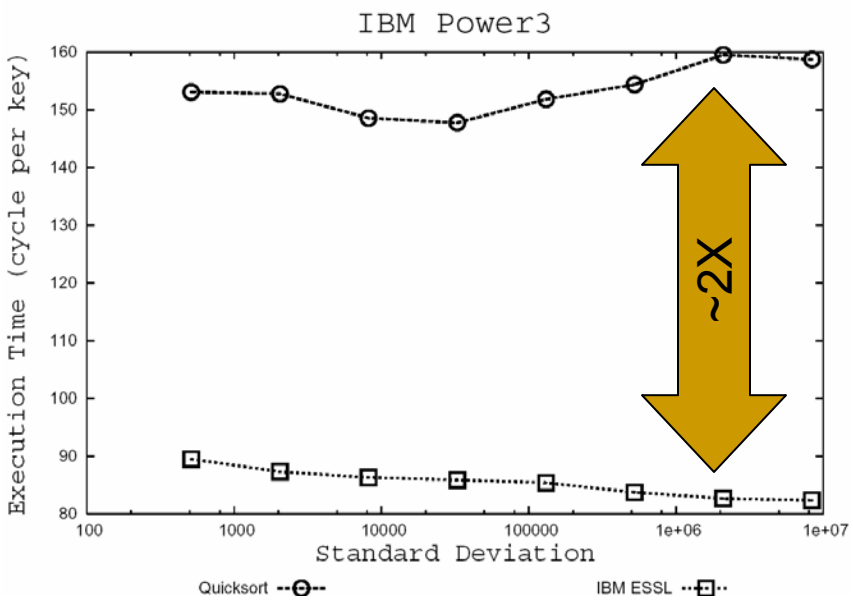
Library Generators and Program Optimization

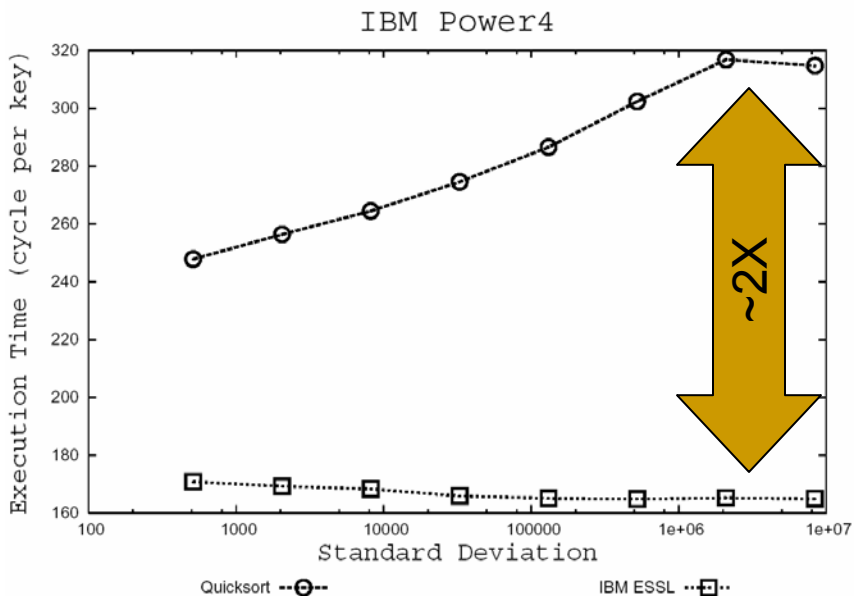
María Garzarán and David Padua
Department of Computer Science
University of Illinois at Urbana-Champaign

Libraries and Productivity

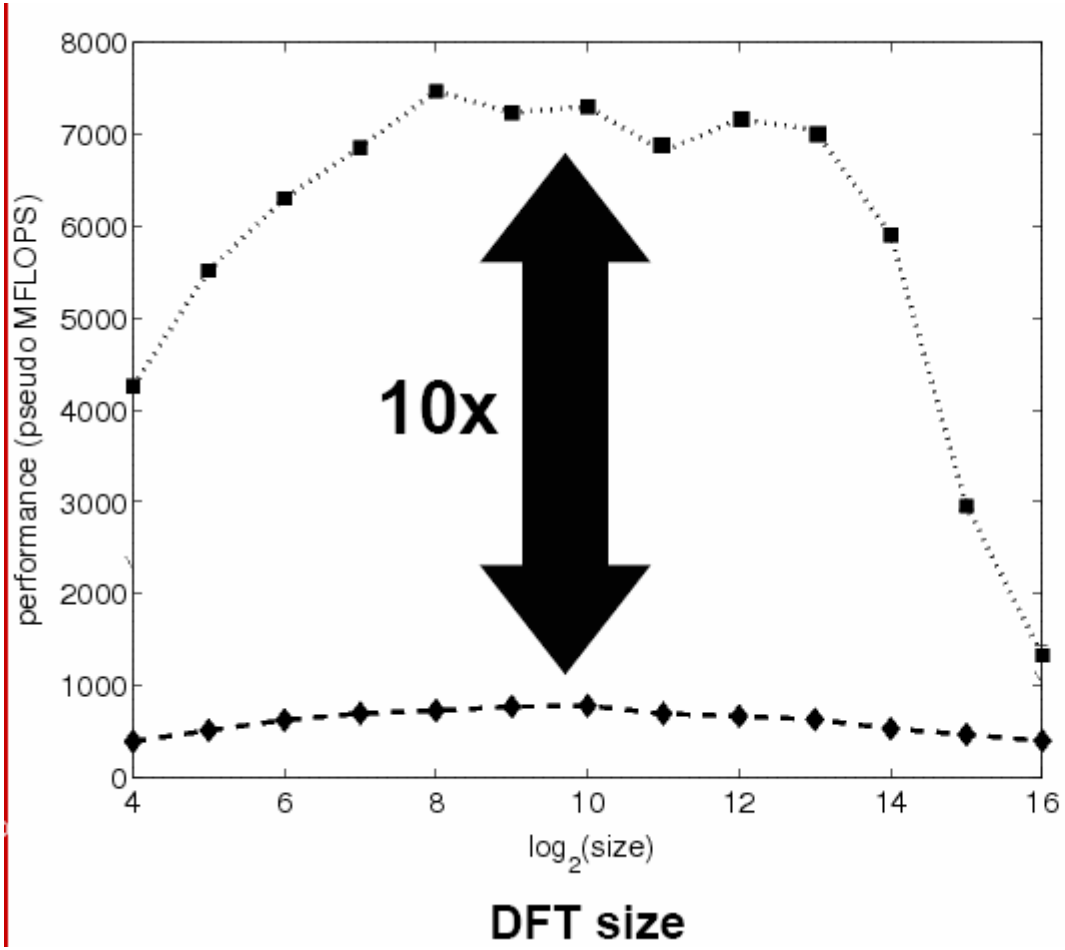
- Building libraries is one of the earliest strategies to improve productivity.
  - Functionality
  - Performance
- Libraries are particularly important for performance
  - High performance is difficult to attain and not portable.

