HPC: Performance

UROŠ LOTRIČ

Key features to performance

Complexities and varieties of architectures

- Data locality and availability of parallel operations
- Per machine tuning can be still necessary

Data locality

- Reuse of nearby locations in time and space
- Important for memory bandwidth and cache usage
  - Data chunks that can fit in the cache
  - Organize data structures and memory access to reuse data locality
    - Access to far memory locations, access to locations power of 2 apart (reduce cache conflicts on caches with low associativity)
  - Avoid accessing too many pages at ones (TLB misses)
  - Align data with cache line boundaries (false sharing)

Key features to performance

Data locality

- Hard to do for unknown architecture
  - parameter to define granularity (manually or by autotuning)
  - Cache oblivious approach (locality at all scales)
- Arithmetic intensity
  - Ratio computation/data transfer, should be high
  - Can be done with fusion and tiling
  - Small granularity can bring more data transfer

Key features to performance

Parallel slack

- Extra parallelism available can be beneficial
- Software and hardware schedulers get more flexibility to exploit machine
  - Number of tasks equal to number of functional units is tempting,
    whole system can wait for a certain task interrupted by OS
  - With more parallelism, when problematic task is waiting, another can jump in
  - Software does not always support such concepts (POSIX)
    - Calling parallel routines from threads can lead to huge numbers of threads
    - OS does not know which should run simultaneously

Performance theory

What is performance?

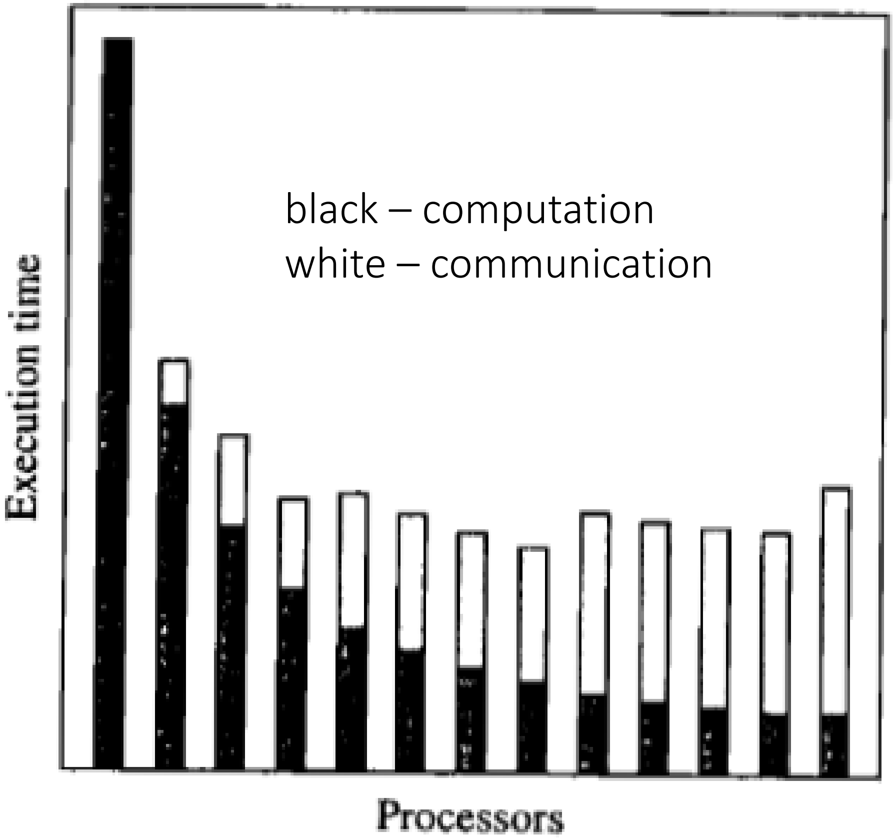
- total time, throughput, power, cost, efficiency, scalabiliity

Total time and throughput

- total time = wall clock time needed to complete the task
- throughput = rate at which tasks are completed
  - Better throughput may increase total time
    (pipeline communication)
- response time (web services)

