MARCIN COPIK <MARCIN.COPIK@INF.ETHZ.CH>

DPHPC: Amdahl's Law, Roofline model

Recitation session

Why do we have to do performance modelling?

- **Will my program scale?**
  "Am I going to run faster on twice larger machine?"
- **Which parts of the program I should improve?**

"Let me parallelize one more loop, that should help… I can't be spending 90% of time on communication and synchronization!"

- **Can my program achieve better performance? How far is it from maximum?**
  "I spent 50 hours on optimizing every memory accesses and I'm 0.5% faster"
- **How should we design a new computing system?**
  "Do I need accelerators? Do I need more memory?"

Why do we have to do performance modelling?

- **Will my program scale?**
  "Am I going to run faster on twice larger machine?"
- **Which parts of the program I should improve?**

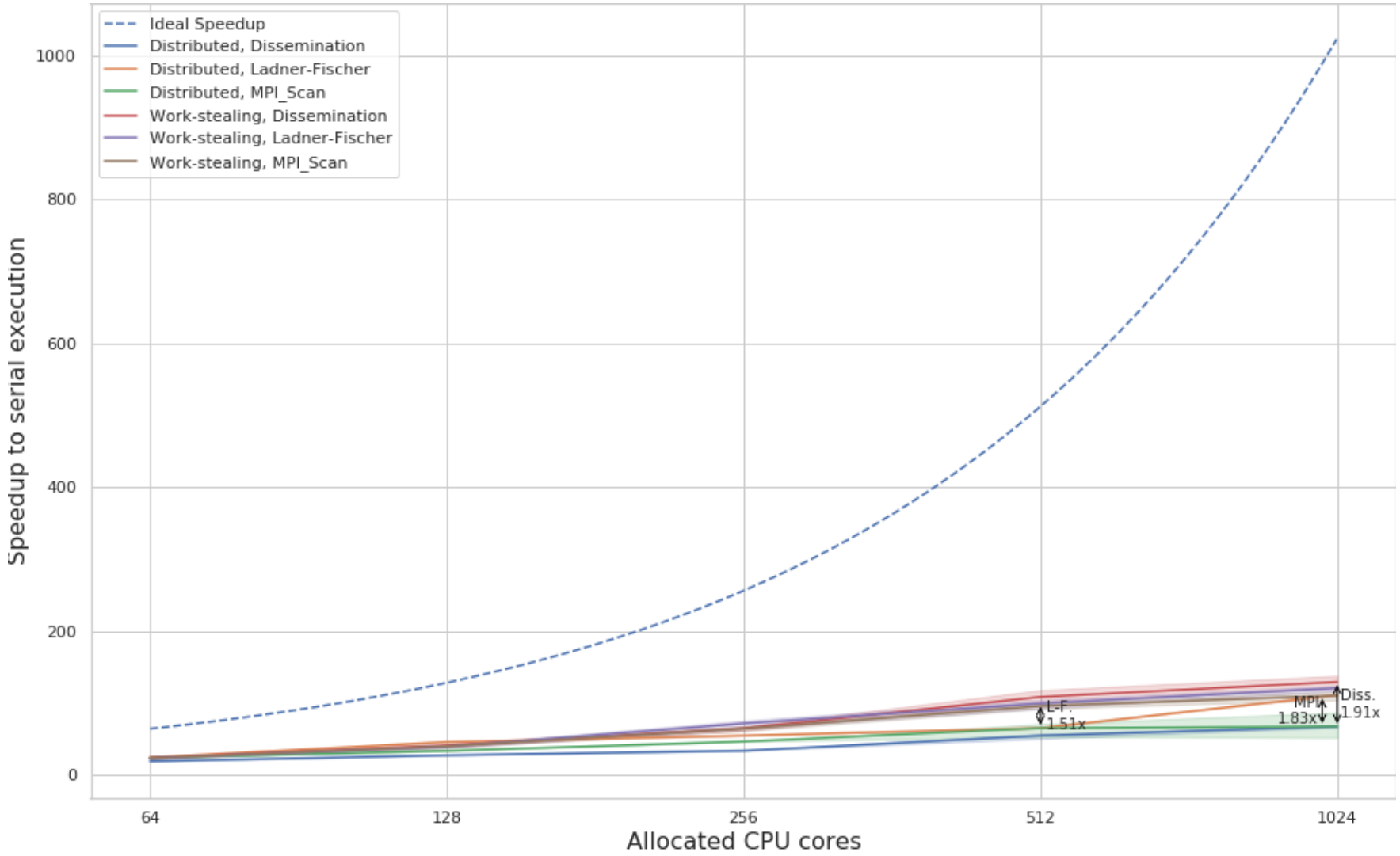
"Let me parallelize one more loop, that should help… I can't be spending 90% of time on communication and synchronization!"

- **Can my program achieve better performance? How far is it from maximum?**
  "I spent 50 hours on optimizing every memory accesses and I'm 0.5% faster"
- **How should we design a new computing system?**
  "Do I need accelerators? Do I need more memory?"

Two things we need to understand

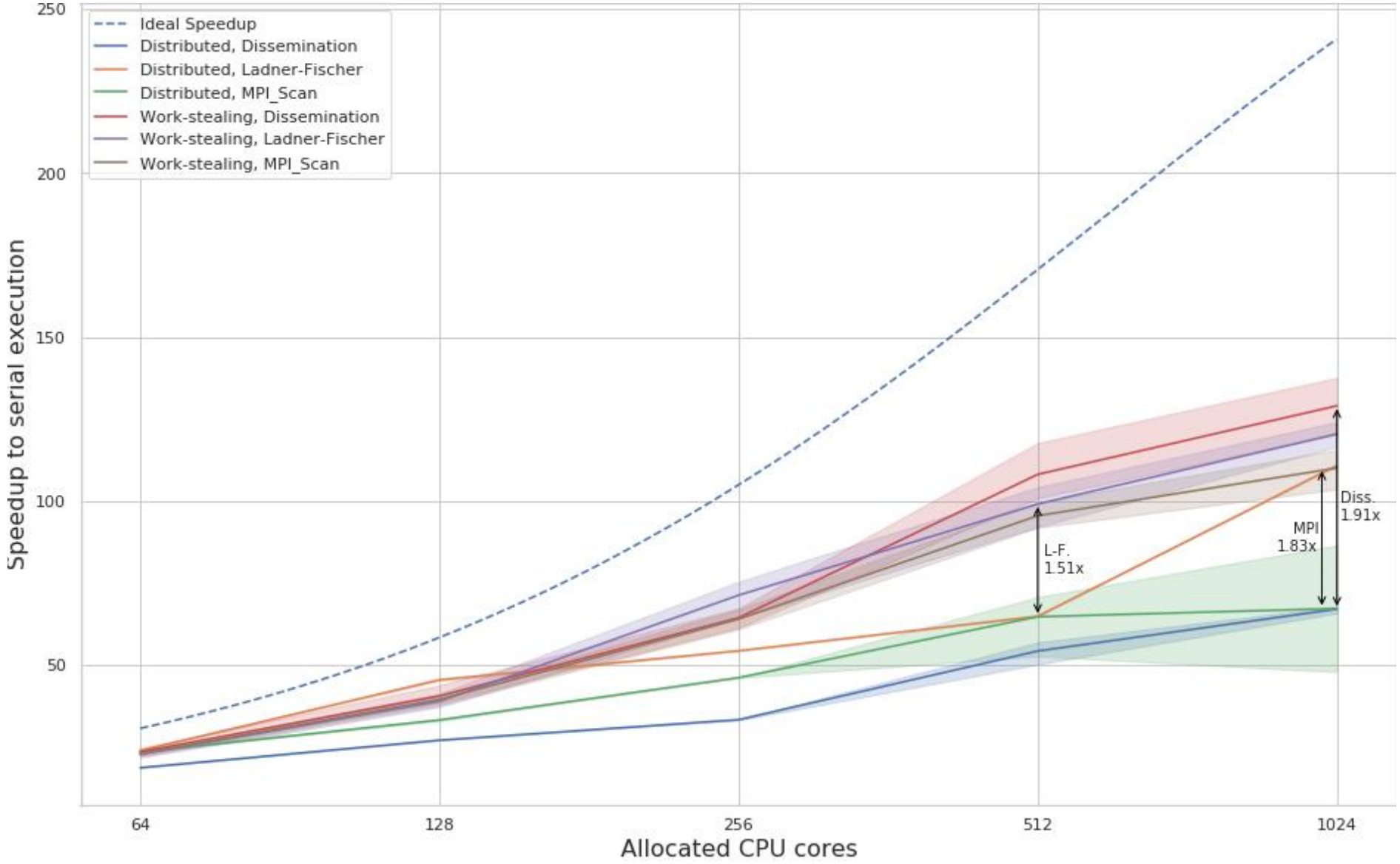
- **Baseline – slow and bad programs tend to scale better.**
- **Upper bound**

Why is upper bound important?

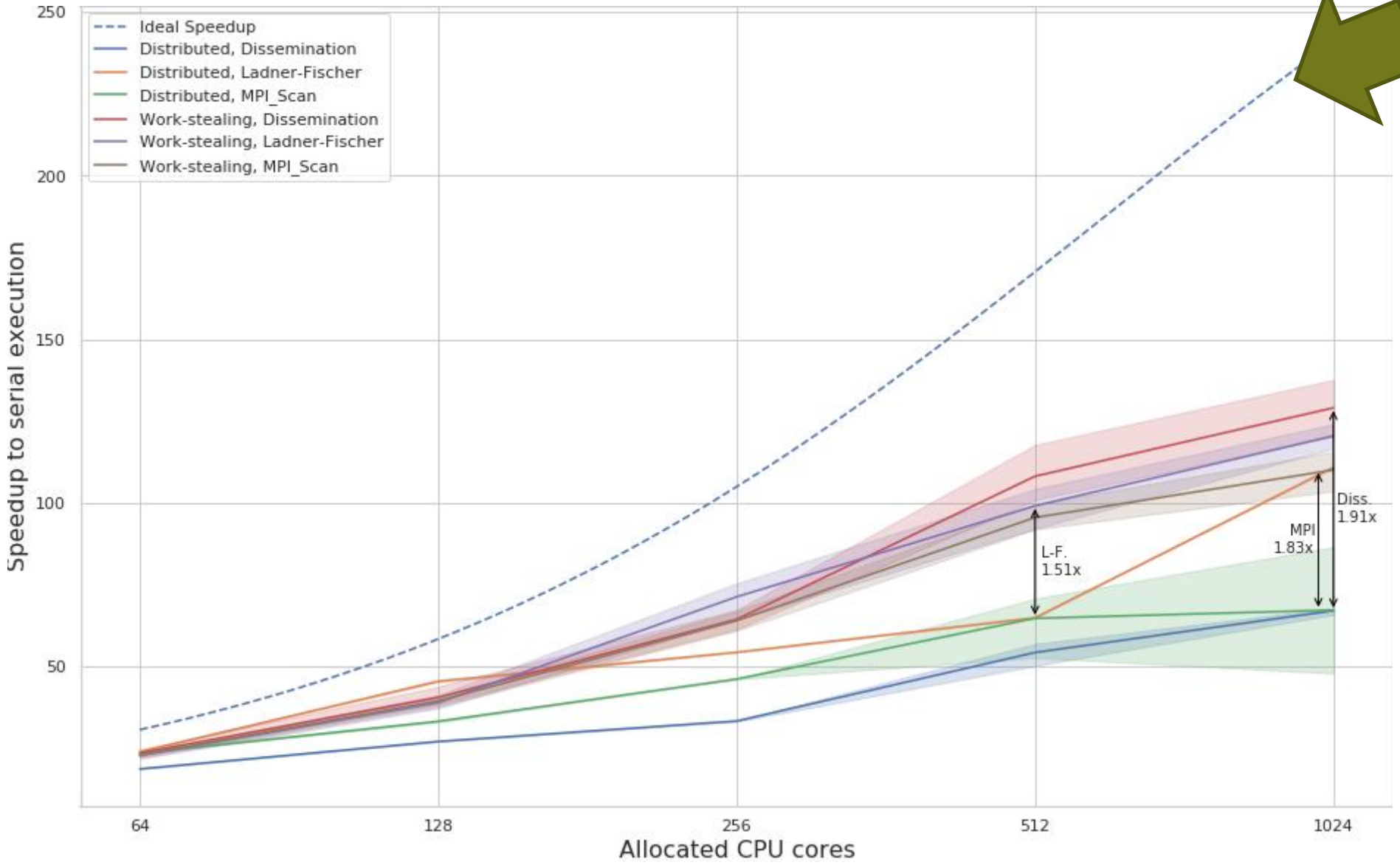


This is pretty pathetic…

Why is upper bound important?

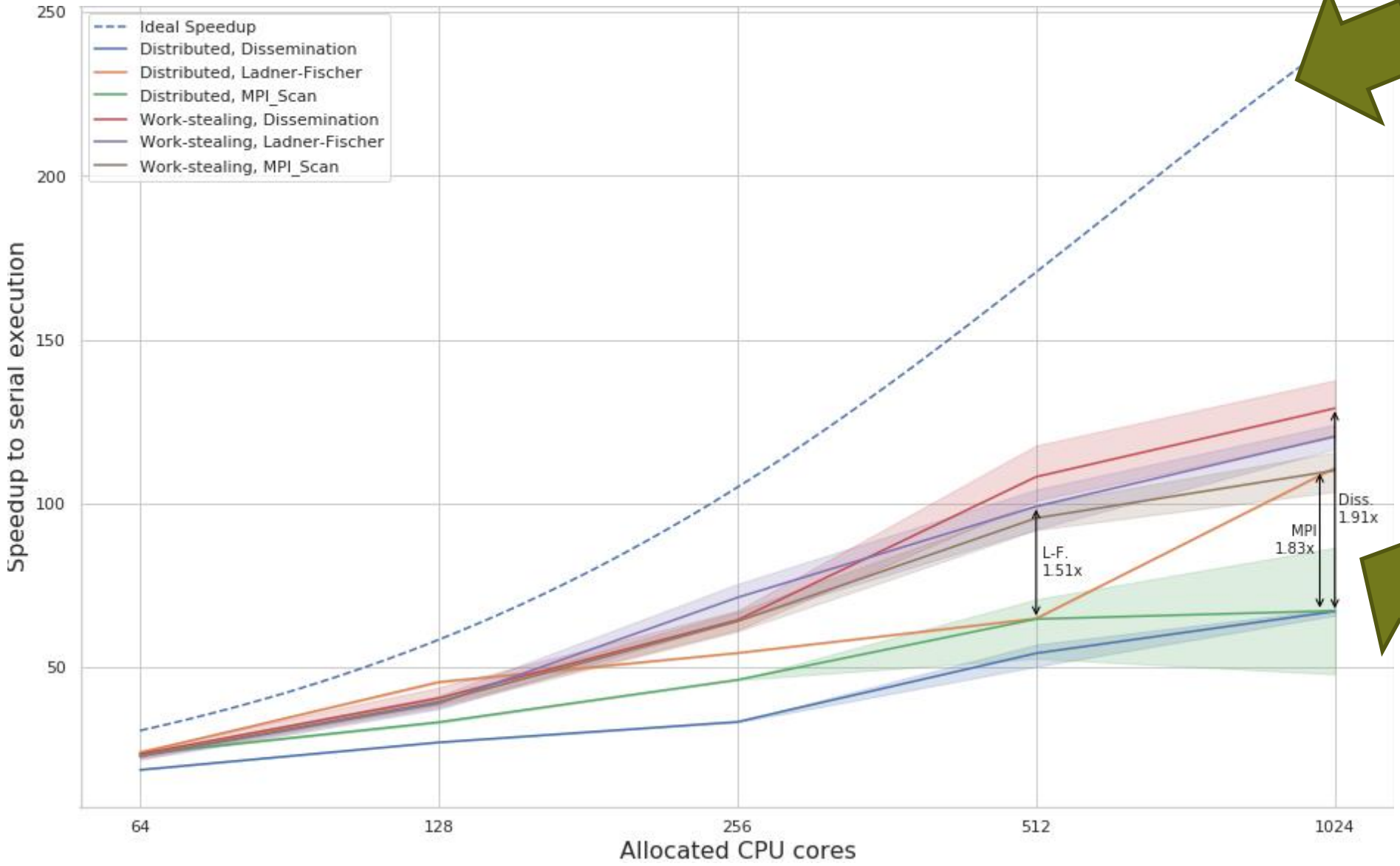


Why is upper bound important?



Upper bound for prefix scan

Why is upper bound important?



Upper bound for prefix scan

We're actually improving!

