Outline

08:30 Welcome and broader context (Saman Amarasinghe)
08:40 Introduction to OpenTuner (Jason Ansel)
09:10 Search Techniques (Kalyan Veeramachaneni)
09:35 In depth example (Jeffrey Bosboom)
10:00 Break
10:15 Applications
   Halide (Jonathan Ragan-Kelley)
   SEJITS (Chick Markley)
   JVM optimization (Tharindu Rusira)
11:00 Hands on session (Shoaib Kamil & Jeffrey Bosboom)
11:45 Discussion
Observation 1: Software Lifetime >> Hardware

• Lifetime of a software application is 30+ years
• Lifetime of a computer system is less than 6 years
• New hardware every 3 years
• Multiple Ports
• “Software Quality deteriorates in each port
• Huge problem for the expert programmers
Observation 2: Too Complex to Model

• Good old days \( \rightarrow \) model based optimization

• Now
  – Machines are too complex to accurately model
  – Compiler passes have many subtle interactions
  – Thousands of knobs and billions of choices

• But…
  – Computers are cheap
  – We can do end-to-end execution of multiple runs
  – Then use machine learning to find the best choice
void __inplace_stable_sort(RandomAccessIterator __first, RandomAccessIterator __last)
{
    if (__last - __first < 15)
    {
        std::__insertion_sort(__first, __last);
        return;
    }
    RandomAccessIterator __middle = __first + (__last - __first) / 2;
    std::__inplace_stable_sort(__first, __middle);
    std::__inplace_stable_sort(__middle, __last);
    std::__merge_without_buffer(__first, __middle, __last,
                                __middle - __first,
                                __last - __middle);
}
Tuning Sort

- Why 15?
- Dates back to at least June 2000 SGI release
- Still in current C++ STL shipped with GCC
- cutoff = 15 survived 13 years
- In the source code for millions of C++ programs
- There is nothing the compiler can do about it

```cpp
/// This is a helper function for the stable sorting routines.
//
//  template<typename RandomAccessIterator>
//  void
//  _inplace_stable_sort(RandomAccessIterator __first, RandomAccessIterator __last) {
//    if (__last - __first < 15) {
//      std::_insertion_sort(__first, __last);
//      return;
//    }
//    RandomAccessIterator __middle = __first + (__last - __first) / 2;
//    std::_inplace_stable_sort(__first, __middle);
//    std::_inplace_stable_sort(__middle, __last);
//    std::_merge_without_buffer(__first, __middle, __last,
//                                __middle - __first,
//                                __last - __middle);
//  }
```
Algorithmic Choice in Sorting

- Mergesort (N-way)
- Insertionsort
- Radixsort
- Quicksort
Algorithmic Choice in Sorting

- Mergesort (N-way)
- Insertionsort
- Radixsort
- Quicksort

STL Algorithm

N=2 @15
Algorithmic Choice in Sorting

Mergesort (N-way)

Insertionsort

Radixsort

Quicksort

Optimized For:
Xeon (1 core)
Algorithmic Choice in Sorting

Mergesort (N-way)

Insertionsort

Radixsort

Quicksort

Optimized For:
Xeon (1 core)
Xeon (8 cores)
Algorithmic Choice in Sorting

Mergesort (N-way)

Insertionsort

Radixsort

Quicksort

Optimized For:
- Xeon (1 core)
- Xeon (8 cores)
- Niagra (8 cores)
Getting Performance Portability

• High Level Languages + standard libraries → functional portability

• Performance tuning of applications
  – Multiple knobs → set at development time with some minimal search

• Autotuning
  – Can search very large spaces (ex: $10^{1000}$) → better initial results
  – Easy to retune → performance portability

• OpenTuner makes it possible for all
  – Very simple interface
  – Can easily describe the tunable knobs in your application
  – Sophisticated machine learning techniques under the hood to efficiently search for your specific problem
Outline

08:30 Welcome and broader context (Saman Amarasinghe)

08:40 **Introduction to OpenTuner (Jason Ansel)**

09:10 Search Techniques (Kalyan Veeramachaneni)

09:35 In depth example (Jeffrey Bosboom)

10:00 Break

10:15 Applications

  - Halide (Jonathan Ragan-Kelley)
  - SEJITS (Chick Markley)
  - JVM optimization (Tharindu Rusira)

11:00 Hands on session (Shoaib Kamil & Jeffrey Bosboom)

11:45 Discussion