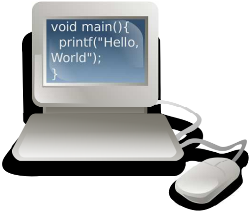
Locality-conscious parallel algorithms

Nodari Sitchinava
Karlsruhe Institute of Technology
University of Hawaii

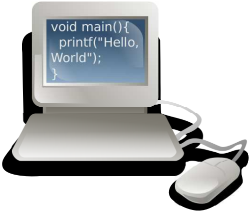
October 17, 2013

1990s: Faster Programs

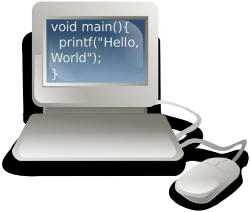
1990s: Faster Programs

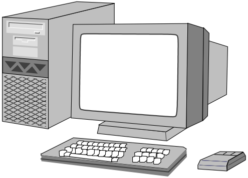


1990s: Faster Programs

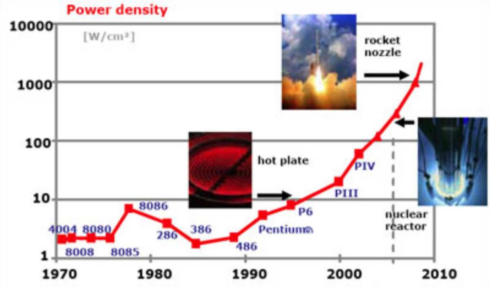


1990s: Faster Programs

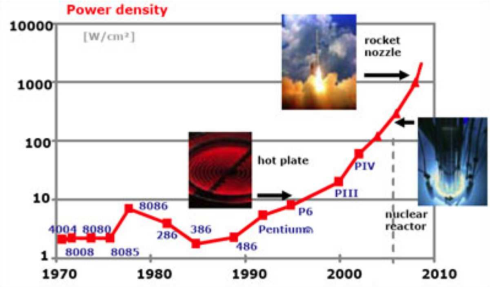


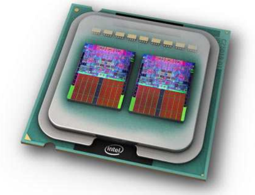


Mid 2000s – Power Dissipation Is a Problem



Mid 2000s – Power Dissipation Is a Problem

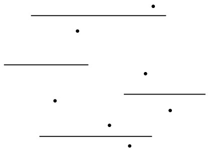




Parallel computing

- 1970-90s: PRAM, BSP, hypercubes...
- Important results on the limits of parallelism
- Are they practical?

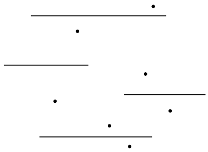
Example: Orthogonal Point Location



Runtime

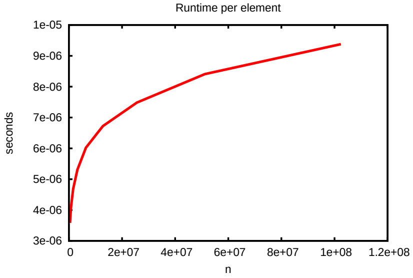
$$\Theta(n \log n)$$

Example: Orthogonal Point Location

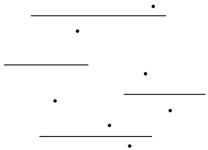


Runtime

$$\Theta(n \log n)$$

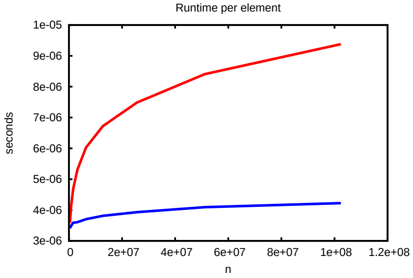


Example: Orthogonal Point Location



Runtime

$$\Theta(n \log n)$$



Θ notation

What does Θ notation represent?