Amdahl's Law

Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1 - f)T_1$$

Amdahl's Law

Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1 - f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1 - f)T_1}{P}$$

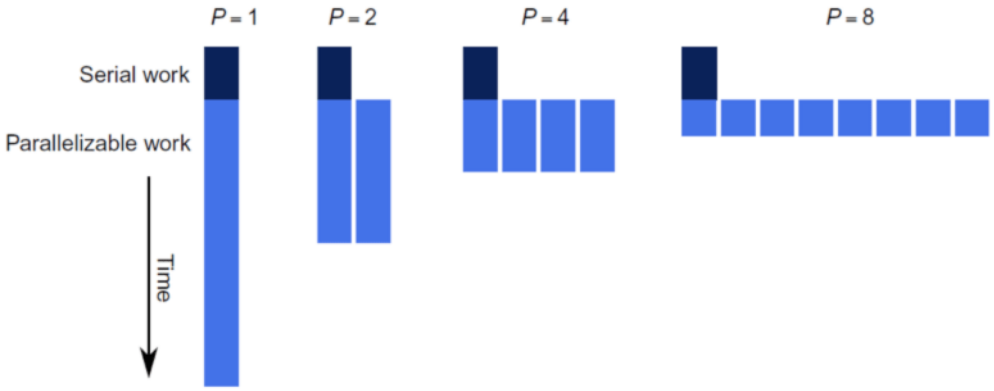
Amdahl's Law

Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1-f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1-f)T_1}{P}$$



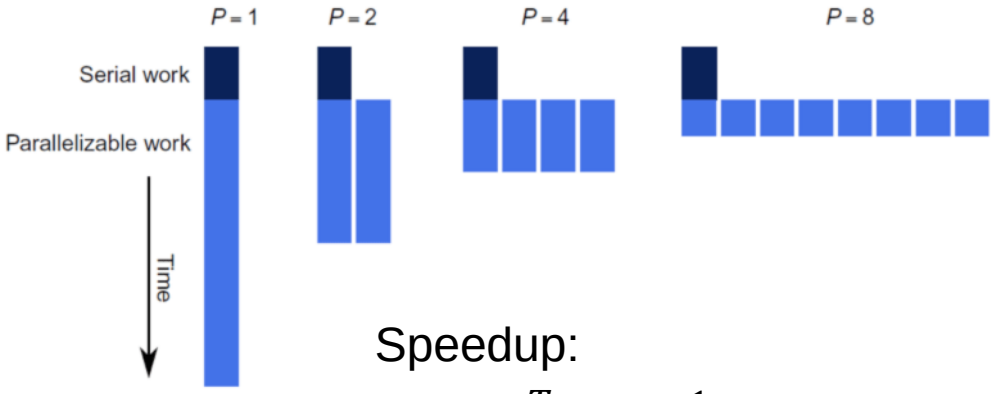
Amdahl's Law

Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1-f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1-f)T_1}{P}$$



Speedup:

$$S_P = \frac{T_1}{T_P} \leq \frac{1}{\frac{1-f}{P}+f}$$ c

Amdahl's Law

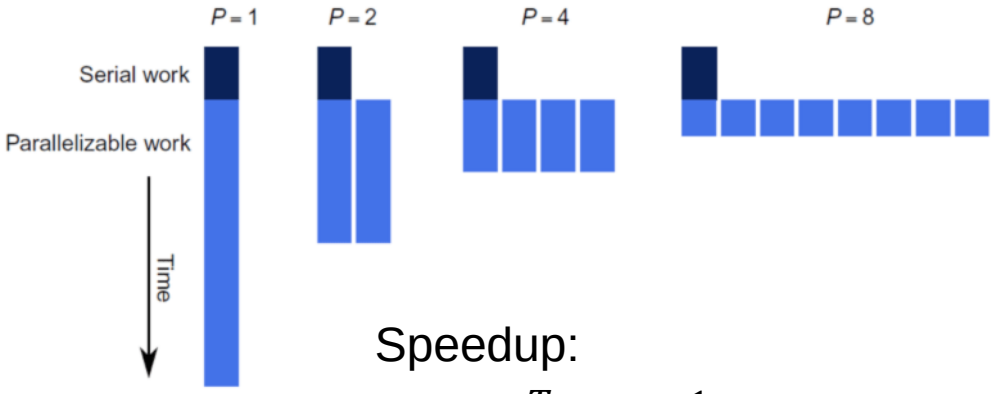
Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1-f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1-f)T_1}{P}$$

$$T_\infty = fT_1$$



Speedup:

$$S_P = \frac{T_1}{T_P} \leq \frac{1}{\frac{1-f}{P}+f}$$ c

Amdahl's Law

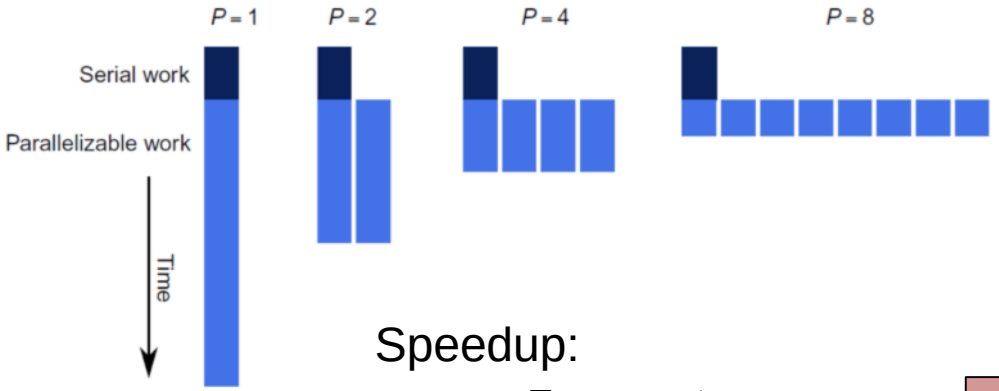
Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1-f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1-f)T_1}{P}$$

$$T_\infty = fT_1$$



Speedup:

$$S_P = \frac{T_1}{T_P} \leq \frac{1}{\frac{1-f}{P}+f}$$ c

$$S_\infty \leq \frac{1}{f}$$

Amdahl's Law

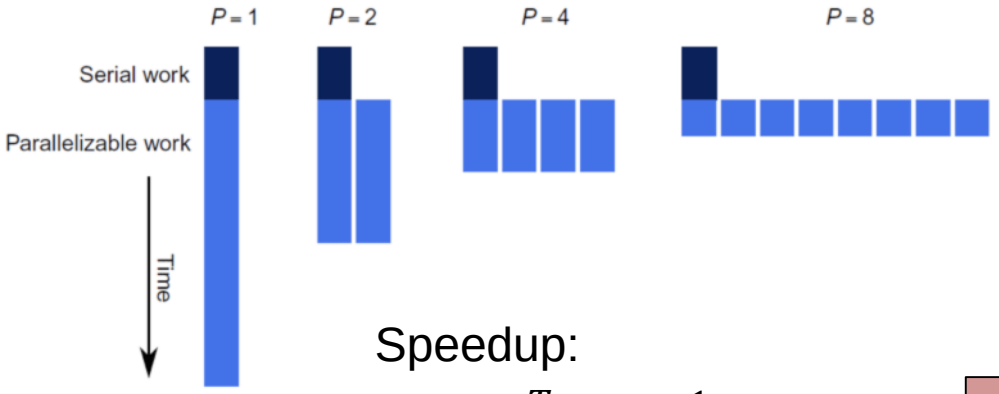
Time of sequential program with f as the fraction not affected by the parallelization:

$$T_1 = fT_1 + (1-f)T_1$$

Time of parallel program:

$$T_P \geq fT_1 + \frac{(1-f)T_1}{P}$$

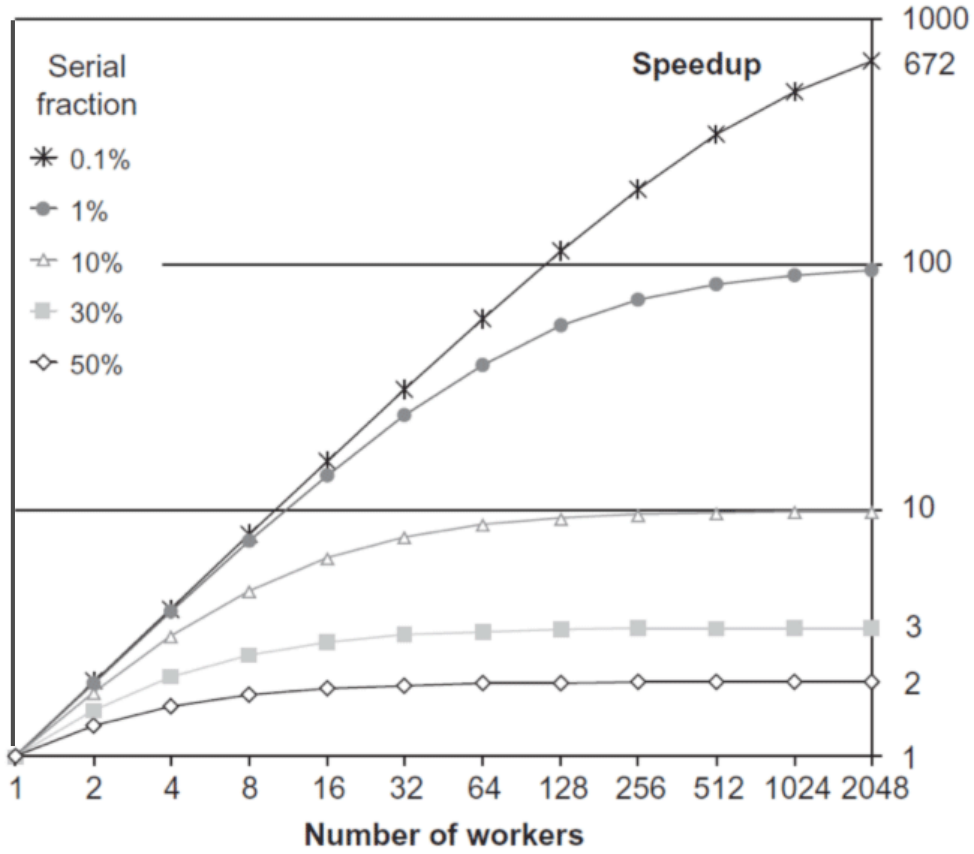
$$T_\infty = fT_1$$



Speedup:

$$S_P = \frac{T_1}{T_P} \leq \frac{1}{\frac{1-f}{P}+f}$$ c

$$S_\infty \leq \frac{1}{f}$$



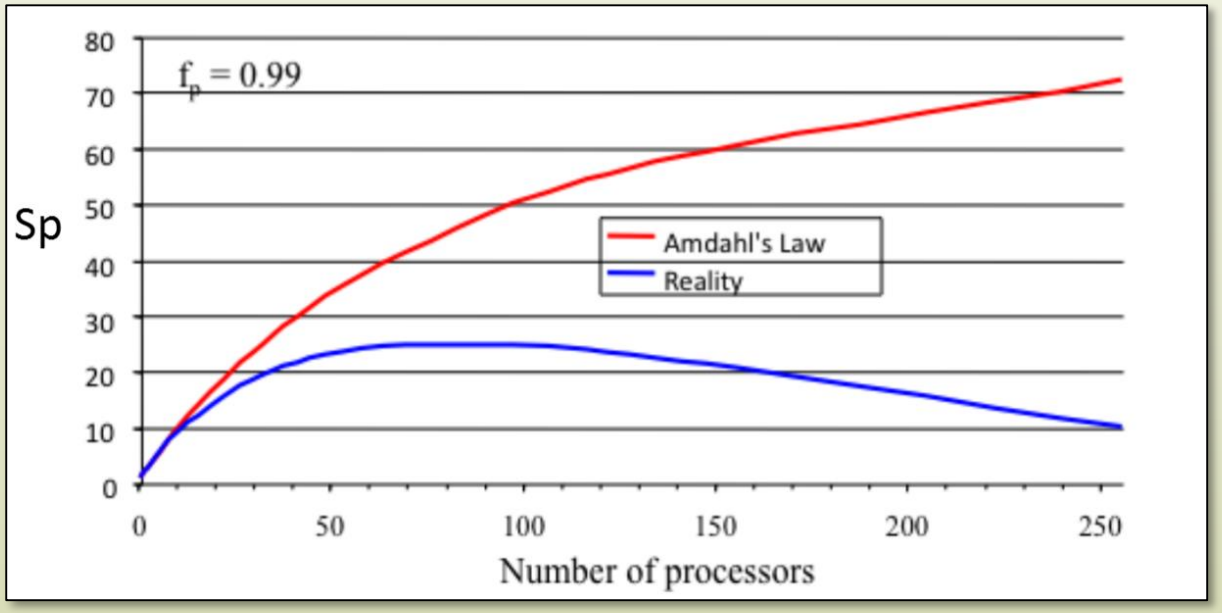
Amdahl's Law

Time of sequenti
by the parallel

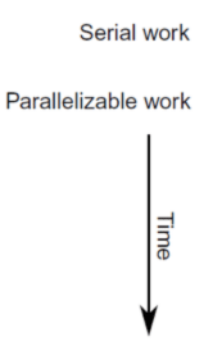
Time of paralle

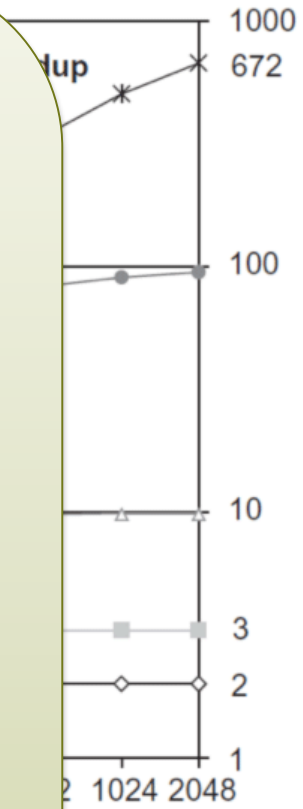
It's like to see the glass as half empty but…

It could be even worse!



Possible factors: *load balancing, communication costs, I/O, scheduling*





Amdahl’s Law vs Gustafson-Barsis' Law

> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Amdahl's Law vs Gustafson-Barsis' Law

> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Time of sequential program with α as the fraction not affected by the parallelization on P-processors machine:

$$T_1 = \alpha T_1 + (1 - \alpha) P T_1$$

Amdahl's Law vs Gustafson-Barsis' Law

> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Time of sequential program with α as the fraction not affected by the parallelization on P-processors machine:

$$T_1 = \alpha T_1 + (1 - \alpha) P T_1$$

Time of parallel program:

$$T_P = \alpha T_1 + (1 - \alpha) T_1$$

Amdahl's Law vs Gustafson-Barsis' Law

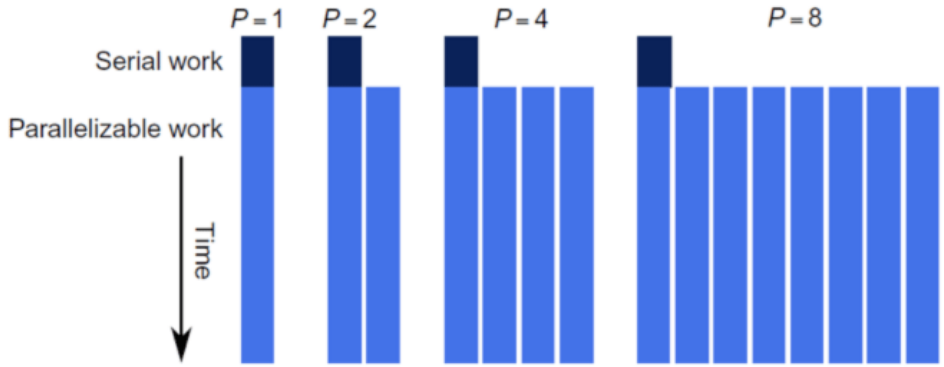
> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Time of sequential program with α as the fraction not affected by the parallelization on P-processors machine:

$$T_1 = \alpha T_1 + (1 - \alpha) P T_1$$

Time of parallel program:

$$T_P = \alpha T_1 + (1 - \alpha) T_1$$



Amdahl's Law vs Gustafson-Barsis' Law

> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Time of sequential program with α as the fraction not affected by the parallelization on P-processors machine:

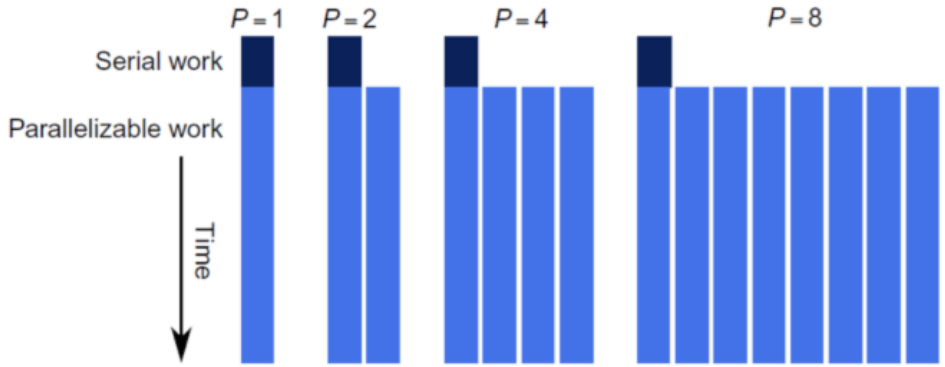
$$T_1 = \alpha T_1 + (1-\alpha)PT_1$$

Time of parallel program:

$$T_P = \alpha T_1 + (1-\alpha)T_1$$

Speedup:

$$S_P = \frac{T_1}{T_P} \leq \alpha + P(1-\alpha)$$



Amdahl's Law vs Gustafson-Barsis' Law

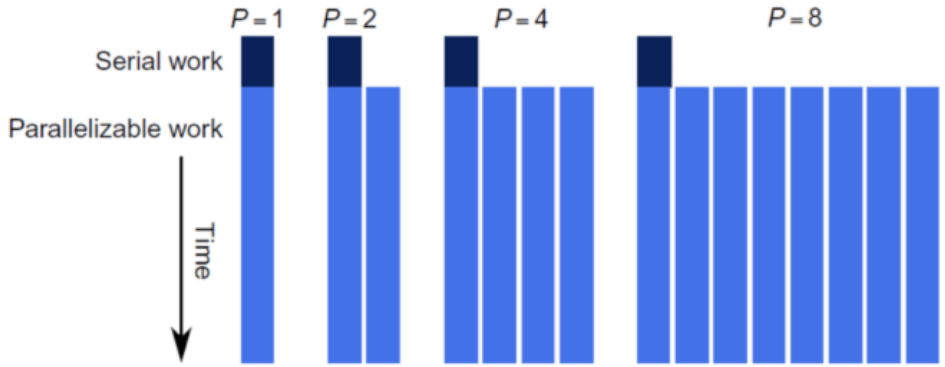
> *…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.*
> — John Gustafson

Time of sequential program with α as the fraction not affected by the parallelization on P-processors machine:

$$T_1 = \alpha T_1 + (1-\alpha)PT_1$$

Time of parallel program:

$$T_P = \alpha T_1 + (1-\alpha)T_1$$



Speedup:

$$S_P = \frac{T_1}{T_P} \leq \alpha + P(1-\alpha)$$

Note: no parallel overheads are taken into account here!

Quiz

- **Speedup**
- **Efficiency**
- **Strong Scaling**
- **Weak Scaling**

Quiz

- **Speedup**
  - How well something responds to adding more resources
  - **What's your base case?** The best serial version or a single parallel process?
- **Efficiency**
- **Strong Scaling**
- **Weak Scaling**

Quiz

- **Speedup**
  - How well something responds to adding more resources
  - **What's your base case?** The best serial version or a single parallel process?

