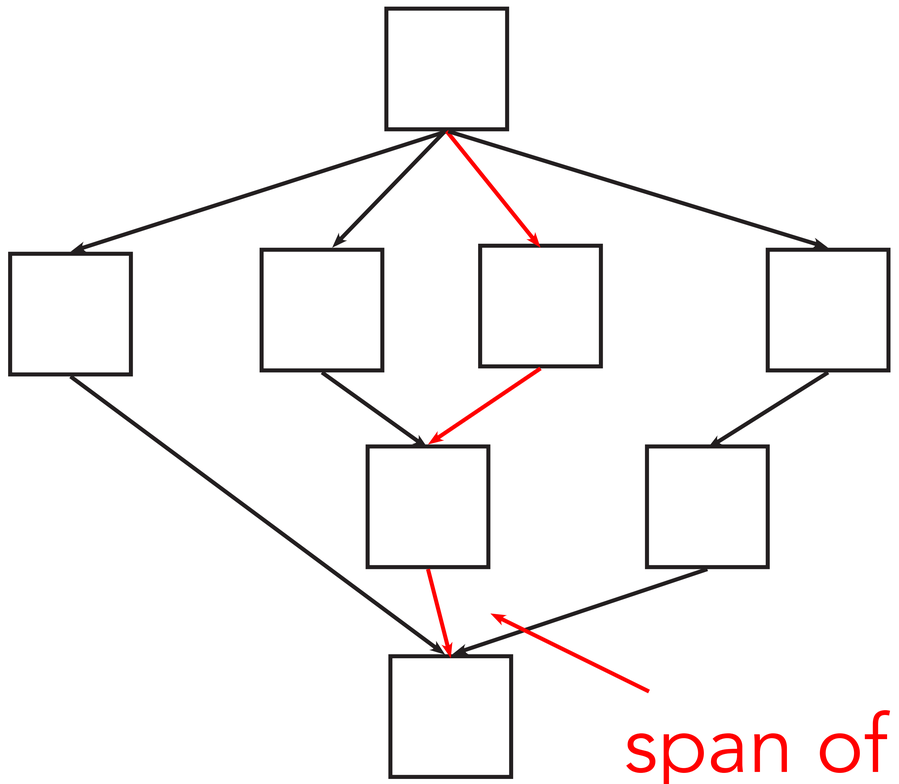
Work-Span Analysis

Amdahl's law:

$$S_\infty = 4$$

But there are dependencies!

$$S_{\max} = 8/4 = 2$$



serial

parallel

serial

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

Minimal execution time with P threads:

$$T_P = \#\text{work}/P$$

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

Minimal execution time with P threads:

$$T_P = \#\text{work}/P$$

But we cannot be faster than T_∞ !

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

Minimal execution time with P threads:

$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

Optimal speedup

$$S_\infty = \frac{T_1}{T_\infty}$$

Minimal execution time with P threads:

$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$

Work-Span Analysis

Serial execution time:

$$T_1 = \#\text{work}$$

Minimal parallel execution time:

$$T_\infty = \text{span}$$

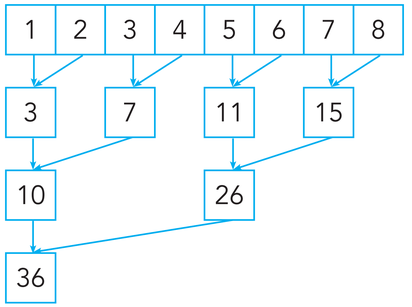
Minimal execution time with P threads:

$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$

speedup with P threads

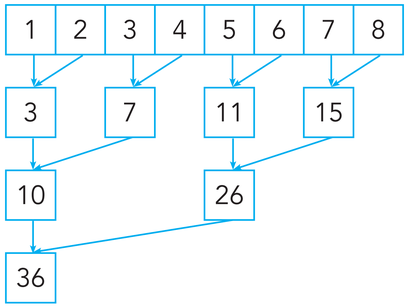
$$S_P = \min(P, S_\infty)$$

Work-Span Analysis



$$S_P = \min(P, S_\infty)$$

Work-Span Analysis



$$S_P = \min\left(2^{11}, 2^{23}/23\right) = 2^{11}$$

quite close to
empirical speedup

Work-Span Analysis

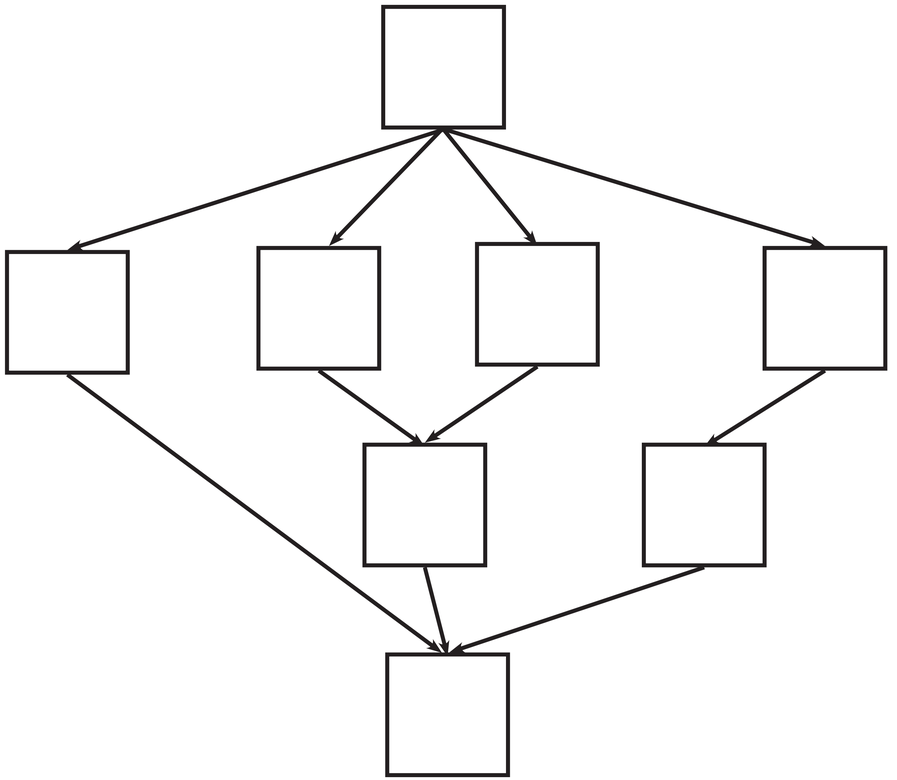
Optimal execution time
with P threads:

$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$

Work-Span Analysis

Optimal execution time
with P threads:

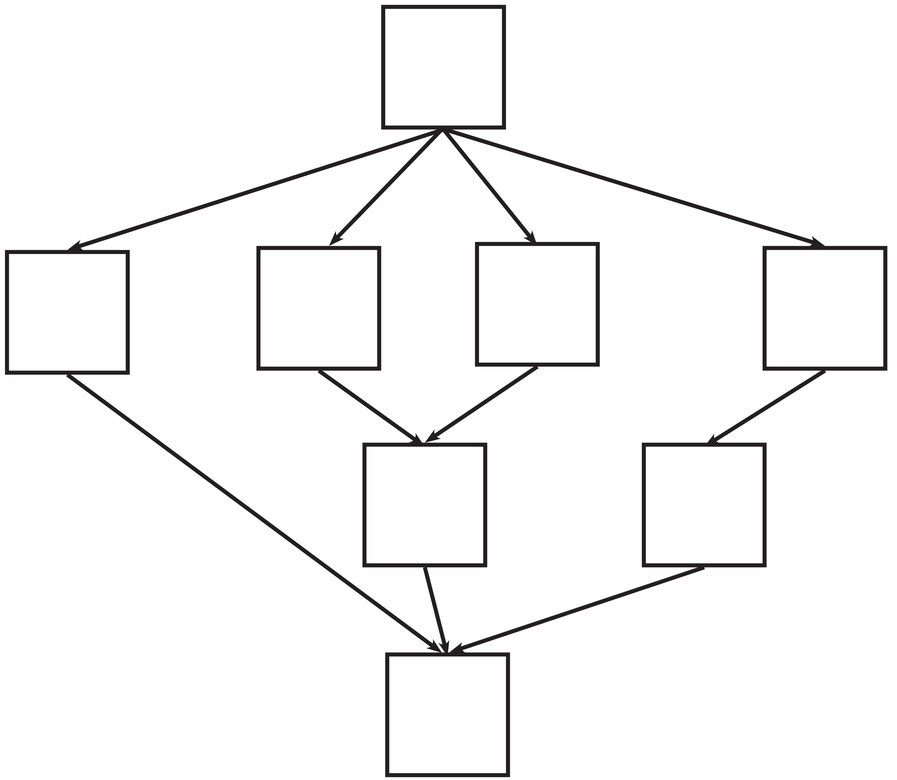
$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$