- Number of comparisons
- Number of arithmetic operations
- Number of memory accesses

Θ notation

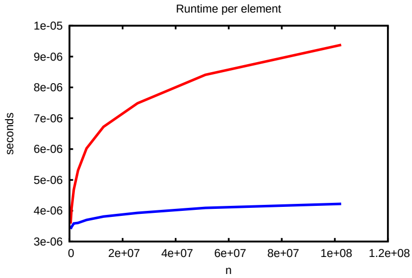
What does Θ notation represent?

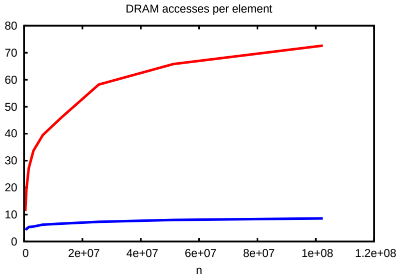
- Number of comparisons
- Number of arithmetic operations
- Number of memory accesses

All of the above in RAM/PRAM models

Example: Orthogonal Point Location

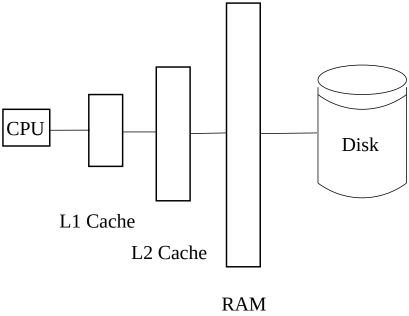
$\Theta(n \log n)$ memory accesses?



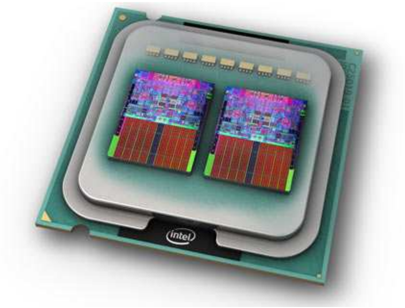


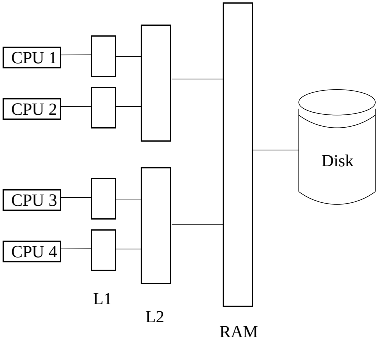
Modern memory design

Caches!