- **Efficiency**

- **Strong Scaling**

- **Weak Scaling**

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost?. In Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems (HOTOS'15), George Candea (Ed.). USENIX Association, Berkeley, CA, USA, 14-14.

Quiz

- **Speedup**
  - How well something responds to adding more resources
  - **What's your base case?** The best serial version or a single parallel process?

- **Efficiency**
  - Gives idea on the "utilization" degree of the computing resources

- **Strong Scaling**

- **Weak Scaling**

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost?. In Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems (HOTOS'15), George Candea (Ed.). USENIX Association, Berkeley, CA, USA, 14-14.

Quiz

- **Speedup**
  - How well something responds to adding more resources
  - **What's your base case?** The best serial version or a single parallel process?

- **Efficiency**
  - Gives idea on the "utilization" degree of the computing resources

- **Strong Scaling**
  - Problem size stays fixed as the number of processing elements are increased

- **Weak Scaling**

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost?. In Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems (HOTOS'15), George Candea (Ed.). USENIX Association, Berkeley, CA, USA, 14-14.

Quiz

- **Speedup**
  - How well something responds to adding more resources
  - **What's your base case?** The best serial version or a single parallel process?

- **Efficiency**
  - Gives idea on the "utilization" degree of the computing resources

- **Strong Scaling**
  - Problem size stays fixed as the number of processing elements are increased

- **Weak Scaling**
  - Problem size increases as the number of processing elements are increased

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost?. In Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems (HOTOS'15), George Candea (Ed.). USENIX Association, Berkeley, CA, USA, 14-14.

Exercise 1

Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of a Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

Exercise 1

Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of a Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

Amdahl's law assumes that a program consists of a serial part and a parallelizable part. The fraction of the program which is serial can be denoted as B — so the parallel fraction becomes $1 - B$. If there is no additional overhead due to parallelization, the speedup can therefore be expressed as

$$S(n) = \frac{1}{B + \frac{1}{n}(1 - B)}$$

For the given value of $B = 0.01$ we get $S(61) = 38.125$.

Exercise 2

Assume 0.1% of the runtime is not parallelizable. The program also invokes a broadcast operation, that add overhead depending on the number of cores involved. There are two broadcast implementations available. One adds a parallel overhead of $0.0001n$, the other one $0.0005 \log n$. For which number of cores do you get the highest speedup for both implementations?

Exercise 2

Assume 0.1% of the runtime is not parallelizable. The program also invokes a broadcast operation, that add overhead depending on the number of cores involved. There are two broadcast implementations available. One adds a parallel overhead of $0.0001n$, the other one $0.0005 \log n$. For which number of cores do you get the highest speedup for both implementations?

$$S_1(n) = \frac{1}{0.001 + \frac{1}{n}0.999 + 0.0001n}$$

$$S_2(n) = \frac{1}{0.001 + \frac{1}{n}0.999 + 0.0005log(n)}$$

Exercise 2

Assume 0.1% of the runtime is not parallelizable. The program also invokes a broadcast operation, that add overhead depending on the number of cores involved. There are two broadcast implementations available. One adds a parallel overhead of $0.0001n$, the other one $0.0005 \log n$. For which number of cores do you get the highest speedup for both implementations?

$$S_1(n) = \frac{1}{0.001 + \frac{1}{n}0.999 + 0.0001n}$$

$$S_2(n) = \frac{1}{0.001 + \frac{1}{n}0.999 + 0.0005log(n)}$$

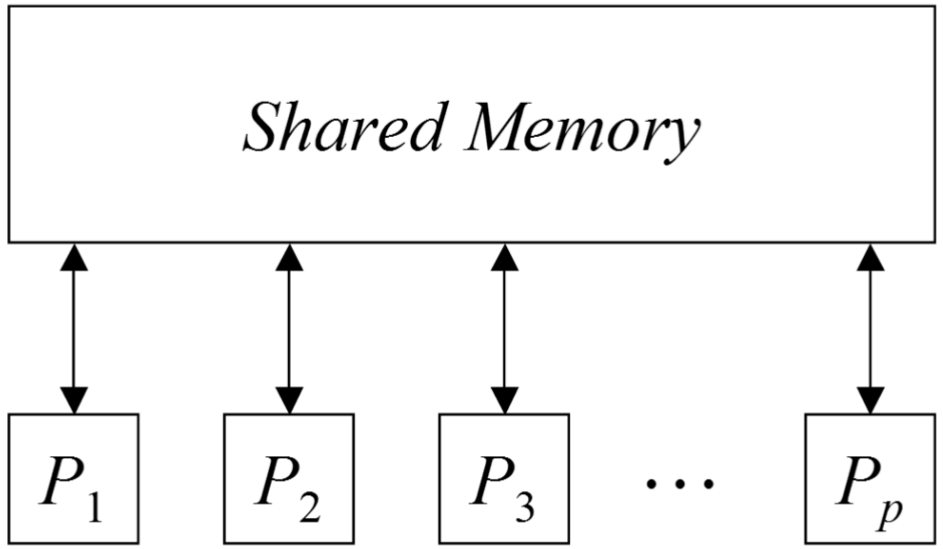
We can get the maximum of these terms if we minimize the term in denominator.

$$\frac{d}{dn}0.001 + \frac{1}{n}0.999 + 0.0001n = 0 \leftrightarrow 0.0001 - \frac{0.999}{n^2} = 0 \leftrightarrow n \approx 100$$

$$\frac{d}{dn}0.001 + \frac{1}{n}0.999 + 0.0005log(n) = 0 \leftrightarrow \frac{0.005n0.999}{n^2} = 0 \leftrightarrow n = 1998$$

PRAM: Parallel Random Access Machine

- **P processes with shared memory**
- **Ignores communications and synchronization**
- **Instruction are composed by 3 phases:**
  - Load data from shared memory (if needed)
  - Perform computation (if any)
  - Store data in shared memory (if needed)
- **Any process can read/write to any memory cell**
  - How conflicts are handled?



PRAM: Conflicting Accesses

- **EREW: Exclusive Read / Exclusive Write**
  - No two processes are allowed to read or write to the same memory cell simultaneously
- **CREW: Concurrent Read / Exclusive Write**
  - Simultaneous reads are allowed; only one process can write
- **CRCW: Concurrent Read / Concurrent Write**
  - Simultaneous reads and write to the same memory cell are allowed
  - Priority CRCW: processors assigned fixed distinct priorities, highest priority wins
  - Random CRCW: one randomly chosen write wins
  - Common CRCW: all processors are allowed to complete write if and only if all the values to be written are equal

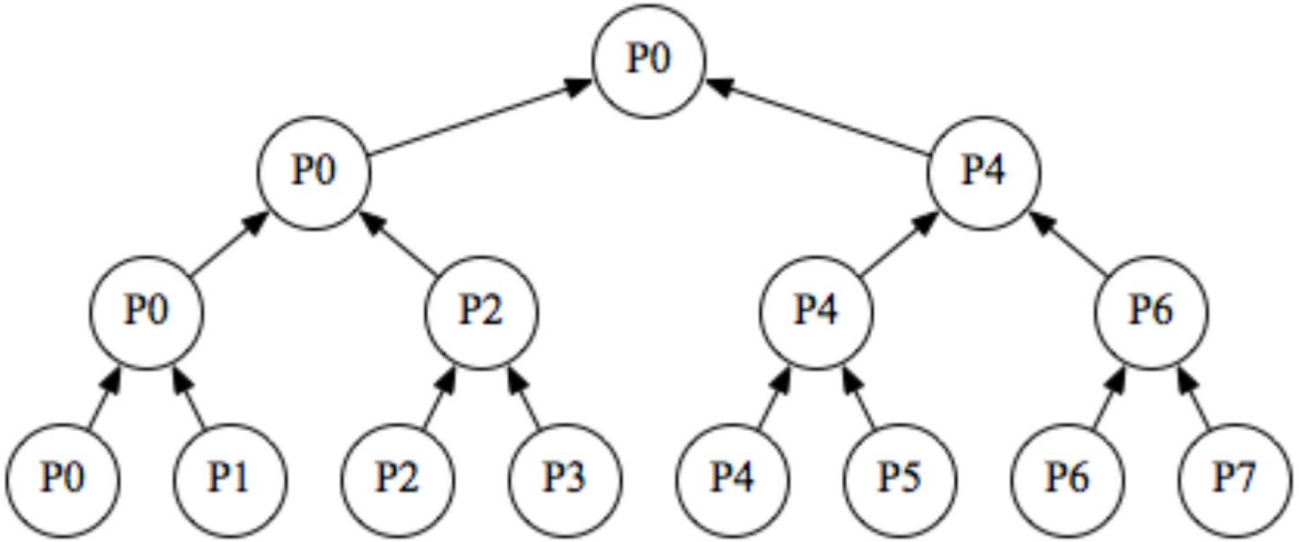
Weak **Strong**

EREW < CREW < CRCW-C < CRCW-R < CRCW-P

http://homes.cs.washington.edu/~arvind/cs424/notes/l2-6.pdf

PRAM: Reduction

- Reduce p values on the p-processor EREW PRAM in $O(logp)$ time
- The algorithm uses exclusive reads and writes
- It's the basis of other EREW algorithms

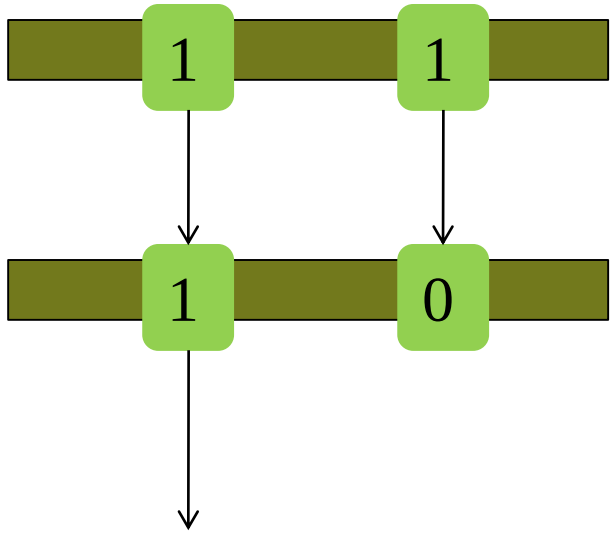


PRAM: First 1

- **Computing the position of the first one in the sequence of 0's and 1's in a constant time.**

Algorithm A
(2 parallel steps and n^2 processors)

```
for each 1≤ i<j ≤ n do in parallel
   if C[i] =1 and C[j]=1 then C[j]:=0
for each 1≤ i ≤ n do in parallel
   if C[i] =1 then FIRST-ONE-POSITION:=i
```

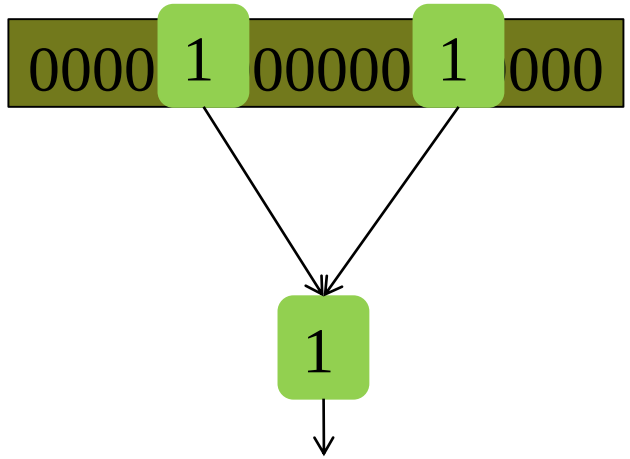


www.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_5.ppt

PRAM: First 1 – Reducing Number of Processors

Algorithm B: it reports if there is any one in the table.

```
There-is-one:=0
for each 1≤ i ≤ n do in parallel
   if C[i] =1 then There-is-one:=1
```

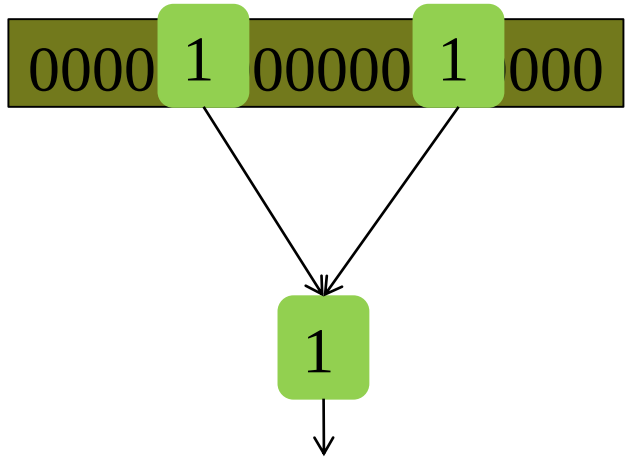


www.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_5.ppt

PRAM: First 1 – Reducing Number of Processors

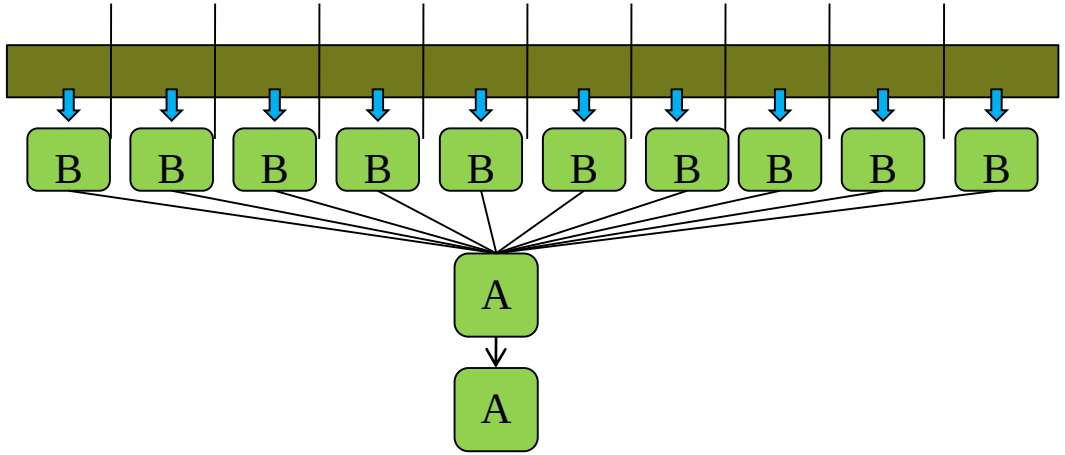
Algorithm B: it reports if there is any one in the table.

```
There-is-one:=0
for each 1≤ i ≤ n do in parallel
   if C[i] =1 then There-is-one:=1
```



Merge A and B

1. Partition table C into segments of size $\sqrt{n}$
2. In each segment apply the algorithm B
3. Find position of the first one in these sequence by applying algorithm A
4. Apply algorithm A to this single segment and compute the final value



www.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_5.ppt

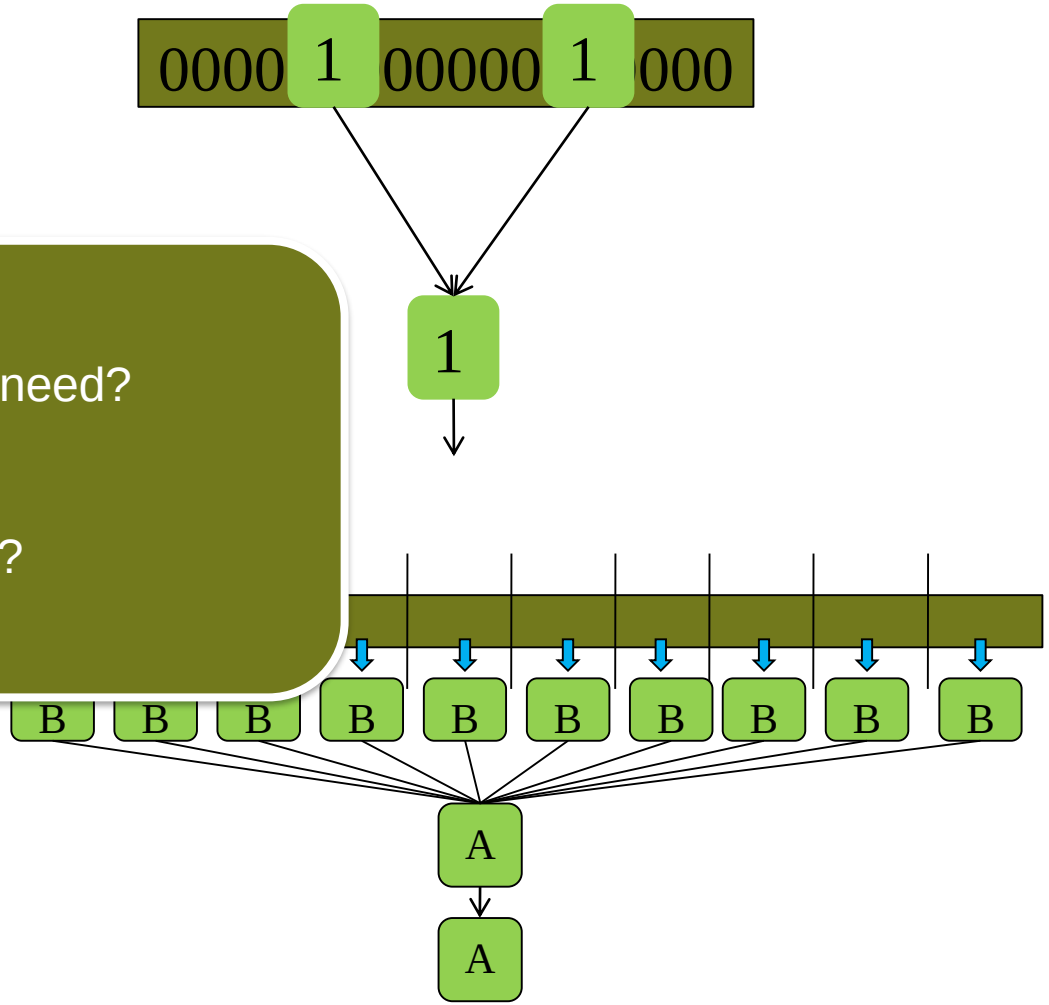
PRAM: First 1 – Reducing Number of Processors

Algorithm B: it reports if there is any one in the table.

```
There-is-one:=0
for each 1≤ i ≤ n do in
   if C[i] =1 then The
```

How many processors we need?