Performance

Total time



Performance theory

Speedup

- Definition
  - $t_s(n)$ - time for sequential computation
  - $t_p(n,p)$ - time for parallel computation
  - n – problem size
  - p – number of workers

$$S(n,p) = \frac{t_s(n)}{t_p(n,p)}$$

- Parallel program is composed of
  - sequential operations, $\sigma(n)$
  - parallel operations, $\varphi(n)$
  - communication $\kappa(n,p)$
- Ideal work distribution

$$S(n,p) = \frac{t_s(n)}{t_p(n,p)} = \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n,p)}$$

Performance theory

Efficiency

- Measures return of hardware investment
  - tells how well the hardware is used
- Usually $0 \leq E(n,p) \leq 1$
- Relative and absolute speedup
  - Absolute when another (better) algorithm is used
- Super-linear speedup
  - most commonly related to better use of cache
  - cooperation between workers can reduce time (earlier stopping)

$$E(n,p) = \frac{t_s(n)}{p \cdot t_p(n,p)} = \frac{S(n,p)}{p}$$

$$E(n,p) = \frac{\sigma(n) + \varphi(n)}{p\sigma(n) + \varphi(n) + p\kappa(n,p)}$$

Performance theory

Price

- Efficient programs contribute to lower computation price

$$P(n,p) = p t_p(n) = \frac{p t_s(n)}{S(n,p)} = \frac{t_s(n)}{E(n,p)}$$

Performance theory

Amdahl's law

- Portion of sequential operations
- Neglects communication

$$f = \frac{\sigma(n)}{\sigma(n) + \varphi(n)}$$

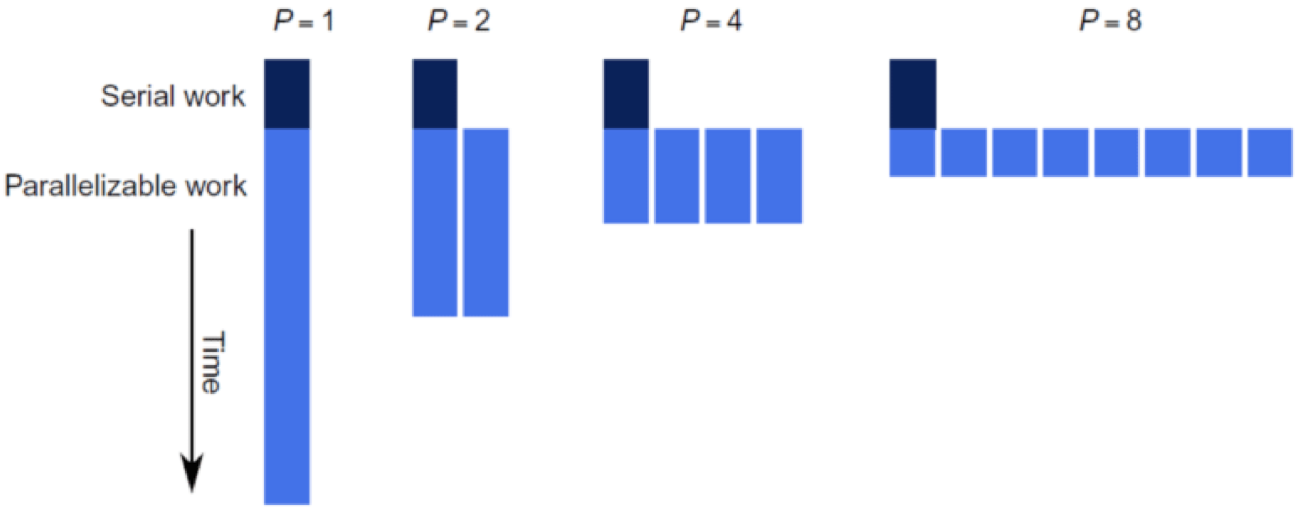
$$S(n,p) = \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n,p)} \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p}$$

$$S(n,p) \leq \frac{1}{f + (1-f)/p}$$

- Maximal speedup depends on code, which cannot be parallelized
- Assumptions
  - Constant problem size
  - Focus of parallelization is to reduce the total time

Performance theory

Amdahl's law



- $n = 10, \sigma(n) = 2, \varphi(n) = 8, f = 0.2$
- $S(n,p) = 1, 1.66, 2.5, 3.33$
- $E(n,p) = 1, 0.83, 0.62, 0.41$