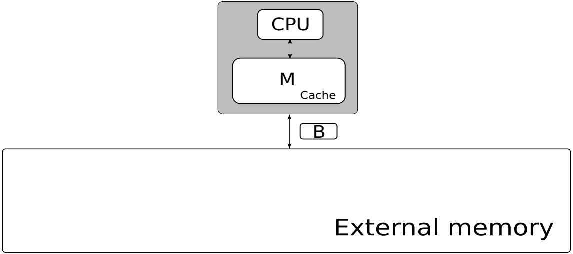
Caches in Multi-cores



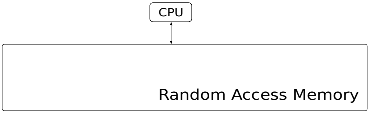


Dealing with Caches

External Memory Model

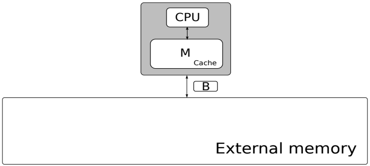


RAM

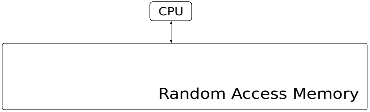


PRAM

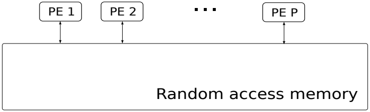
External Memory



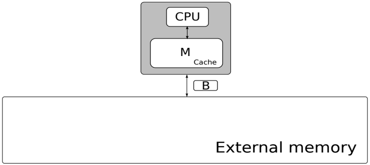
RAM



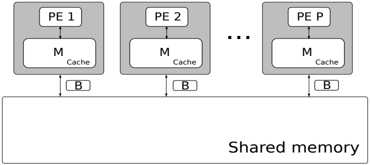
PRAM



External Memory

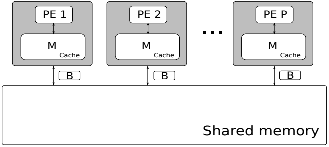


Parallel External Memory



Parallel External Memory (PEM) Model [SPAA '08]

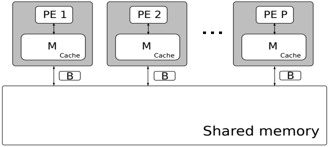
PEM: A simple model of locality and parallelism



- Explicit cache replacement
- Up to P block transfers = 1 I/O
- Block-level CREW access

Parallel External Memory (PEM) Model [SPAA '08]

PEM: A simple model of locality and parallelism



- Explicit cache replacement
- Up to P block transfers $= 1$ I/O
- Block-level CREW access

Complexity: Parallel I/O complexity – number of *parallel* block transfers

Solutions in PEM

Algorithms

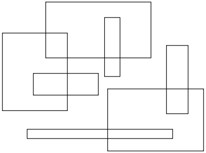
- Sorting [SPAA'08]
- Orthogonal Computational Geometry [ESA'10, IPDPS'11]
- Problems on Graphs [IPDPS'10]

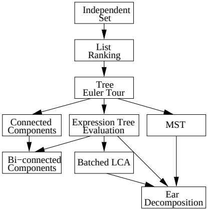
PEM Data Structures [SPAA'12]

- Parallel Buffer/Range Tree

Exp. Validation [ESA'13]

- Orthogonal Computational Geometry

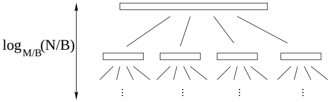




PEM Algorithms

Sorting [SPAA '08]

$$d = \min\{M/B, \sqrt{n/P}\}$$

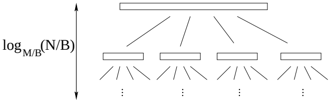


Sorting

- d-way mergesort
- d-way distribution sort

Sorting [SPAA '08]

$$d = \min\{M/B, \sqrt{n/P}\}$$



Sorting

- d-way mergesort
- d-way distribution sort

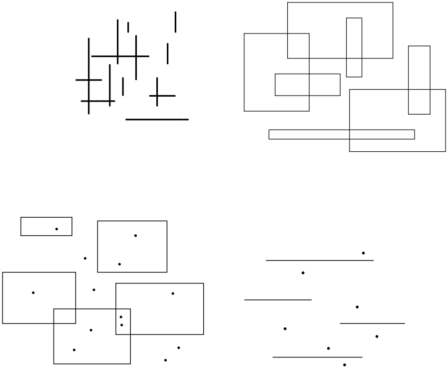
$$\Theta\left(\frac{n}{PB}\log_{M/B}\frac{n}{B}\right) = \mathrm{sort}_P(n) \text{ I/Os}$$

$$\Theta(n \log n) \text{ comparisons}$$

Geometric Algorithms [ESA '10, IPDPS '11]

Solved Problems

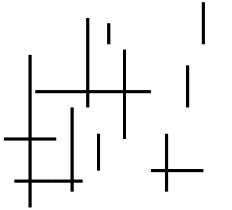
- Line segment intersections
- Rectangle intersections
- Range Query
- Orthogonal point location



$$\Theta\left(\frac{n}{PB}\log_{M/B}\frac{n}{B}\right) = \mathrm{sort}_P(n) \text{ I/Os}$$

