Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

MCS 572 Lecture 14
Introduction to Supercomputing
Jan Verschelde, 27 September 2024

Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

time metrics

The goal is to characterize parallel performance.

Metrics are determined from performance measures.

Time metrics are obtained from time measurements.

Time measurements are

1. execution time
   - CPU time and system time
   - I/O time
2. overhead time
   - communication
   - synchronization

The wall clock time measures execution time plus overhead time.

derived metrics

Time metrics come directly from time measurements.

Derived metrics are results of arithmetical metric expressions.

- flops are the number of floating-point operations per second:

$$\frac{\text{number of floating-point operations done}}{\text{execution time}}$$

- communication-to-computation ratio:

$$\frac{\text{communication time}}{\text{execution time}}$$

- memory access-to-computation ratio:

$$\frac{\text{time spent on memory operations}}{\text{execution time}}$$

parallelism metrics and use of metrics

Speedup and efficiency depend on the number of processors and are called parallelism metrics.

Metrics used in performance evaluation:

- Peak speed is the maximum flops a computer can attain.
  Fast Graphics Processing Units achieve teraflop performance.
- Benchmark metrics use representative applications.
  The LINPACK benchmark ranks the Top 500 supercomputers.
- Tuning metrics include bottleneck analysis.

  For task-based parallel programs,
  the application of *critical path analysis* techniques finds
  the longest path in the execution of a parallel program.

Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

efficiency and scalability

For p processors:

$$\text{Speedup} = \frac{\text{serial time}}{\text{parallel time}} = S(p) \to p$$

$$\text{Efficiency} = \frac{\text{Speedup}}{p} = \frac{S(p)}{p} = E(p) \to 1$$

Let T_s denote the serial time, T_p the parallel time, and T_O the overhead, then: $pT_p = T_s + T_O$.

$$E(p) = \frac{T_s}{pT_p} = \frac{T_s}{T_s + T_O} = \frac{1}{1 + T_O/T_s}$$

The scalability analysis of a parallel algorithm measures its capacity to effectively utilize an increasing number of processors.

relating efficiency to work and overhead

Let W be the problem size.

The overhead T_O depends on W and p: $T_O = T_O(W, p)$.

The parallel time equals $T_p = \dfrac{W + T_O(W, p)}{p}$

Speedup $S(p) = \dfrac{W}{T_p} = \dfrac{Wp}{W + T_O(W, p)}$.

Efficiency $E(p) = \dfrac{S(p)}{p} = \dfrac{W}{W + T_O(W, p)} = \dfrac{1}{1 + T_O(W, p)/W}$.

The goal is for $E(p) \to 1$ as $p \to \infty$.

The algorithm *scales badly* if W must grow exponentially to keep efficiency from dropping. If W needs to grow only moderately to keep the overhead in check, then the algorithm *scales well*.

isoefficiency relates work to overhead

$$E = \frac{1}{1 + T_O(W,p)/W} \quad \Rightarrow \quad \frac{1}{E} = \frac{1 + T_O(W,p)/W}{1}$$
$$\Rightarrow \quad \frac{1}{E} - 1 = \frac{T_O(W,p)}{W}$$
$$\Rightarrow \quad \frac{1-E}{E} = \frac{T_O(W,p)}{W}.$$

Definition (the isoefficiency function)

For problem size W, number of processors p, efficiency E, and overhead $T_O(W,p)$, *the isoefficiency function* is

$$W = \left(\frac{E}{1-E}\right) T_O(W,p) \quad \text{or} \quad W = K\, T_O(W,p).$$

Keeping K constant, isoefficiency relates W to T_O.

isoefficiency related to Amdahl and Gustafson

The isoefficiency function is

$$W = \left(\frac{E}{1-E} \right) T_O(W,p) \quad \text{or} \quad W = K\, T_O(W,p).$$

Keeping K constant, isoefficiency relates W to T_O.

- Amdahl's Law: keep W fixed and let p grow.

  $$\text{Speedup } S(p) \leq \frac{1}{R + \frac{1-R}{p}},$$

  where R is fraction of time done sequentially.

- Gustafson's Law: keep p fixed and let W grow.

  $$\text{Scaled speedup } S_s(p) \leq p + (1-p)s,$$

  where s is fraction of time done sequentially.

Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

isoefficiency applied to parallel FFT

The isoefficiency function: $W = K\,T_O(W,p)$.

For FFT: $T_s = n\log(n)t_c$,
where t_c is the time for complex multiplication and adding a pair.

Let t_s denote the startup cost and t_w denote the time to transfer a word.

