How to Write Fast Code

18-645, spring 2008

27th Lecture, Apr. 23rd

Instructor: Markus Püschel

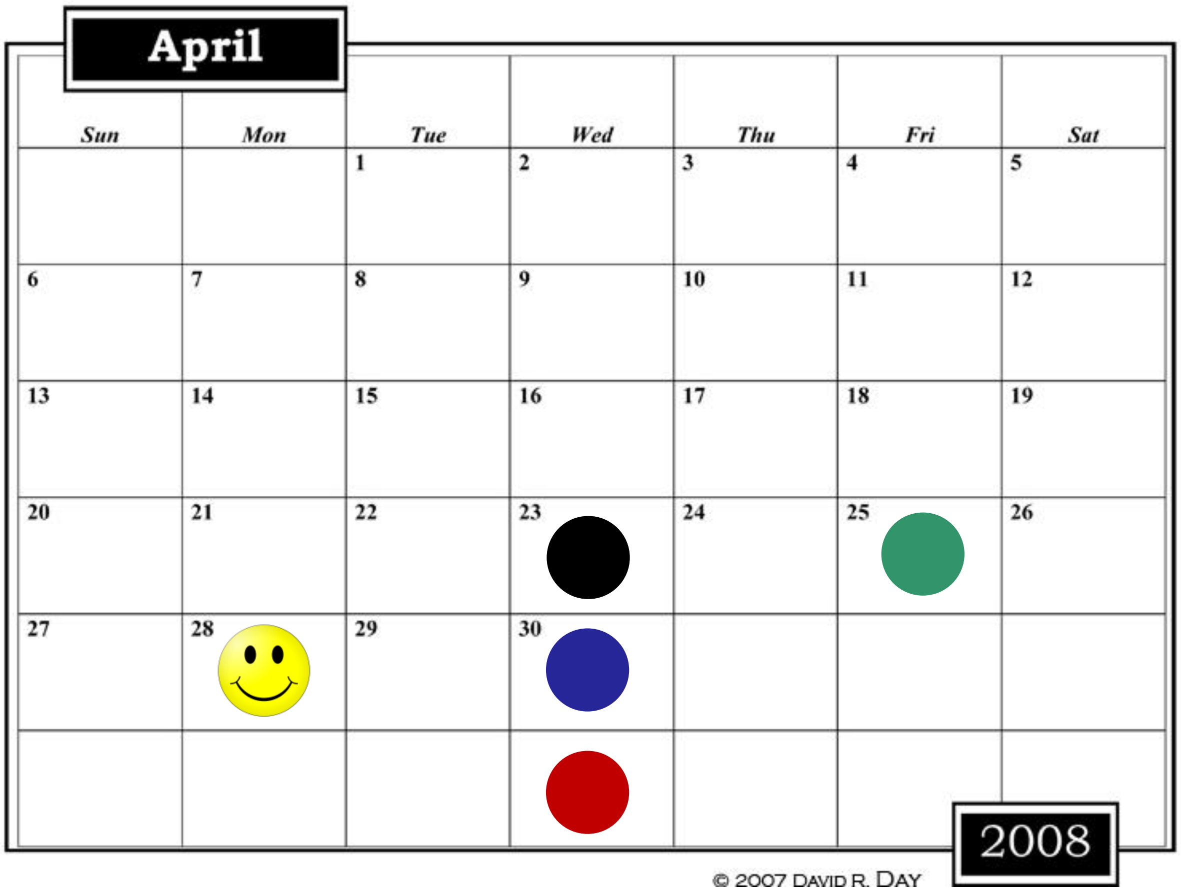
TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)

Course Evaluations

- Are open now
- Please fill it out

Research Project

- Project expectations
- Paper templates and instructions on the website
- Poster template will be uploaded tonight



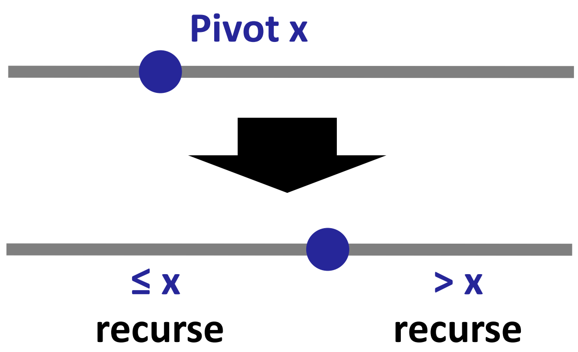
- Today
- Papers due (6 pm)
- Last class:
  poster session
  Scaife Hall
  5:30 – 8:30 pm
- Due:
  - Final papers
  - Final code