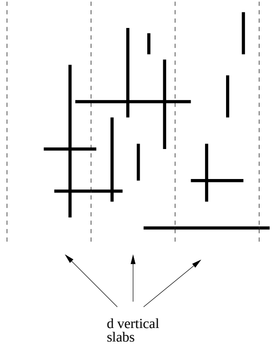
Parallel Distribution Sweeping

$$d = \min\{M/B, \sqrt{n/P}\}$$



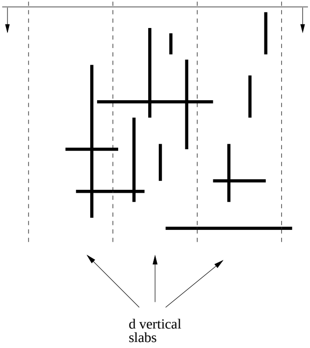
Parallel Distribution Sweeping

$d = \min\{M/B, \sqrt{n/P}\}$



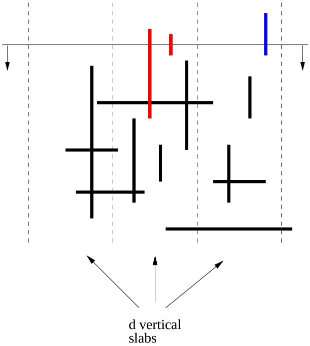
Parallel Distribution Sweeping

$$d = \min\{M/B, \sqrt{n/P}\}$$



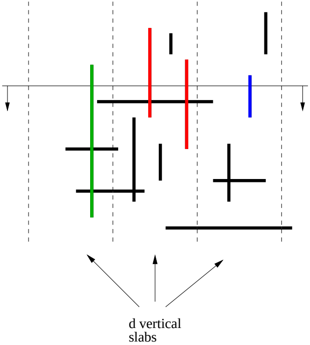
Parallel Distribution Sweeping

$$d = \min\{M/B, \sqrt{n/P}\}$$



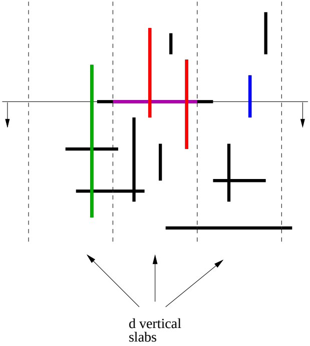
Parallel Distribution Sweeping

$$d = \min\{M/B, \sqrt{n/P}\}$$



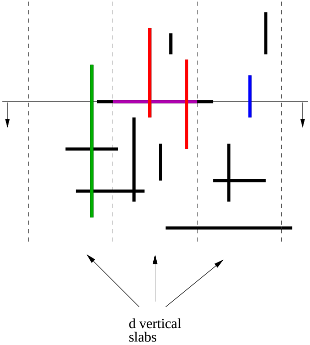
Parallel Distribution Sweeping

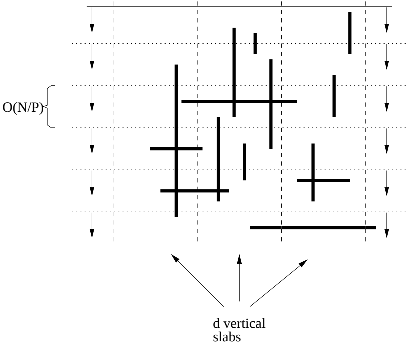
$$d = \min\{M/B, \sqrt{n/P}\}$$



Parallel Distribution Sweeping

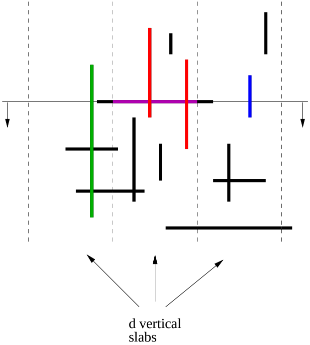
$$d = \min\{M/B, \sqrt{n/P}\}$$

