Compiler-Based Autotuning Technology

Lecture 5: Autotuning High-End Applications

Mary Hall

July, 2011

*** This work has been partially sponsored by DOE SciDAC as part of the Performance Engineering Research Institute (PERI), DOE Office of Science, the National Science Foundation, DARPA and Intel Corporation.**

THE UNIVERSITY OF UTAH

Motivation

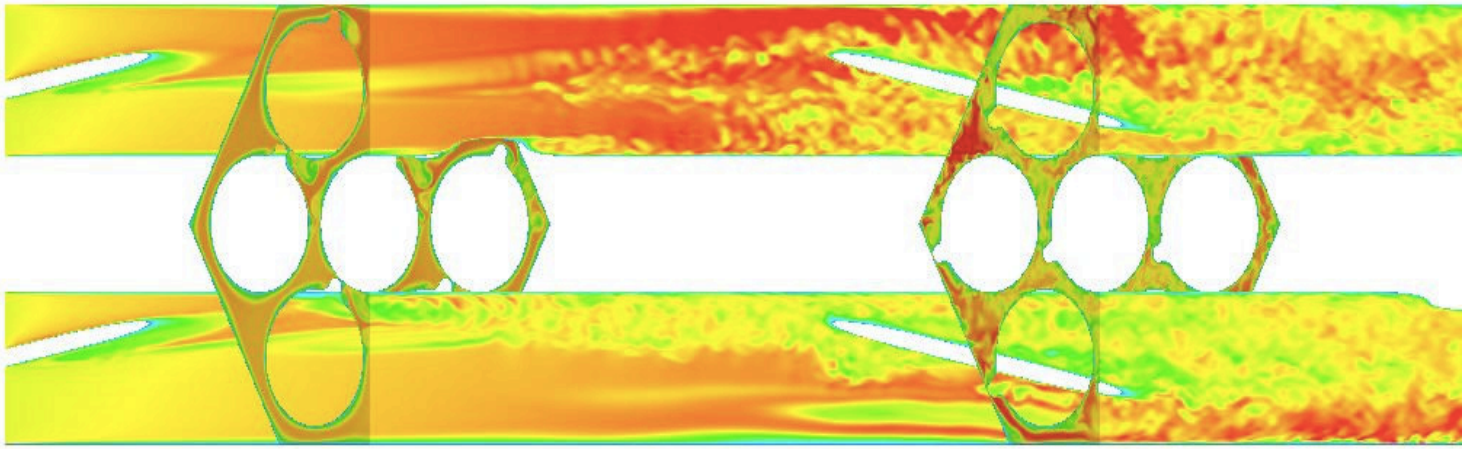
- Much of our focus is making compiler technology accessible to users seeking high performance
- Essential goal is to demonstrate impact on production scientific codes
- Today we look at a few examples
- Also, role of autotuning in future tens of petaflops and exaflops systems

Outline for Today's Lecture

1. Remainder of nek5000 discussion (from Tuesday)
2. Navigating large search spaces: smg2000
   - Combining CHiLL with Active Harmony
3. Library specialization for sparse computations: PFLOTRAN
4. Migrating applications to Exascale (and tens of Petaflops)