What's the complexity?



Merge A and B

1. Partition table C into se
2. In each segment apply t
3. Find position of the first one in these sequence by applying algorithm A
4. Apply algorithm A to this single segment and compute the final value

www.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_5.ppt

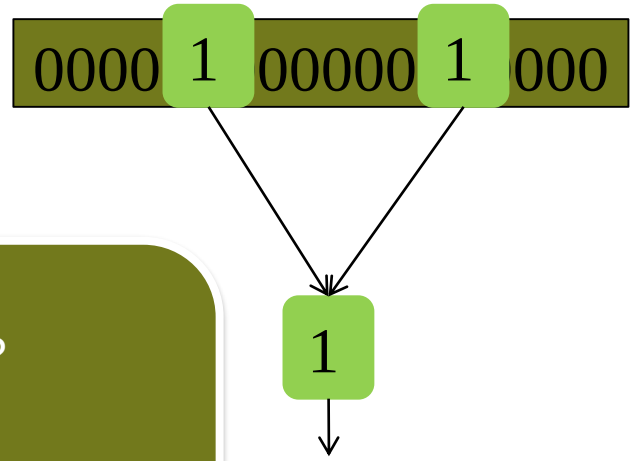
PRAM: First 1 – Reducing Number of Processors

Algorithm B: it reports if there is any one in the table.

```
There-is-one:=0
for each 1≤ i ≤ n do in
   if C[i] =1 then The
```

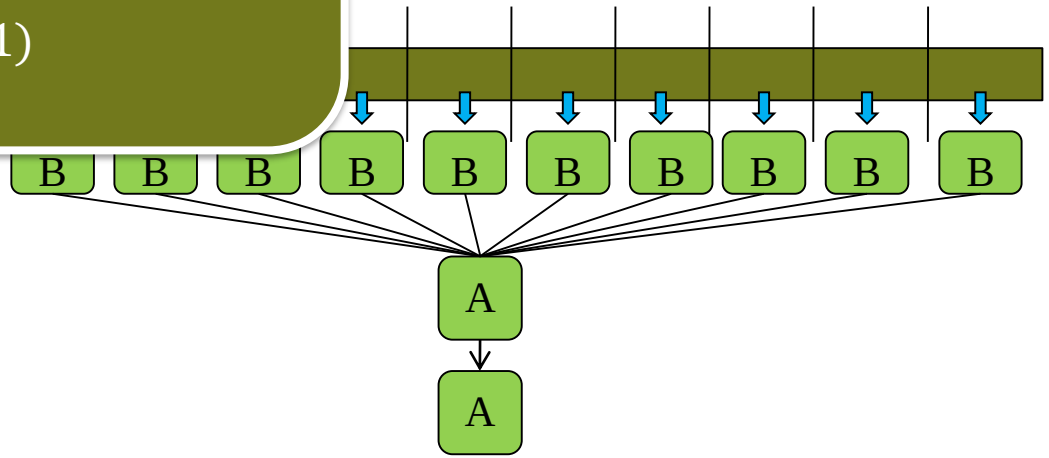
How many processors we need?
$(\sqrt{n})^2 = n$

What's the complexity?
$3\ parallel\ steps \rightarrow O(1)$



Merge A and B

1. Partition table C into se
2. In each segment apply t
3. Find position of the first one in these sequence by applying algorithm A
4. Apply algorithm A to this single segment and compute the final value



www.csc.liv.ac.uk/~igor/COMP308/ppt/Lect_5.ppt

Exercise 3

How can we find the minimum from an unordered collection of n natural numbers on EREW-PRAM machine?

Exercise 3

How can we find the minimum from an unordered collection of n natural numbers on EREW-PRAM machine?

We can find the minimum from an unordered collection of n natural numbers by performing a reduction along a binary tree: In each round, each processor compares two elements, and the smaller element gets to the next round, the bigger one is discarded. What is the work and depth of this algorithm?

Exercise 3

How can we find the minimum from an unordered collection of n natural numbers on EREW-PRAM machine?

We can find the minimum from an unordered collection of n natural numbers by performing a reduction along a binary tree: In each round, each processor compares two elements, and the smaller element gets to the next round, the bigger one is discarded. What is the work and depth of this algorithm?

The dependency graph of this computation is a tree with $log_2(n)$ levels. Therefore the longest path, which is equal to the depth/span has length $log_2(n)$. The tree contains $2n - 1$ nodes, which is equal to the work.

Exercise 4

Develop an algorithm which can find the minimum in an unordered collection of n natural numbers in $O(1)$ time on a CRCW-PRAM machine.

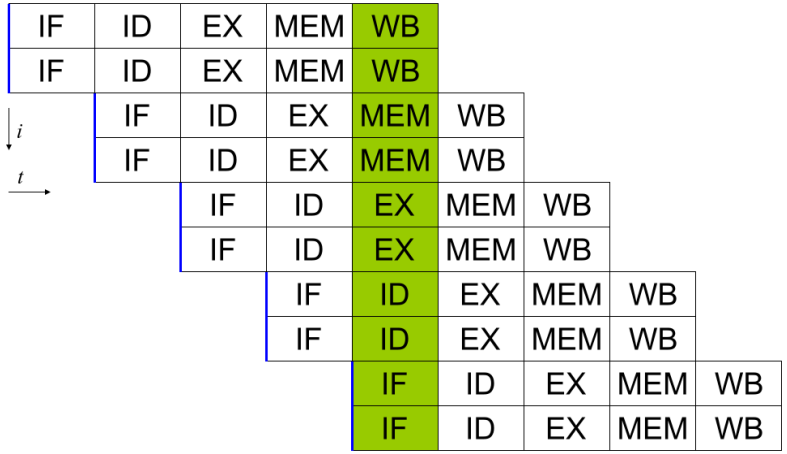
Exercise 4

Develop an algorithm which can find the minimum in an unordered collection of n natural numbers in $O(1)$ time on a CRCW-PRAM machine.

- Assume the list is stored in an array A.
- Create an additional array $tmp[n]$ initialized with $true$.
- We use $O(n^2)$ processors, labelled $p(i,j)$ with $0 \leq i,j \leq n$.
- Each processor $p(i,j)$ checks if $A[i] > A[j]$.
  - If true then $tmp[i]$ is set to false (it cannot be the minimum)
  - Otherwise nothing is done
- At the end we have only one element of tmp set to true, say $tmp[k]$. The minimum element of A is $A[k]$.

Computation

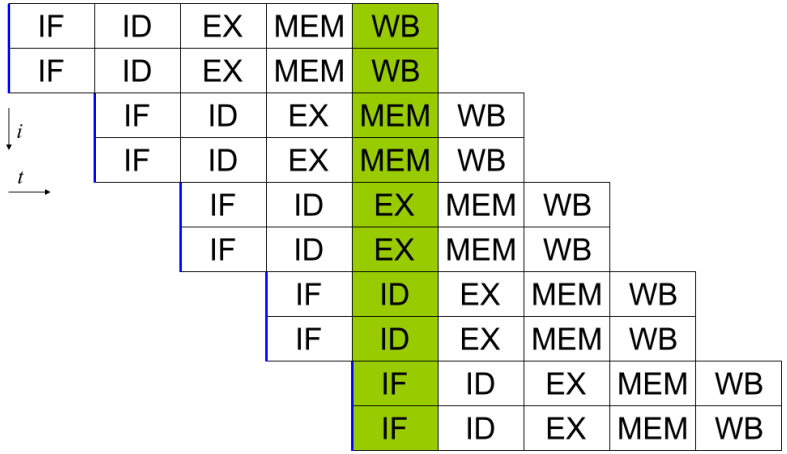
- **Usually, floating point performance (Gflop/s) is the metric of interest**
- **Road to peak in-core performance:**



Instruction Level Parallelism (ILP)

Computation

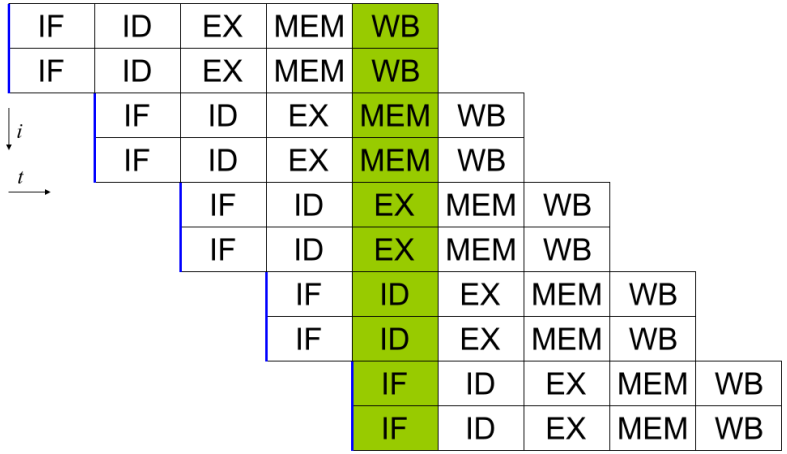
- **Usually, floating point performance (Gflop/s) is the metric of interest**
- **Road to peak in-core performance:**
  - ***Improve ILP and apply SIMD***



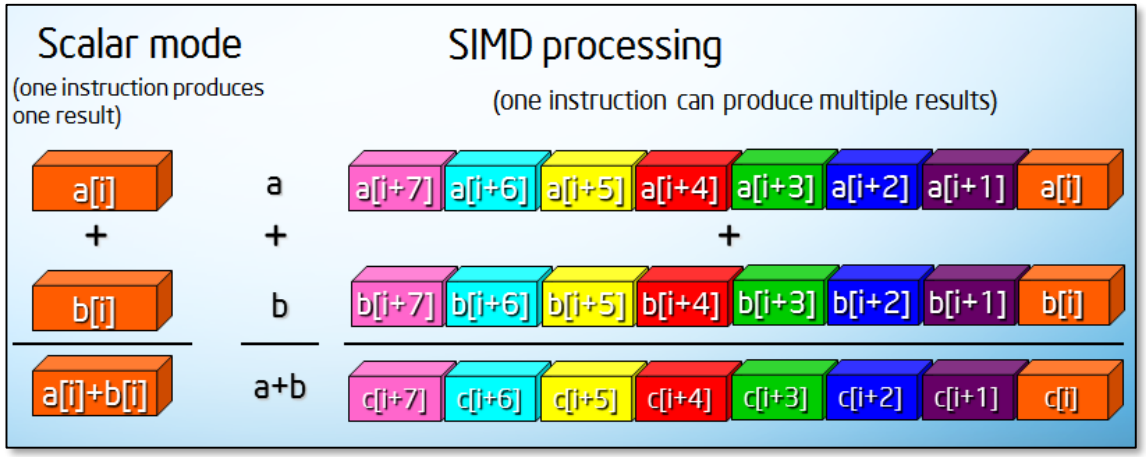
Instruction Level Parallelism (ILP)

Computation

- **Usually, floating point performance (Gflop/s) is the metric of interest**
- **Road to peak in-core performance:**
  - ***Improve ILP and apply SIMD***



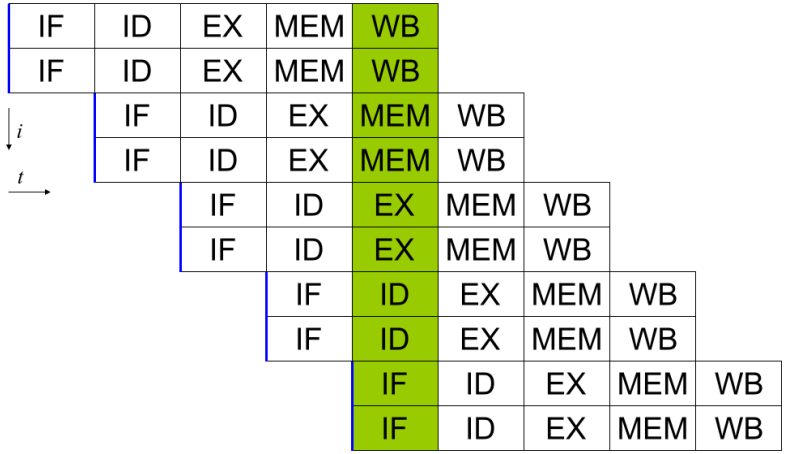
Instruction Level Parallelism (ILP)



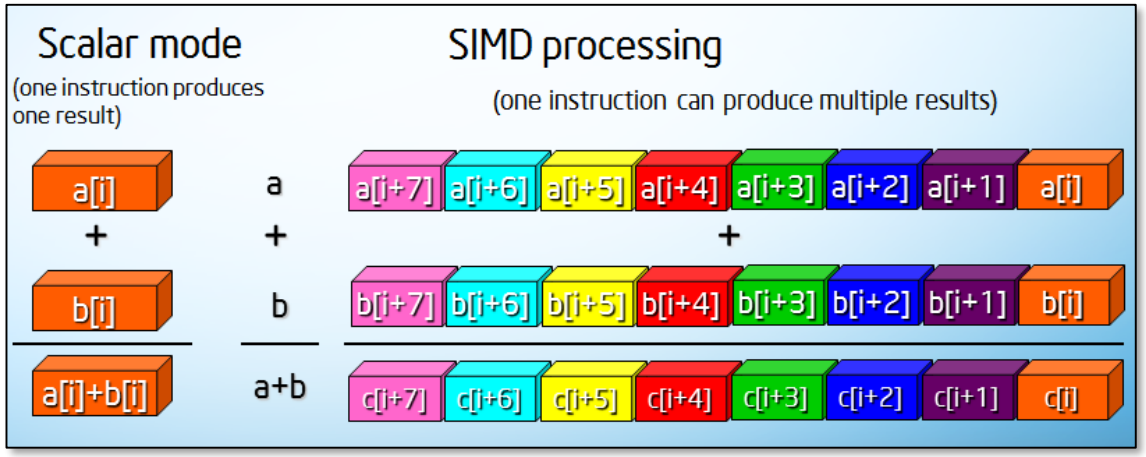
Single Instruction Multiple Data (SIMD)

Computation

- **Usually, floating point performance (Gflop/s) is the metric of interest**
- **Road to peak in-core performance:**
  - ***Improve ILP and apply SIMD***



Instruction Level Parallelism (ILP)



Single Instruction Multiple Data (SIMD)

- ***Balance floating-point operation mix:*** equal number of additions and multiplications

  Hardware may have Fused Multiple-Add instructions (FMA) or equal number of adders/multipliers

Communication

- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

Communication

- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

- **Restructure loops for unit stride accesses**
  - Engages the hardware prefetcher

Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

Communication

- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

- **Restructure loops for unit stride accesses**
  - Engages the hardware prefetcher

- **Ensure memory affinity**
  - E.g., two multicore chips with local memory controller

Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

Communication

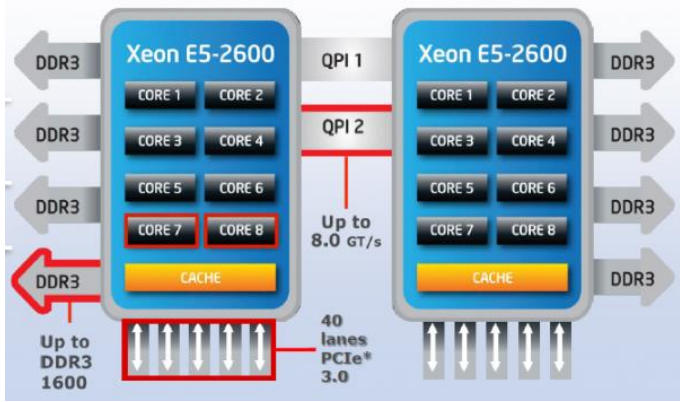
- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

- **Restructure loops for unit stride accesses**
  - Engages the hardware prefetcher

- **Ensure memory affinity**
  - E.g., two multicore chips with local memory controller



Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

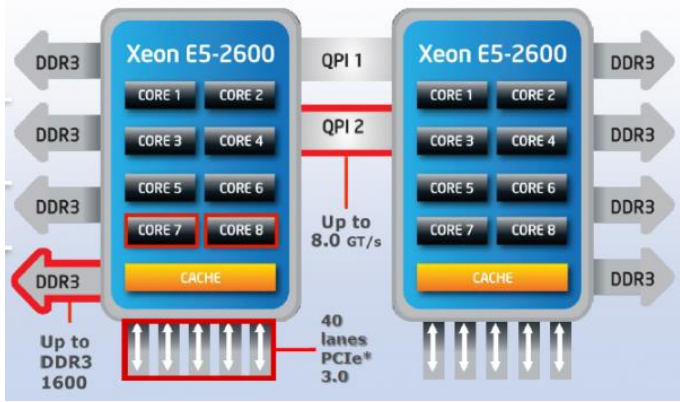
Communication

- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

- **Restructure loops for unit stride accesses**
  - Engages the hardware prefetcher



- **Ensure memory affinity**
  - E.g., two multicore chips with local memory controller

- **Use software prefetching**
  - Depending on the architecture, HW prefetcher can take time (e.g., 5 loads) to start prefetching
  - SW prefetching can provide speedups for complex access patterns

Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

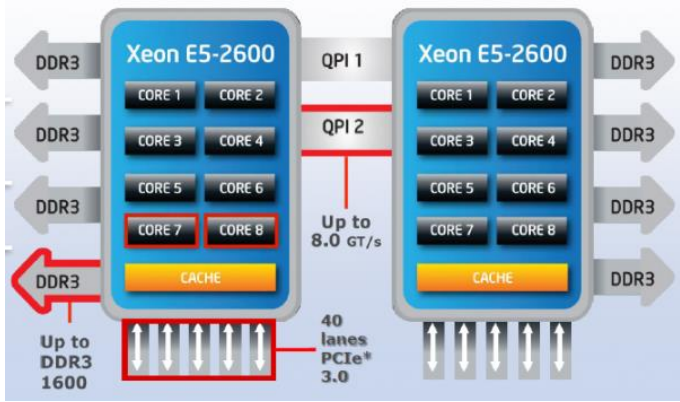
Communication

- **DRAM bandwidth (GB/s) is the metric of interest**

```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[i][j] = a[i][j] + c[i][j] * d;
```