Time for a parallel FFT:

$$T_p = \underbrace{t_c\left(\frac{n}{p}\right)\log(n)}_{\text{computation time}} + \underbrace{t_s\log(p)}_{\text{start up time}} + \underbrace{t_w\left(\frac{n}{p}\right)\log(p)}_{\text{transfer time}}.$$

start up cost versus computation

Using the expression for T_p in the efficiency $E(p)$:

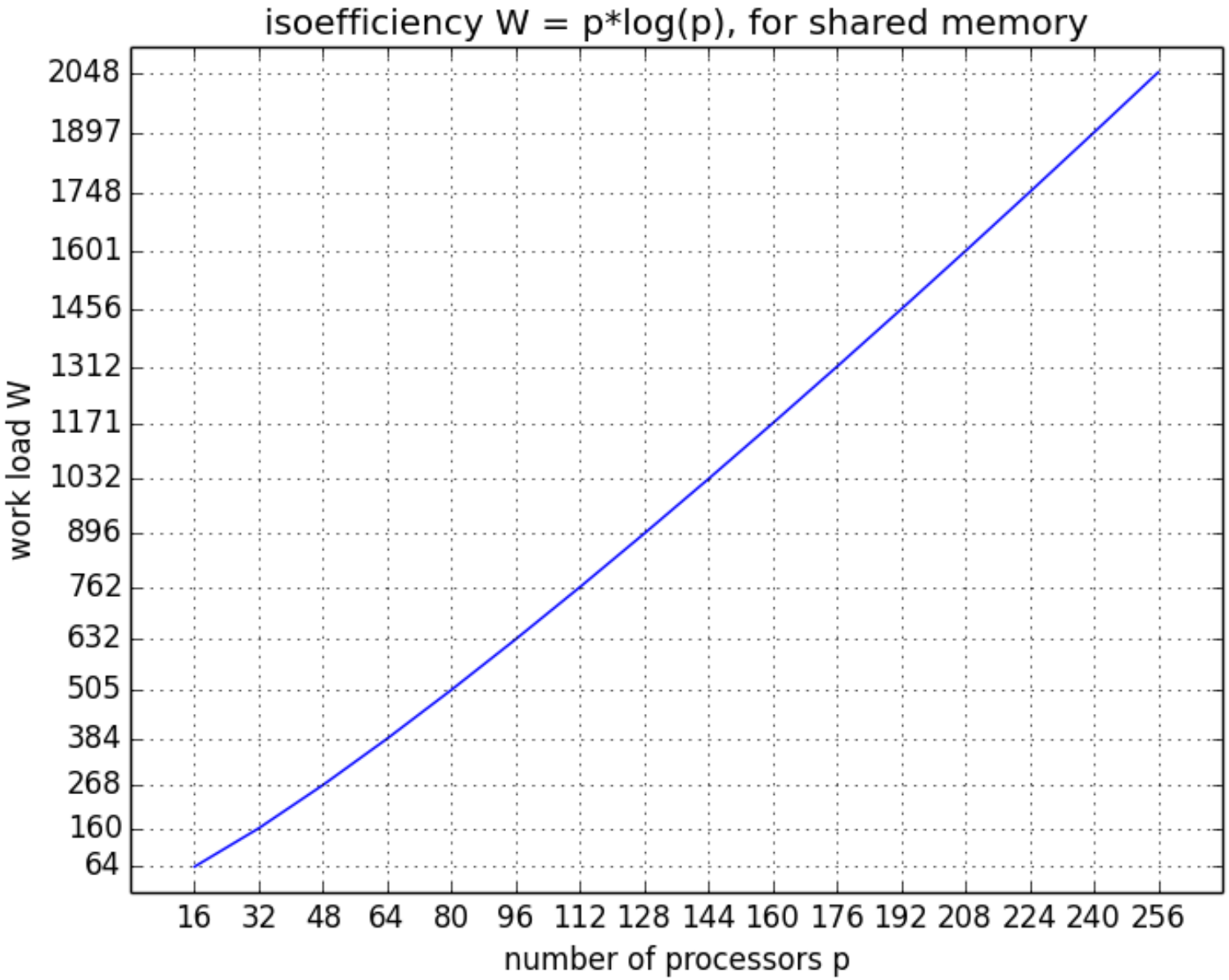
$$E(p) = \frac{T_s}{pT_p} = \frac{n\log(n)t_c}{n\log(n)t_c + p\log(p)t_s + n\log(p)t_w}$$
$$= \frac{Wt_c}{Wt_c + p\log(p)t_s + n\log(p)t_w}, \quad W = n\log(n).$$

Assume $t_w = 0$ (shared memory): $E(p) = \dfrac{Wt_c}{Wt_c + p\log(p)t_s}$.

