



Multicore Chips and Parallel Programming

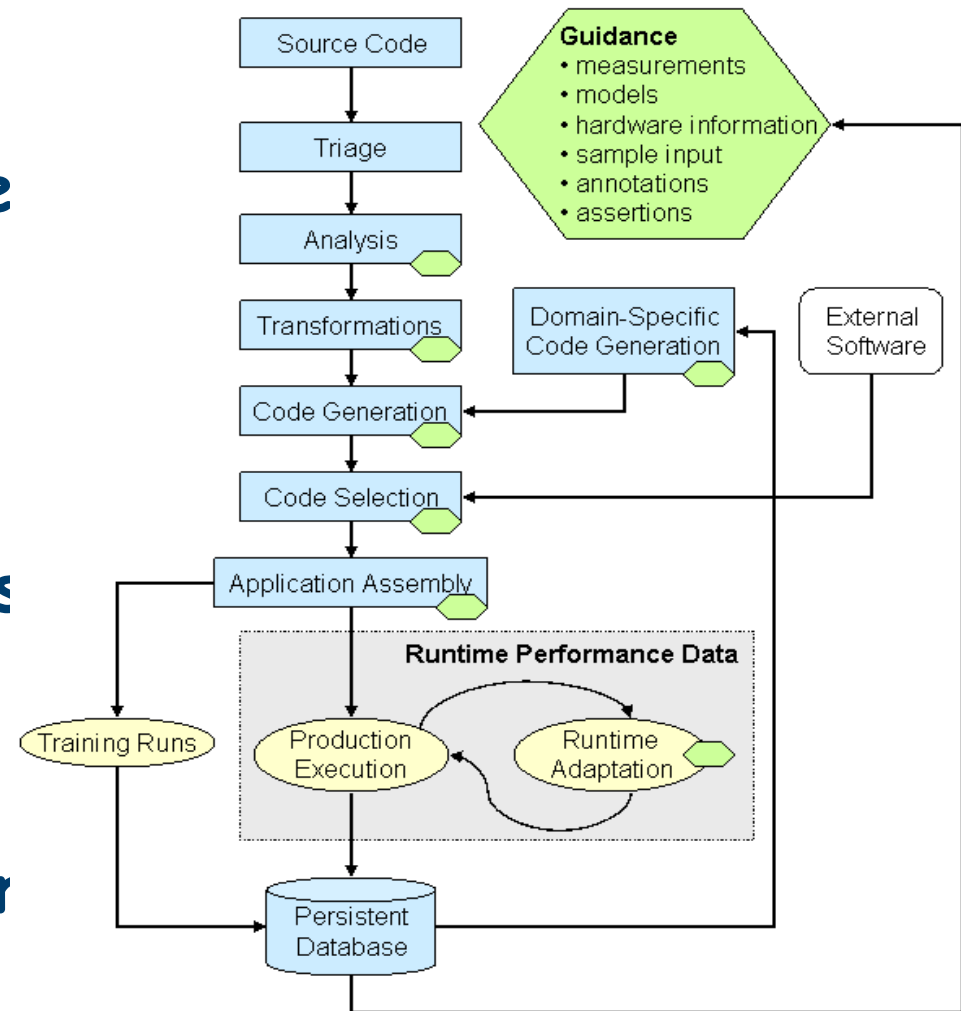
Mary Hall

USC/ISI



Context: Performance Engineering Research Institute (SciDAC-2)

- Long-term goal is to automate the process of tuning software to maximize its performance
- Reduces performance portability challenge for computational scientists.
- Addresses the problem that performance experts are in short supply.
- Builds on forty years of human experience and recent success with linear algebra libraries.