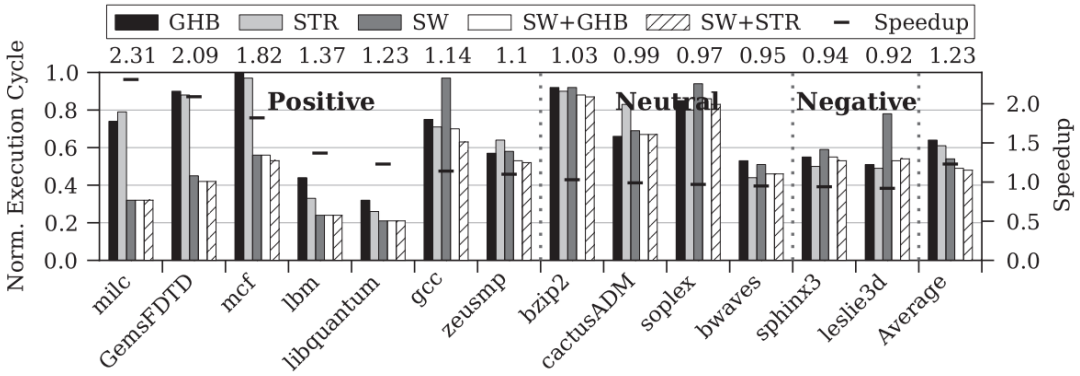
```
for (i=0; i<n; i++)
   for (j=0; j<n; j++)
      a[j][i] = a[j][i] + c[j][i] * d;
```

- **Restructure loops for unit stride accesses**
  - Engages the hardware prefetcher



- **Ensure memory affinity**
  - E.g., two multicore chips with local memory controller

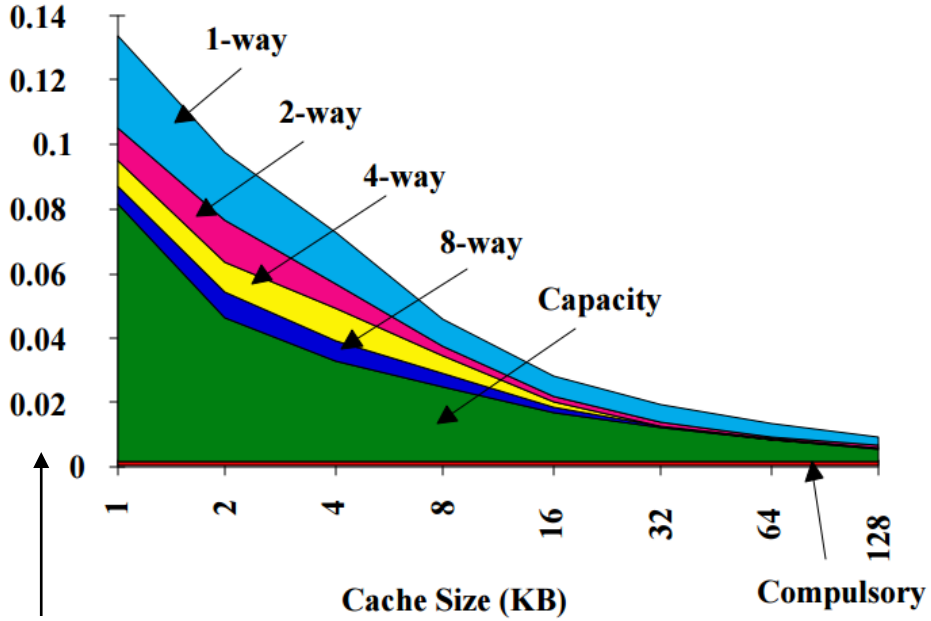
- **Use software prefetching**
  - Depending on the architecture, HW prefetcher can take time (e.g., 5 loads) to start prefetching
  - SW prefetching can provide speedups for complex access patterns



Lee, Jaekyu, Hyesoon Kim, and Richard Vuduc. "When prefetching works, when it doesn't, and why." ACM Transactions on Architecture and Code Optimization (TACO) 9.1 (2012): 2.

Locality

- **3Cs Model**
  - **Compulsory:** *On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.*
  - **Capacity:** *Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).*
  - **Conflict:** *In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.*



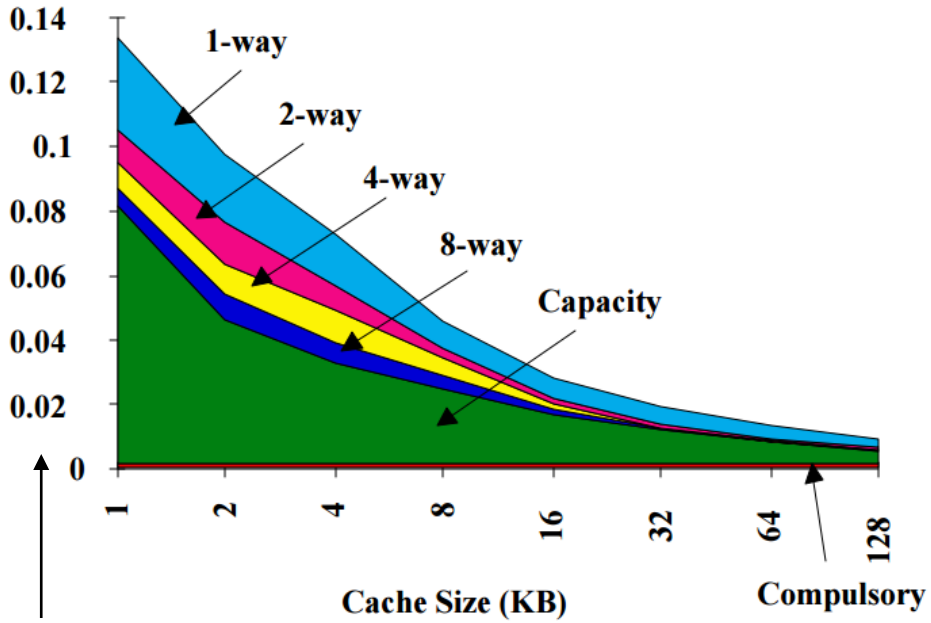
Absolut Miss Rates
on SPEC92

http://meseec.ce.rit.edu/eecc551-winter2001/551-1-30-2002.pdf

Locality

- **3Cs Model**
  - **Compulsory:** *On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.*
  - **Capacity:** *Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).*
  - **Conflict:** *In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.*

What is the lower bound to the number of memory operations?



Absolut Miss Rates
on SPEC92

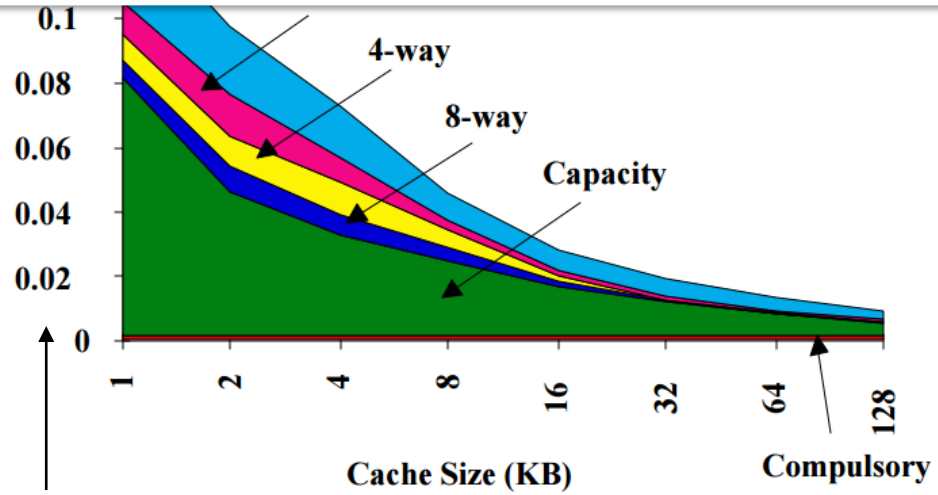
http://meseec.ce.rit.edu/eecc551-winter2001/551-1-30-2002.pdf

Locality

- **3Cs Model**
  - **Compulsory:** *On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.*
  - **Capacity:** *Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).*
  - **Conflict:** *In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.*

What is the lower bound to the number of memory operations?

How to lower capacity misses?



Absolut Miss Rates
on SPEC92

http://meseec.ce.rit.edu/eecc551-winter2001/551-1-30-2002.pdf

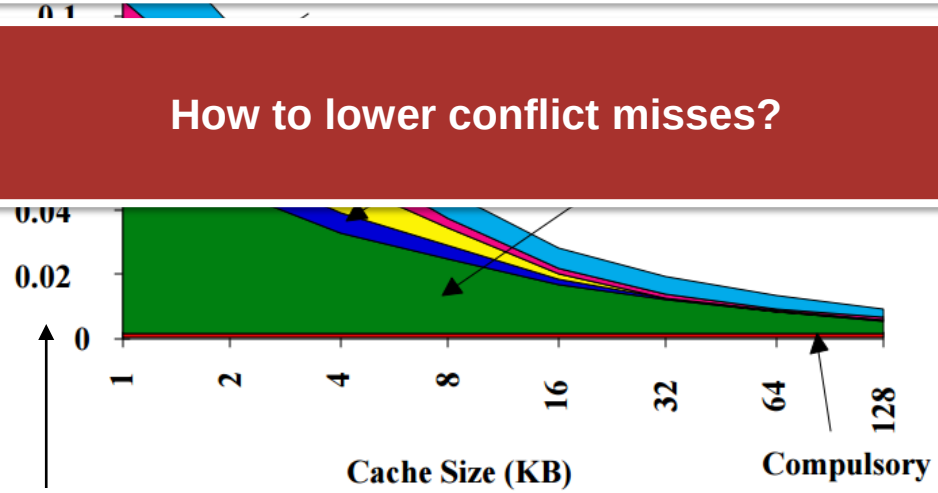
Locality

- **3Cs Model**
  - **Compulsory:** *On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.*
  - **Capacity:** *Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).*
  - **Conflict:** *In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.*

What is the lower bound to the number of memory operations?

How to lower capacity misses?

How to lower conflict misses?



Absolut Miss Rates
on SPEC92

http://meseec.ce.rit.edu/eecc551-winter2001/551-1-30-2002.pdf

Locality

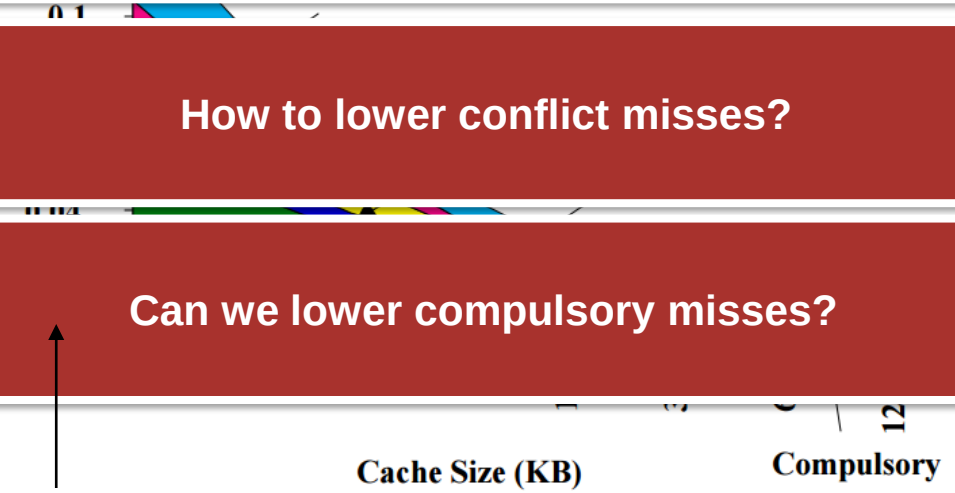
- **3Cs Model**
  - **Compulsory:** *On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.*
  - **Capacity:** *Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).*
  - **Conflict:** *In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.*

What is the lower bound to the number of memory operations?

How to lower capacity misses?

How to lower conflict misses?

Can we lower compulsory misses?



Absolut Miss Rates
on SPEC92

http://meseec.ce.rit.edu/eecc551-winter2001/551-1-30-2002.pdf

How to Improve Locality?

- **Merging Arrays**

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
   int val;
   int key;
};
struct merge merged_array[SIZE];
```

- **Loop Interchange**
- **Loop Fusion**
- **Blocking or “tiling”**

How to Improve Locality?

- **Merging Arrays**

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
   int val;
   int key;
};
struct merge merged_array[SIZE];
```

- Reduce conflicts between key and val
- Improve spatial locality

- **Loop Interchange**
- **Loop Fusion**
- **Blocking or “tiling”**

How to Improve Locality?

- **Merging Arrays**
- **Loop Interchange**

```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

- **Loop Fusion**
- **Blocking or “tiling”**

How to Improve Locality?

- **Merging Arrays**
- **Loop Interchange**

```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
       for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

Improves spatial locality: sequential access instead of striding through memory every 100 words

- **Loop Fusion**
- **Blocking or "tiling"**

How to Improve Locality?

- **Merging Arrays**
- **Loop Interchange**
- **Loop Fusion**

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {   a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

- **Blocking or “tiling”**

How to Improve Locality?

- **Merging Arrays**
- **Loop Interchange**
- **Loop Fusion**

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {   a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

- From two missies per access to a & c to one miss per access
- Improve temporal locality

- **Blocking or "tiling"**

How to Improve Locality?

- **Merging Arrays**
- **Loop Interchange**
- **Loop Fusion**
- **Blocking or “tiling”**
  - Example: matrix multiplication
  - Goal: reduce the working set

Compute/Memory Bound

- **What do we mean by “compute bound”?**
- **What to we mean by “memory bound”?**

Compute/Memory Bound

- **What do we mean by “compute bound”?**
  - It has high operations intensity
- **What to we mean by “memory bound”?**

Compute/Memory Bound

- **What do we mean by “compute bound”?**
  - It has high operations intensity
- **What to we mean by “memory bound”?**
  - It has low operational intensity

Compute/Memory Bound

- **What do we mean by “compute bound”?**
  - It has high operations intensity
- **What to we mean by “memory bound”?**
  - It has low operational intensity

- **They’re not very precise definitions…**

Compute/Memory Bound

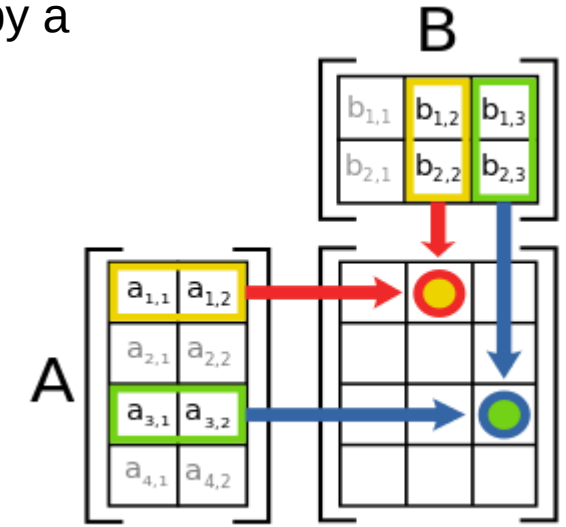
- **What do we mean by "compute bound"?**
  - It has high operations intensity
- **What to we mean by "memory bound"?**
  - It has low operational intensity

- **They're not very precise definitions…**
- **Roofline model helps to clarify**
  - Plots the performance (GFlops/second) as a function of the Operational Intensity (GFlops/byte)
  - What's Operational Intensity?

Operational Intensity

How many Flops per byte does your code show?

- **Work:** W is the number of operations performed by a given program
- **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

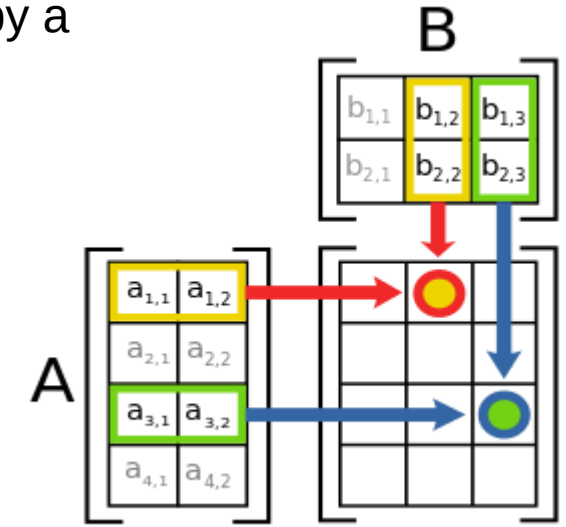


Operational Intensity

How many Flops per byte does your code show?

- **Work:** W is the number of operations performed by a given program
- **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**



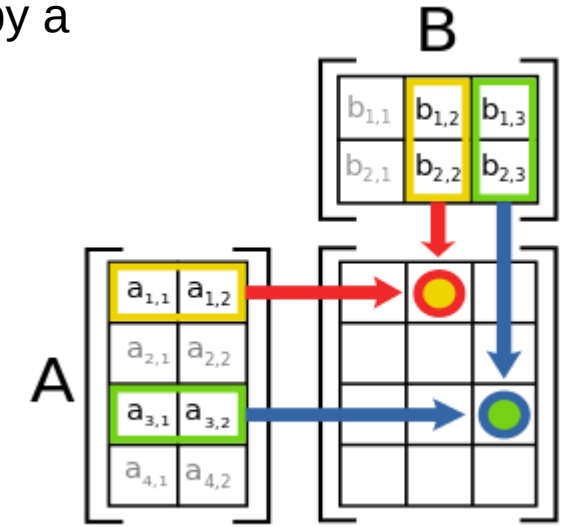
Operational Intensity

How many Flops per byte does your code show?