We want to express $K = \dfrac{E}{1-E}$, using $\dfrac{1}{K} = \dfrac{1-E}{E} = \dfrac{1}{E} - 1$:

$$\frac{1}{K} = \frac{Wt_c + p\log(p)t_s}{Wt_c} - \frac{Wt_c}{Wt_c} \quad \Rightarrow \quad W = K\left(\frac{t_s}{t_c}\right)p\log(p).$$

isoefficiency for shared memory



transfer cost versus computation

Taking another look at the efficiency $E(p)$:

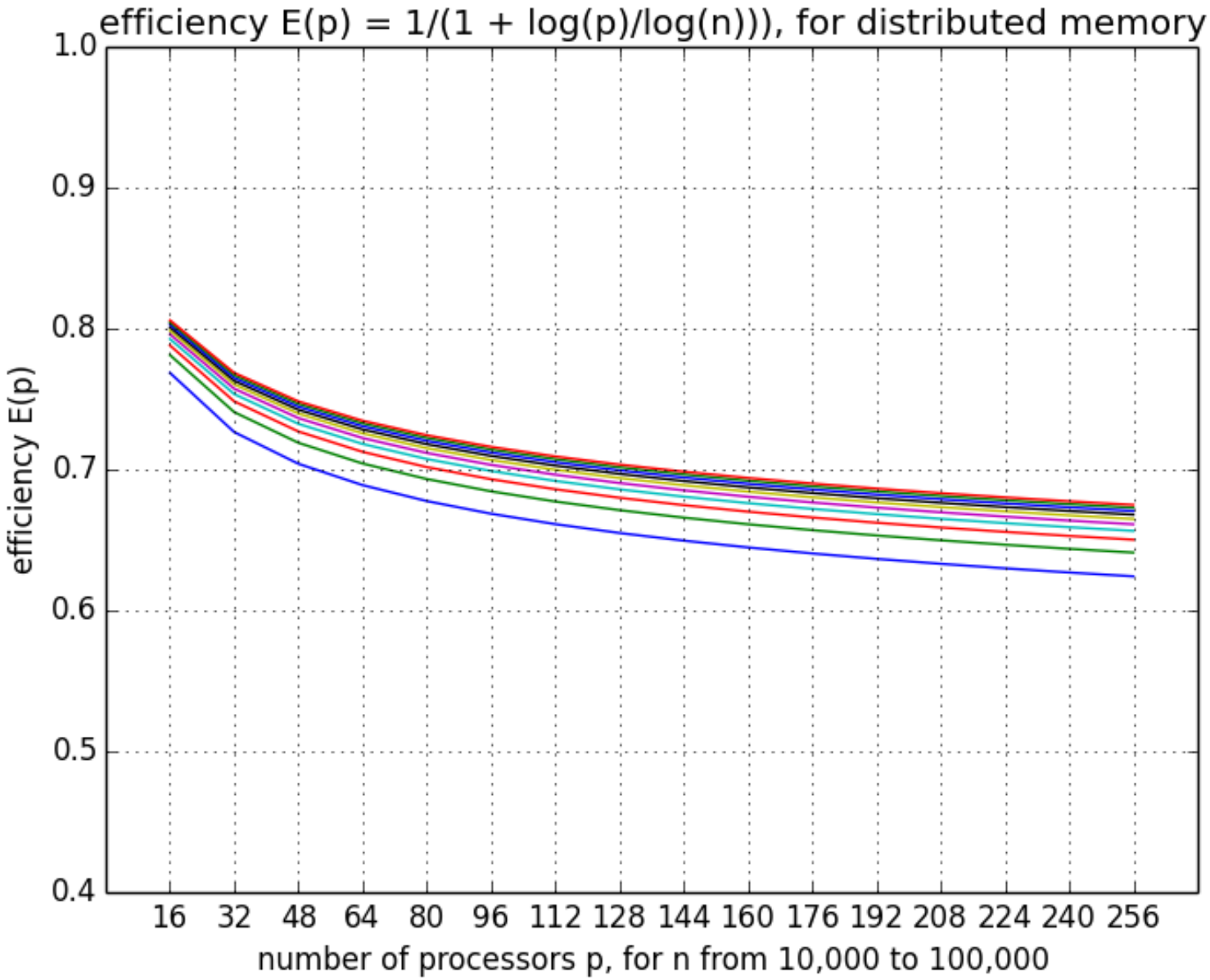
$$E(p) = \frac{Wt_c}{Wt_c + p\log(p)t_s + n\log(p)t_w}, \quad W = n\log(n).$$

Assume $t_s = 0$ (no start up): $E(p) = \dfrac{Wt_c}{Wt_c + n\log(p)t_w}$.