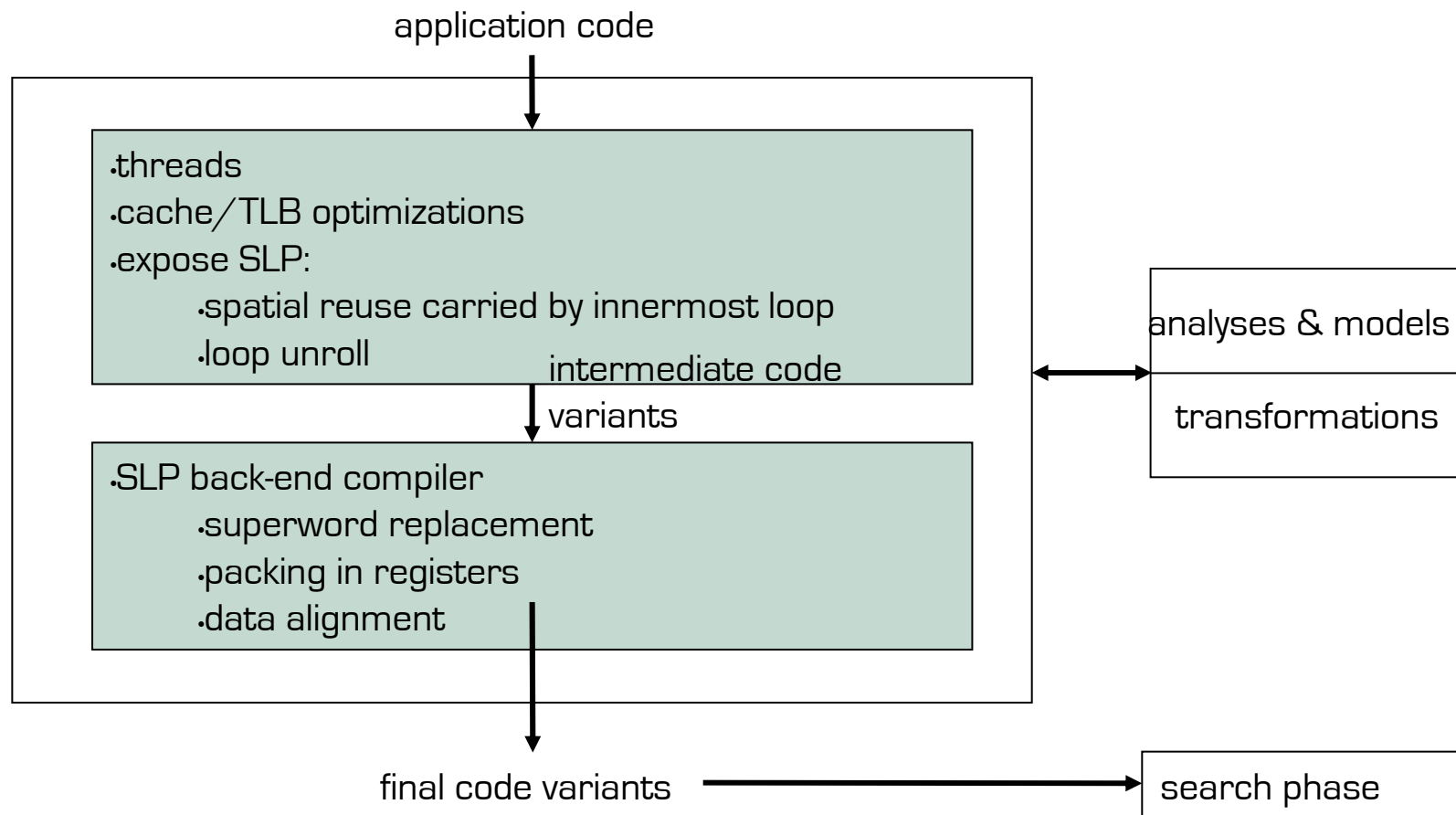


PERI automatic tuning framework

Slide source: Bob Lucas and David Bailey

Context: TUNE (3/15/08 start!)

- Goals:
 - High "socket" performance on Cray XT4
 - Locality, SSE and multi-core

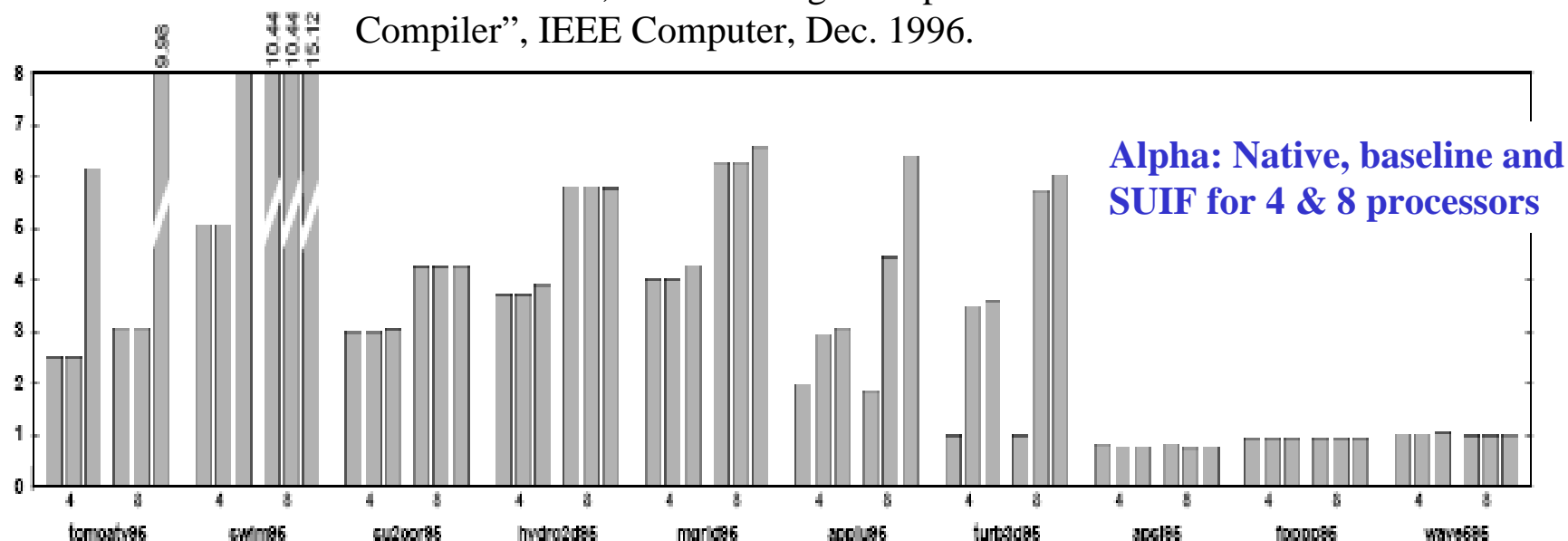


- **Convergence of high-end, conventional and embedded computing**
 - Application development and compilation strategies for high-end (supercomputers) are now becoming important for *the masses*
- **Looking to the future**
 1. Automatically generating parallel code is useful, but insufficient.
 2. Parallel computing for the masses demands better parallel programming paradigms.
 3. New tunable library and component technology.
 4. Compiler technology will become increasingly important to deal with a diversity of optimization challenge... and must be engineered for managing complexity and adapting to new architectures.
 5. Potential to exploit vast machine resources to automatically compose applications and systematically tune application performance.



1. Automatic Parallelization

From Hall et al., "Maximizing Multiprocessor Performance with the SUIF Compiler", IEEE Computer, Dec. 1996.