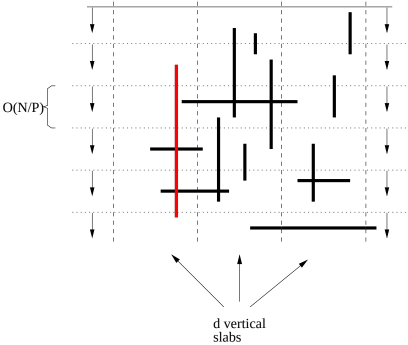




Parallel Distribution Sweeping

$$d = \min\{M/B, \sqrt{n/P}\}$$

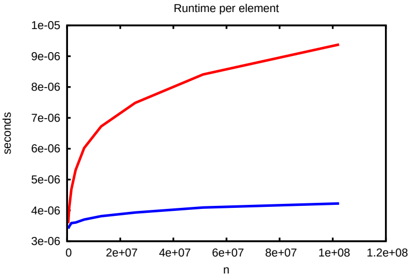


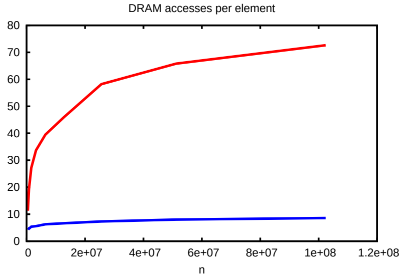


Example: Orthogonal Point Location

Complexity

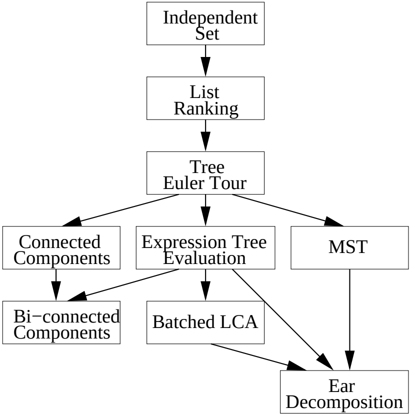
- $\mathcal{O}\left(\frac{n}{PB}\log_{M/B}\frac{n}{B}\right)$ I/Os
- $\mathcal{O}(n\log n)$ comparisons



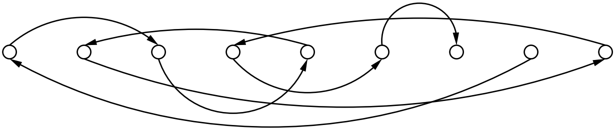


$$\log_{M/B}(n/B) \approx 2$$

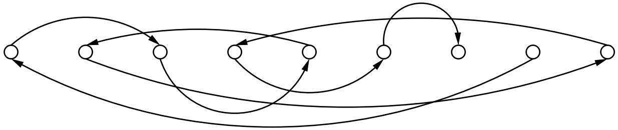
Graph Problems [IPDPS '10]



Pointers And Caches Don't Get Along

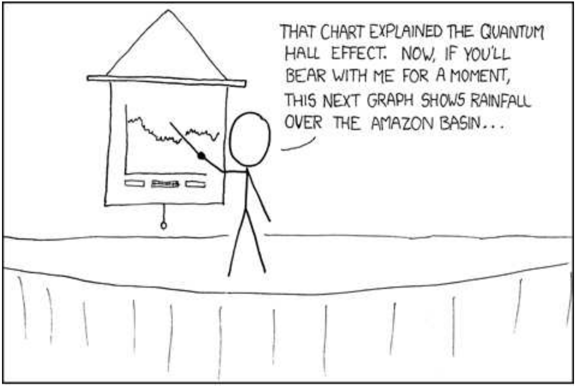


Pointers And Caches Don't Get Along



$$\min\left\{\Theta\left(\frac{n}{P}+\log n\right),\Theta\left(\frac{n}{PB}\log_{M/B}\frac{n}{B}\right)\right\}$$ I/Os for list ranking.

If you bear with me for a moment...