Compilers vs. Libraries in Sorting





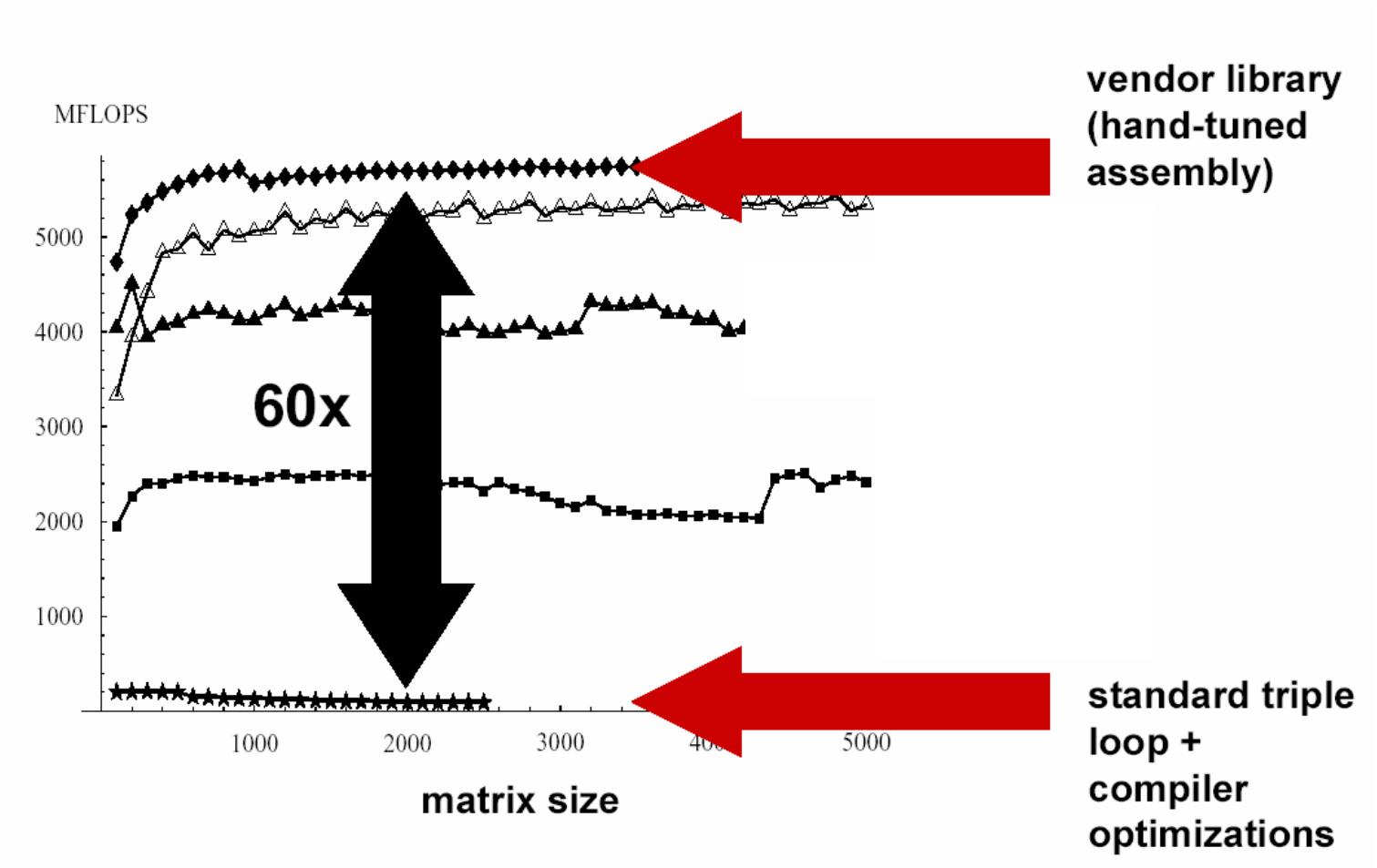
Compilers versus libraries in DFT



vendor library
(hand-tuned assembly)
but also
FFTW (adaptable library)
SPIRAL (generated code)

reasonable
implementation
(Numerical recipes.
GNU scientific library)