We want to express $K = \dfrac{E}{1-E}$, using $\dfrac{1}{K} = \dfrac{1-E}{E} = \dfrac{1}{E} - 1$:

$$\frac{1}{K} = \frac{Wt_c + n\log(p)t_w}{Wt_c} - \frac{Wt_c}{Wt_c} \quad \Rightarrow \quad W = K\left(\frac{t_w}{t_c}\right) n\log(p).$$

efficiency plot for distributed memory



Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

task graph scheduling

A task graph is a Directed Acyclic Graph (DAG):

- nodes are tasks, and
- edges are precedence constraints between tasks.

Task graph scheduling or DAG scheduling maps the task graph onto a target platform.

The scheduler

1. takes a task graph as input,
2. decides which processor will execute what task,
3. with the objective to minimize the total execution time.

an example: forward substitution

Consider $L\mathbf{x} = \mathbf{b}$, an n-by-n lower triangular linear system, where $L = [\ell_{i,j}] \in \mathbb{R}^{n \times n}$, $\ell_{i,i} \neq 0$, $\ell_{i,j} = 0$, for $j > i$.

For $n = 3$:

$$
\begin{array}{rcll}
\ell_{1,1}x_1 & = & b_1 & \Rightarrow \quad x_1 := b_1/\ell_{1,1} \\
\ell_{2,1}x_1 + \ell_{2,2}x_2 & = & b_2 & \Rightarrow \quad x_2 := (b_2 - \ell_{2,1}x_1)/\ell_{2,2} \\
\ell_{3,1}x_1 + \ell_{3,2}x_2 + \ell_{3,3}x_3 & = & b_3 & \Rightarrow \quad x_3 := (b_3 - \ell_{3,1}x_1 - \ell_{3,2}x_2)/\ell_{3,3}
\end{array}
$$

Pseudo code with tasks labeled for each instruction:

```
task T_{1,1}: x_1 := b_1/l_{1,1}
for i from 2 to n do
    for j from 1 to i - 1 do
        task T_{i,j}: b_i := b_i - l_{i,j}x_j
    task T_{i,i}: x_i := b_i/l_{i,i}
```

applying Bernstein's conditions

Each task T has an input set $\text{in}(T)$, and an output set $\text{out}(T)$.

Tasks T_1 and T_2 are independent if

$$\begin{aligned} \text{in}(T_1) \cap \text{out}(T_2) &= \emptyset, \\ \text{out}(T_1) \cap \text{in}(T_2) &= \emptyset, \\ \text{out}(T_1) \cap \text{out}(T_2) &= \emptyset. \end{aligned}$$

Applied to forward substitution:

task $T_{1,1}$: $x_1 := b_1/\ell_{1,1}$ $\quad \text{in}(T_{1,1}) = \{b_1, \ell_{1,1}\}$, $\text{out}(T_{1,1}) = \{x_1\}$

for i from 2 to n do

  for j from 1 to $i-1$ do

    task $T_{i,j}$: $b_i := b_i - \ell_{i,j} x_j$ $\quad \text{in}(T_{i,j}) = \{x_j, b_i, \ell_{i,j}\}$, $\text{out}(T_{i,j}) = \{b_i\}$

  task $T_{i,i}$: $x_i := b_i/\ell_{i,i}$ $\quad \text{in}(T_{i,i}) = \{b_i, \ell_{i,i}\}$, $\text{out}(T_{i,i}) = \{x_i\}$

the task graph of forward substitution

task $T_{1,1}$: $x_1 := b_1/\ell_{1,1}$ $\quad \text{in}(T_{1,1}) = \{b_1, \ell_{1,1}\}$, $\text{out}(T_{1,1}) = \{x_1\}$