Today

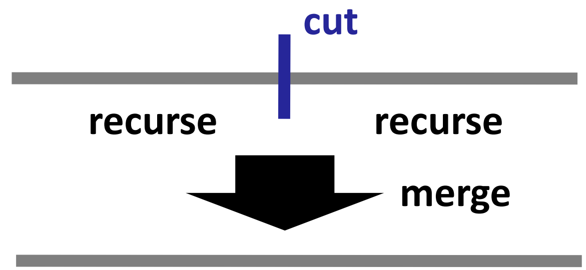
- **Sorting, part 2**
  (Example of a non-numerical problem)

Sorting large arrays

Quicksort



(Multiway-) Mergesort



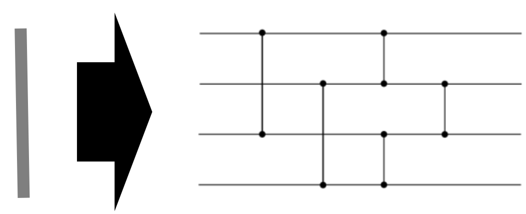
- Temporal and spatial locality
- Simple, array based (no complicated data structures)

Sorting small arrays

Insertion sort

- Good for “almost sorted” list

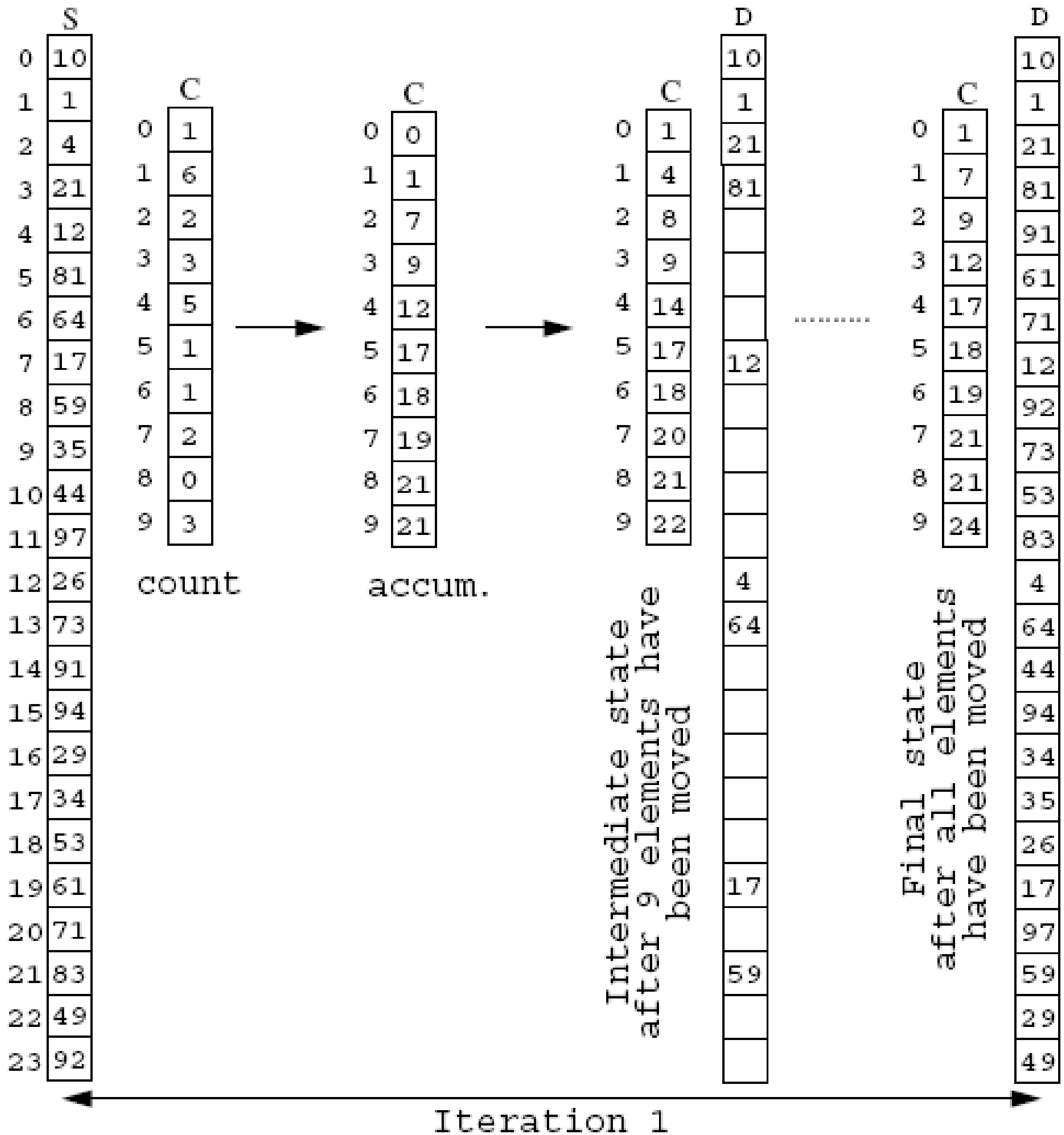
Sorting Networks



- Suitable for unrolling

Radix Sort

- **Basic idea**
- **Second iteration:**
  - Sort for next digit
  - D → S
- **Discussion: blackboard**



Plot: D. Jimenez-Gonzalez, J. Navarro, and J. Larriba-Pey. CC-Radix: A Cache Conscious Sorting Based on Radix Sort. In *Euromicro Conf. on Parallel Distributed and Network based Processing, pp.* 101–108, 2003