- **Work:** W is the number of operations performed by a given program
- **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**
  - For some kernels, OI is a function of the input size

    e.g., dense matrix multiplication



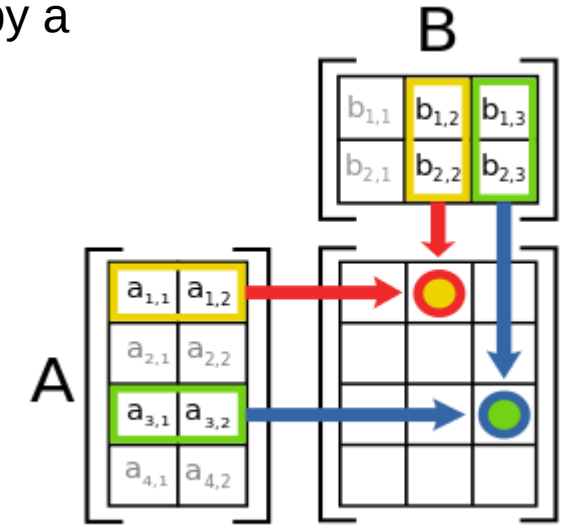
Operational Intensity

How many Flops per byte does your code show?

- **Work:** W is the number of operations performed by a given program
- **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**
  - For some kernels, OI is a function of the input size

    e.g., dense matrix multiplication
  - What else?



Operational Intensity

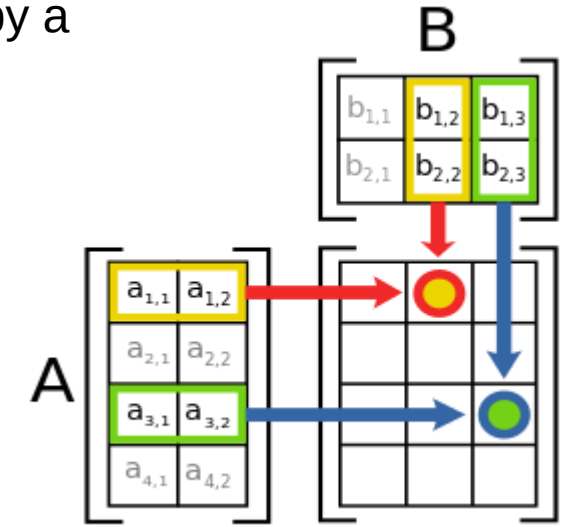
How many Flops per byte does your code show?

- **Work:** W is the number of operations performed by a given program
- **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**
  - For some kernels, OI is a function of the input size

    e.g., dense matrix multiplication
  - What else?

    Improve locality



Operational Intensity

- **How many Flops per byte does your code show?**
  - **Work:** W is the number of operations performed by a given program
  - **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**
  - For some kernels, OI is a function of the input size

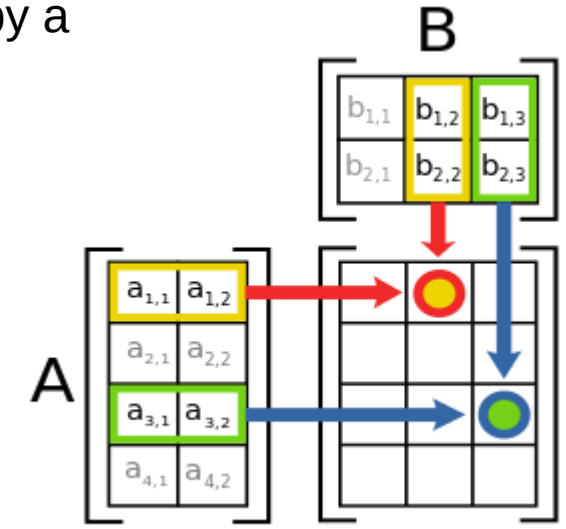
    e.g., dense matrix multiplication
  - What else?

    Improve locality

  - **Example:** matrix multiplication (3 nested loops)

$$W(n) = \sim n^3$$
$$Q(n) = n^2$$

$$I(n) = \frac{W(n)}{Q(n)} = \sim n$$



Operational Intensity

- **How many Flops per byte does your code show?**
  - **Work:** W is the number of operations performed by a given program
  - **Memory Traffic:** Q is the number of bytes transferred from memory by a given program

- **Can you increase it?**
  - For some kernels, OI is a function of the input size

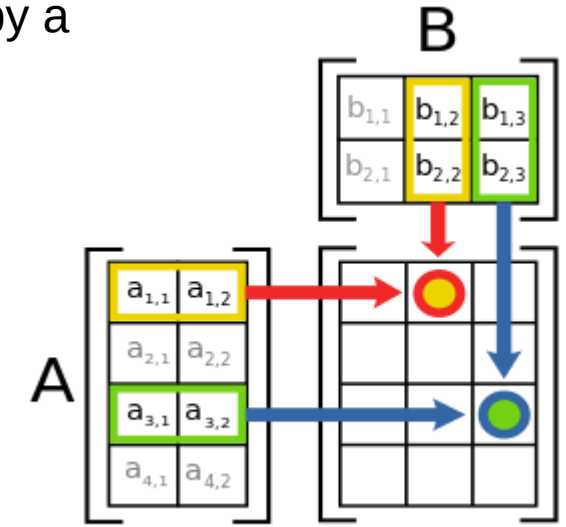
    e.g., dense matrix multiplication
  - What else?

    Improve locality

  - **Example:** matrix multiplication (3 nested loops)

$$W(n) = \sim n^3$$
$$Q(n) = n^2$$

$$I(n) = \frac{W(n)}{Q(n)} = \sim n$$



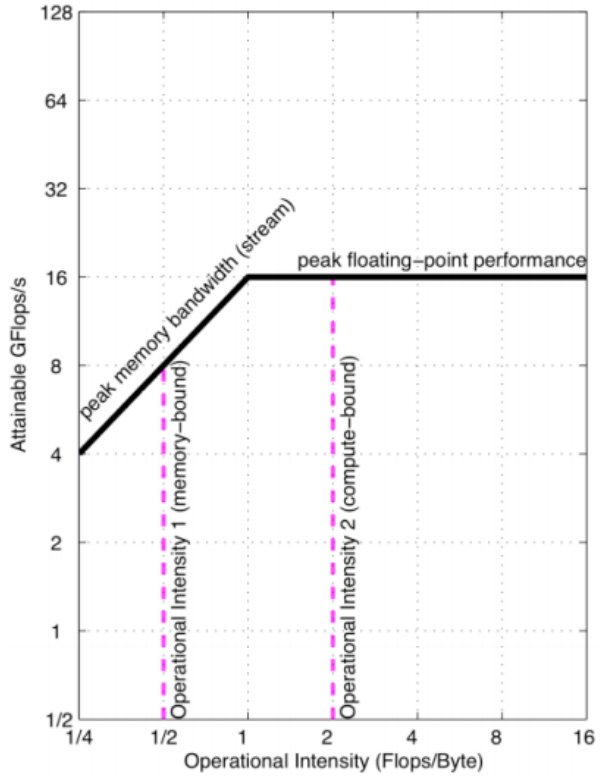
Measures the traffic between the caches and DRAM. But why?

Roofline Model

Attainable GFlops/sec = Min(Peak Floating Point Performance,
Peak Memory Bandwidth x Operational Intensity)

- **A kernel with a given OI lies somewhere in the vertical line with x=OI**

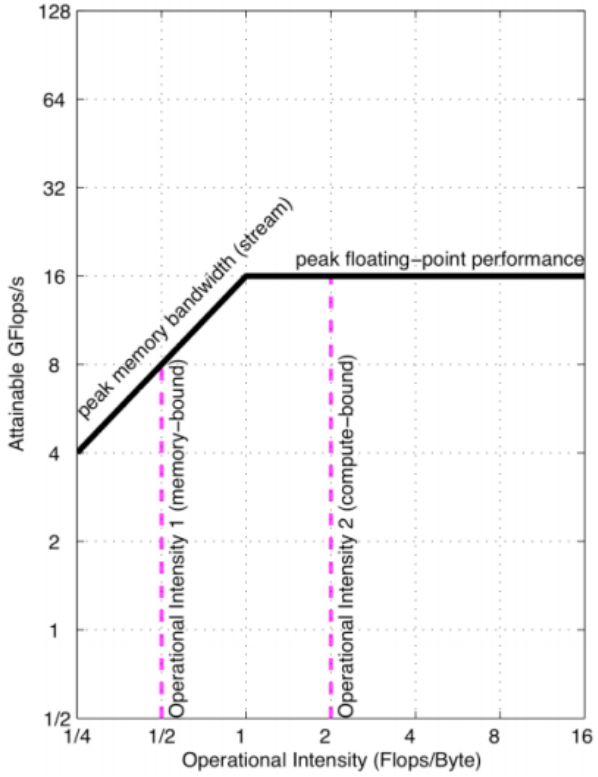
- **Ridge point:** intersection of the diagonal and horizontal roof
  - *Its x-coordinate is the minimum operational intensity required to achieve maximum performance*
  - *It suggests the level of difficulty for programmers and compiler writers to achieve peak performance*

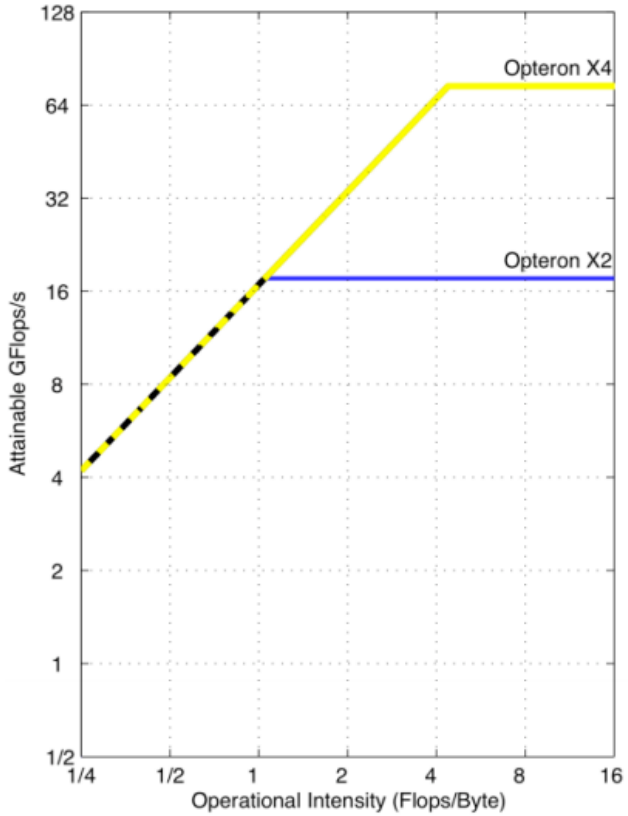


Roofline Model

Attainable GFlops/sec = Min(Peak Floating Point Performance,
Peak Memory Bandwidth x Operational Intensity)

- **A kernel with a given OI lies somewhere in the vertical line with x=OI**
- **Ridge point:** intersection of the diagonal and horizontal roof
  - *Its x-coordinate is the minimum operational intensity required to achieve maximum performance*
  - *It suggests the level of difficulty for programmers and compiler writers to achieve peak performance*





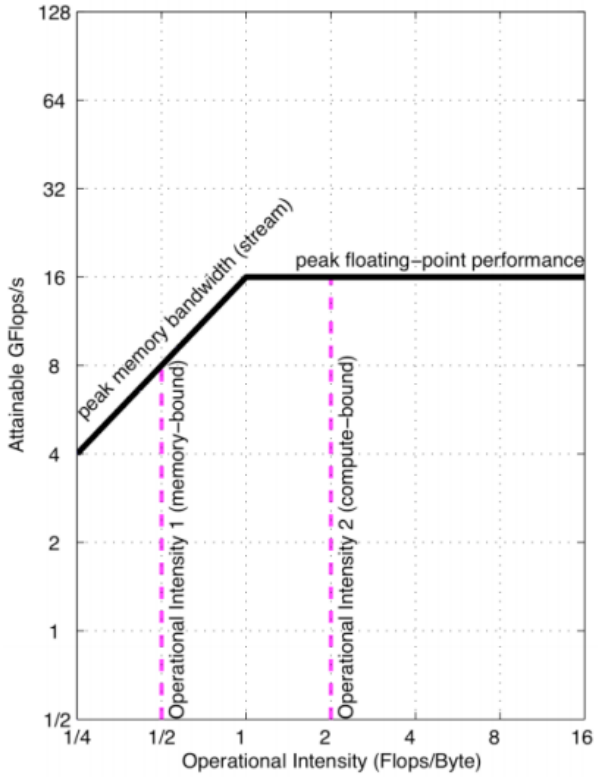
Opteron X4:

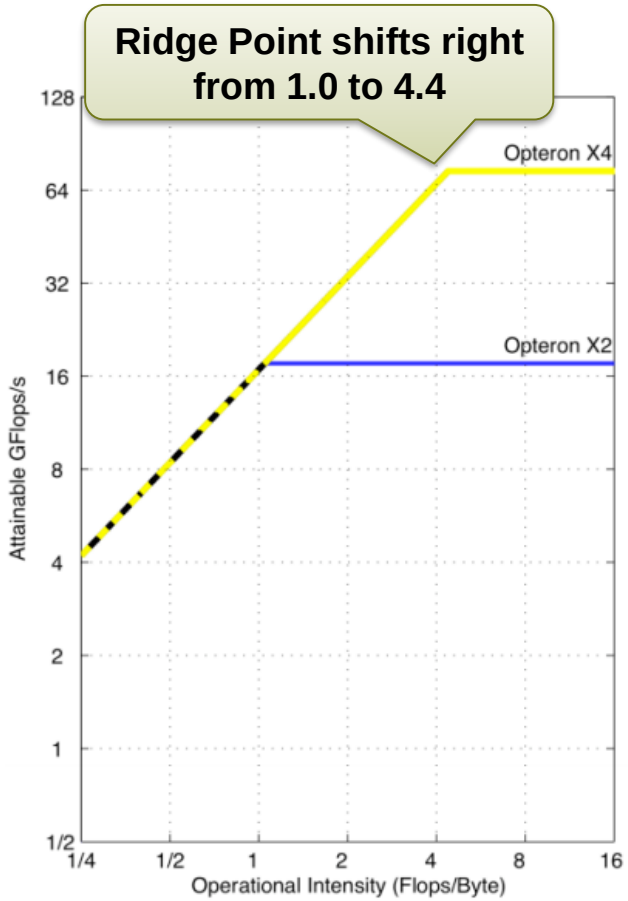
- Can issue 2 FP SSE2 instructions per cycle
- Slightly faster clock rate
- >4x gain in peak performance w.r.t. X2

Roofline Model

Attainable GFlops/sec = Min(Peak Floating Point Performance,
Peak Memory Bandwidth x Operational Intensity)

- **A kernel with a given OI lies somewhere in the vertical line with x=OI**
- **Ridge point:** intersection of the diagonal and horizontal roof
  - *Its x-coordinate is the minimum operational intensity required to achieve maximum performance*
  - *It suggests the level of difficulty for programmers and compiler writers to achieve peak performance*



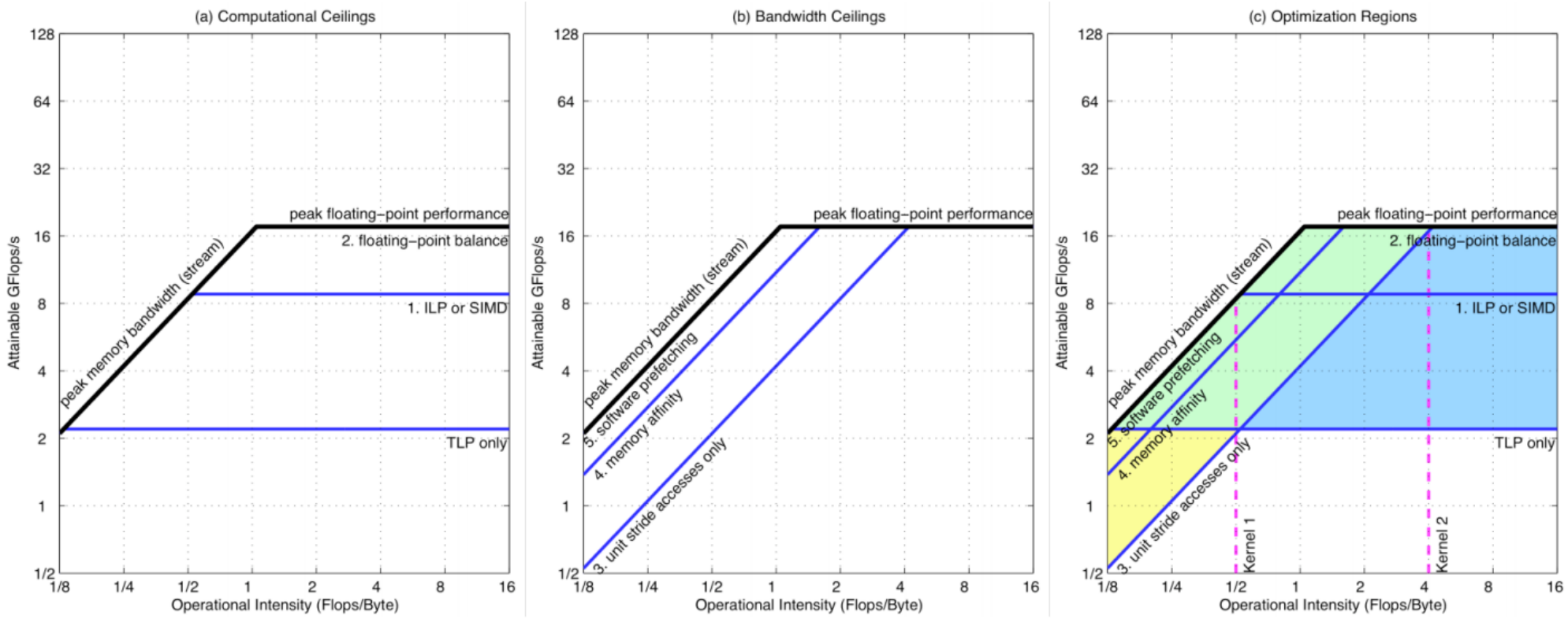


Opteron X4:

- Can issue 2 FP SSE2 instructions per cycle
- Slightly faster clock rate
- >4x gain in peak performance w.r.t. X2