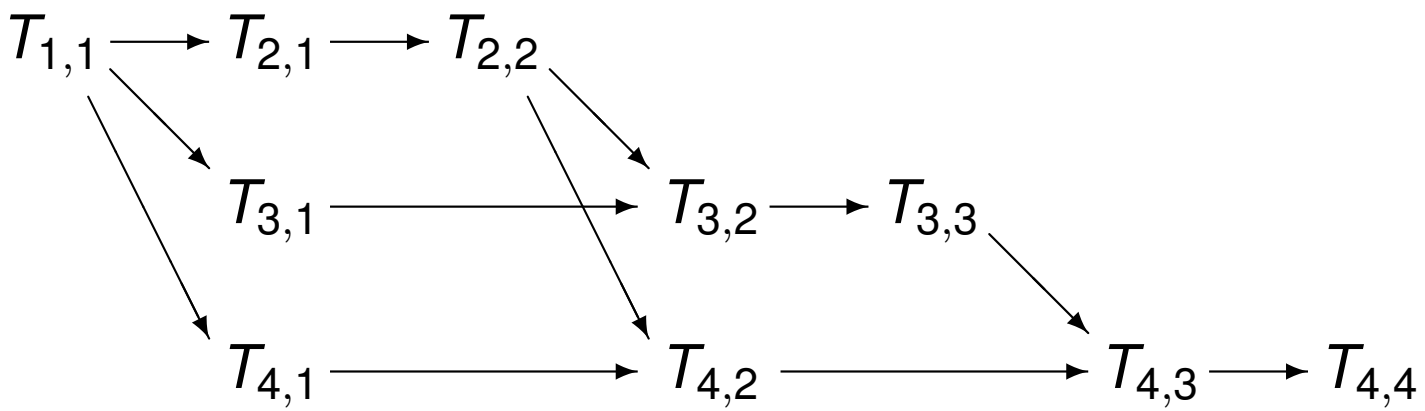
for i from 2 to n do

  for j from 1 to $i-1$ do

    task $T_{i,j}$: $b_i := b_i - \ell_{i,j} x_j$ $\quad \text{in}(T_{i,j}) = \{x_j, b_i, \ell_{i,j}\}$, $\text{out}(T_{i,j}) = \{b_i\}$

  task $T_{i,i}$: $x_i := b_i/\ell_{i,i}$ $\quad \text{in}(T_{i,i}) = \{b_i, \ell_{i,i}\}$, $\text{out}(T_{i,i}) = \{x_i\}$

The task graph for $n = 4$:

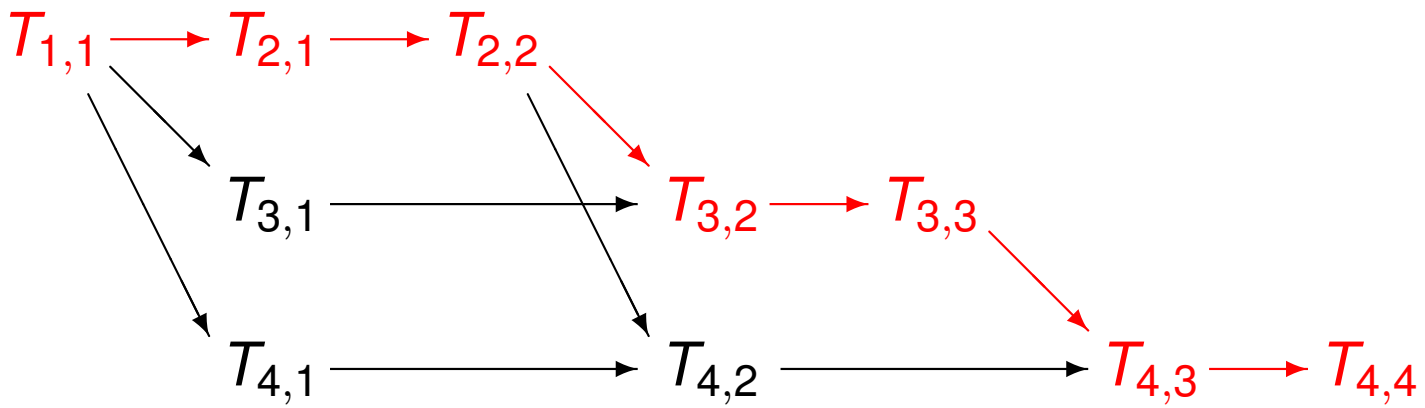


Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

one critical path

In the task graph, a *critical path* is colored in red.



Recall that $T_{i,i}$ computes x_i.