Performance theory

Gustafson-Barsis's Law

- Speedup should be measured by scaling the problem to the number of workers, not by fixing the problem size.
- Applications scale to exploit better and better computers.
- Portion of sequential tasks in parallel computation

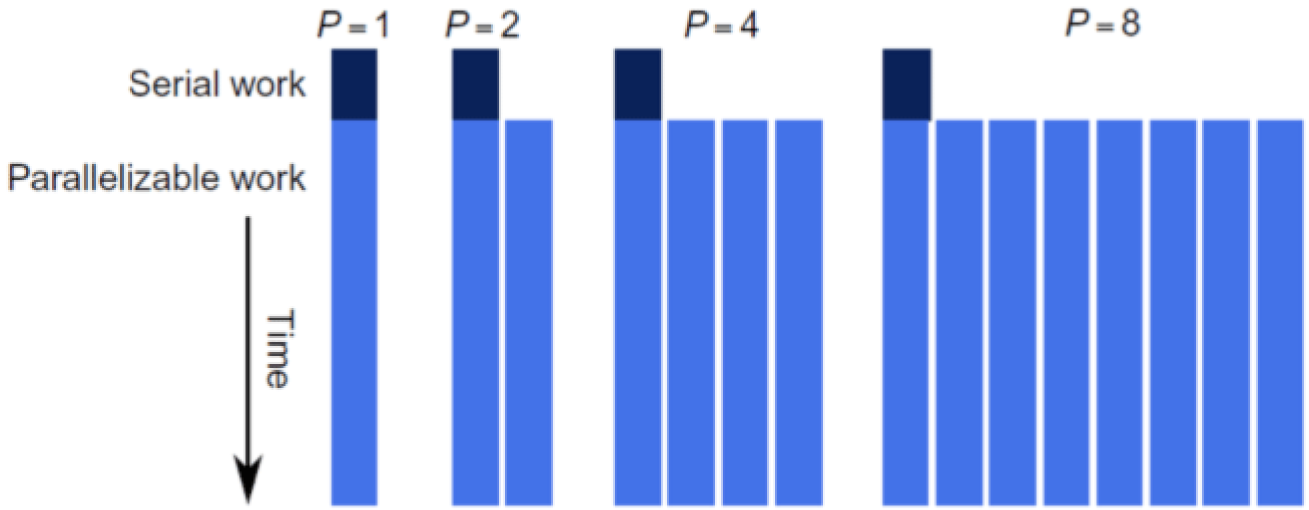
$$s = \frac{\sigma(n)}{\sigma(n) + \varphi(n)/p}$$