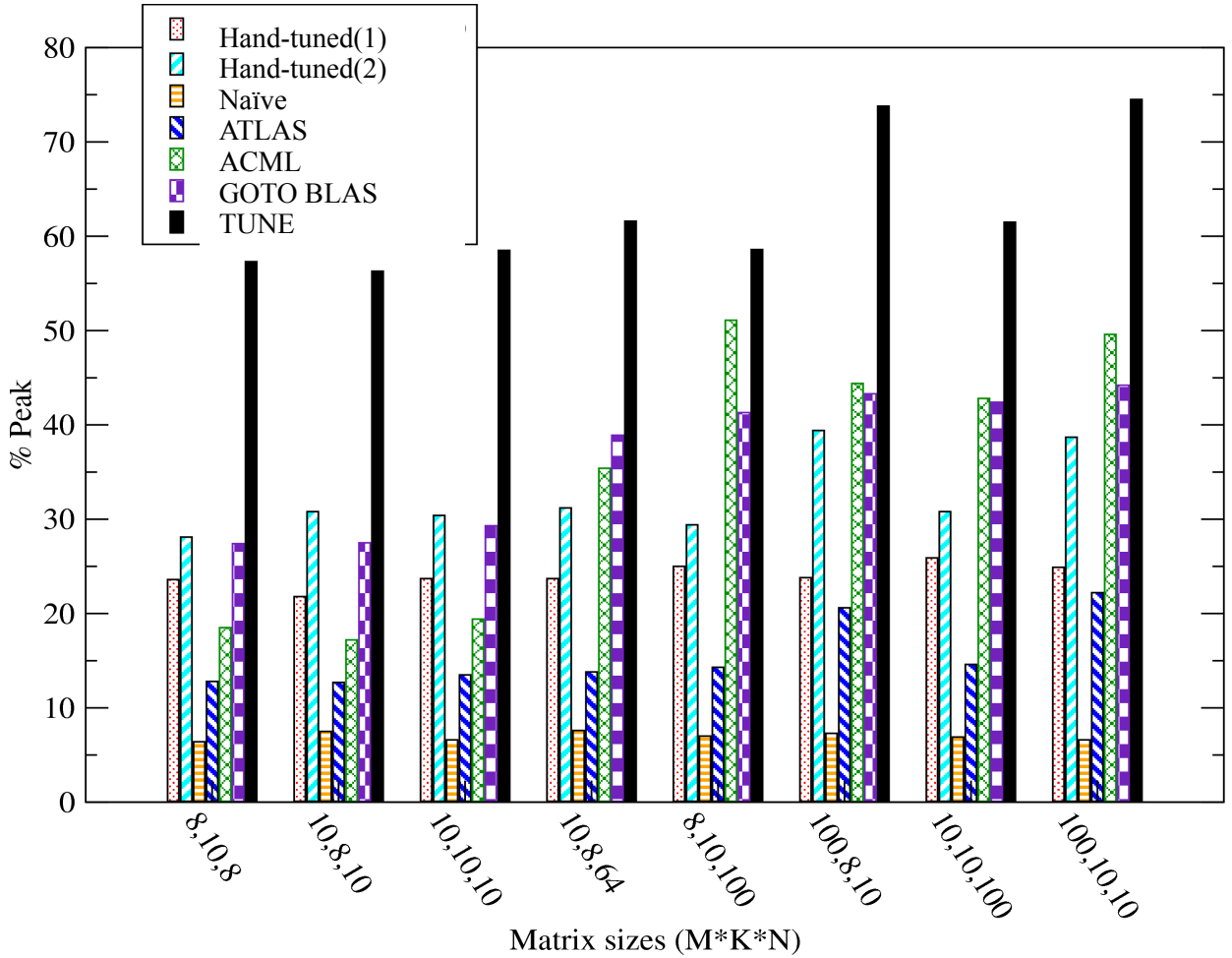
1. Autotuning of Nek5000

Spectral element code: turbulence in wire-wrapped subassemblies



- Applications: nuclear energy, astrophysics, ocean modeling, combustion, bio fluids, ...
- Scales to P > 10,000 (Cray XT5, BG/P)
- > 75% of time spent on manually optimized mxm
  - matrix multiply of very small, rectangular matrices
  - matrix sizes remain the same for different problem sizes

1. nek5000: Automatically-Generated Code is Faster than Manually-Tuned Libraries



2.2X speedup for DGEMM

Target architecture: AMD Phenom, 2.5 GHz, data fits in 64 KB L1,
4 double-precision floating point operations / cycle ➔ 10 GFlops / core peak