Work-Span Analysis

Optimal execution time
with P threads:

$$T_P = \max\left(\#\text{work}/P\,,\, T_\infty\right)$$



At each of the T_∞
steps we need time

$$T_i = W_i/P$$

Work-Span Analysis

Brent's theorem:

$$T_P \leq T_\infty + \frac{T_1 - T_\infty}{P}$$

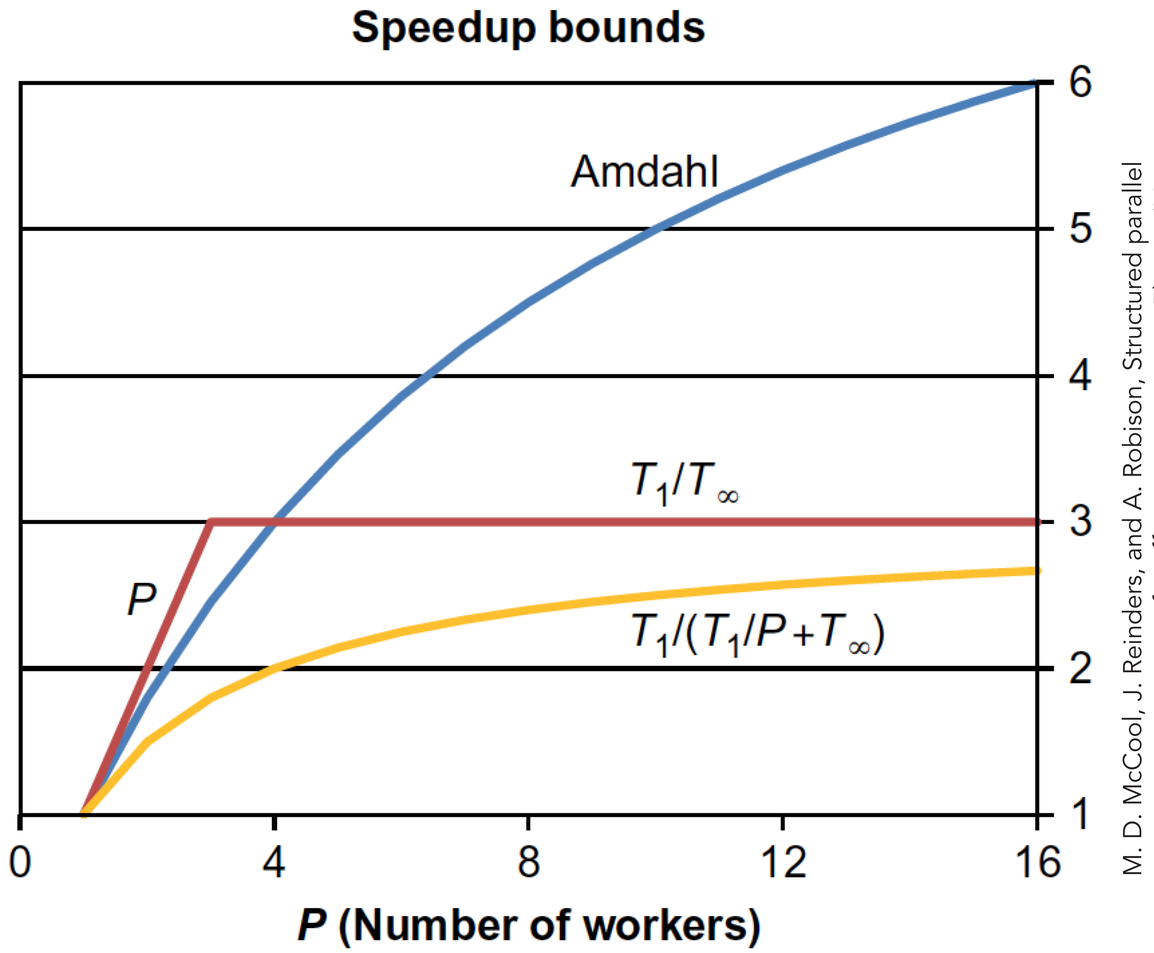
Work-Span Analysis

Brent's theorem:

$$T_P \leq T_\infty + \frac{T_1 - T_\infty}{P}$$

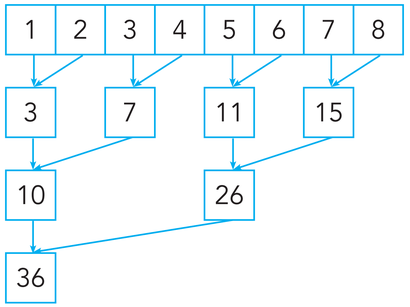
Upper bound on execution time

Work-Span Analysis