- Speedup

$$S(n,p) \leq p - s(p-1)$$

Performance theory

Gustafson-Barsis's Law

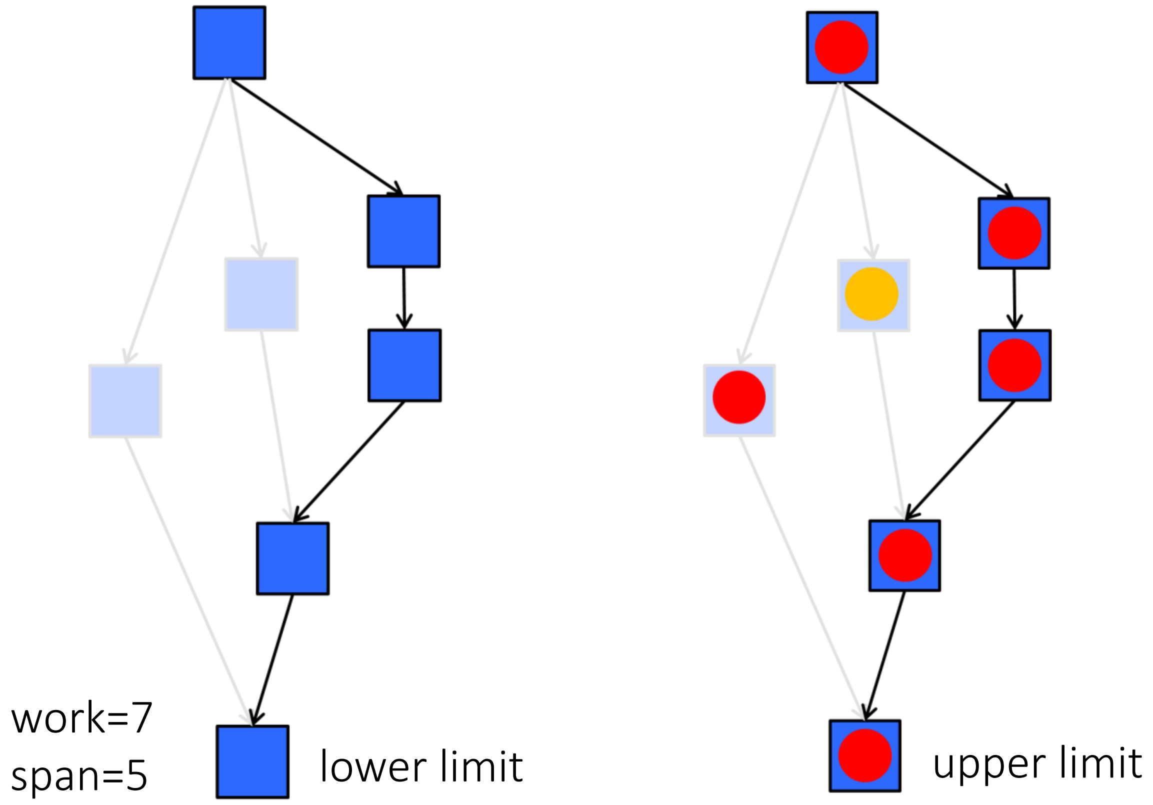


- $n = 10, 18, 34, 66, \sigma(n) = 2, \varphi(n) = 8, 16, 32, 64$
- $s = 0.2$, constant
- $S(n, p) = 1, 1.8, 3.4, 6.6$
- $E(n, p) = 1, 0.9, 0.85, 0.825$

Performance theory

Work-span model

- Optimistic assumptions in previous speedup computation
  - all parallelisable work can be ideally parallelized
- Tasks presented as direct acyclic graph
  - Ignore communication and memory access
  - Assumes greedy scheduling
    - work, $t_s(n)$ computation time on sequential machine
    - span, $t_p(n, \infty)$ computation time on ideal machine
      - also critical path, step complexity, depth



Performance theory

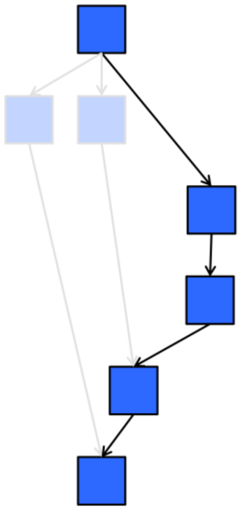
Work-span model

- Upper limit on speedup
  - on ideal machine with greedy scheduling, adding processors never slows down an algorithm
- Lower limit on speedup
  - to achieve good parallelization f << 1 in Amdahl's law
  - Make a replacement $f = span/work$
  - lower bound on speedup
- Example

$$S(n,p) = \frac{t_s(n)}{t_p(n,p)} \le \frac{t_s(n)}{t_p(n,\infty)} = \frac{work}{span}$$