- Old approaches:
 - Limited to loops and array computations
 - Difficult to find sufficient *granularity* (parallel work between synchronization)
 - Success but from fragile, complex software
- New ideas in this area (e.g., August's work):
 - Finer granularity of parallelism -- more plentiful
 - Combine with hardware support (e.g., speculation and multithreading)

2. Parallel Programming

State of the Art

Three dominant classes of applications

Domains	Appl. Characteristics	Programming Paradigms
Scientific Computing	Very large arrays representing simulation region, loops, data parallel	MPI dominant, Also, OpenMP, PGAS Grids & distributed computing
Databases	Queries over large data sets, often distributed	Query languages like SQL
Systems and Embedded Software	Fine-grain threads, small number of processors	Low-level threading such as Pthreads, Ada

Domain-specific, intellectually challenging and low-level programming models not suitable for the masses.



2. New Parallel Programming Paradigms

- Transactional memory
 - Section of code executes *atomically* with subsequent *commit* or *rollback*
 - Programming model + hardware support
- Streams and data-parallel models
 - Data streams describe the flow of data
 - More general data parallel well-suited for certain applications and hardware (IBM Cell, GPUs)
- Domain-specific languages and libraries
 - Parallelism implicit within implementation

Different applications and users demand different solutions. Convergence unlikely. Architecture independence?



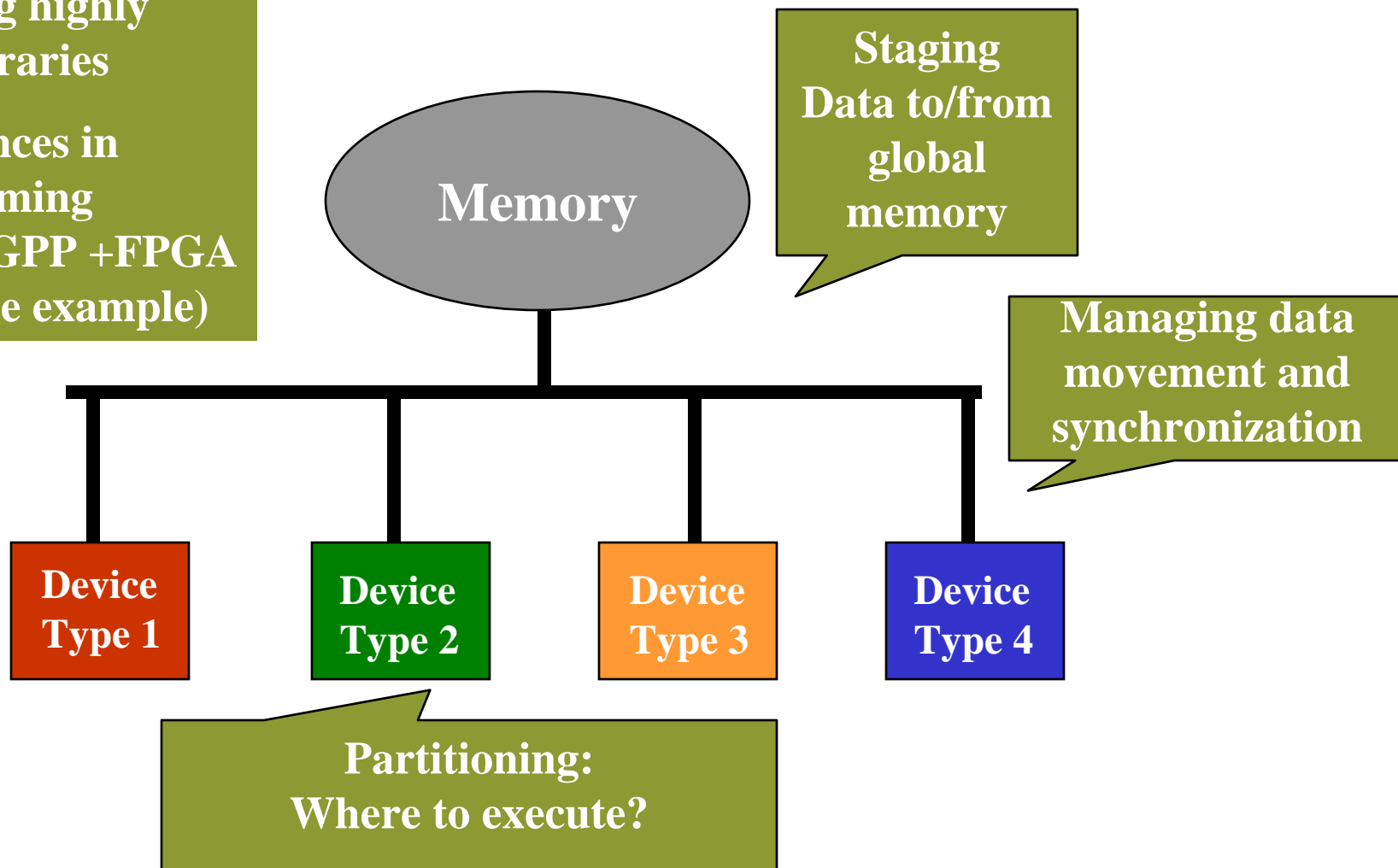
3. Libraries and Components

- Role of components (*e.g.*, CCA, SmartApps)
 - Encapsulate reusable software
 - Abstraction allows combining multiple different programming models
 - Uniform mechanism for heterogeneous programming
- How to obtain good performance?
 - Component describes a set of distinct implementations
 - Tune components *in context*