Cache-Conscious (CC) Radix Sort (Jimenez et al. 2003)

- **Basic idea: Blackboard**
- **Pseudocode (Bucket = array)**

CC-Radix(bucket,b)
begin
    if fits_in_cache_L_i(bucket) then
        Radix_sort(bucket,b)
    else
        sub-buckets = Reverse_sorting(bucket,b)
        for each sub-bucket in sub-buckets
            CC-Radix(sub-bucket,$b - b_r$)
        endfor
    endif
end

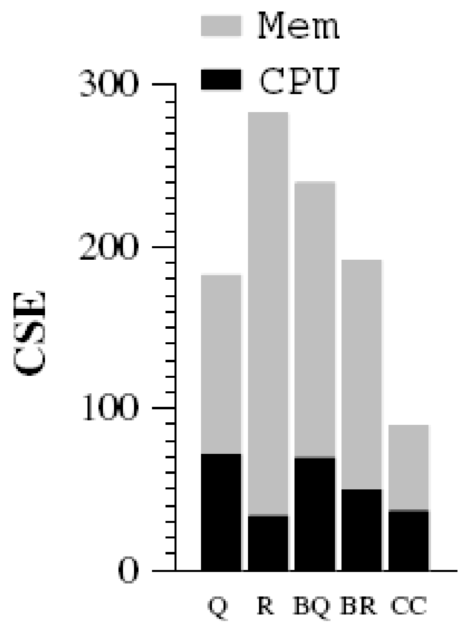
1 step w.r.t. most significant bits b_r

Choose to avoid TLB misses

Source: D. Jimenez-Gonzalez, J. Navarro, and J. Larriba-Pey. CC-Radix: A Cache Conscious Sorting Based on Radix Sort. In *Euromicro Conf. on Parallel Distributed and Network based Processing, pp.* 101–108, 2003