1. nek5000: Construct wrapper to select among specialized DGEMM kernels

```
(1) mxm(a, m, b, k, c, n){
(2)  if (all a, b and c are aligned to the SIMD register width){
(3)    if (k == 10){
(4)      if (m == 10){
(5)        if (n == 10){ m10_10_10(a,b,c); return;}
(6)        if (n == 100){ m10_10_100(a,b,c); return;}}
(7)      else if (n == 2){
(8)        if (m == 2){ m2_10_2(a,b,c); return;}
(9)        if (m == 4){ m4_10_2(a,b,c); return;}}
(10)     else if (m == 100 && n == 10){m100_10_10(a,b,c); return;}
(11)     else if (n == 16){
(12)       if (m == 16){ m16_10_16(a,b,c); return;}
(13)       if (m == 256){ m256_10_16(a,b,c); return;}}
(14)     else if (m == 16 && n == 100){m16_10_100(a,b,c); return;}}
(15)   else if (k == 2){
(16)     if (n == 10){
(17)       if (m == 10){ m10_2_10(a,b,c); return;}
(18)       if (m == 100){ m100_2_10(a,b,c); return;}}
(19)     else if (m == 10 && n == 88){m10_2_88(a,b,c); return;}}
(20)   else if (k == 16){
(21)     if (m == 16){
(22)       if (n == 16){ m16_16_16(a,b,c); return;}
(23)       if (n == 256){ m16_16_256(a,b,c); return;}}
(24)     if (m == 10){
(25)       if (n == 10){ m10_16_10(a,b,c); return;}
(26)       if (n == 256){ m10_16_256(a,b,c); return;}}
(27)     else if (m == 256 && n == 16){m256_16_16(a,b,c); return;}
(28)     else if (m == 100 && n == 10){m100_16_10(a,b,c); return;}}}
(29) mxm44_0(a, m, b, k, c, n);}
```

Wrapper calls ordered according to execution frequency to minimize overhead

Slide source: Jaewook Shin, ICS '10

1.nek5000: Higher-Level Kernels, Multiple Calls

```
do iz=1,10
  call mxm(A(1,1,iz),10,B,10,C(1,1,iz),10)
enddo
```