$$S(n,p) = \min\left(p, \frac{work}{span}\right)$$

$$\frac{1}{f + (1-f)/p} > \frac{1}{\frac{span}{work} + \frac{1}{p}} \le S(n,p)$$

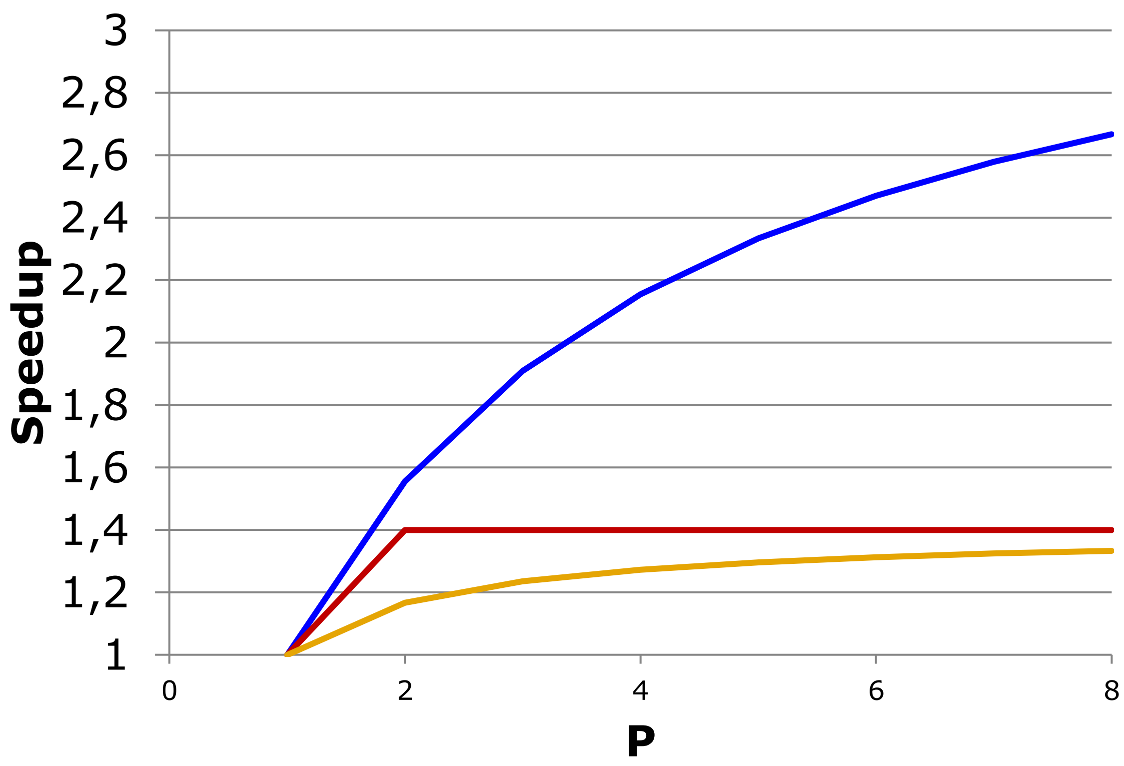


$$\frac{7}{6} = 1.17 \le S(n,2) \le 1.4 = \frac{7}{5}$$

Performance theory

Work-span model

- Red: upper limit
- Yellow: lower limit
- Blue: Amdahl, *f* = 2/7
  - All but first and last task in example can be executed in parallel



Parallel slack

- $PS = \frac{S(n,\infty)}{p} = \frac{t_s\,(n)}{p\; t_p(n,\infty)}$
- In practice *PS* = 8 works well

Performance theory

Scalability (iso-efficiency)

- Speedup grows with problem size
- Efficiency on more workers can be maintained by increasing the problem size

$$S(n,p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n,p)}$$

- Total cost of parallelism

$$S(n,p) \leq \frac{p(\sigma(n) + \varphi(n))}{\sigma(n) + \varphi(n) + (p-1)\sigma(n) + p\kappa(n,p)}$$

$$T_{overhead} = (p-1)\sigma(n) + p\kappa(n,p)$$

Performance theory

Scalability (iso-efficiency)

- Efficiency should be maintained

$$E(n,p) = \frac{S(n,p)}{p} \leq \frac{T_s(n)}{T_s(n) + T_{overhead}(n,p)} = \frac{1}{1 + \frac{T_{overhead}(n,p)}{T_s(n)}}$$