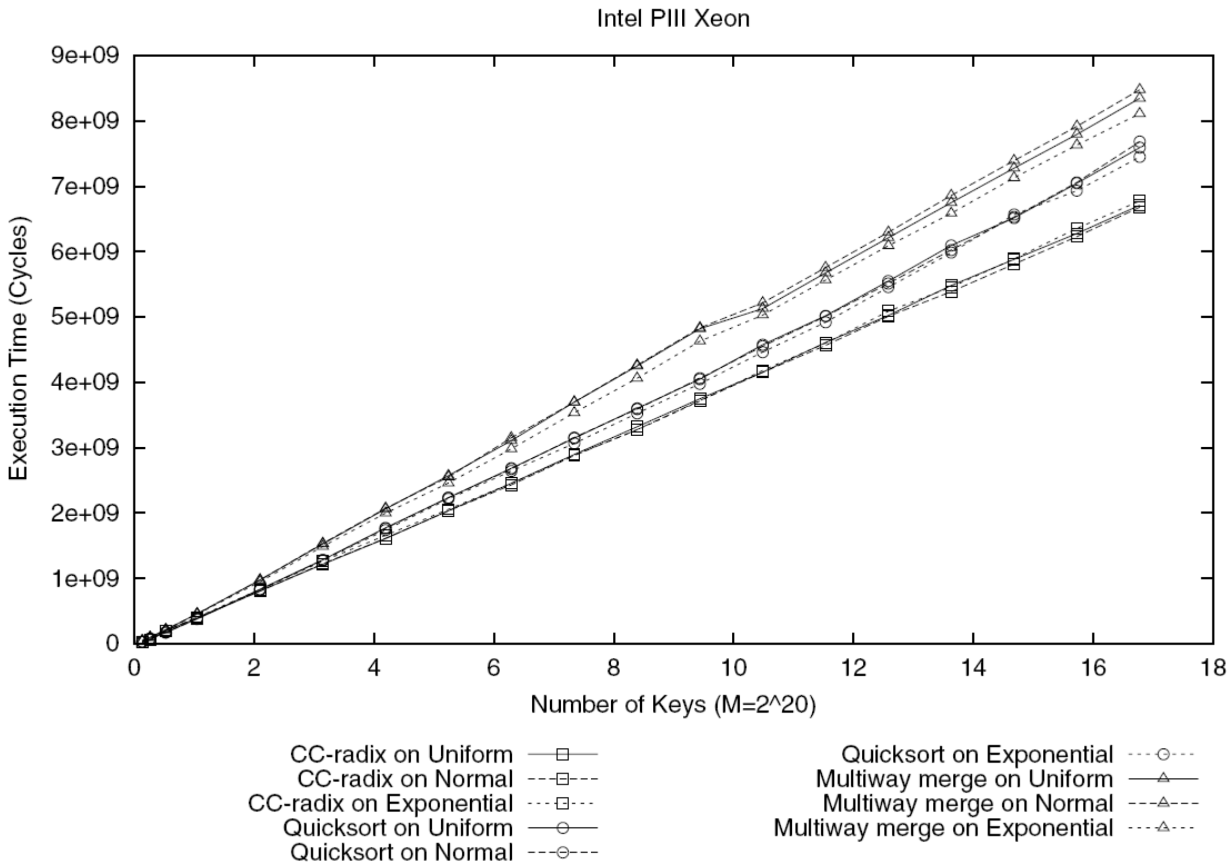
CC Radix Sort: Results

1M keys	CSE	L2 misses	TLB misses
CC-Radix sort	89.0	305,242	1,762
EBT (11)	132.0	434,000	121,000
PLSB (11)	159.6	744,638	10,278
LSB sort (6)	163.0	779,329	172,000
MSB-Radix sort	203.2	796,891	974,008
Radix sort	282.0	502,883	2,607,023



Plots: D. Jimenez-Gonzalez, J. Navarro, and J. Larriba-Pey. CC-Radix: A Cache Conscious Sorting Based on Radix Sort. In *Euromicro Conf. on Parallel Distributed and Network based Processing, pp.* 101–108, 2003