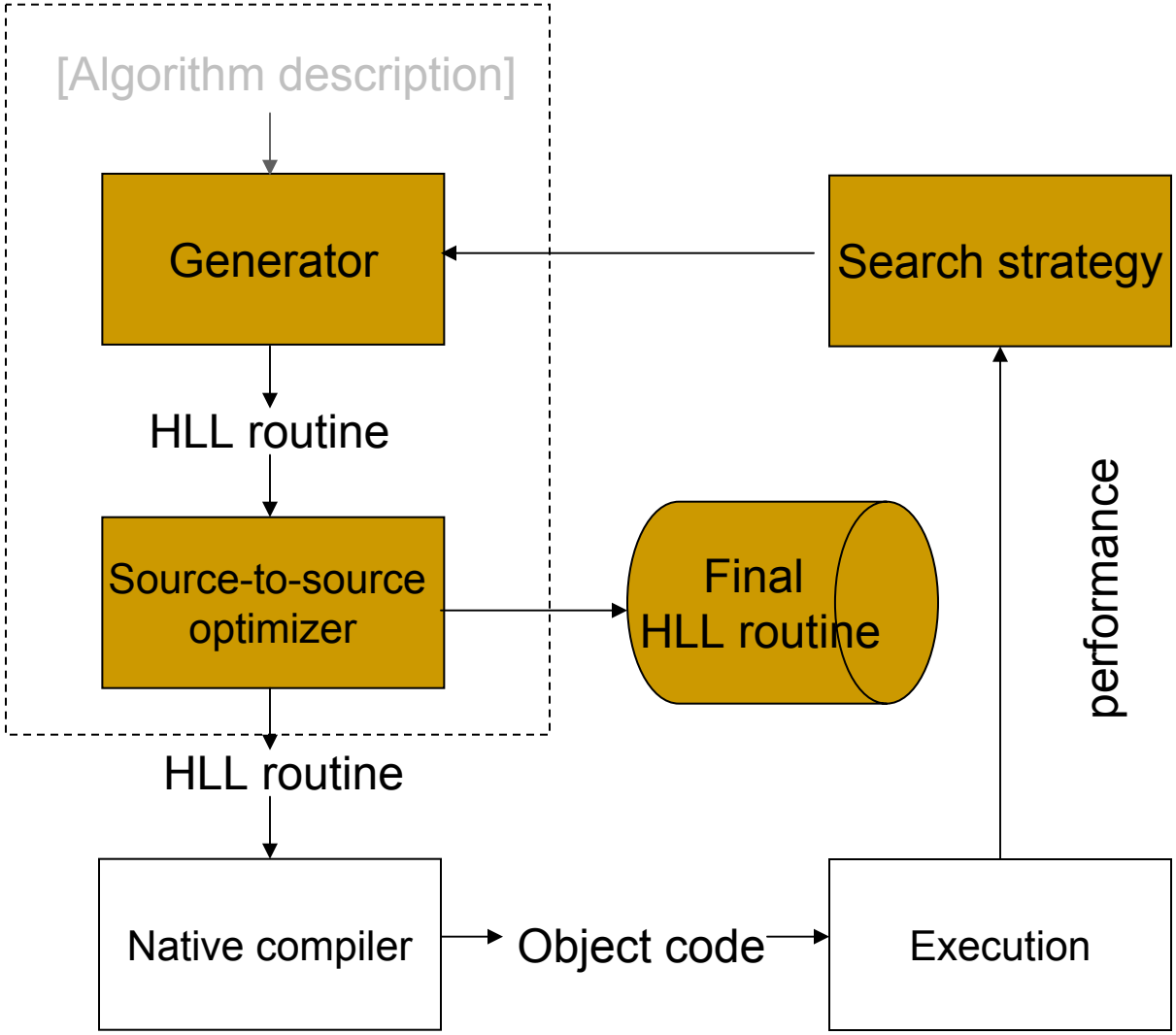
Compilers vs. Libraries in Matrix-Matrix Multiplication (MMM)



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.