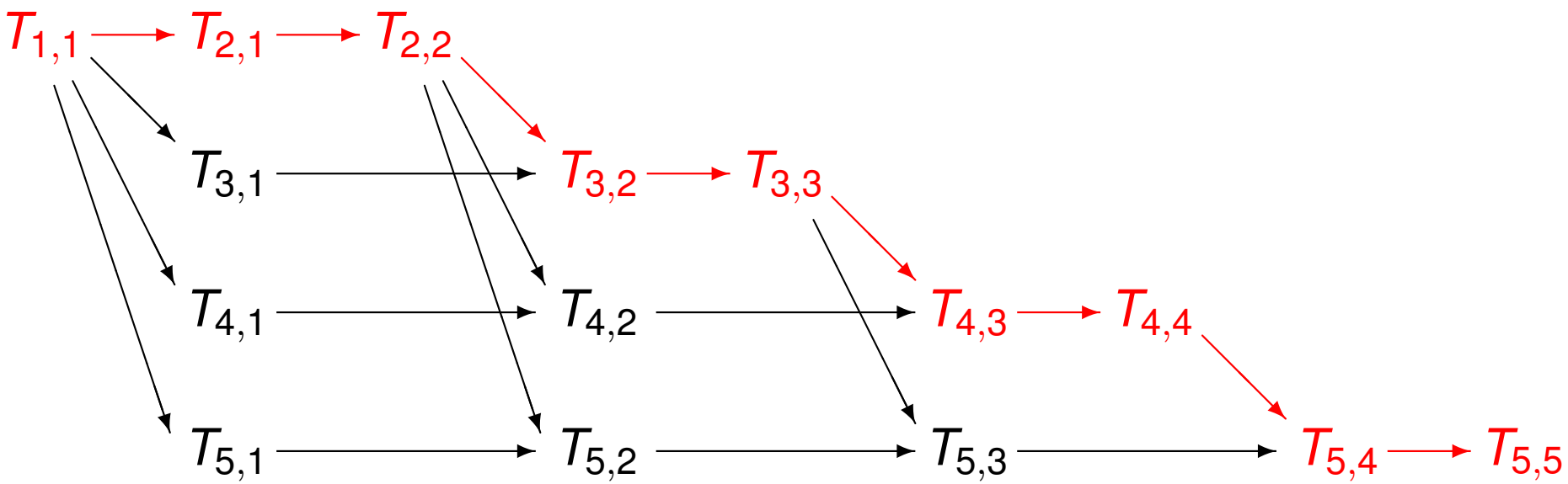
The length of a critical path limits the speedup.
For the above example, a sequential execution

$$T_{1,1}, T_{2,1}, T_{3,1}, T_{4,1}, T_{2,2}, T_{3,2}, T_{4,2}, T_{3,3}, T_{4,3}, T_{4,4}$$

takes 10 steps. The length of a critical path is 7.
At most three threads can compute simultaneously.

the DAG for $n = 5$

In the task graph, a *critical path* is colored in red.



For $n = 4$, we found 7. For $n = 5$, the length of the critical path is 9.

For any n, the length of the critical path is $2n - 1$.
At most $n - 1$ threads can compute simultaneously.

recommended reading

- Alan D. Malony: **Metrics.** In *Encycopedia of Parallel Computing*, edited by David Padua, pages 1124–1130, Springer 2011.
- Yves Robert: **Task Graph Scheduling.** In *Encycopedia of Parallel Computing*, edited by David Padua, pages 2013–2024, Springer 2011.
- Vipin Kumar and Anshul Gupta: **Analyzing Scalability of Parallel Algorithms and Architectures**. *Journal of Parallel and Distributed Computing* 22: 379–391, 1994.
- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar: **Introduction to Parallel Computing**. 2nd edition, Pearson 2003. Chapter 13 is devoted to the Fast Fourier Transform.
- Thomas Decker and Werner Krandick: **On the Isoefficiency of the Parallel Descartes Method**. In *Symbolic Algebraic Methods and Verification Methods* 2001, pages 55–67, Springer 2001.

Evaluating Parallel Performance

1. Metrics
   - time metrics and derived metrics
2. Isoefficiency
   - efficiency and scalability
   - an illustration of isoefficiency
3. Task Graph Scheduling
   - mapping tasks to threads
   - critical path analysis
4. The Roofline Model
   - arithmetic intensity

arithmetic intensity

Performance is measured in *flops*:
the number of floating-point operations per second.

Definition (arithmetic intensity)

The *arithmetic intensity* of a computation
is the number of floating-point operations per byte.

Example: $z := x + y$, assign $x + y$ to z.

- One floating point operation involving 64-bit doubles, and
- each double occupies 8 bytes,

so the arithmetic intensity is $1/24$.

memory bound and compute bound

Do you want faster memory or faster processors?

Definition (memory bound)

A computation is *memory bound*
if the peak memory bandwidth determines the performance.