Evaluation: Quicksort, Mergesort, CC-Radix

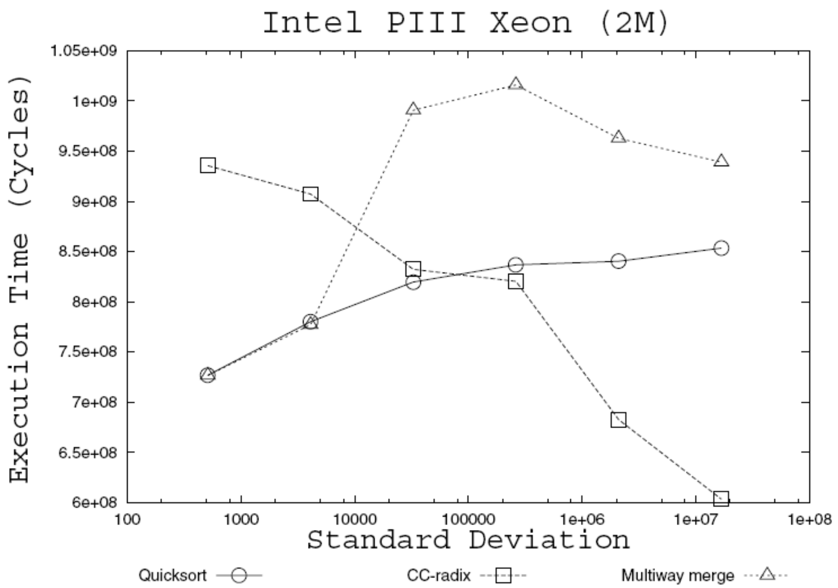


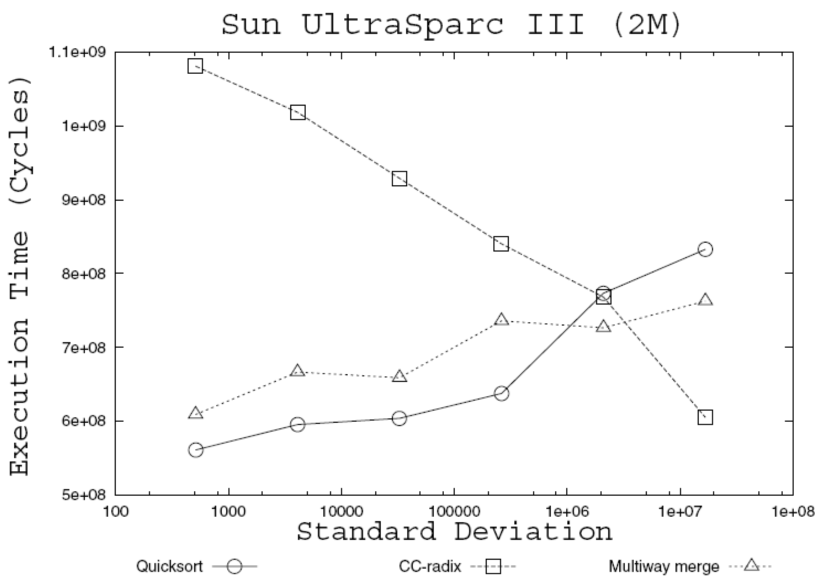
So everything solved?

Plots: Xiaoming Li, María J. Garzarán and David Padua, A Dynamically Tuned Sorting Library, *Proc. International Symposium on Code Generation and Optimization (CGO),* pp. 111-124, 2004

Performance versus Standard Deviation

- **Performance may depend on**
  - the distribution of input data
  - the computing platform





CC-Radix: Smaller stddev = data distributes over fewer buckets = more steps to fit into cache

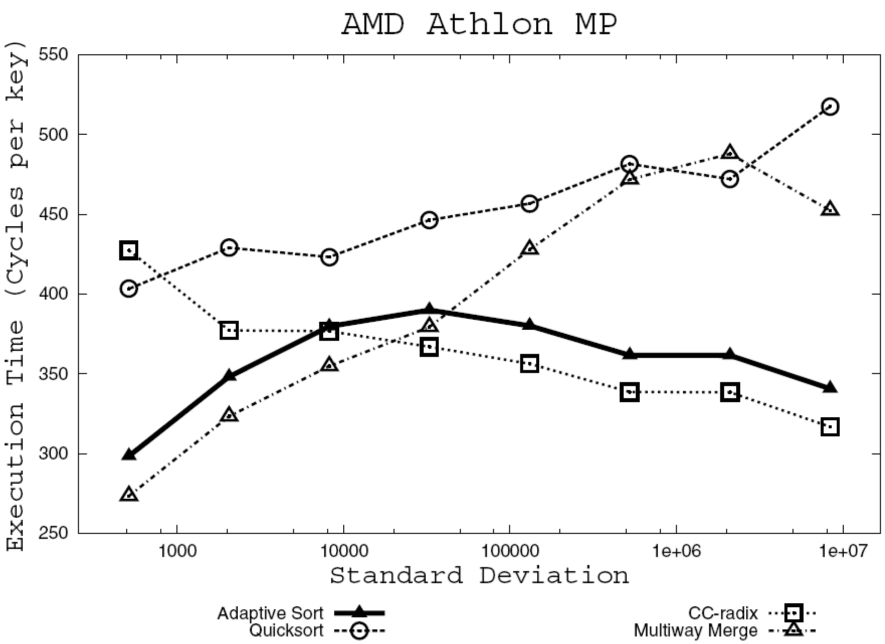
Plots: Xiaoming Li, María J. Garzarán and David Padua, A Dynamically Tuned Sorting Library, *Proc. International Symposium on Code Generation and Optimization (CGO),* pp. 111-124, 2004