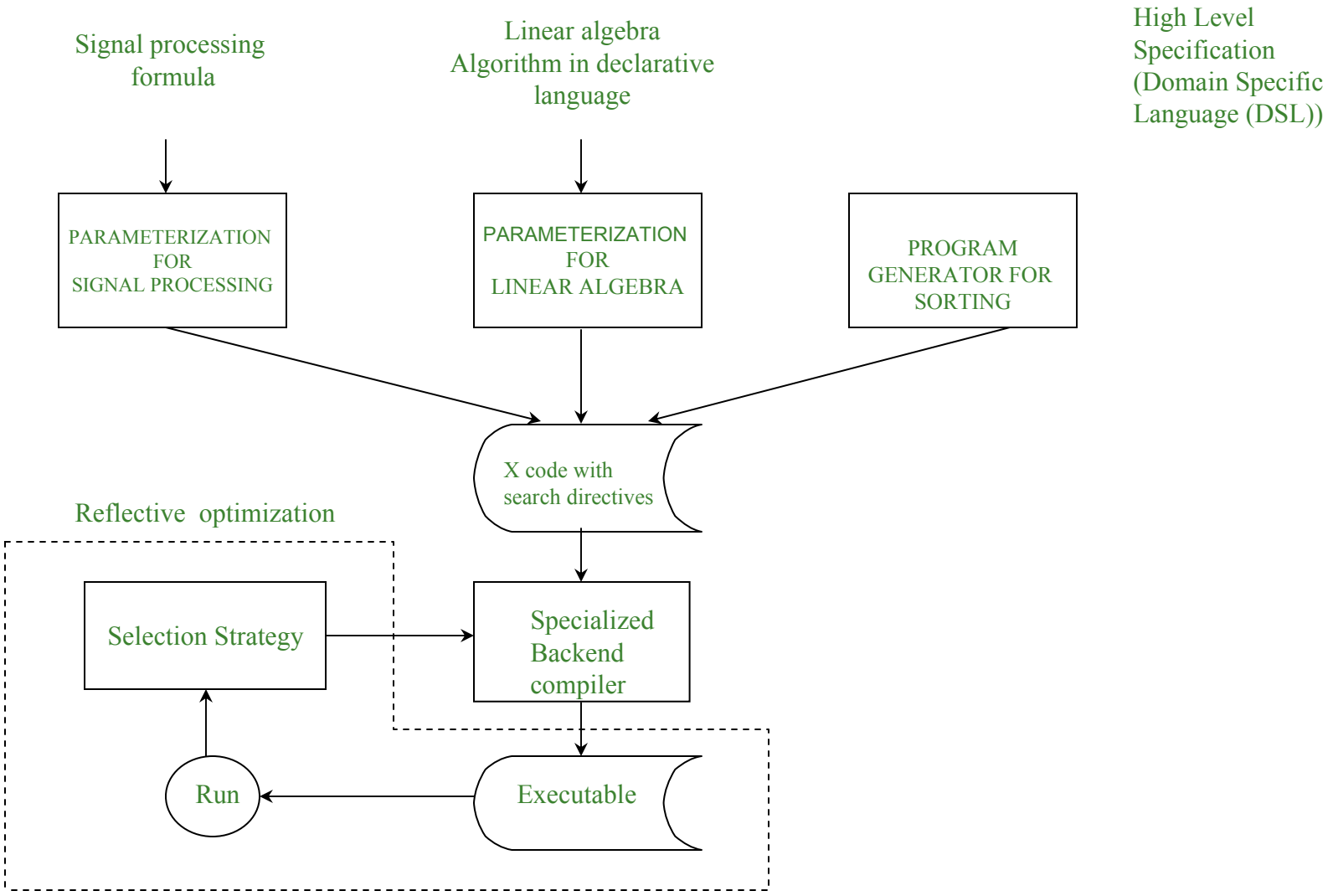
Today's Library Generators



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



X: An intermediate representation for black belt macho programmers and library generators

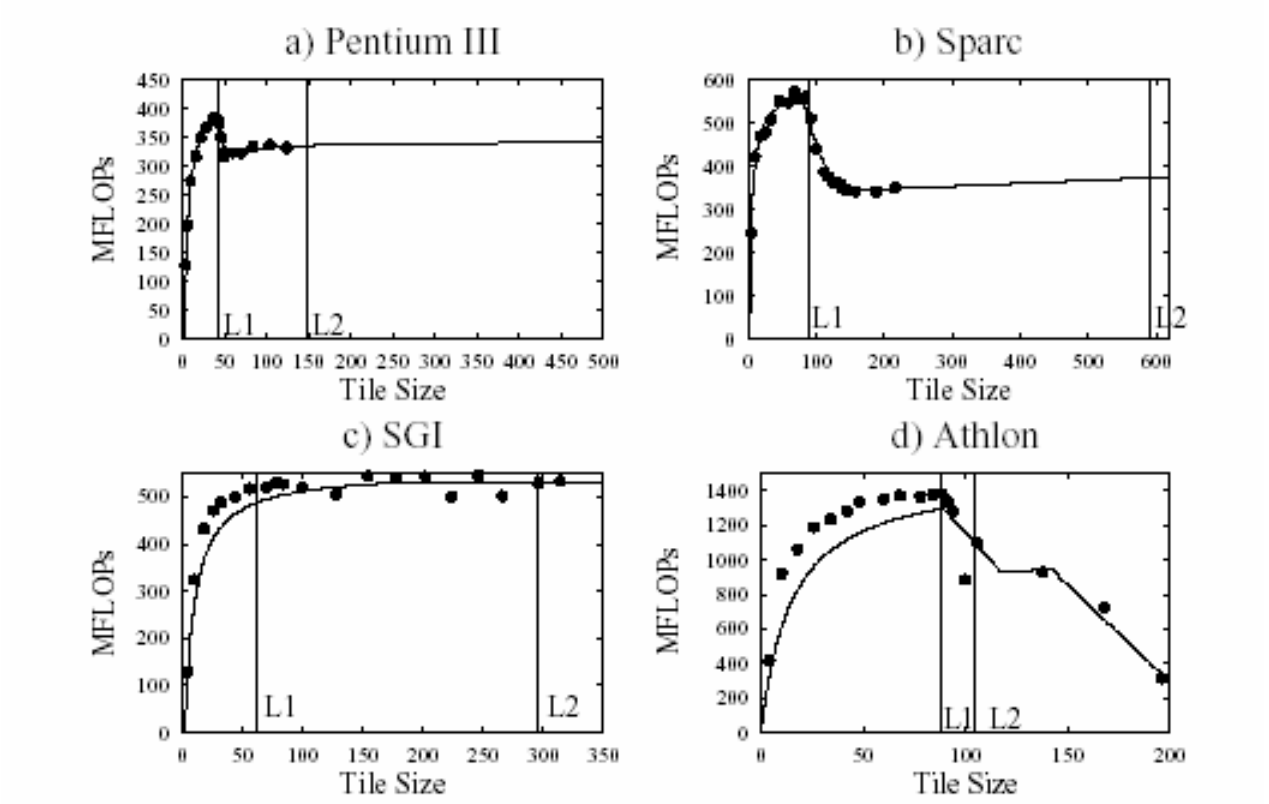
- Language directives to specify in a compact form the search space and the search procedure.
- Three classes of directives.
  - Specification of program transformations (rewriting rules)
  - Application of program transformations
  - Search strategy