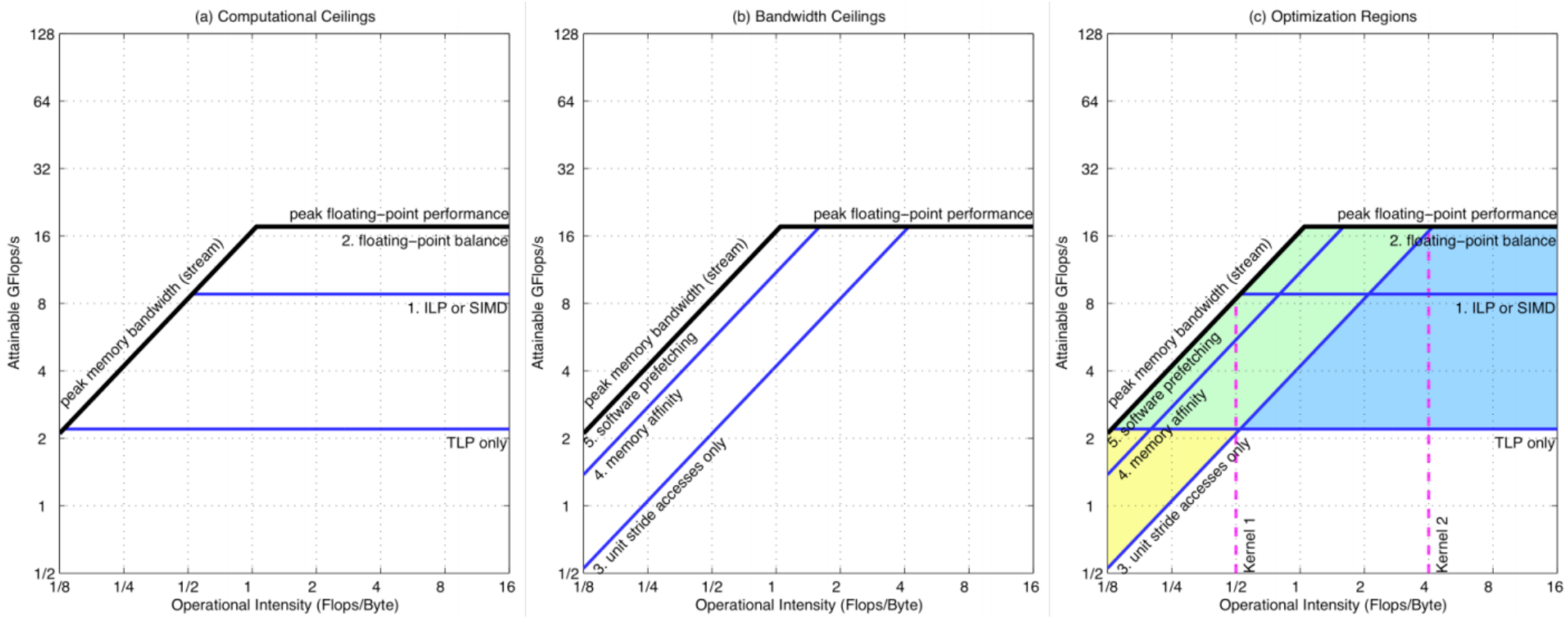
Adding Ceilings

- **What if your program is far from the roofline?**



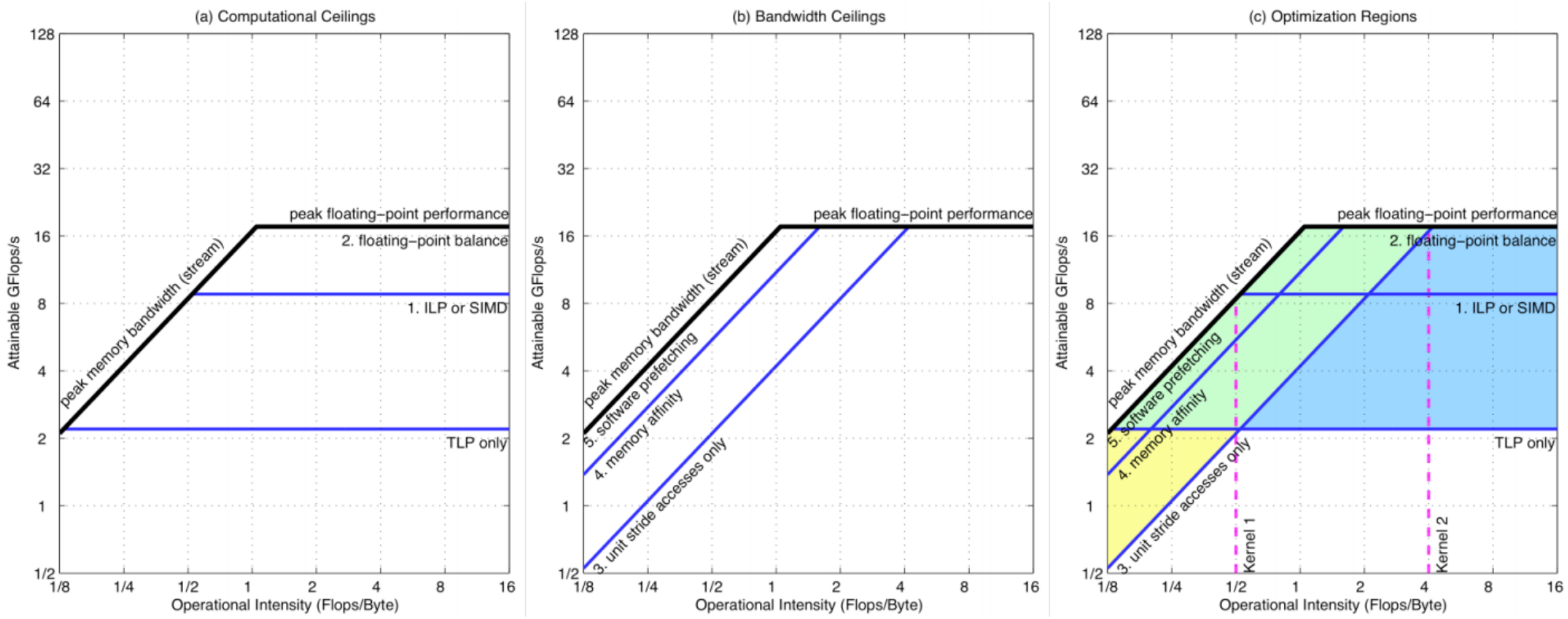
Adding Ceilings

- **What if your program is far from the roofline?**
  - Ceilings can help us: you cannot break through a ceiling without performing the associated optimization.



Adding Ceilings

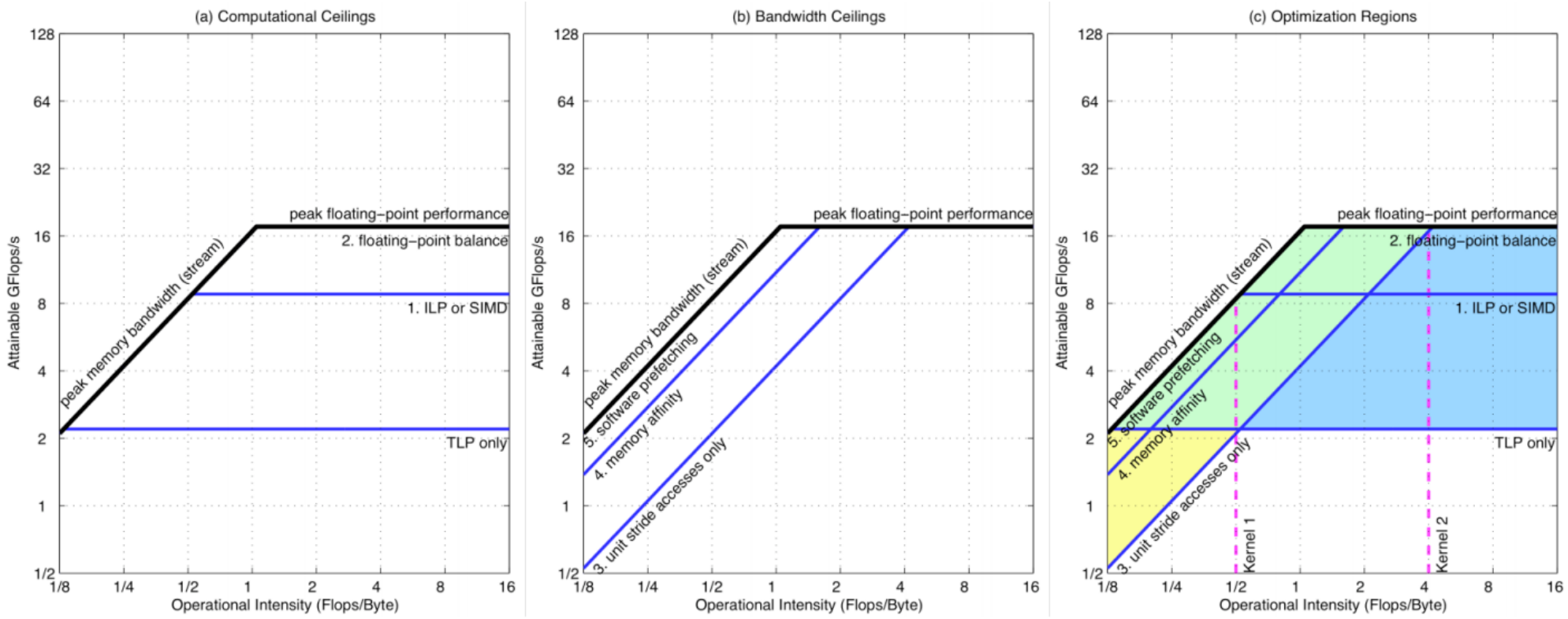
- **What if your program is far from the roofline?**
  - Ceilings can help us: you cannot break through a ceiling without performing the associated optimization.



  - The height of the gap between a ceiling and the next higher one is the potential reward for trying that optimization

Adding Ceilings

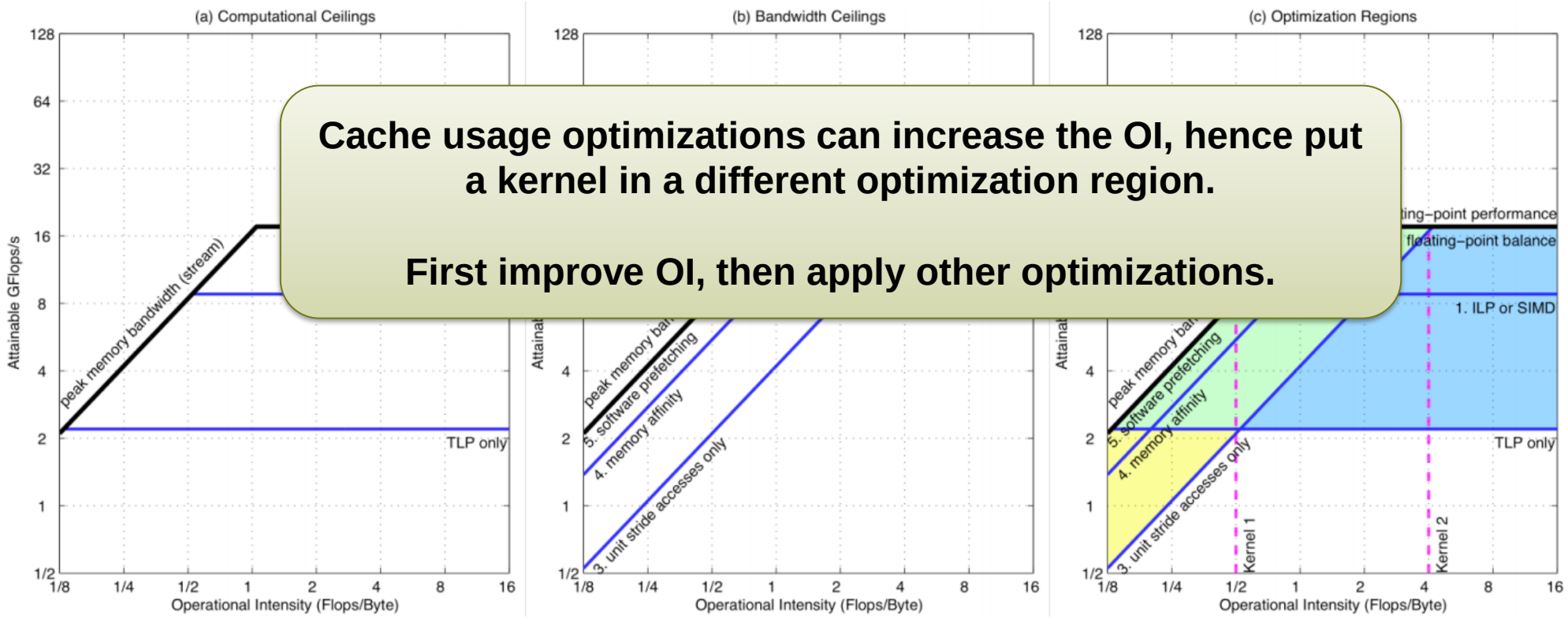
- **What if your program is far from the roofline?**
  - Ceilings can help us: you cannot break through a ceiling without performing the associated optimization.



  - The height of the gap between a ceiling and the next higher one is the potential reward for trying that optimization
  - Their order suggests the optimization order. Lower ceilings: easy to implement by the programmer or likely realized by the compiler.

Adding Ceilings

- **What if your program is far from the roofline?**
  - Ceilings can help us: you cannot break through a ceiling without performing the associated optimization.

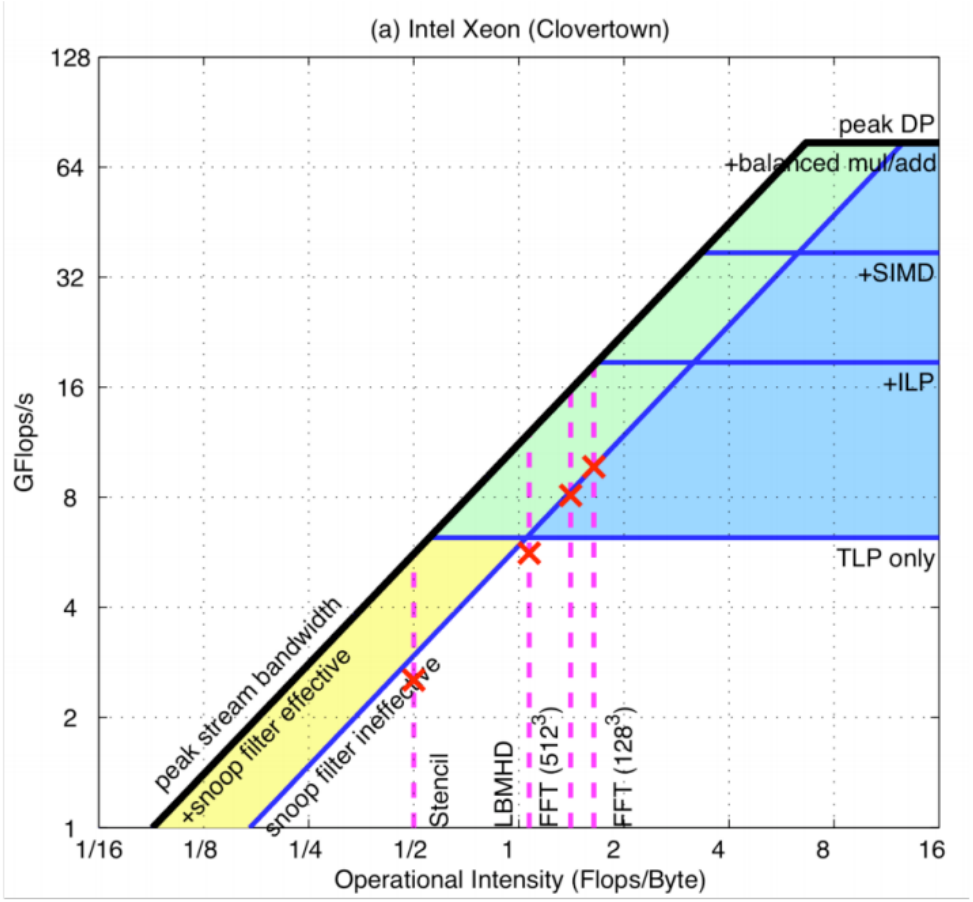


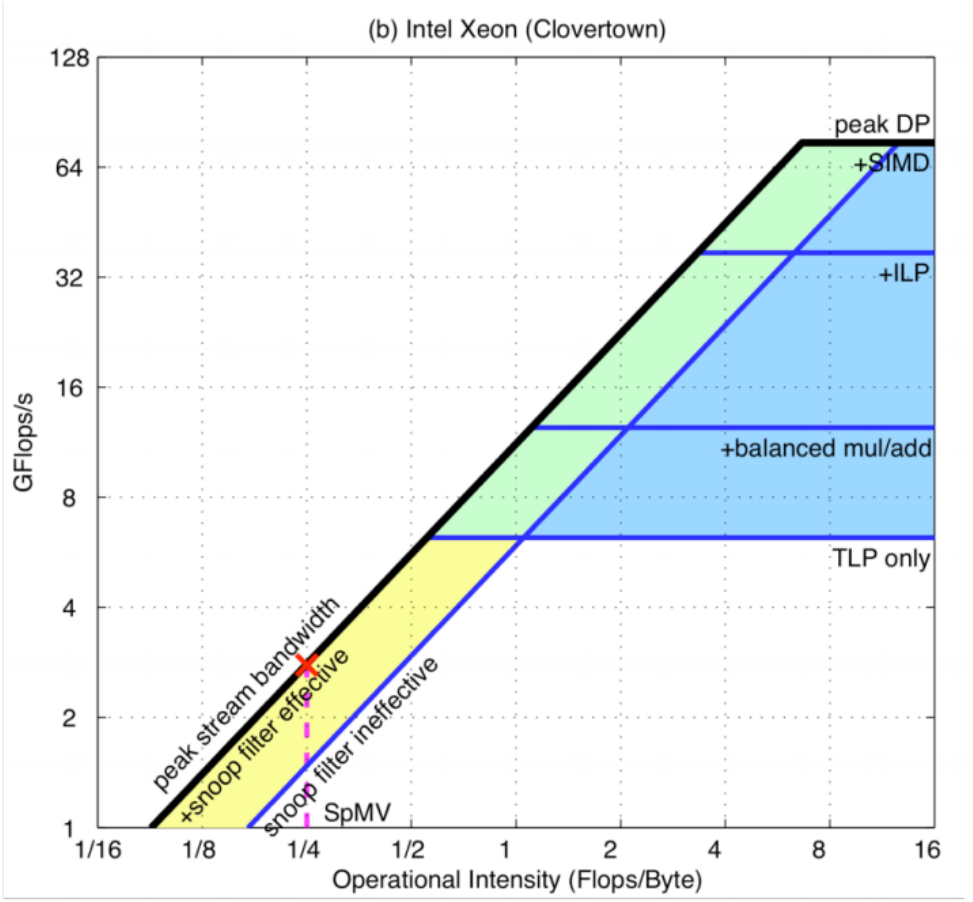
Cache usage optimizations can increase the OI, hence put a kernel in a different optimization region.

