Library Generators and Program Optimization

María Garzarán and David Padua
Department of Computer Science
University of Illinois at Urbana-Champaign
Libraries and Productivity

- Building libraries is one of the earliest strategies to improve productivity.
  - Functionality
  - Performance
- Libraries are particularly important for performance
  - High performance is difficult to attain and not portable.
Compilers vs. Libraries in Sorting

Graphs showing the comparison between QuickSort and IBM ESSL on IBM Power3 and IBM Power4, indicating a ~2X difference in execution time.
Compilers versus libraries in DFT

![Graph comparing performance of vendor libraries and adaptable libraries]

- Vendor library (hand-tuned assembly)
- FFTW (adaptable library)
- SPIRAL (generated code)

- Reasonable implementation
  - Numerical recipes
  - GNU scientific library

10x improvement in performance at certain DFT sizes.
Compilers vs. Libraries in Matrix-Matrix Multiplication (MMM)

![Graph showing performance comparison between compiler and library implementations]

- **Vendor library (hand-tuned assembly):**
- **Standard triple loop + compiler optimizations:**

The graph illustrates a performance improvement of 60x between the two implementations as the matrix size increases.
Libraries and Productivity

- Libraries are not a universal solution.
  - Not all algorithms implemented.
  - Not all data structures.

- Automatic generation of libraries should improve the situation by
  - Reducing implementation cost
  - For a fixed cost, enabling a wider range of implementations and thus make libraries more usable.
Today’s Library Generators

[Algorithm description]

Generator

Source-to-source optimizer

Search strategy

HLL routine

Final HLL routine

Native compiler

Object code

Execution

Execution performance
Important research issues

- Infrastructure library generators.
  - Backend compiler specialized for “a few” classes of problems
- Learning about search strategies.
  - Reducing search time with minimal impact on performance.
- Adaptation to the input data (not needed for dense linear algebra, FFTs)
- Tuning in context.
- More flexible generators
  - algorithms
  - data structures
  - classes of target machines
An infrastructure for library generators

- Signal processing
  - Formula
  - Algorithm in declarative language
  - PARAMETERIZATION FOR SIGNAL PROCESSING

- Linear algebra
  - Algorithm in declarative language
  - PARAMETERIZATION FOR LINEAR ALGEBRA

- High Level Specification
  (Domain Specific Language (DSL))

- PARAMETERIZATION FOR SIGNAL PROCESSING
  - X code with search directives
  - Specialized Backend compiler
  - PROGRAM GENERATOR FOR SORTING
  - Reflective optimization

- Selection Strategy
  - Run
  - Executable

- Reflective optimization
  - Selection Strategy
X: An intermediate representation for black belt macho programmers and library generators

- Language directives to specify in a compact form the search space and the search procedure.

- Three classes of directives.
  - Specification of program transformations (rewriting rules)
  - Application of program transformations
  - Search strategy

Sebastien Donadio, James Brodman, Thomas Roeder, Kamen Yotov, Denis Barthou, Albert Cohen, Maria Jesus Garzaran, David Padua, and Keshav Pingali  
A Language for the Compact Representation of Multiple Program Versions. LCPC 2005
Search strategies

- Numerous possibilities
  - Exhaustive search
  - Random
  - Hill climbing
  - Genetic algorithms
  - Simplex

- A possible strategy: explanation-based learning
  - Use understanding of expected behavior to search for optimal point.
Three library generation projects

1. Spiral and the impact of compilers
2. ATLAS and analytical model
3. Sorting and adaptation to the input
ESSL on Power3

![Graph showing execution time vs. standard deviation for Quicksort, CC-radix, and IBM ESSL on Power3.](image)
ESSL on Power4

IBM Power4

Execution Time (Cycle. per key)

Standard Deviation

Quicksort
CC-radix
IBM ESSL
Motivation

- No universally best sorting algorithm
- Can we automatically generate and tune sorting algorithms for each platform?
- Performance of sorting depends not only on the platform but also on the input characteristics.
A first strategy: Algorithm Selection

- Select the best algorithm from Quicksort, Multiway Merge Sort and CC-radix.

- Relevant input characteristics: number of keys, entropy vector.
Algorithm Selection

![Graph showing execution time for different algorithms on IBM Power3](image)
Algorithm selection for sparse banded solvers

- We have applied this approach to SPIKE, a parallel environment for solving banded linear systems. (A. Sameh, E. Polizzi, Purdue U.)
  - Many algorithms choices.
  - Best choice depends on characteristics of the input matrix (bandwidth, degree of diagonal dominance, size of the matrix) and number of processors.
  - During installation time we build a table to select the best algorithm at runtime.
A better Solution

- We can use different algorithms for different partitions
- Build Composite Sorting algorithms
  - Identify primitives from the sorting algorithms
  - Design a general method to select an appropriate sorting primitive at runtime
  - Design a mechanism to combine the primitives and the selection methods to generate the composite sorting algorithm
Performance of Classifier Sorting

IBM Power3

![Graph showing the performance of different sorting algorithms for the IBM Power3 processor.]
Power4
Sorting

- Again divide-and-conquer.
- But could not find formulas like Spiral.
- Adaptation to input data crucial.
  - Need to deal with other features of the input data – degree of “sortedness”
Conclusions

- Much exploratory work today
- General principles are emerging, but much remains to be done.
- This new exciting area of research should teach us much about program optimization.