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

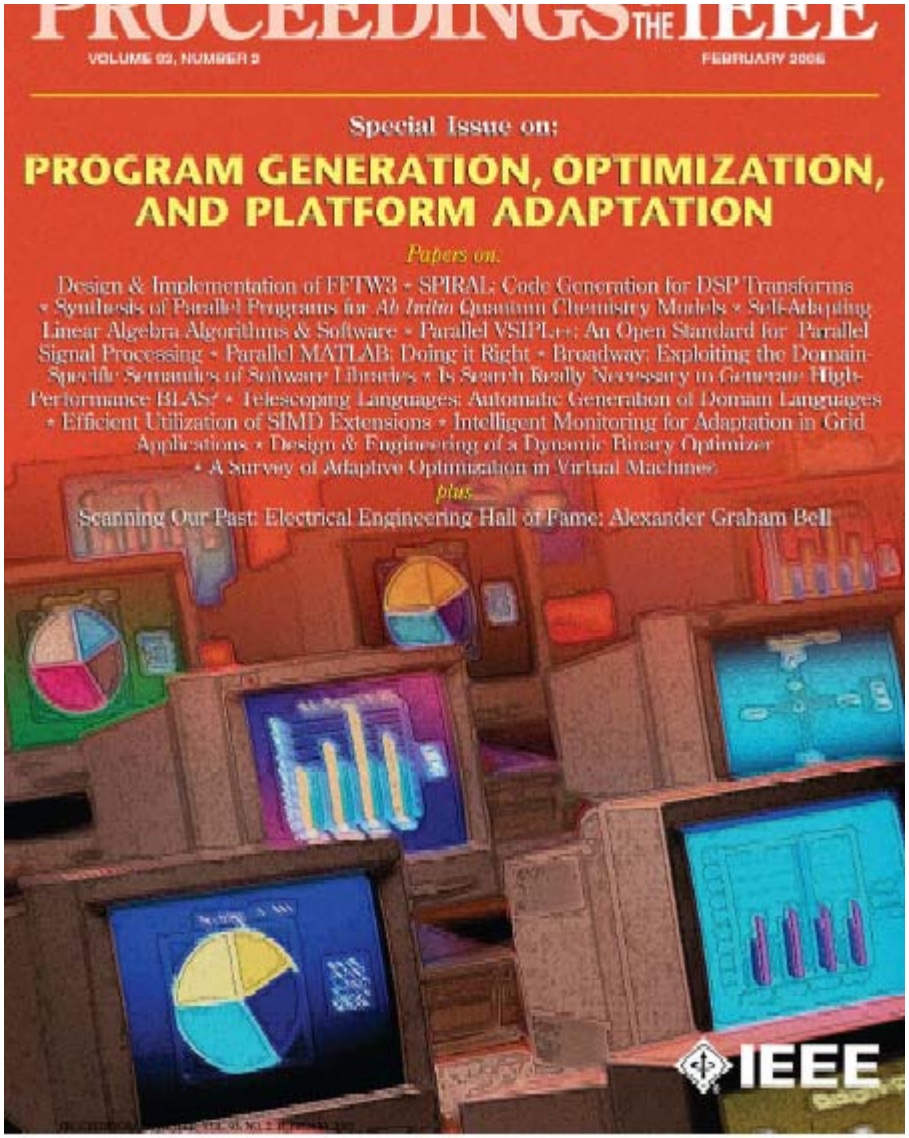
- Numerous possibilities
  - Exhaustive search
  - Random
  - Hill climbing
  - Genetic algorithms
  - Simplex
- A possible strategy: explanation-based learning
  - Use understanding of expected behavior to search for optimal point.

Arkady Epshteyn, Maria Garzaran, Gerald DeJong, David Padua, Gang Ren, Xiaoming Li, Kamen Yotov, and Keshav Pingali Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization. LCPC 2005


a) Pentium III
MFLOPs
Tile Size
L1
L2
b) Sparc
MFLOPs
Tile Size
L1
L2
c) SGI
MFLOPs
Tile Size
L1
L2
d) Athlon
MFLOPs
Tile Size
L1
L2

Three library generation projects

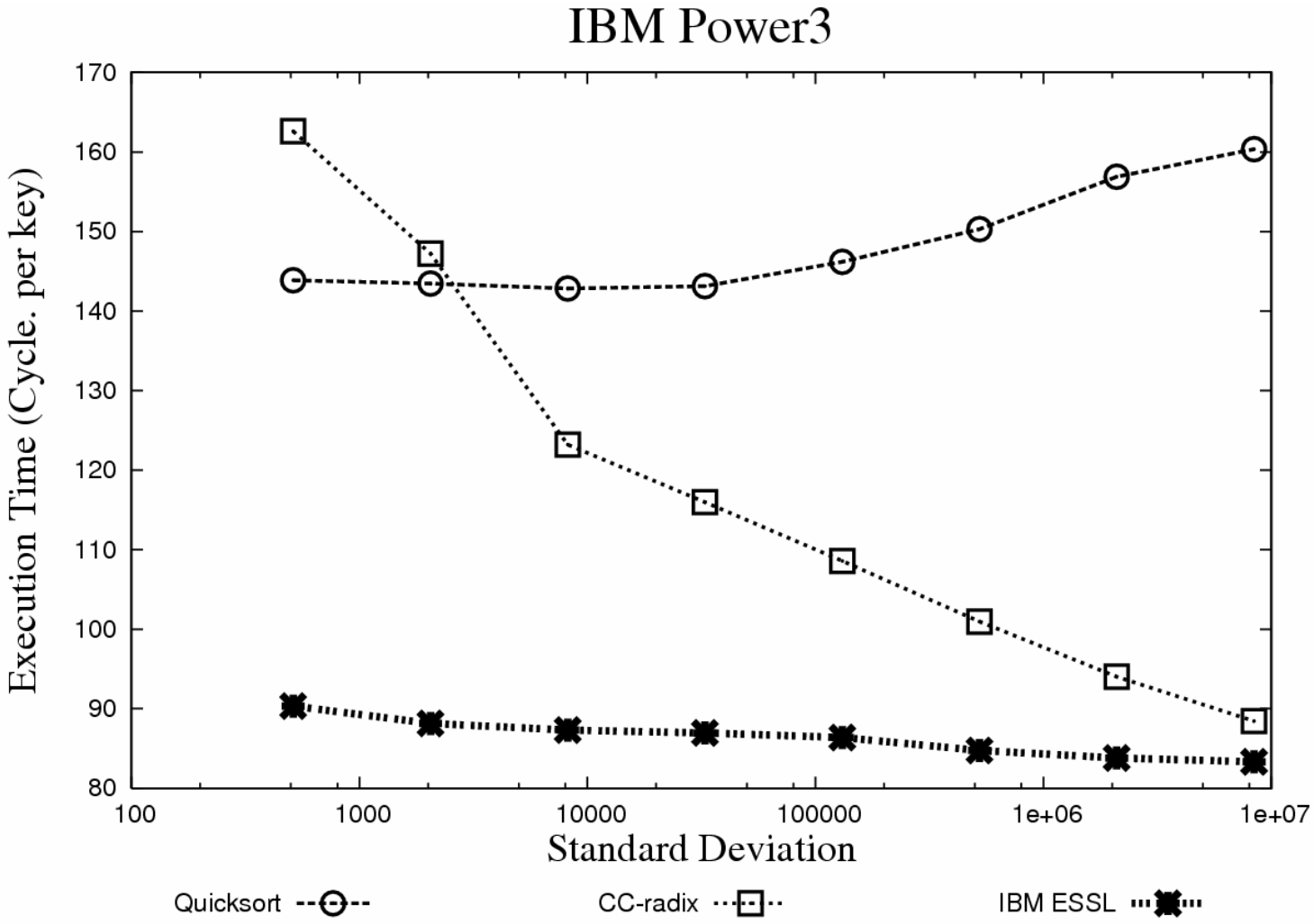
1. Spiral and the impact of compilers
2. ATLAS and analytical model
3. Sorting and adaptation to the input