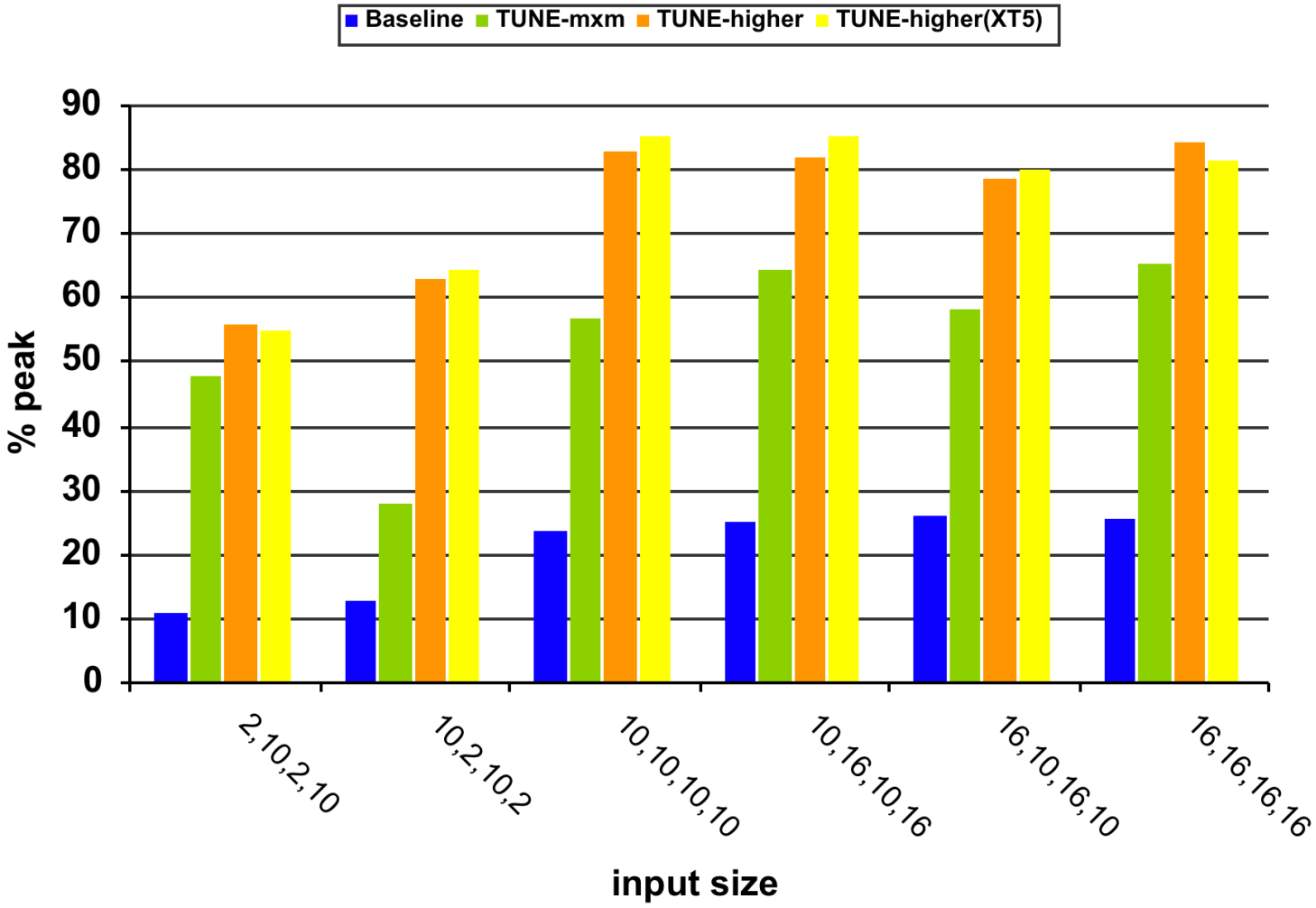
(a) original

```
do iz=1,10
   do i=1,10
      do j=1,10
         C(i,j,iz) = 0.0d0
         do l=1,10
            C(i,j,iz) = C(i,j,iz) + A(i,l,iz)*B(l,j)
         enddo
      enddo
   enddo
enddo
```