- Sequential time

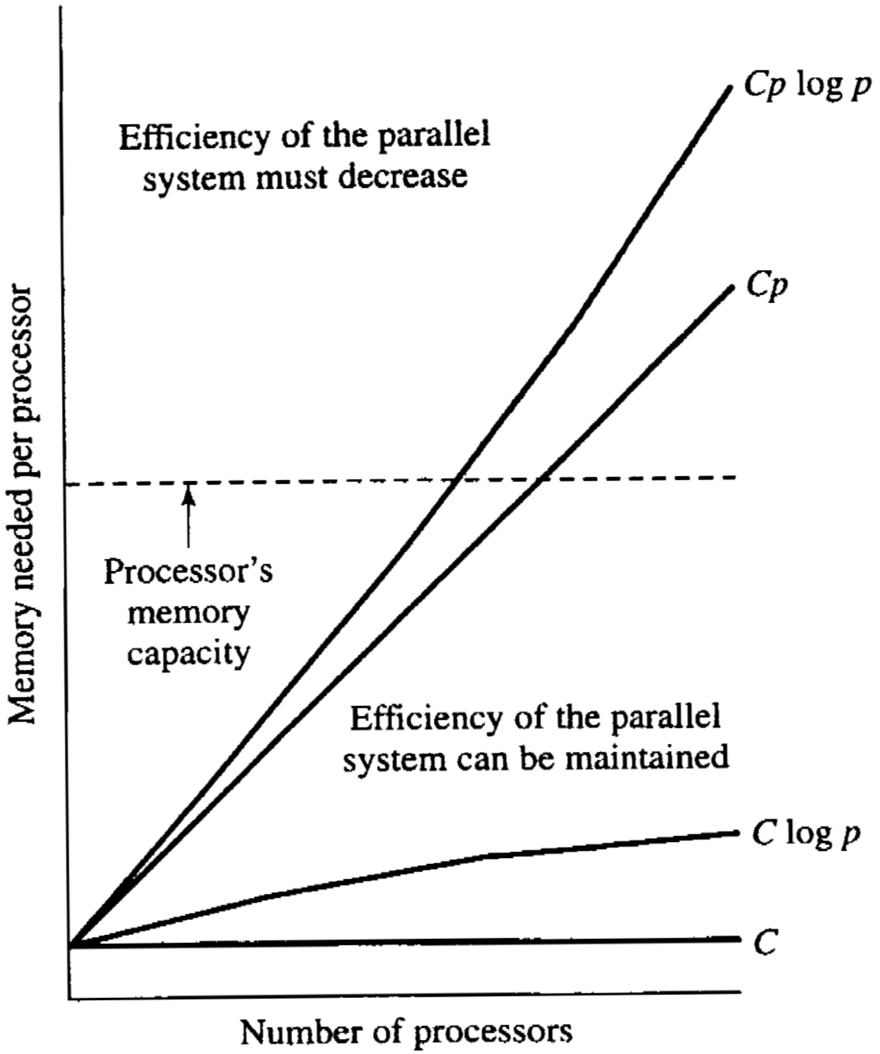
$$T_s(n) \geq \frac{E(n,p)}{1 - E(n,p)} T_{overhead}(n,p) = C T_{overhead}(n,p)$$

  - For good scalability efficiency should be constant
  - $T_{overhead}$ increases with p
  - Inequality can be satisfied only by increasing problem size

Performance theory

Scalability (iso-efficiency)

- Suppose the former relations gives $n \geq g(p)$
- With large problem sizes memory becomes a bottleneck
  - Memory requirements are given by function $M(n)$
  - To maintain efficiency, memory requirements per worker become $M(g(p))/p$