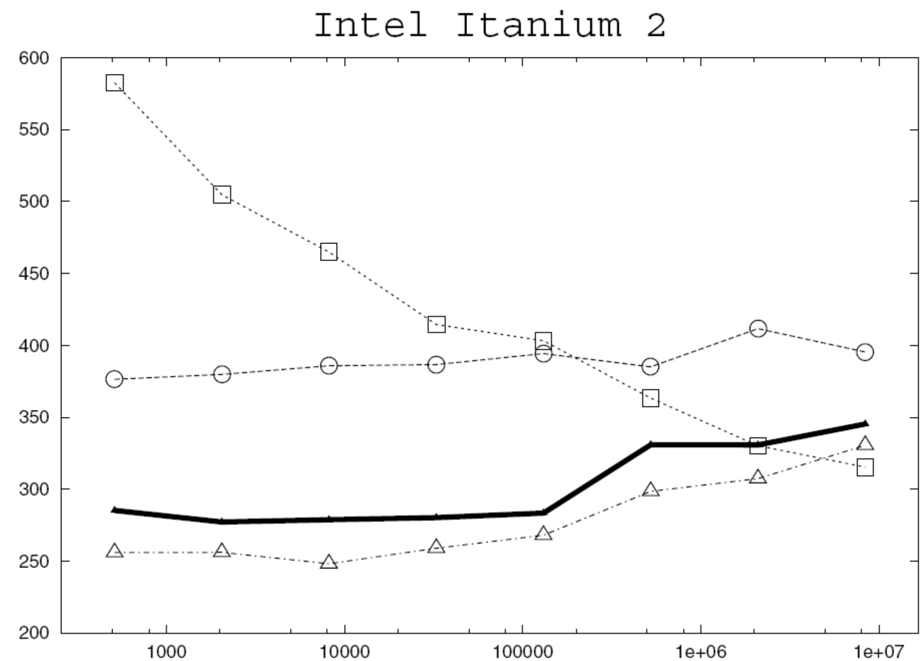
Adaptive Sorting (Li et al.)

- **Basic idea: Adapt algorithm to platform and input data**
- **Algorithm space and parameters:**
  - Quicksort recursively,
    once data sets < t, use insertion or sorting network
  - CC-Radix recursively,
    once data sets < u, use insertion or sorting network
  - Multiway-mergesort (one step) with p subsets and fanout f
    then CC-radix as above
- **Input characteristics: Use entropy E (of digits)**
- **At installation time:**
  - find t and u
  - Use machine learning to learn a decision function:
    decision: (N, E) → {Q, CC, MM(f, p)}

Xiaoming Li, María J. Garzarán and David Padua, A Dynamically Tuned Sorting Library,
Proc. International Symposium on Code Generation and Optimization (CGO), pp. 111-124, 2004

Example Result (Sorting 12M Records)