(b) inlined

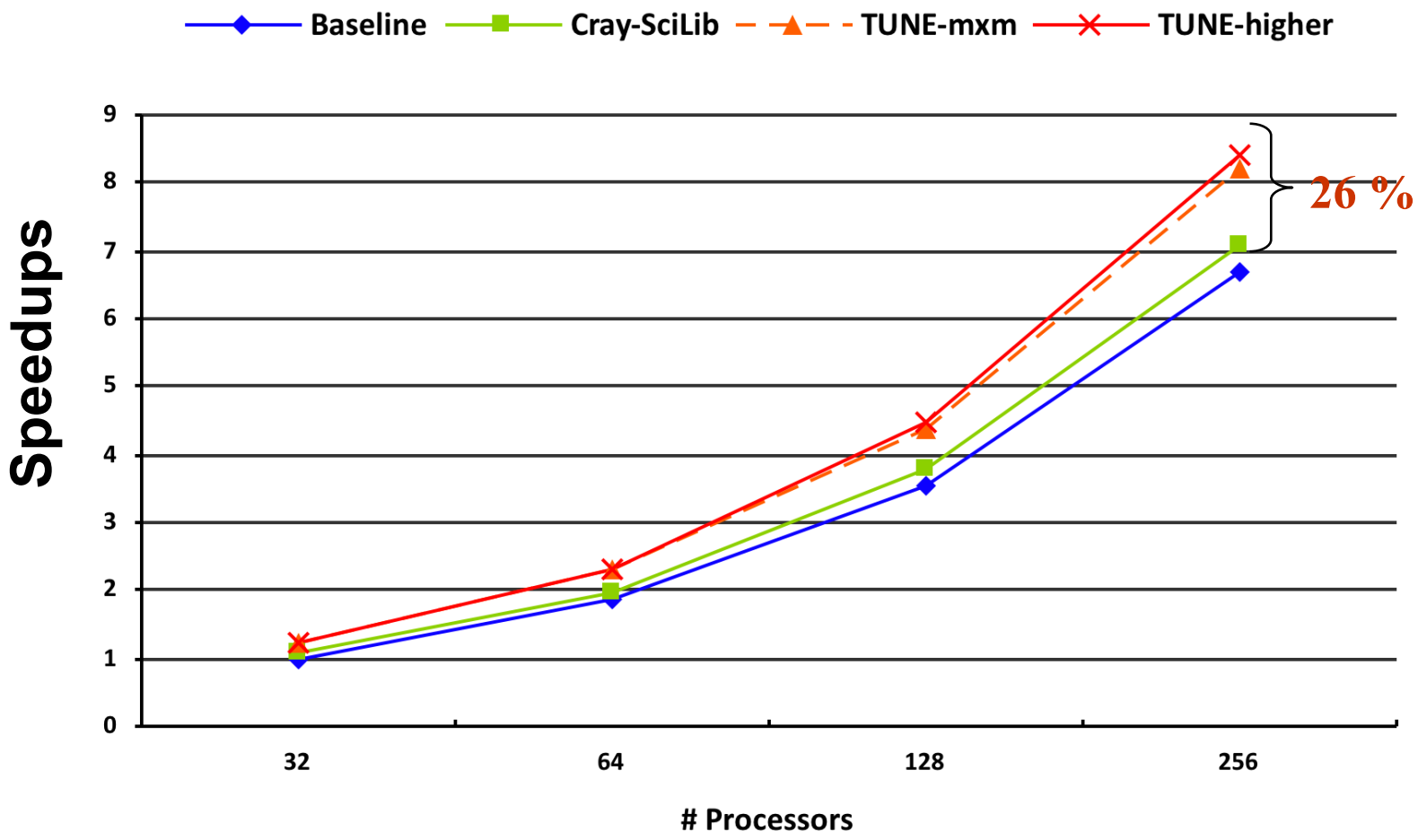
Slide source: Jaewook Shin, ICS '10

1. nek5000: Higher-Level Kernel Performance



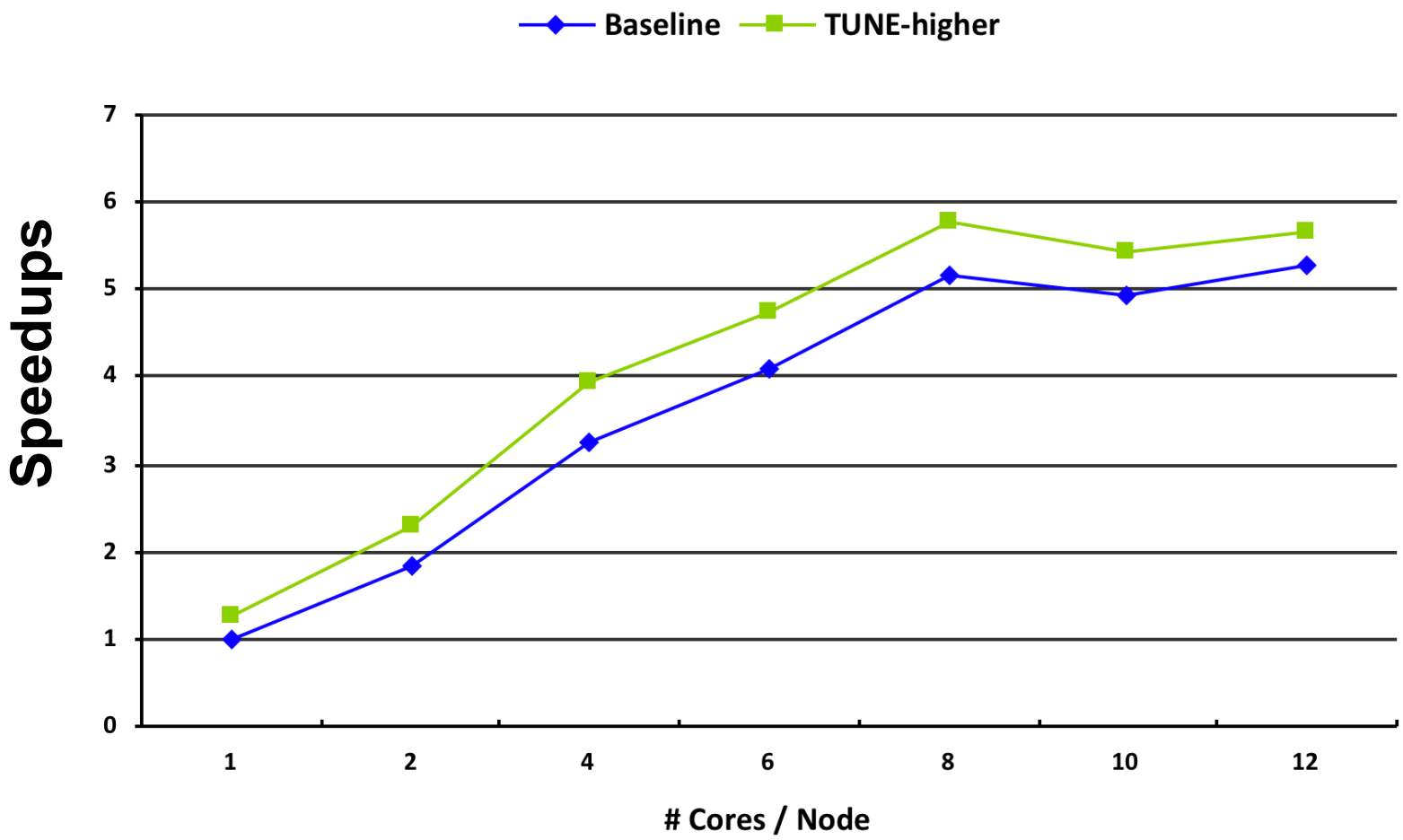
Slide source: Jaewook Shin, ICS '10

1. nek5000: (g6a input) on Jaguar



THE UNIVERSITY OF UTAH

1. Multiple Cores / Node: Nek5000 (g6a) on 32 nodes of Jaguar

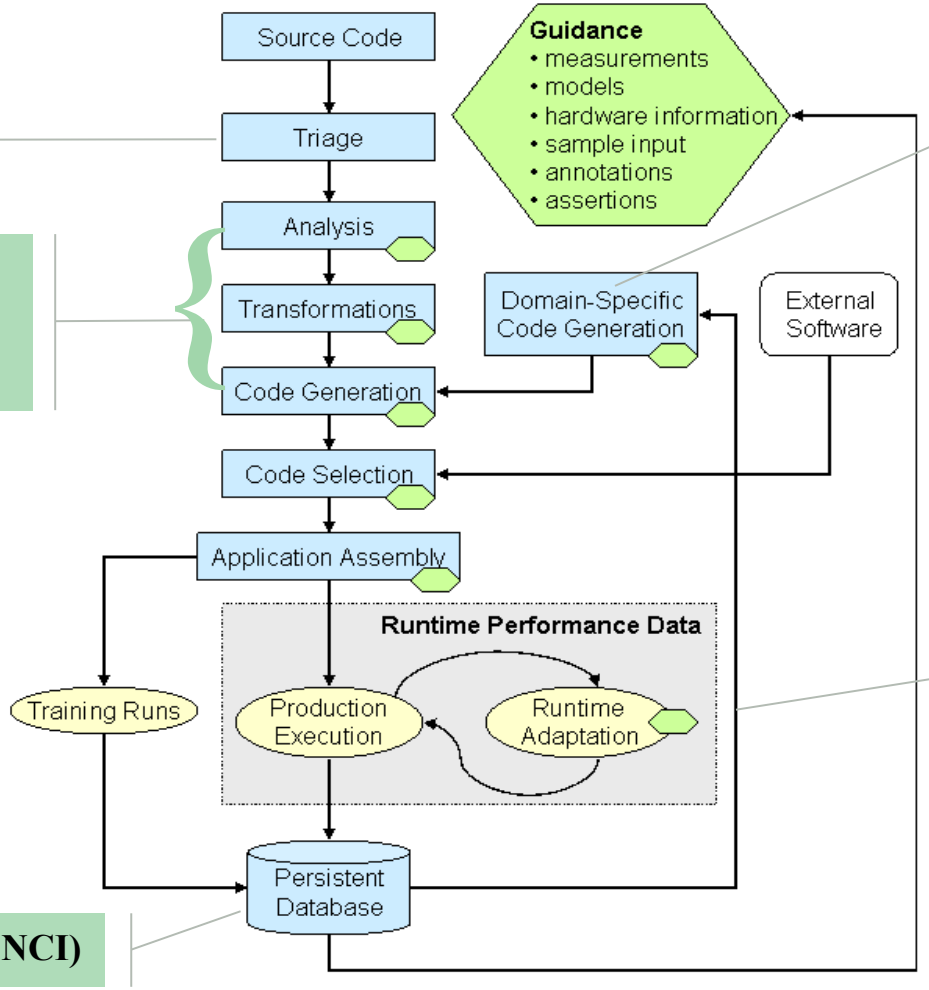


THE UNIVERSITY OF UTAH

2. Navigating Large Search Spaces

- Autotuning can lead to very large search spaces