IF YOU KEEP SAYING "BEAR WITH ME FOR A MOMENT", PEOPLE TAKE A WHILE TO FIGURE OUT THAT YOU'RE JUST SHOWING THEM RANDOM SLIDES.

GPGPU Computing

GPU computing

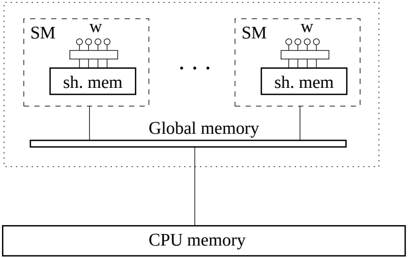
- GPGPU – graphics cards for computation
- Hundreds of cores
- Thousands of threads

GPUs

Global Memory ($n \leq 3$GB)

Shared (Local) Memory ($M \approx 48$ kB)

Registers (≈ 32 kB)



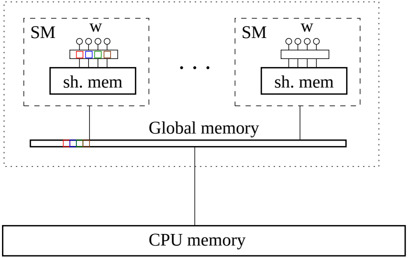
GPUs

Global Memory ($n \leq 3$GB)

- Coalesced accesses ($w = 32$)

Shared (Local) Memory ($M \approx 48$ kB)

Registers (≈ 32 kB)



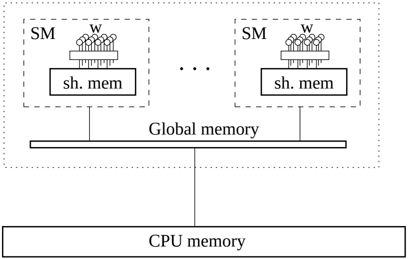
GPUs

Global Memory ($n \leq 3$GB)

- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48$ kB)

Registers (≈ 32 kB)



GPUs

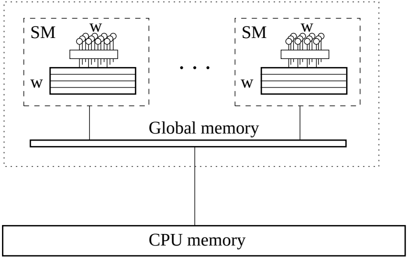
Global Memory ($n \leq$ 3GB)

- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48$ kB)

- Memory banks

Registers ($\approx$ 32 kB)



GPUs

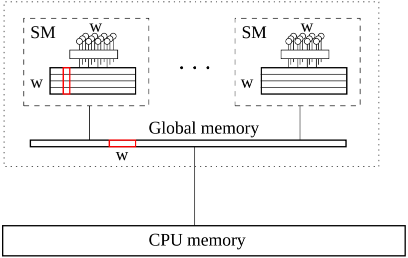
Global Memory ($n \leq 3$GB)

- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48$ kB)

- Memory banks

Registers (≈ 32 kB)



GPUs

Global Memory ($n \leq$ 3GB)

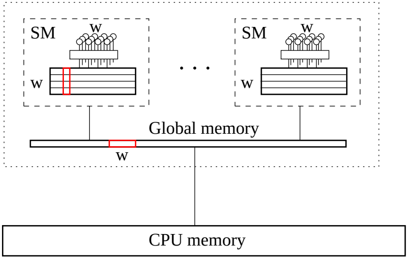
- Coalesced accesses ($w = 32$)
- Hyperthreading

Shared (Local) Memory ($M \approx 48$ kB)

- Memory banks

Registers ($\approx$ 32 kB)