M. D. McCool, J. Reinders, and A. Robison, Structured parallel programming: patterns for efficient computation. Elsevier/Morgan Kaufmann, 2012.

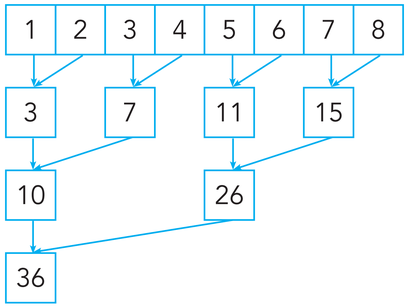
Work-Span Analysis



Brent's theorem:

$$T_P \leq T_\infty + \frac{T_1 - T_\infty}{P}$$

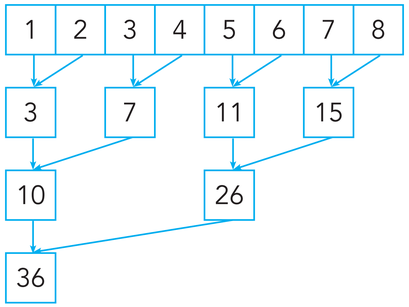
Work-Span Analysis



Brent's theorem:

$$T_P \leq 23 + \frac{(2^{23} - 1) - 23}{P}$$

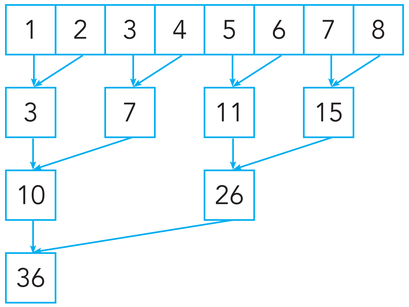
Work-Span Analysis



Brent's theorem:

$$T_P \leq 23 + \frac{8\,388\,584}{2^{15}} = 279$$

Work-Span Analysis

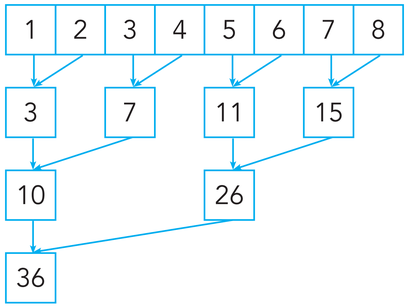


Brent's theorem:

$$T_P \leq 23 + \frac{8\,388\,584}{2^{15}} = 279$$

$$\Rightarrow S_P = 30\,067$$

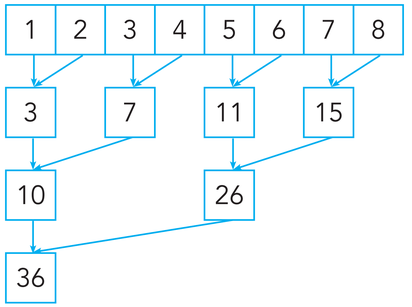
Performance analysis



Cost:

$$C = T \cdot P$$

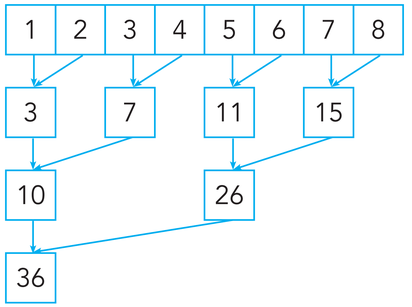
Performance analysis



$$T \approx \mathrm{ld}(2^k) = k$$

$$P \approx 2^k$$

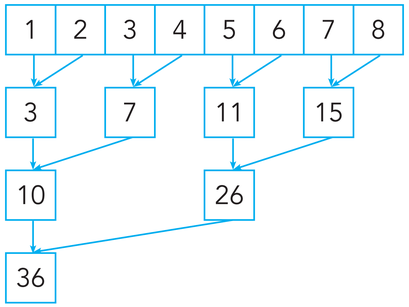
Performance analysis



Cost:

$$C \approx \mathrm{ld}(2^k) \cdot 2^k = k \cdot 2^k$$

Performance analysis

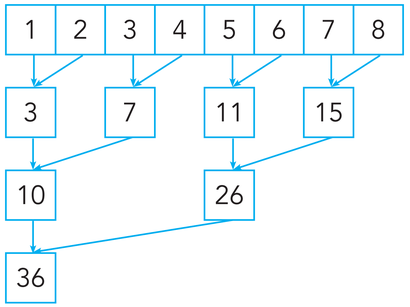


Cost:

$$C \approx \mathrm{ld}(2^k) \cdot 2^k = k \cdot 2^k$$

Cost of serial algorithm is 2^k, i.e. of the order of the input size, so parallel reduction is not cost efficient