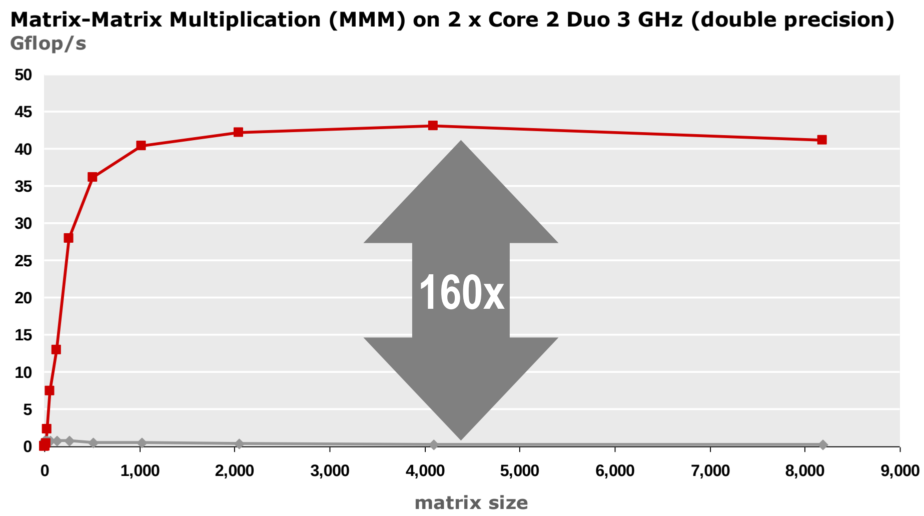
Plots: Xiaoming Li, María J. Garzarán and David Padua, A Dynamically Tuned Sorting Library, *Proc. International Symposium on Code Generation and Optimization (CGO),* pp. 111-124, 2004

	MMM *Atlas*	Sparse MVM *Sparsity/Bebop*	DFT *FFTW*	*Sorting* *Adaptive sorting*
Cache optimization	Blocking	Blocking (rarely useful)	recursive FFT, fusion of steps	Recursive, array-based sorting algorithms
Register optimization	Blocking	Blocking (sparse format)	Scheduling small FFTs	Scheduling sorting networks
Optimized basic blocks	Unrolling, instruction ordering, scalar replacement, simplifications (for FFT), different algorithm (for sorting)			
Other optimizations	—	—	Precomputation of constants	Sorting specific
Adaptivity	Search: blocking parameters	Search: register blocking size	Search: recursion strategy	Search: recursion strategy

Course Summary:
What I hope you have learned

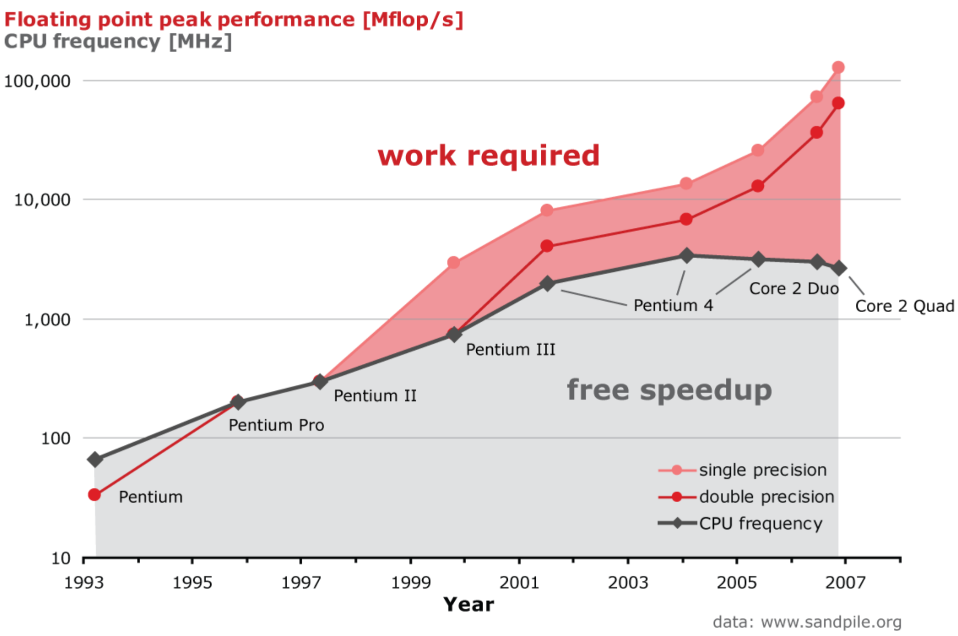
Understand the problem (symptoms)

- Minimizing operations count ≠ minimizing runtime (and not even close)
- A straightforward implementation is usually 10-100x suboptimal
- Optimal performance on one machine does mean optimal performance on another
- End of automatic speedup for legacy software is near