- Fastest if addressable at compile time



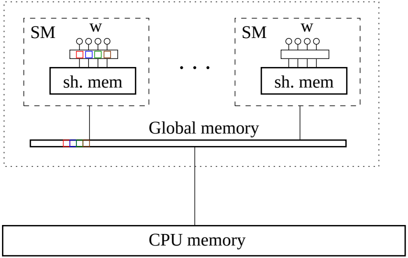
GPU Programming Challenges

Type of memory

- Coalescing complicates algorithm design
- Shared memory bank conflicts
- Registers private to threads

Hyperthreading

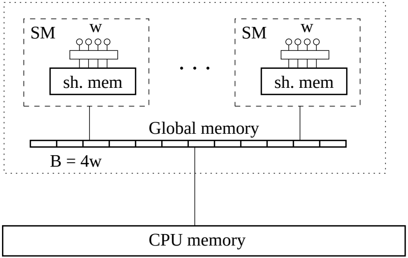
- Resource sharing tradeoffs



GPU: PEM with multiprocessors [under submission]

Model

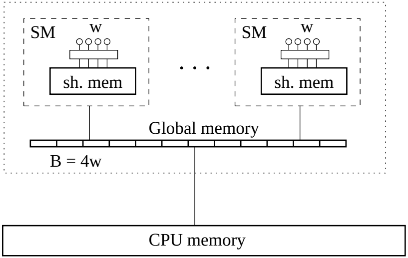
- Hyperthreading: k vSMs per SM
- Always load data into shared memory of size $M' = M/k$
- Process shared memory using SIMD algorithms



GPU algorithm design

Algorithmic framework

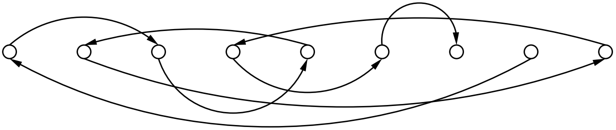
- Pick minimum k to saturate I/O bandwidth
- Minimize I/Os as in PEM
- Independently design (bank conflict free) SIMD algorithm on M' items and w processors



Example: List Ranking

Problem

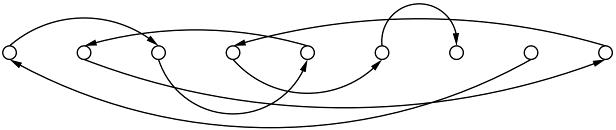
List ranking with coalesced accesses?



Example: List Ranking

Problem

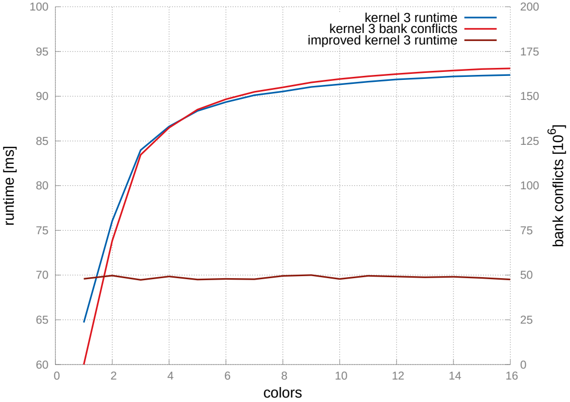
List ranking with coalesced accesses?



From the PEM model:

$$\min\left\{\Theta\left(\frac{n}{P}+\log n\right),\Theta\left(\frac{n}{Pw}\log_{M'/w}\frac{n}{w}\right)\right\}\text{ I/Os.}$$

Effect of bank conflicts on runtime



Modeling bank conflicts

Matrix view of shared memory

- Column-major layout in $w \times (M'/w)$ matrix
- One thread per row
- Convert column- to row-major to process columns