PROCEEDINGS THE IEEE
VOLUME 93, NUMBER 2
FEBRUARY 2005
Special Issue on:
PROGRAM GENERATION, OPTIMIZATION, AND PLATFORM ADAPTATION
Papers on:
Design & Implementation of FFTW3 • SPIRAL: Code Generation for DSP Transforms • Synthesis of Parallel Programs for *Ab Initio* Quantum Chemistry Models • Self-Adapting Linear Algebra Algorithms & Software • Parallel VSIPL++: An Open Standard for Parallel Signal Processing • Parallel MATLAB: Doing it Right • Broadway: Exploiting the Domain-Specific Semantics of Software Libraries • Is Search Really Necessary to Generate High-Performance BLAS? • Telescoping Languages: Automatic Generation of Domain Languages • Efficient Utilization of SIMD Extensions • Intelligent Monitoring for Adaptation in Grid Applications • Design & Engineering of a Dynamic Binary Optimizer • A Survey of Adaptive Optimization in Virtual Machines
plus:
Scanning Our Past: Electrical Engineering Hall of Fame: Alexander Graham Bell
IEEE

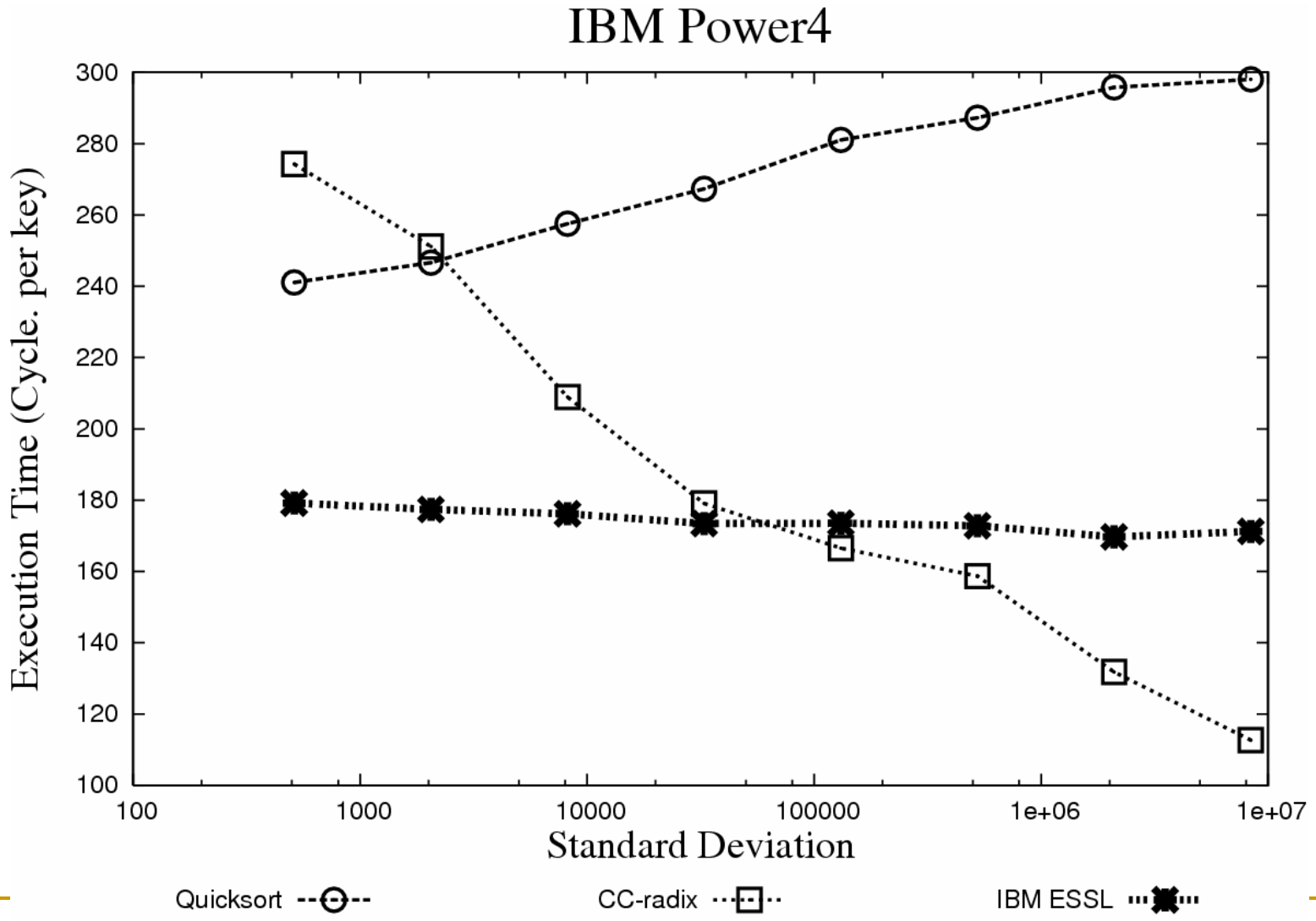
Sorting

Xiaoming Li, María Jesús Garzarán, and David Padua. Optimizing Sorting with Genetic Algorithms. In *Proc. of the International Symposium on Code Generation and Optimization*, pages 99-110, March 2005.