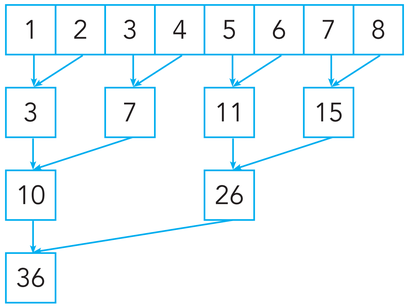
Performance analysis



$$T \approx \mathrm{ld}(2^k) = k$$

$$P \approx 2^k$$

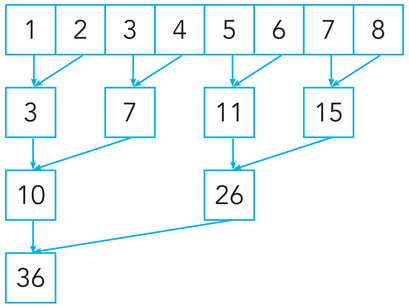
Performance analysis



$$T \approx \mathrm{ld}(2^k) = k$$

$$P \approx \cancel{2^k}\ 2^k/k$$

Performance analysis

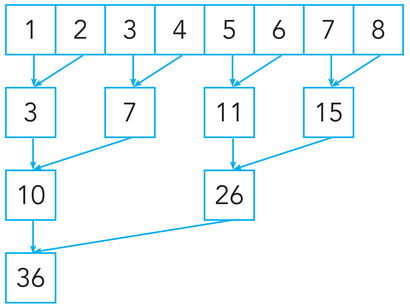


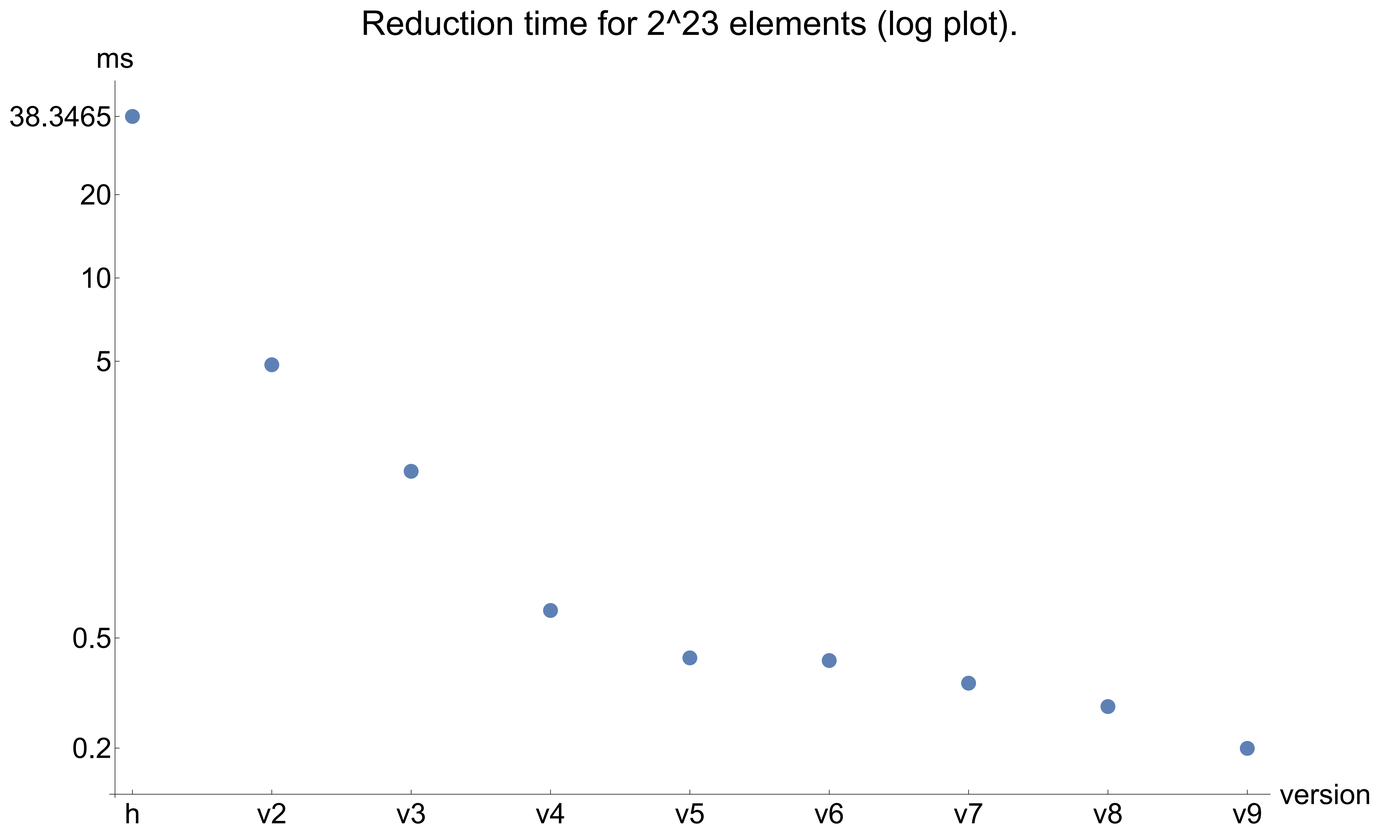
$$T \approx \mathrm{ld}(2^k) = k$$

$$P \approx \cancel{2^k}\ 2^k / k$$

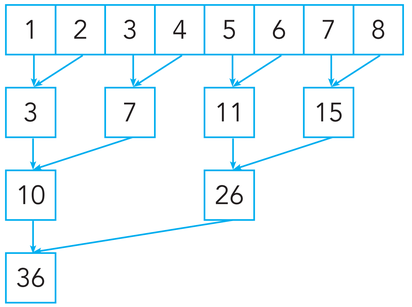
Process O(k) elements in one thread!