First improve OI, then apply other optimizations.

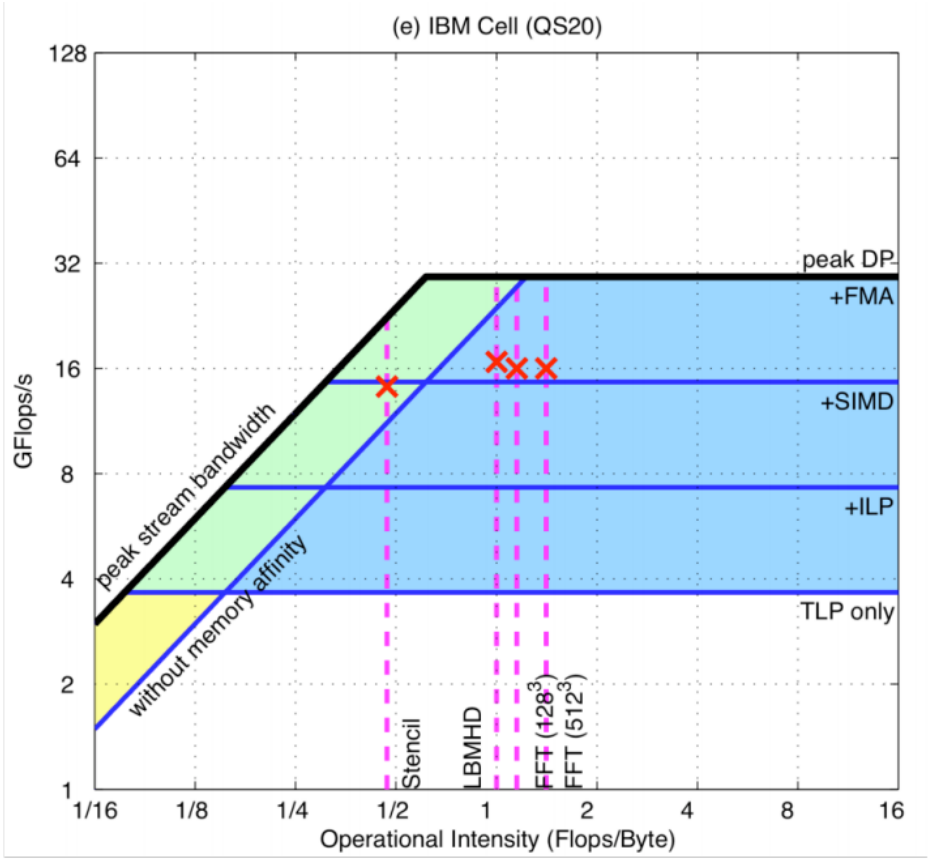
  - The height of the gap between a ceiling and the next higher one is the potential reward for trying that optimization
  - Their order suggests the optimization order. Lower ceilings: easy to implement by the programmer or likely realized by the compiler.

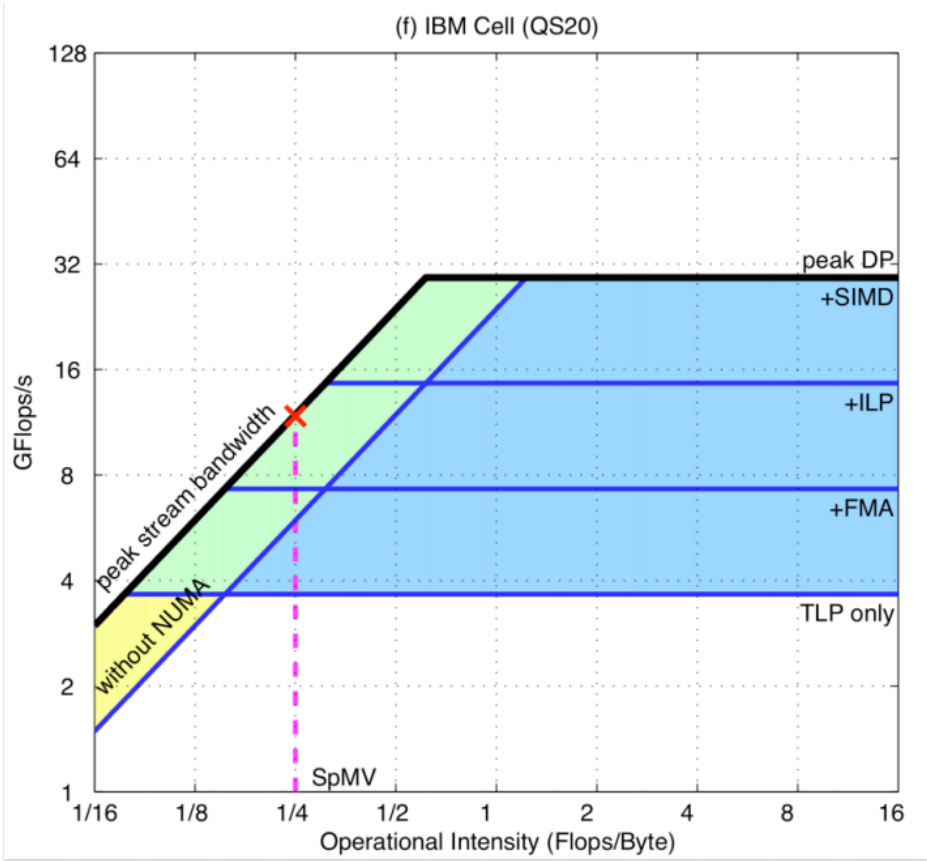
Models & Results



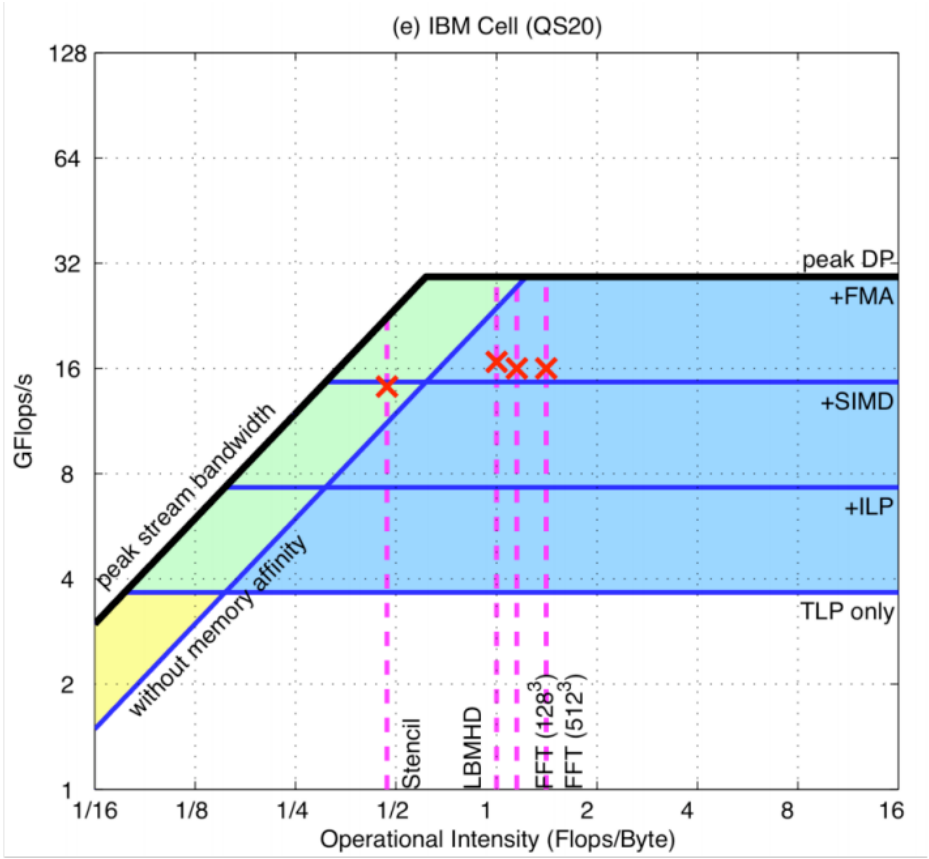


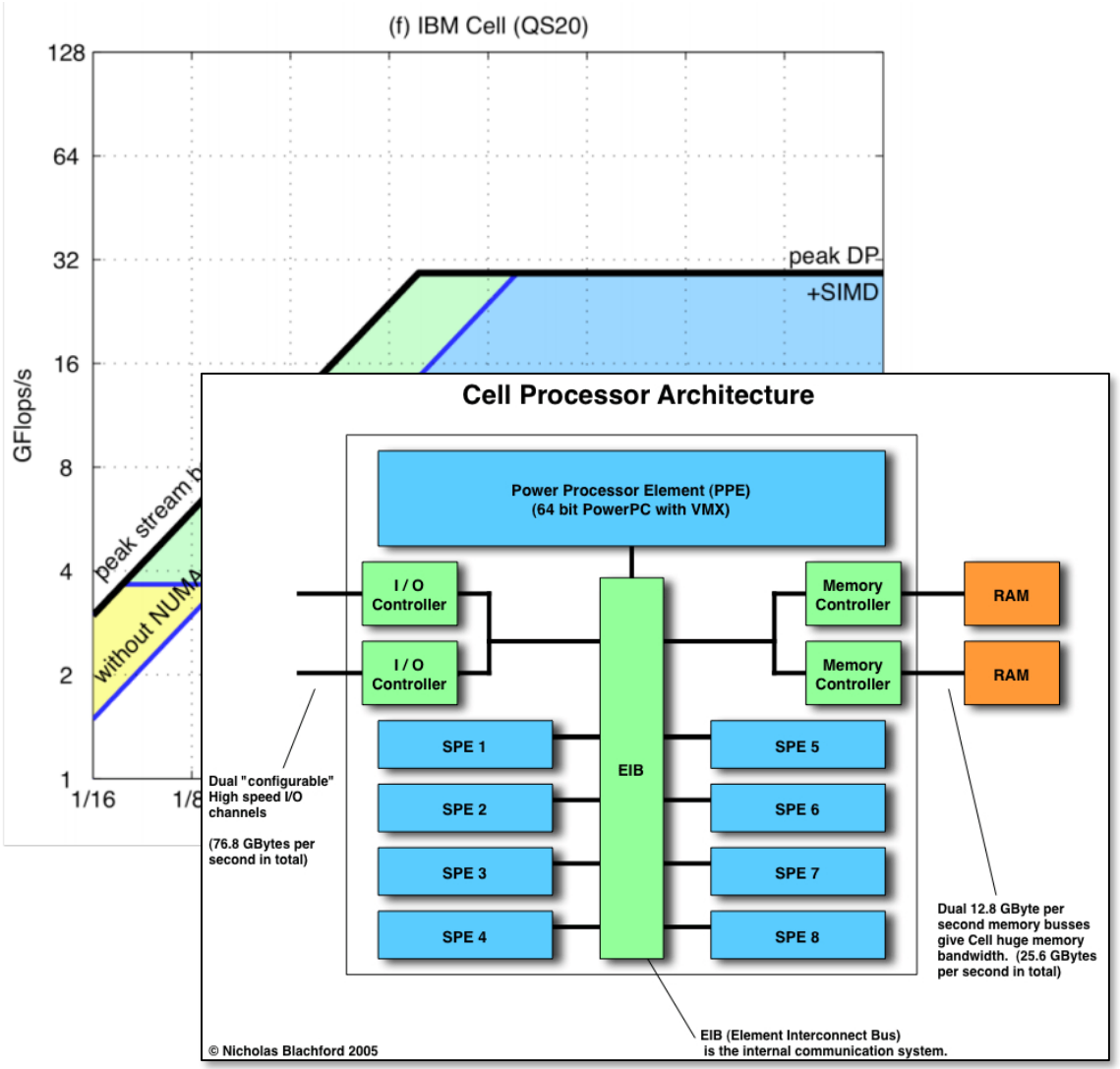
Models & Results



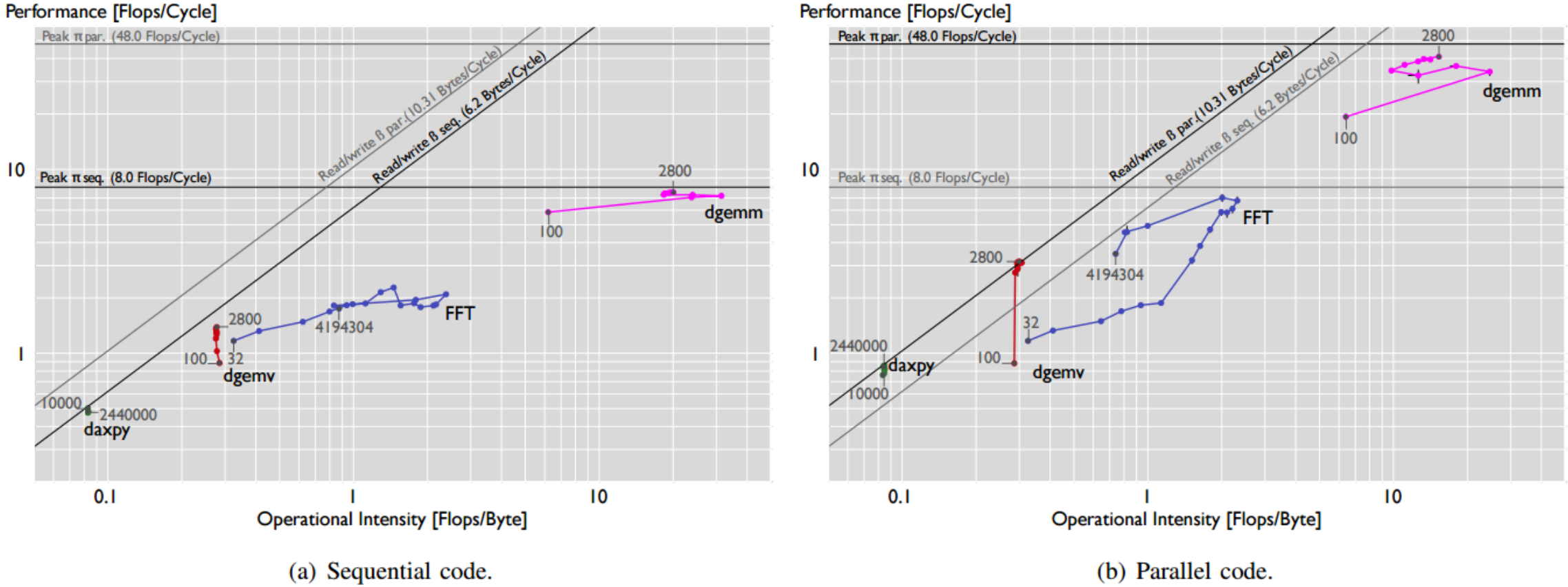


Models & Results





Multithreading



- **The ridge point shifts from 1.3 to 4.6**
- **Increasing the input makes parallelization gain efficiency**
  - Until when the working set gets too big to stay in cache

Ofenbeck, Georg, et al. "Applying the roofline model." Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014.

Applying the Roofline Model

- **For each kernel, we need to measure:**
  - **The work W**

    Counters for floating point operations

  - **The runtime T**

    Read Time Stamp Counter (RDTSC) is still a right choice

  - **The memory traffic Q**

    LLC misses can be an underestimation

    Measure raw traffic on the memory controller if possible (i.e.,Intel PCM)

- **For each architecture, we need to measure:**
  - The peak performance π: microbenchmarks or manual
  - The memory bandwidth β: microbenchmarks, most challenging

Ofenbeck, Georg, et al. "Applying the roofline model." Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014.

Applying the Roofline Model

- **For each kernel, we need to measure:**
  - **The work W**

    Counters for floating point operations

    $$W = \text{Scalar\_double} + \text{SSE\_double} \times 2 + \text{AVX\_double} \times 4$$

    E.g., W on a Sandy Bridge platform

  - **The runtime T**

    Read Time Stamp Counter (RDTSC) is still a right choice

  - **The memory traffic Q**

    LLC misses can be an underestimation

    Measure raw traffic on the memory controller if possible (i.e.,Intel PCM)

- **For each architecture, we need to measure:**
  - The peak performance π: microbenchmarks or manual
  - The memory bandwidth β: microbenchmarks, most challenging

Ofenbeck, Georg, et al. "Applying the roofline model." Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014.

Applying the Roofline Model

- **For each kernel, we need to measure:**
  - **The work W**

    Counters for floating point operations

$$W = \text{Scalar\_double} + \text{SSE\_double} \times 2 + \text{AVX\_double} \times 4$$

E.g., W on a Sandy Bridge platform

  - **The runtime T**

    Read Time Stamp Counter (RDTSC) is still a right choice

  - **The memory traffic Q**

    LLC misses can be an underestimation

    Measure raw traffic on the memory controller if possible (i.e.,Intel PCM)

- **For each architecture, we need to measure:**
  - The peak performance π: microbenchmarks or manual
  - The memory bandwidth β: microbenchmarks, most challenging

LibLSB: https://spcl.inf.ethz.ch/Research/Performance/LibLSB/

Ofenbeck, Georg, et al. "Applying the roofline model." Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014.