ESSL on Power3



ESSL on Power4



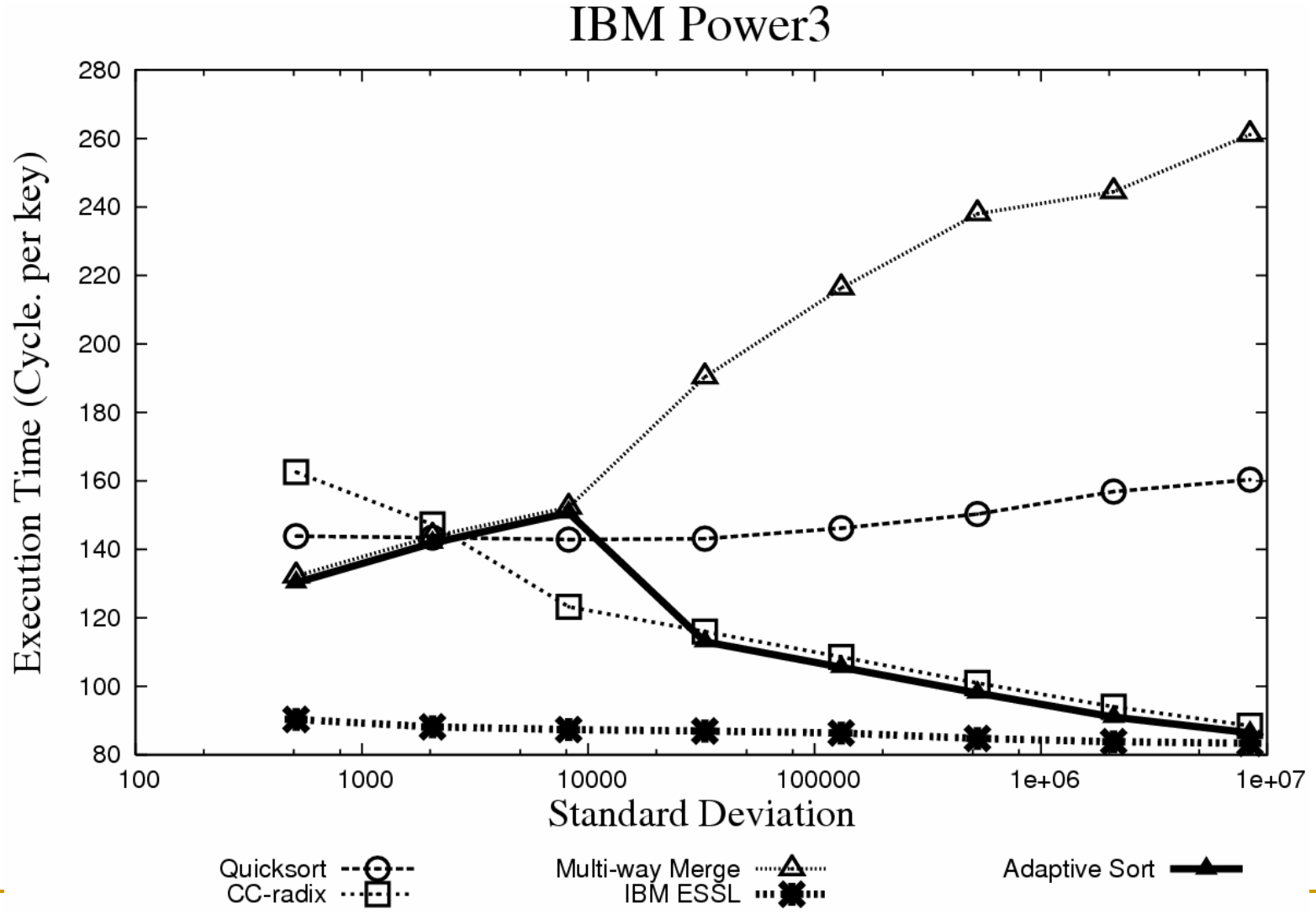
Motivation

- 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



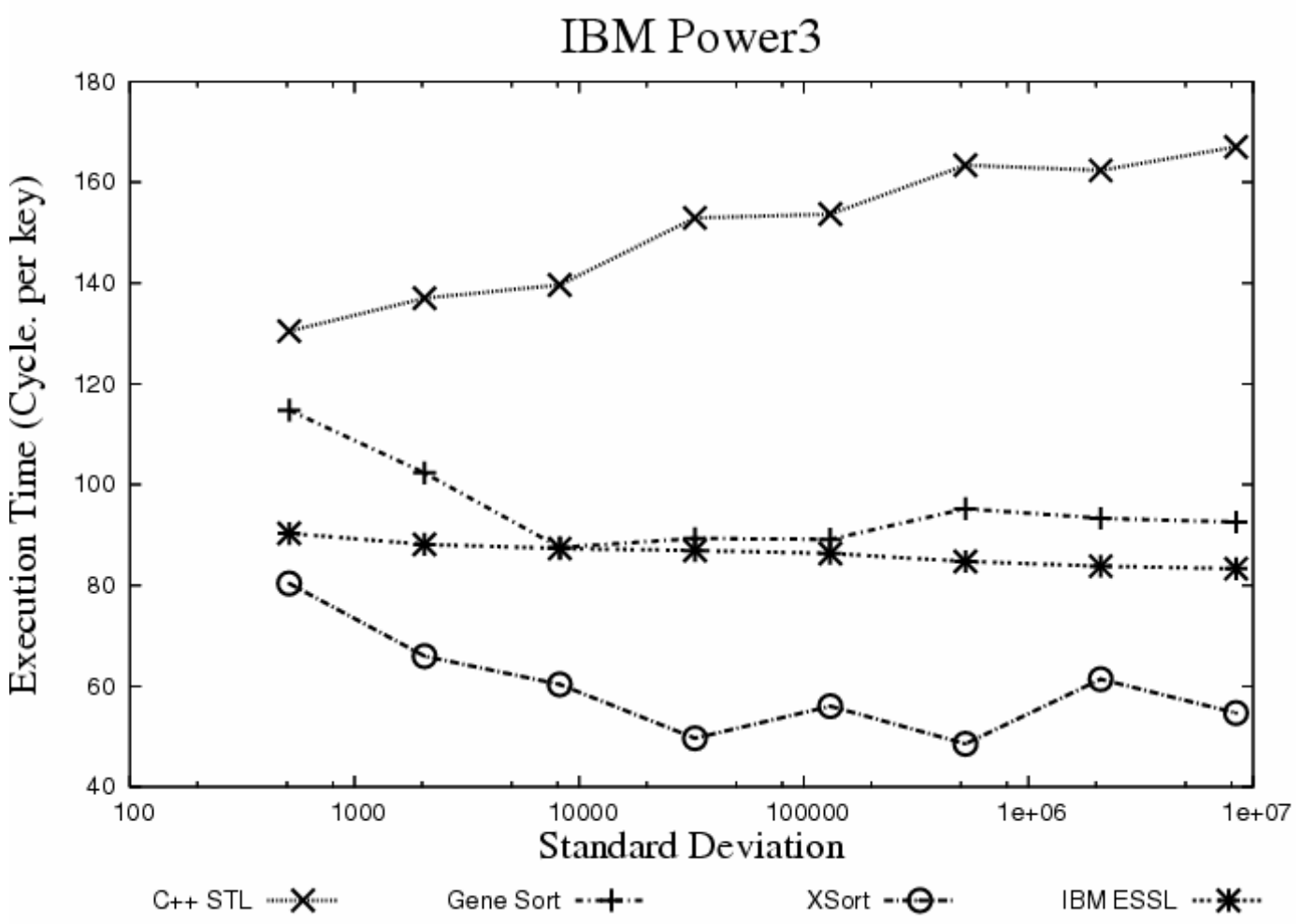
Algorithm selection for sparse banded solvers