Conversion for square matrices is a matrix transposition

Memory Bank 0	A[0]	A[8]	A[16]	A[24]	A[32]	A[40]	A[48]	A[56]				
Memory Bank 1	A[1]	A[9]	A[17]	A[25]	A[33]	A[41]	A[49]	A[57]				
Memory Bank 2	A[2]	A[10]	A[18]	A[26]	A[34]	A[42]	A[50]	A[58]				
Memory Bank 3	A[3]	A[11]	A[19]	A[27]	A[35]	A[43]	A[51]	A[59]				
Memory Bank 4	A[4]	A[12]	A[20]	A[28]	A[36]	A[44]	A[52]	A[60]				
Memory Bank 5	A[5]	A[13]	A[21]	A[29]	A[37]	A[45]	A[53]	A[61]				
Memory Bank 6	A[6]	A[14]	A[22]	A[30]	A[38]	A[46]	A[54]	A[62]				
Memory Bank 7	A[7]	A[15]	A[23]	A[31]	A[39]	A[47]	A[55]	A[63]				

Example: Bank Conflict Free Sorting

ShearSort [Sen et al. 1986]

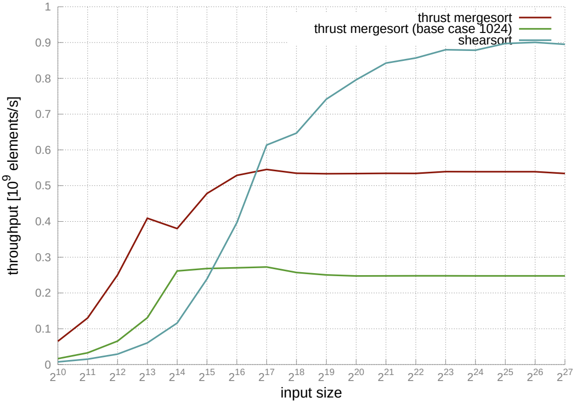
Repeat $\log(M'/w)$ times:

- Sort columns in alternate order
- Sort rows in increasing order

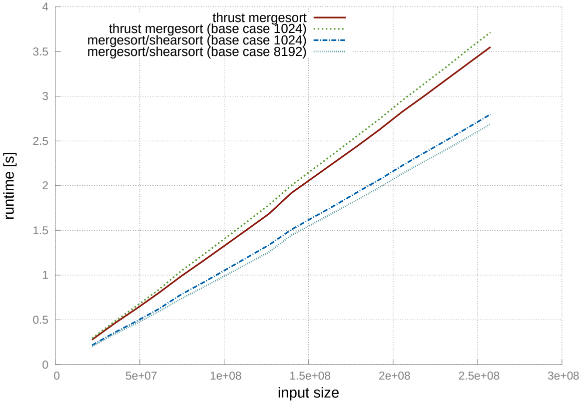
Result: Sorted matrix in column-major order

Memory Bank 0	A[0]	A[8]	A[16]	A[24]	A[32]	A[40]	A[48]	A[56]				
Memory Bank 1	A[1]	A[9]	A[17]	A[25]	A[33]	A[41]	A[49]	A[57]				
Memory Bank 2	A[2]	A[10]	A[18]	A[26]	A[34]	A[42]	A[50]	A[58]				
Memory Bank 3	A[3]	A[11]	A[19]	A[27]	A[35]	A[43]	A[51]	A[59]				
Memory Bank 4	A[4]	A[12]	A[20]	A[28]	A[36]	A[44]	A[52]	A[60]				
Memory Bank 5	A[5]	A[13]	A[21]	A[29]	A[37]	A[45]	A[53]	A[61]				
Memory Bank 6	A[6]	A[14]	A[22]	A[30]	A[38]	A[46]	A[54]	A[62]				
Memory Bank 7	A[7]	A[15]	A[23]	A[31]	A[39]	A[47]	A[55]	A[63]				

Example: Bank Conflict Free Sorting



Example: Bank Conflict Free Sorting



Conclusions

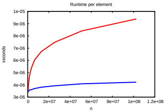
Theory can be useful

With a good model

- Faster implementations
- Solutions can become obvious
- Don't need to reinvent the wheel
- Lower bounds save the frustration

Simplicity vs Accuracy

- (P)RAM: uniform memory
- (P)EM: NUMA in the presence of caches



Thank you!