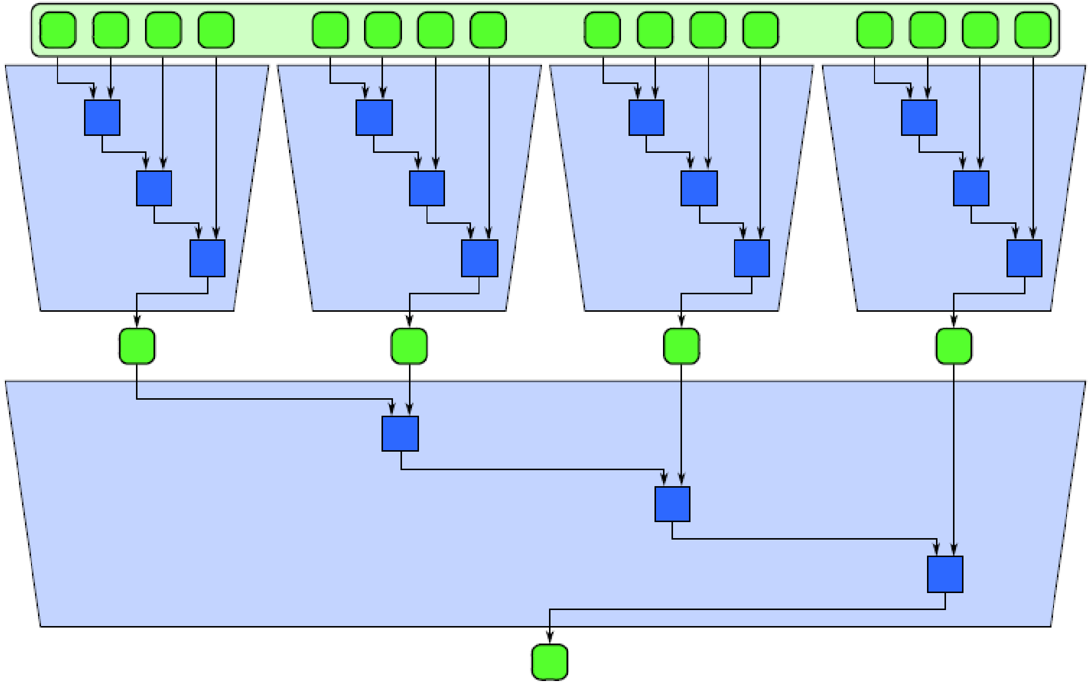
Reduce: theoretical considerations

Reduce with tiling

Use serial algorithm where possible

Do tree-like reduction to reduce communication costs

Process
- Break the work to tiles
- Operate on tiles separately
- Combine partial results from tiles



Serial and tree algorithms
- use the same number of application of the reduce function
- Serial algorithm requires less storage for intermediate results

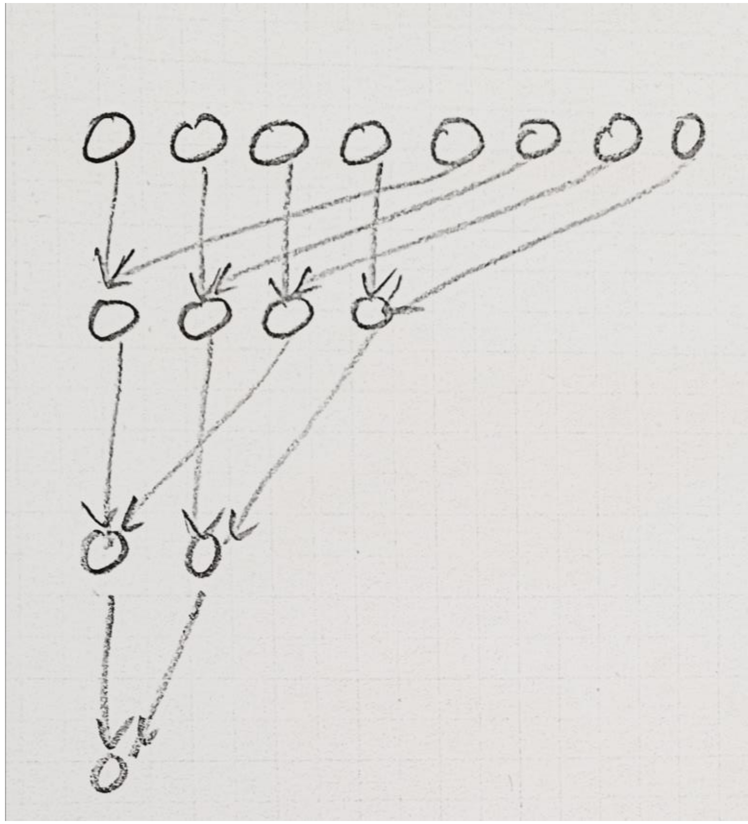
Reduce: theoretical considerations

Sequential reduction of n operands

- $n - 1$ reductions
- Each invocation of reduce function costs χ
- Total execution time $t_s(n) = \chi(n-1)$