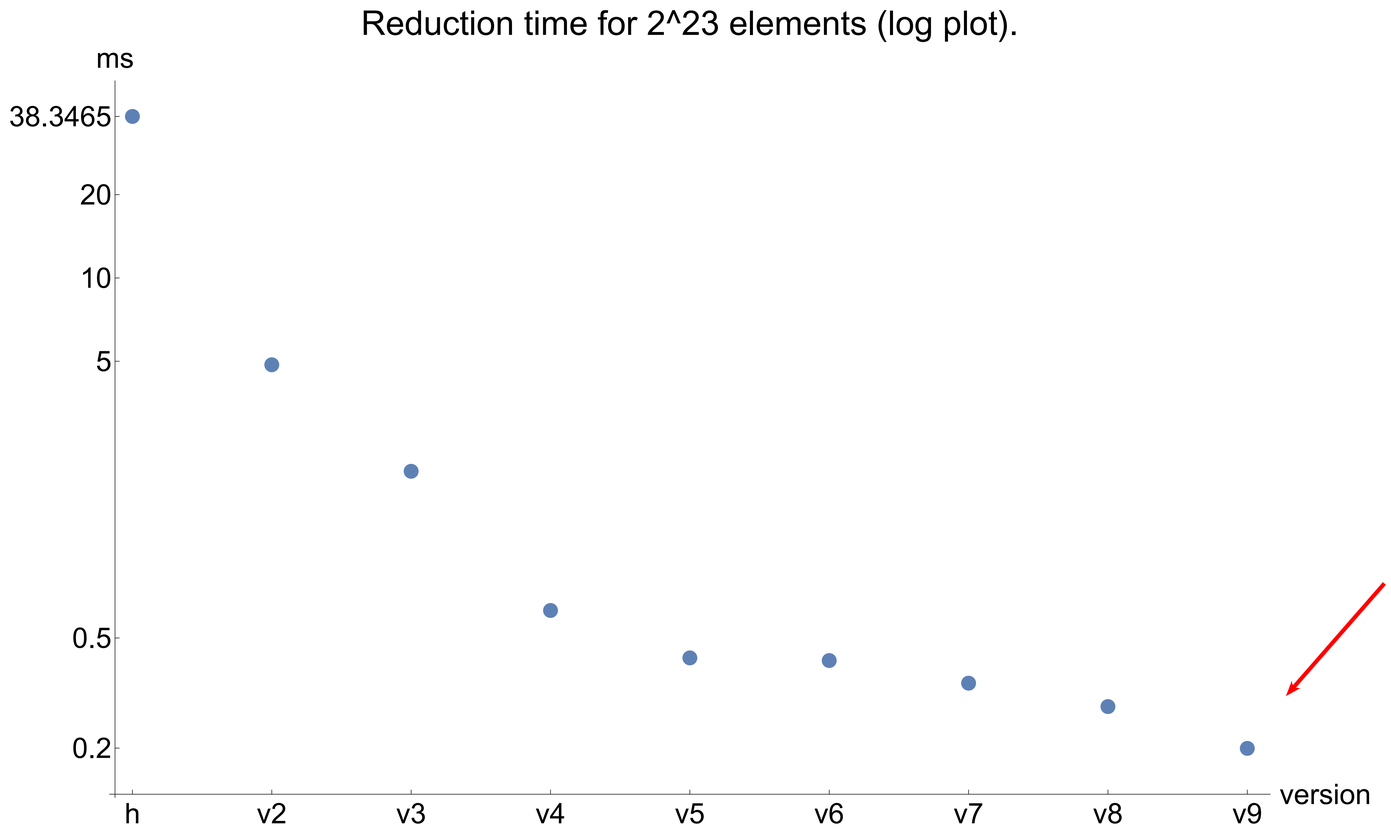
Performance analysis





Performance analysis





Additional 44% improvement with $ld(T_1)$ threads