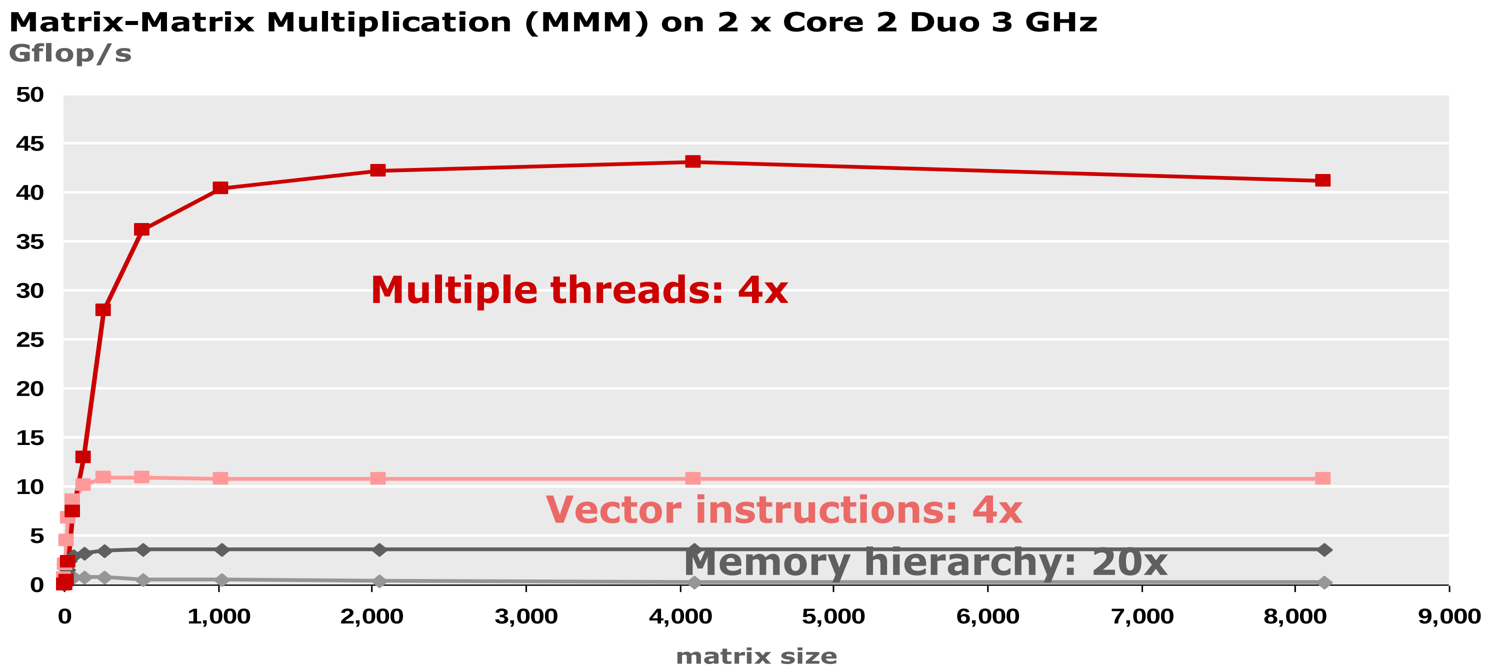
And the Cause

- **Evolution of computing platforms:**
  - End of CPU frequency scaling (power density)
  - Deep memory hierarchies
  - Vector instructions
  - Multiple cores



Understand what to optimize for

- First remove obvious performance killers
- Then memory hierarchy
- And only then vector instructions and multithreading



Understand how to optimize given code

- Proper timing of code
- Find runtime bottleneck
- Analyze cost (cost measure)
- Determine performance and percentage of peak (efficiency)
- Understand cache behavior of code (walking through the code)
- Apply techniques from class
- Repeat procedure

- Understand inherent limitations (degree of reuse, temporal/spatial locality, memory bound/CPU bound)

Understand how to write fast code

- Start with the right algorithm (proper structure)!!!!!
- Continue as in previous slide

Understand how to benchmark and how to report it

- Precise description of procedure
- Correct
- Fair
- Proper analysis
- And if the plots are nice even better ☺