- 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.**

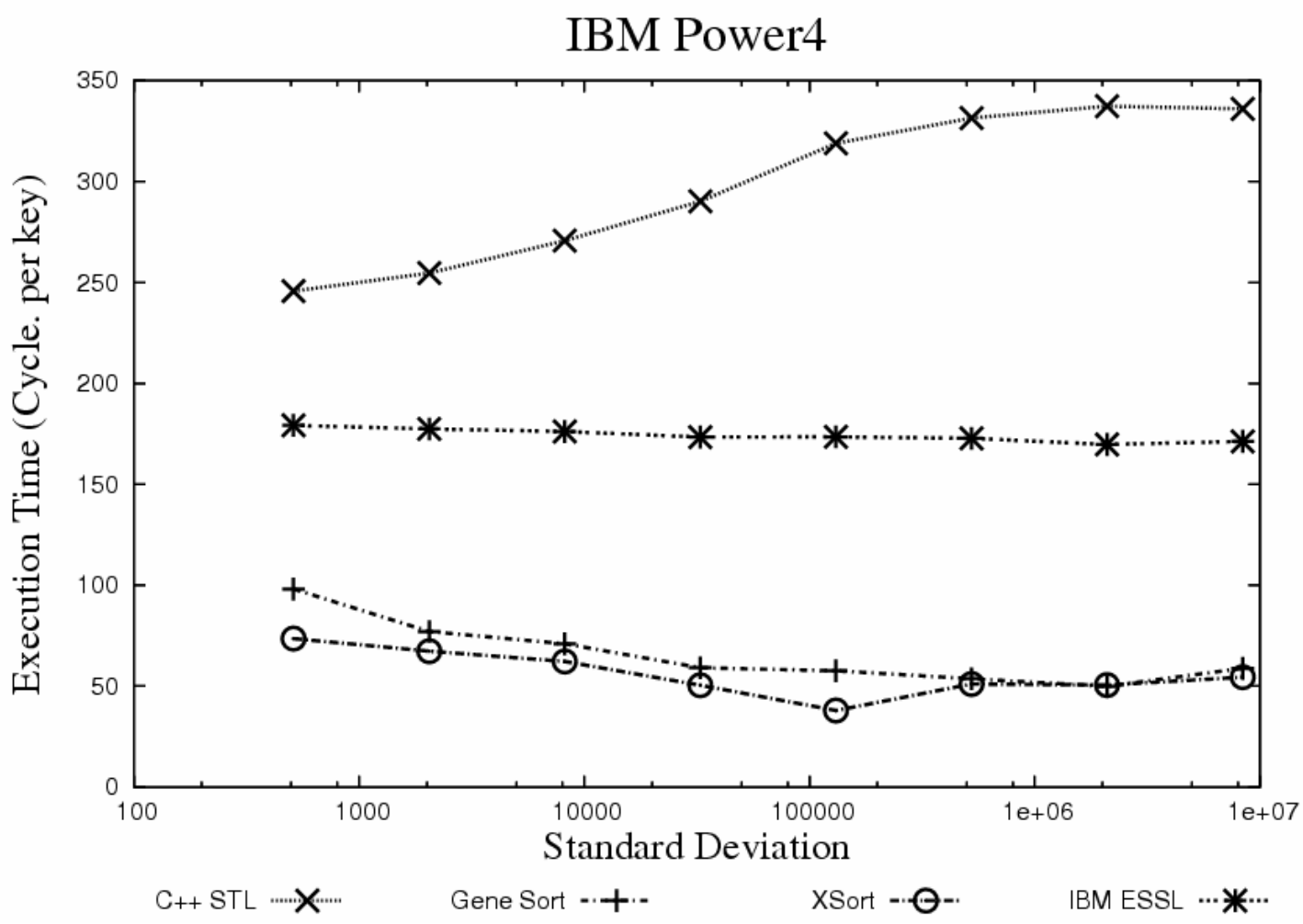
A better Solution

- 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



Power4


IBM Power4
Execution Time (Cycle. per key)
350
300
250
200
150
100
50
0
100
1000
10000
100000
1e+06
1e+07
Standard Deviation
C++ STL
Gene Sort
XSort
IBM ESSL

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.