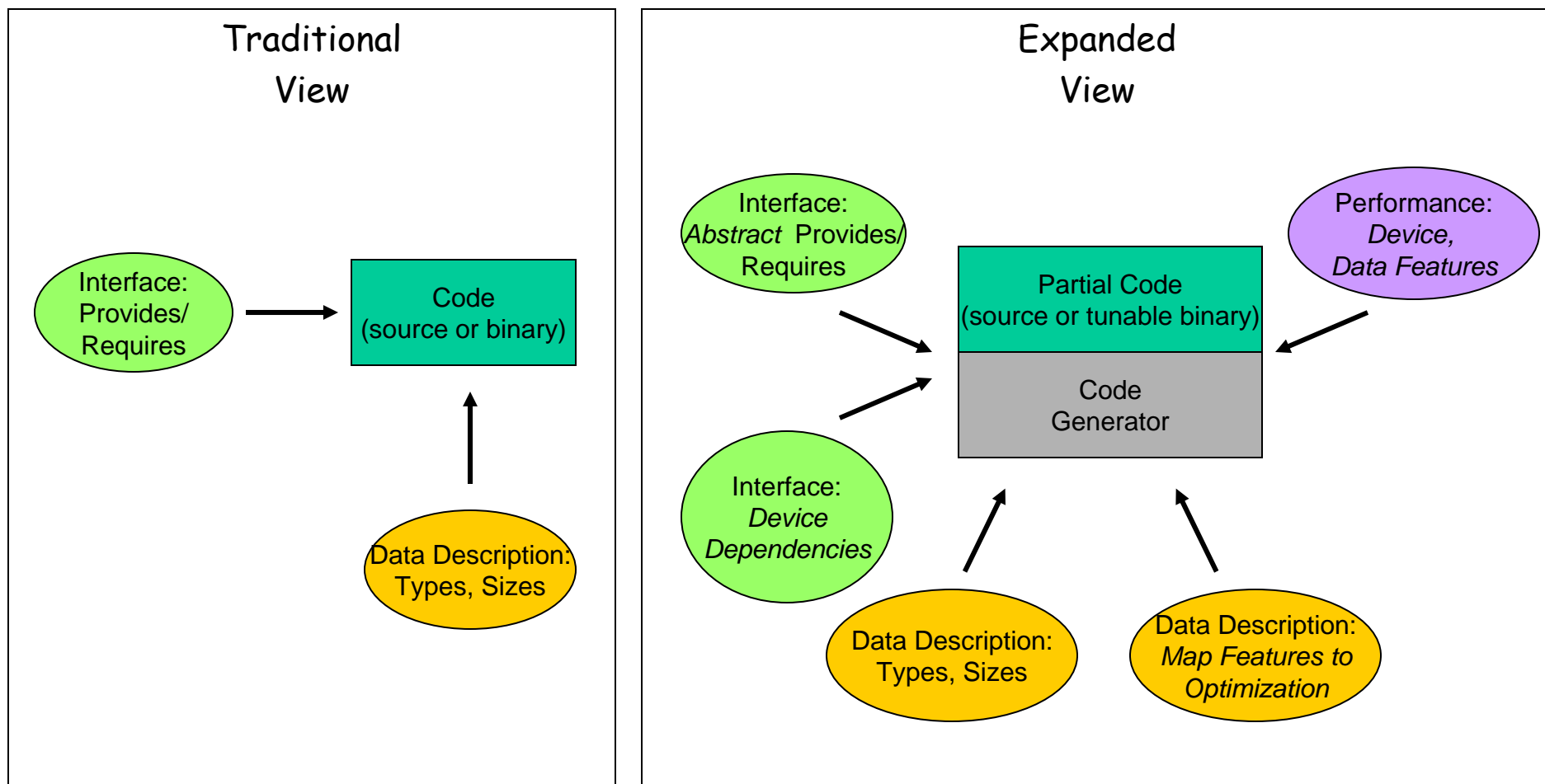
Heterogeneous: Additional Complexity beyond Multi-Core

Other:

- Utilizing highly tuned libraries
- Differences in programming models (GPP +FPGA is extreme example)



