Class 17 – Auto-tuning, or “empirical optimization”

- Definition and motivation

- Auto-tuning applied to sorting
  - Li, Garzaran, and Padua, “A dynamically tuned sorting library”

- Auto-tuning applied to FFT
  - Frigo and Johnson, “The design and implementation of FFTW3”
Auto-tuning, or “empirical optimization”

• Finding an optimized program by trying several versions in a realistic setting (rather than relying on analytic methods)

• Related to program generation, because often the candidates are generated (either at compile time or run time)
Motivation

Programmers try to write the fastest code, and compilers try to apply the most appropriate optimizations. But there are problems:

- Need to have an accurate model of the target machine
- No one code is going to be best on all machines. Instead, generate likely candidates and choose correct one at run time (or install time).

As normally defined, does not include “value-based” code generation (as we use).
Choosing candidates

Major problem in empirical optimization is choosing likely candidates. There are a huge number – possibly infinite – of candidates; need to narrow search.

Allowing for value-based program generation expands the set of candidates.

Examples:
  – Sorting
  – FFT
Sorting

Li, Garzaran, and Padua, “A dynamically tuned sorting library”

• Different sort algorithms are better on different types of data

• Variations on specific sort algorithms can have a big impact
  – E.g. for quicksort, when recursion reaches a small sub-array, it is best to use insertion sort (IS) or a fixed sorting network (SN). But how small, and is IS or SN better?
Sort methods in this paper

- Quicksort (with variations just noted)
- Radix sort – sort data into fixed number of buckets based on higher-order byte, then sort each bucket on next byte, etc. For data of fixed size (e.g. 32-bit ints), radix sort is linear in data size.
- Multiway merge sort – divide data into separate chunks, sort (with radix sort), and merge.
Efficiency of sorting methods

• Quicksort (with variations just noted)
• Radix sort – sort data into fixed number of buckets based on higher-order byte, then sort each bucket on next byte, etc. For data of fixed size (e.g. 32-bit ints), radix sort is linear in data size.
• Multiway merge sort – divide data into separate chunks, sort (with radix sort), and merge.
Efficiency of sorting methods

Features to consider

• Architectural
  – Cache size, # of registers, line size

• Data
  – Size
  – Standard deviation – affects radix sort
    (Paper instead calculates entropy, which is similar to std. dev. but easier to calculate.)
At install time…

Generate data of various sizes, entropies

- For quicksort and radix sort: try various threshold values to switch to either IS or SN.
- Multiway merge: try various entropies

Feed results into a learning algorithm that determines appropriate configuration for real input data
At run time...

Using learning algorithm: Get size and entropy of input data, choose correct algorithm and configuration.