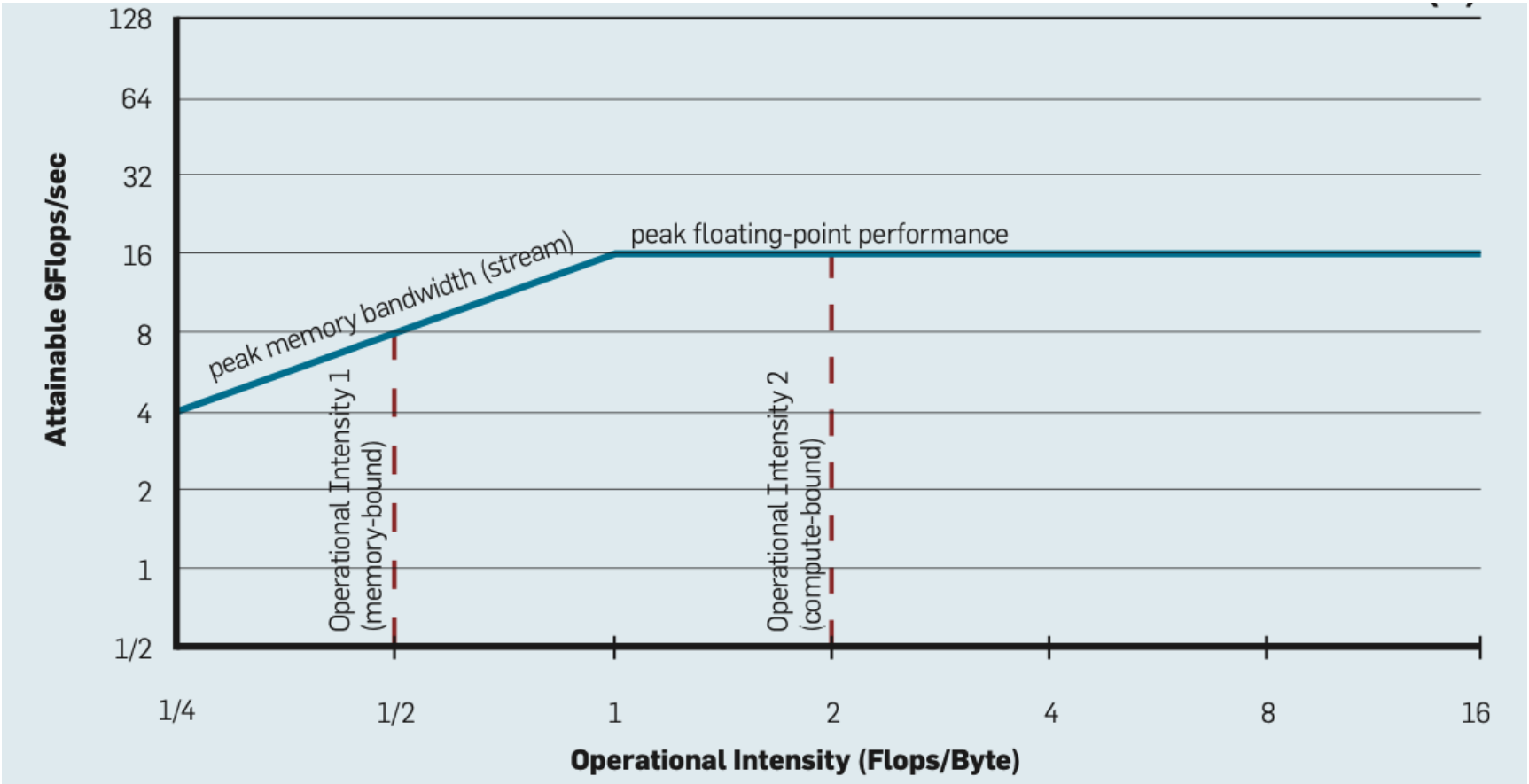
Memory bandwidth is the number of bytes per second
that can be read or stored in memory.

Definition (compute bound)

A computation is *compute bound*
if the peak floating-point performance determines the performance.

A high arithmetic intensity is needed for a compute bound computation.

the roofline model



copied from S. Williams, A. Waterman, and D. Patterson:
Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

the formula for attainable performance

$$\text{attainable GFlops/sec} = \min \begin{cases} \text{peak floating point performance} \\ \text{peak memory bandwidth} \times \text{operational intensity} \end{cases}$$

Observe the difference between arithmetic and operational intensity:

- arithmetic intensity measures the number of floating point operations per byte,
- operational intensity measures the number of operations per byte.