3. Libraries and Component Technology, cont.



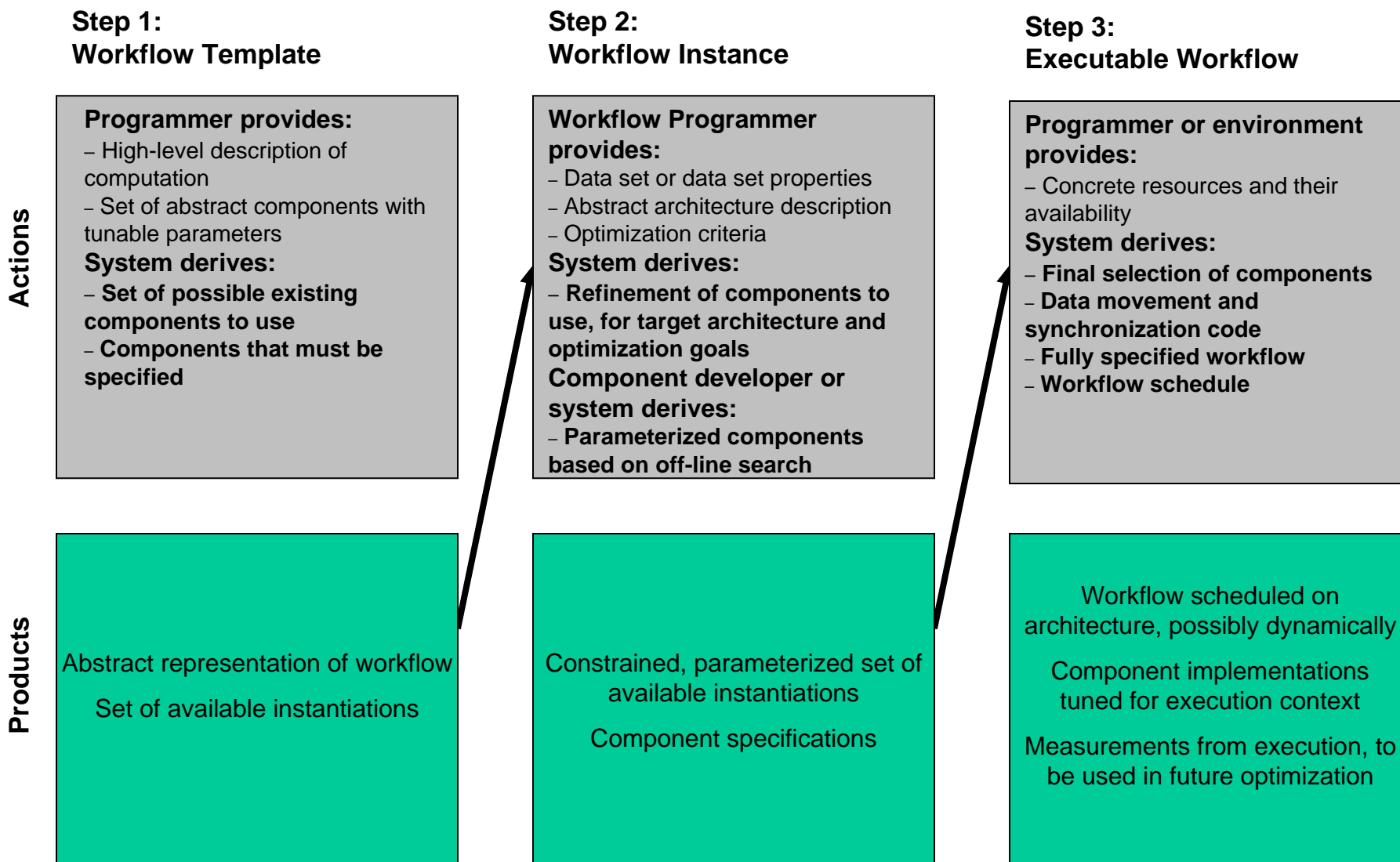
Support for automatic selection, tuning, scheduling, etc.

“Programming for Edge Computing: Using Cognitive Techniques to Manage Heterogeneous Resources,” M. Hall, Y. Gil and R. Lucas, Proceedings of the IEEE, Mar. 2008.