Parallel reduction, $n = 2^k, k \in \mathbb{N}$

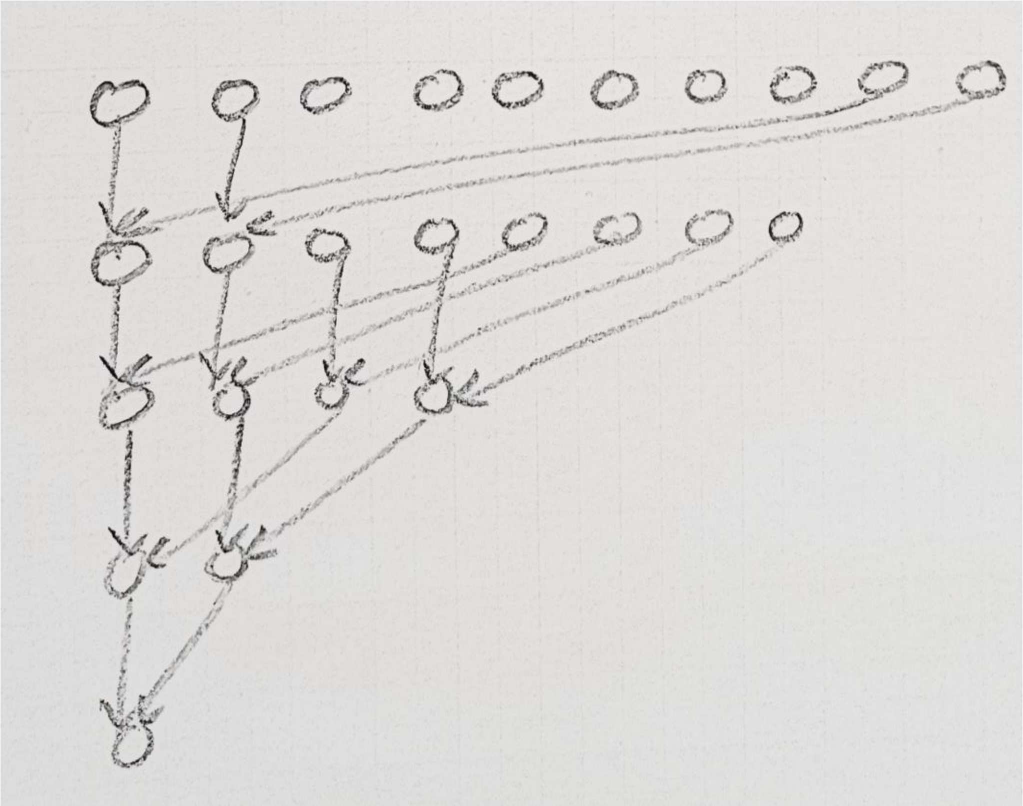
- Communication costs λ
- $n/2$ reductions in the first stage can go in parallel,
  $n/4$ in the second stage can go in parallel …
  1 reduction in the last stage
- altogether we have $\log_2 n$ stages with $n - 1$ reductions
- Total execution time $t_p(n) = (\chi + \lambda)\log_2 n$



Reduce: theoretical considerations

Parallel reduction, $n = 2^k + r, k, r \in \mathbb{N}$

- Additional stage with r reductions the at the very beginning to come to the previous scheme
- Total execution time $t_p(n) = (\chi + \lambda)\lceil \log_2 n \rceil$



Tiled parallel reduction using p tasks

- Each task performs $\lceil n/p \rceil - 1$ sequential reduce operations
- Intermediate results are reduced by three-like scheme in $\lceil \log_2 p \rceil$ steps
- Total execution time

$$t_p(n,p) = \chi\left(\left\lceil \frac{n}{p} \right\rceil - 1\right) + (\chi + \lambda)\lceil \log_2 p \rceil$$

Reduce: theoretical considerations

Scalability (iso-efficiency)

- Iso-efficiency condition $t_s(n) \geq C t_{overhead}(n, p)$
- $t_{overhead}(n, p) = (p-1)\sigma(n) + p\kappa(n, p)$
  - $\sigma(n)$ = 0 all tasks can run in parallel
  - $\kappa(n, p)$ = $\lambda\lceil \log_2 p \rceil$ communication costs
- $t_s(n) \geq C t_{overhead}(n, p) \Rightarrow \chi(n-1) \geq Cp\lambda\lceil \log_2 p \rceil \Rightarrow n \geq C' p \log_2 p$
- According to previous lectures $n \geq g(p) \Rightarrow g(p) = C' p \log_2 p$
- Memory requirements $M(n) = Kn$
- Scalability function $\frac{M(n)}{p} \geq \frac{M(g(p))}{p} = \frac{KC' p \log_2 p}{p} \Rightarrow \frac{M(n)}{p} = C'' \log_2 p$
- Suppose work is doubled and number of processes is doubled
  - Quantity of work per processors remains equal $\chi\left(\left\lceil \frac{2n}{2p} \right\rceil - 1\right) = \chi\left(\left\lceil \frac{n}{p} \right\rceil - 1\right)$
  - One additional reduction step is required $(\chi + \lambda)\lceil \log_2 2p \rceil$= $(\chi + \lambda)(\lceil \log_2 p \rceil + 1)$