applying the roofline model

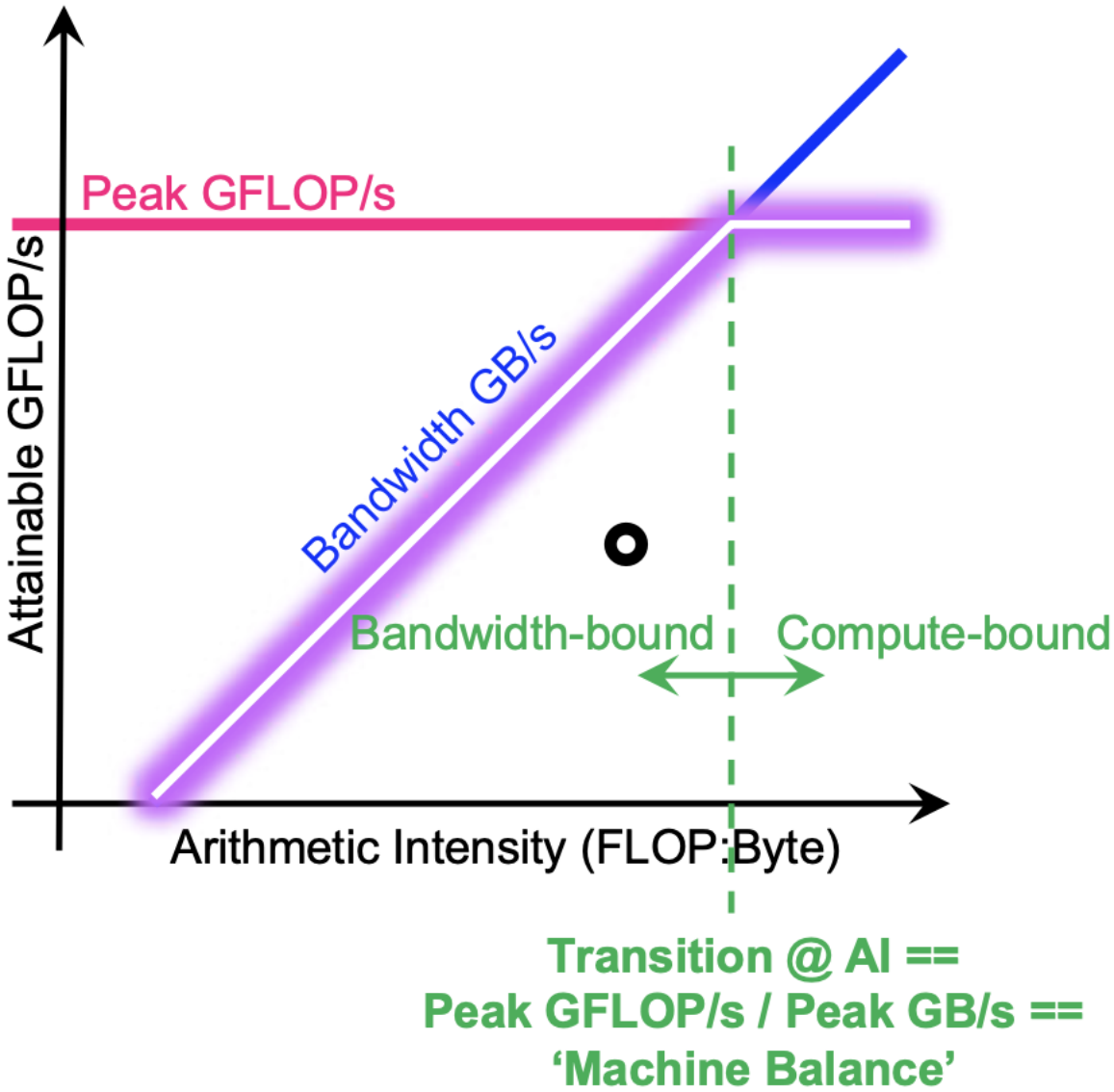
1. The horizontal line is the theoretical peak performance, expressed in gigaflops per second, the units of the vertical axis.
2. The units of the horizontal coordinate axis are flops per byte.

   The *ridge point* is the ratio of the theoretical peak performance and the memory bandwidth.
3. For any particular computation, record the pair (x, y)
   1. x is the arithmetic intensity, number of flops per byte,
   2. y is the performance defined by the number of flops per second.

   If (x, y) lies under the horizontal part of the roof,
   then the computation is compute bound,
   otherwise, the computation is memory bound.

memory bound or compute bound?

from tutorial slides by Charlene Yang, LBNL, 16 Jun 2019



Summary and Exercises

We covered metrics to evaluate parallel performance, derived isoefficiency, and studied an example of a critical path analysis.

Exercises:

1. Consider the isoefficiency formulas we derived for a parallel version of the FFT. Suppose an efficiency of 0.6 is desired. For values $t_c = 1$, $t_s = 25$ and $t_w = 4$, make plots of the speedup for increasing values of n, taking $p = 64$. Interpret the plots relating to the particular choices of the parameters t_c, t_s, t_w, and the desired efficiency.
2. Apply the Bernstein conditions to justify the DAG presented for $n = 4$ in the forward substitution example. In particular,
   1. demonstrate the need for every edge in the DAG; and
   2. for nodes in the DAG that can be executed independently, verify that Bernstein's conditions are met.