3. Composing Multi-Core

Applications Using "Workflow" Technology

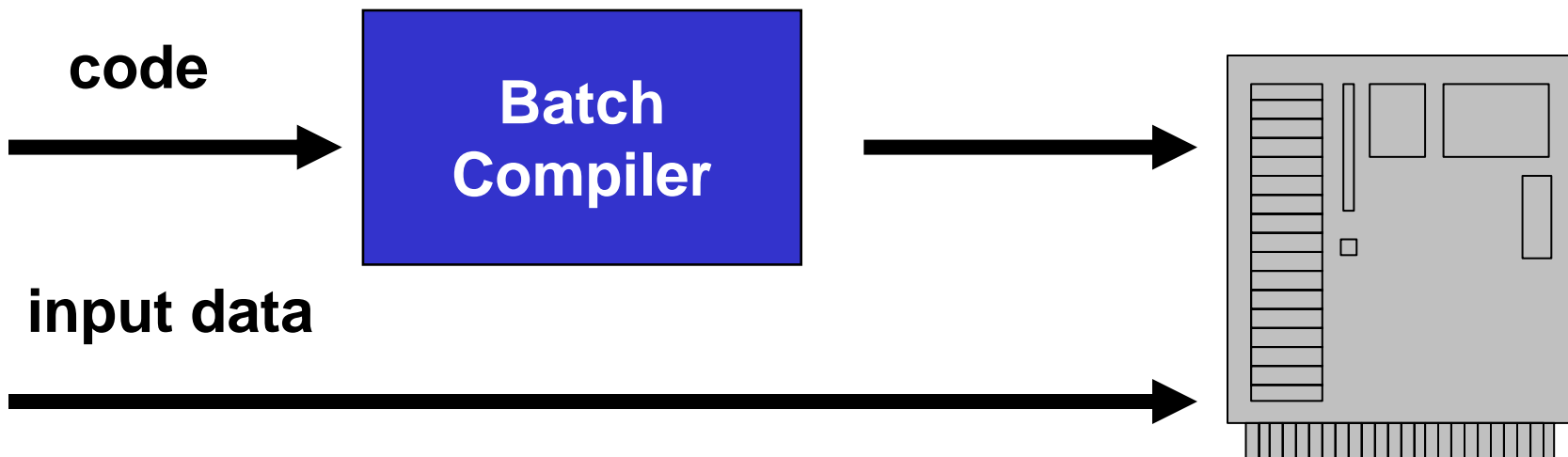




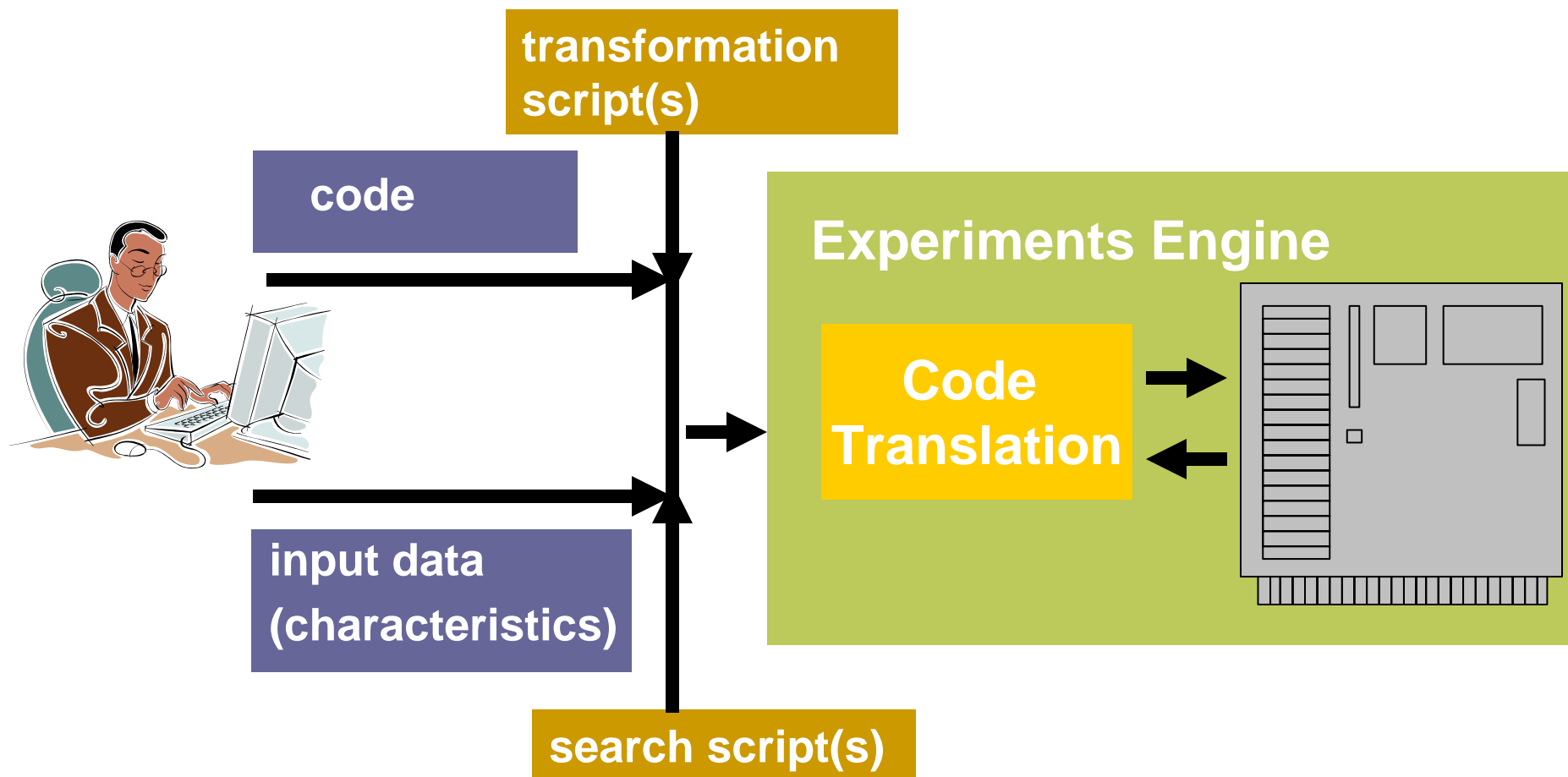
3. New Compiler Technology

- Compiler research will play a crucial role in achieving performance and programmability of multi-core hardware.
 - What is the state of compilers today?
 - Roughly 5 year lag between introducing a new architecture and a robust compiler
 - Many interesting new architectures fail in the marketplace due to inadequate software tools
 - Today's compilers are complex and somewhat monolithic
 - SUIF has ~500K LOC, Open64 has ~12M LOC
 - Difficult to incorporate new algorithms and compare approaches
- The best research ideas do not always make it into practice**

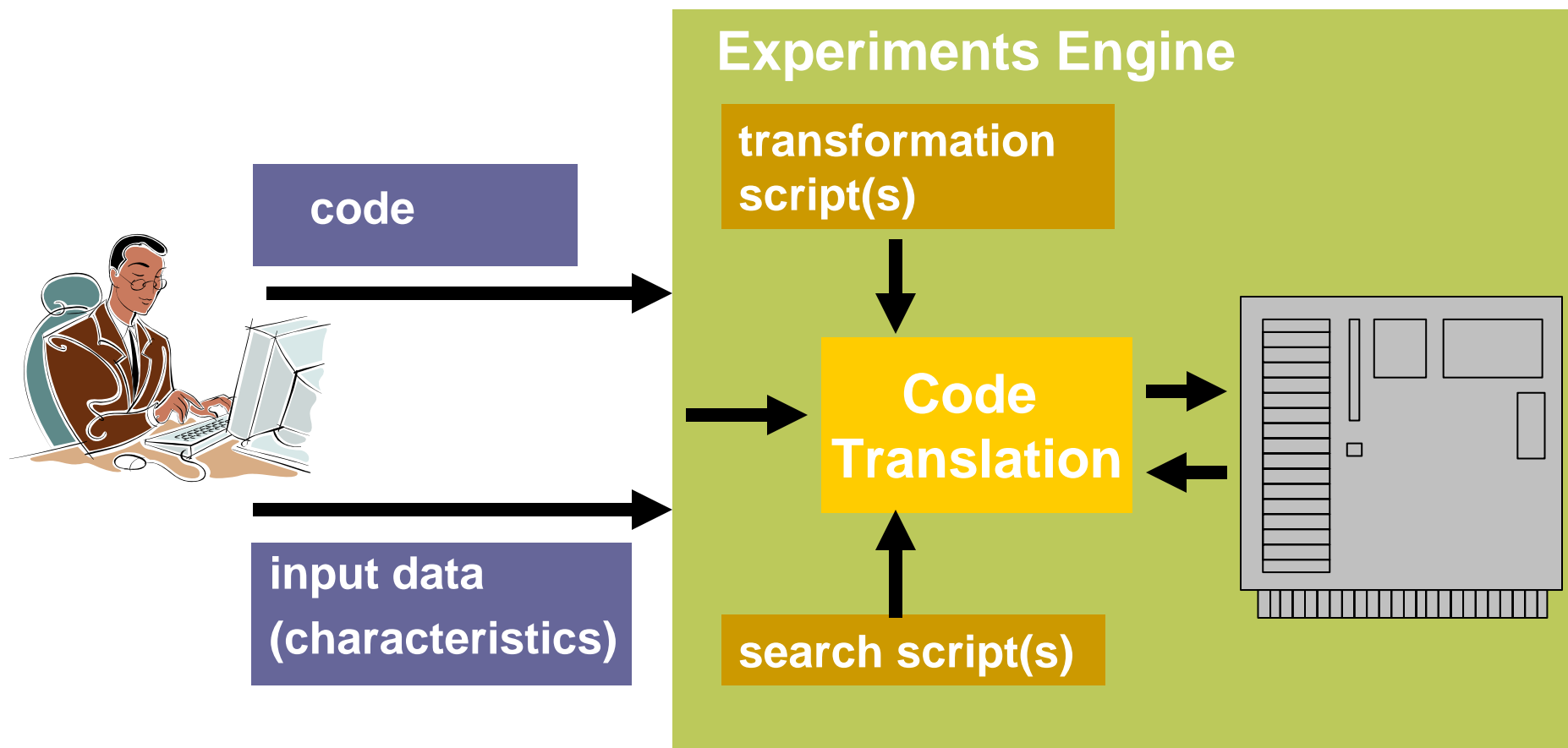
Traditional view:



4 & 5. Performance Tuning "Compiler"



5. Auto-tuners





Automatically-Generated Code Example

Original code

```
DO K = 1, N
  DO J = 1, N
    DO I = 1, N
      C(I,J) = C(I,J) + A(I,K) * B(K,J)
```

Transformation Script

```
permute([I,J,K])
tile(0,J,TJ, JJ)
tile(0,I,TI, II)
tile(0,K,TK, KK)
datacopy(0,A,Q,II,transpose)
datacopy(0,B,P,JJ)
unroll(0,I,UI)
unroll(0,J,UJ)
```

Automatically-generated code variant with locality optimizations

```
new P[TK,TJ]
new Q[TK,TI]
DO KK = 1,N,TK
  DO II = 1,N,TI
    Q[1:TK,1:TI] = A[II:II+TI-1,KK:KK+TK-1]
    DO JJ = 1,N,TJ
      P[1:TK,1:TJ] = B[KK:KK+TK-1,JJ:JJ+TJ-1]
      DO I = II,min(II+TI-1,N)
        DO J = JJ,min(JJ+TJ-1,N),4
          T1 = C[I,J]
          T2 = C[I,J+1]
          T3 = C[I,J+2]
          T4 = C[I,J+3]
          DO K = KK,min(KK+TK-1,N)
            T1 = T1+Q[K-KK+1,I-II+1]*P[K-KK+1,J-JJ+1]
            T2 = T2+Q[K-KK+1,I-II+1]*P[K-KK+1,J-JJ+2]
            T3 = T3+Q[K-KK+1,I-II+1]*P[K-KK+1,J-JJ+3]
            T4 = T4+Q[K-KK+1,I-II+1]*P[K-KK+1,J-JJ+4]
          C[I,J] = T1
          C[I,J+1] = T2
          C[I,J+2] = T3
          C[I,J+3] = T4
```



Automatically-Generated Code Example

Final result: output of SLP compiler with reduction transformation

```
new P[TK,TJ]
new Q[TK,TI]
DO KK = 1,N,TK
  DO II = 1,N,TI
    Q[1:TK,1:TI] = A[II:II+TI-1,KK:KK+TK-1]
    DO JJ = 1,N,TJ
      P[1:TK,1:TJ] = B[KK:KK+TK-1,JJ:JJ+TJ-1]
      DO I = II,min(II+TI-1,N)
        DO J = JJ,min(JJ+TJ-1,N),4
          T1[1:4] = {C[I,J], 0, 0, 0}
          T2[1:4] = {C[I,J+1], 0, 0, 0}
          T3[1:4] = {C[I,J+2], 0, 0, 0}
          T4[1:4] = {C[I,J+3], 0, 0, 0}
          DO K = KK,min(KK+TK-1,N),4
            T1[1:4] = madd(T1[1:4],Q[K-KK+1:K-KK+4,I-II+1]*P[K-KK+1:K-KK+4,J-JJ+1])
            T2[1:4] = madd(T2[1:4],Q[K-KK+1:K-KK+4,I-II+1]*P[K-KK+1:K-KK+4,J-JJ+2])
            T3[1:4] = madd(T3[1:4],Q[K-KK+1:K-KK+4,I-II+1]*P[K-KK+1:K-KK+4,J-JJ+3])
            T4[1:4] = madd(T4[1:4],Q[K-KK+1:K-KK+4,I-II+1]*P[K-KK+1:K-KK+4,J-JJ+4])
          C[I,J] = sum(T1[1:4])
          C[I,J+1] = sum(T2[1:4])
          C[I,J+2] = sum(T3[1:4])
          C[I,J+3] = sum(T4[1:4])
```



Pentium M:

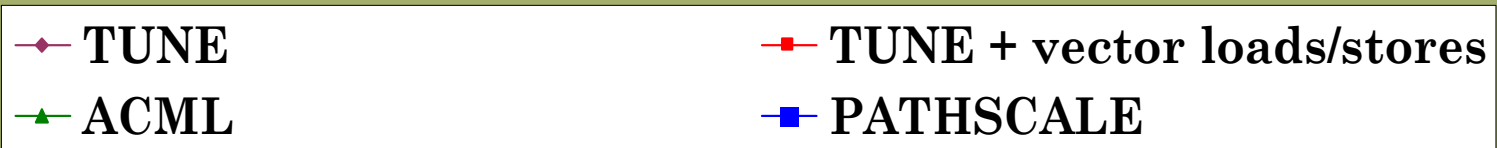
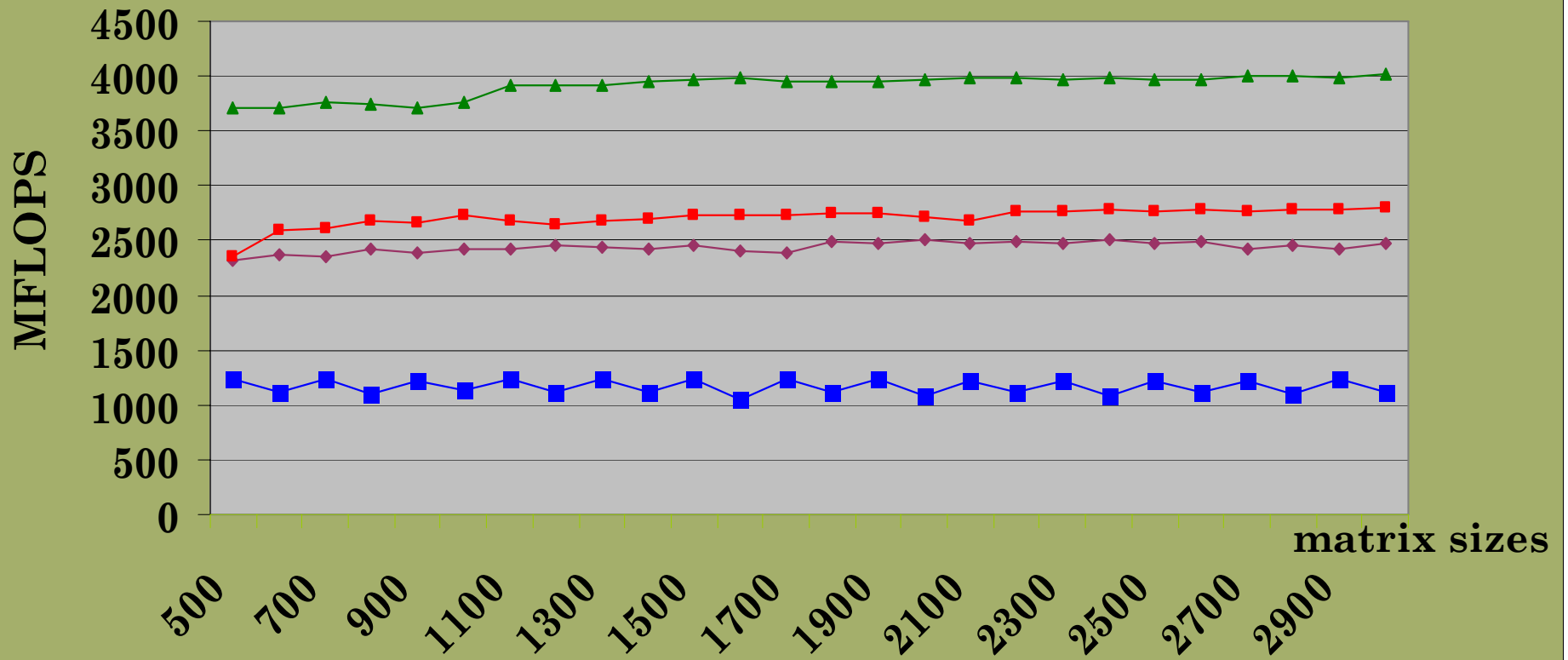
Combined Locality + SLP Compiler

```
do i
  do j
    do k
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
```

MM Version (3200x3200) Single Precision	Automatically- Generated	Intel MKL	ATLAS	Intel ifort compiler
Performance	2.957 Gflops	2.895 Gflops	3.076 Gflops	0.692 Gflops

“Model-Guided Empirical Optimization for Multimedia Extension Architectures: A Case Study,”
C. Chen, J. Shin, S. Kintali, J. Chame and M. Hall, Proceedings of the Workshop on Performance
Optimization of High-Level Languages, held in conjunction with IPDPS '07, March, 2007.

Matrix Multiply on Jacquard (NERSC)



TUNE: TUNE for locality, PATHSCALE for vectorization

ACML: hand-tuned vendor library

PATHSCALE: not vectorized (alignment issues)



Compiler-Generated Tunable Code

- Where compilers can beat libraries
 - PERI&TUNE: Auto-tuning of application code
 - Libraries used in unusual ways (e.g., MM on long, skinny matrices)
 - Composing library calls
- Where compilers can make programmers more productive in tuning their code
 - Search for best values of application-level parameters



Summary

- Parallel computing is everywhere!
 - And we need software tools
 - Can we find some common ground?
- Strategies
 - Automatic parallelization
 - New programming languages
 - Libraries and domain-specific tools that hide parallelism
→ **component technology**
 - Compiler technology
 - Auto-tuners to “test” alternative solutions
- General approach to solving challenges
 - **Education:** CS503 , Parallel Programming
 - Organize the community to support incremental LONG TERM development.