- Overview of other tools in the performance tuning process
- Strategies in PERI for navigating large search spaces (Active Harmony)
  - Heuristic search to prune search space
  - Search multiple points in parallel
- Example code: SMG2000

2. (DOE SciDAC) PERI Autotuning Tools



HPC Toolkit (Rice)
ROSE (LLNL)

CHiLL (USC/ISI and Utah)
ROSE (LLNL)
Orio (Argonne)

OSKI (LBNL)

Active Harmony (UMD)
GCO (UTK)

PerfTrack (LBNL, SDSC, RENCI)

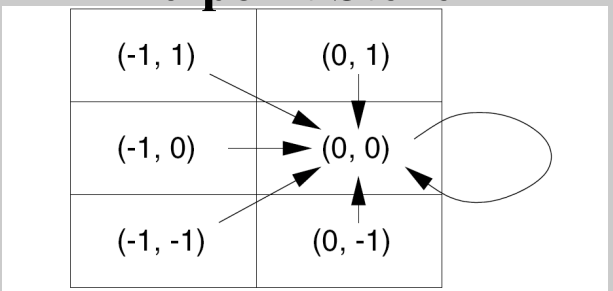
THE UNIVERSITY OF UTAH

2. Parallel Heuristic Search: SMG2000 Optimization in PERI

- Semi-coarsening multigrid on structured grids
  - Residual computation contains sparse matrix-vector multiply bottleneck, expressed in 4-deep loop nest
  - Key computation identified by HPCToolkit

```
for si = 0 to NS-1
  for k = 0 to NZ-1
    for j = 0 to NY-1
      for i = 0 to NX-1
        r[i + j*JR + k*KR] -=
              A[i + j*JA + k*KA + SA[si]]
            * x[i + j*JX + k*KX + Sx[si]]
```

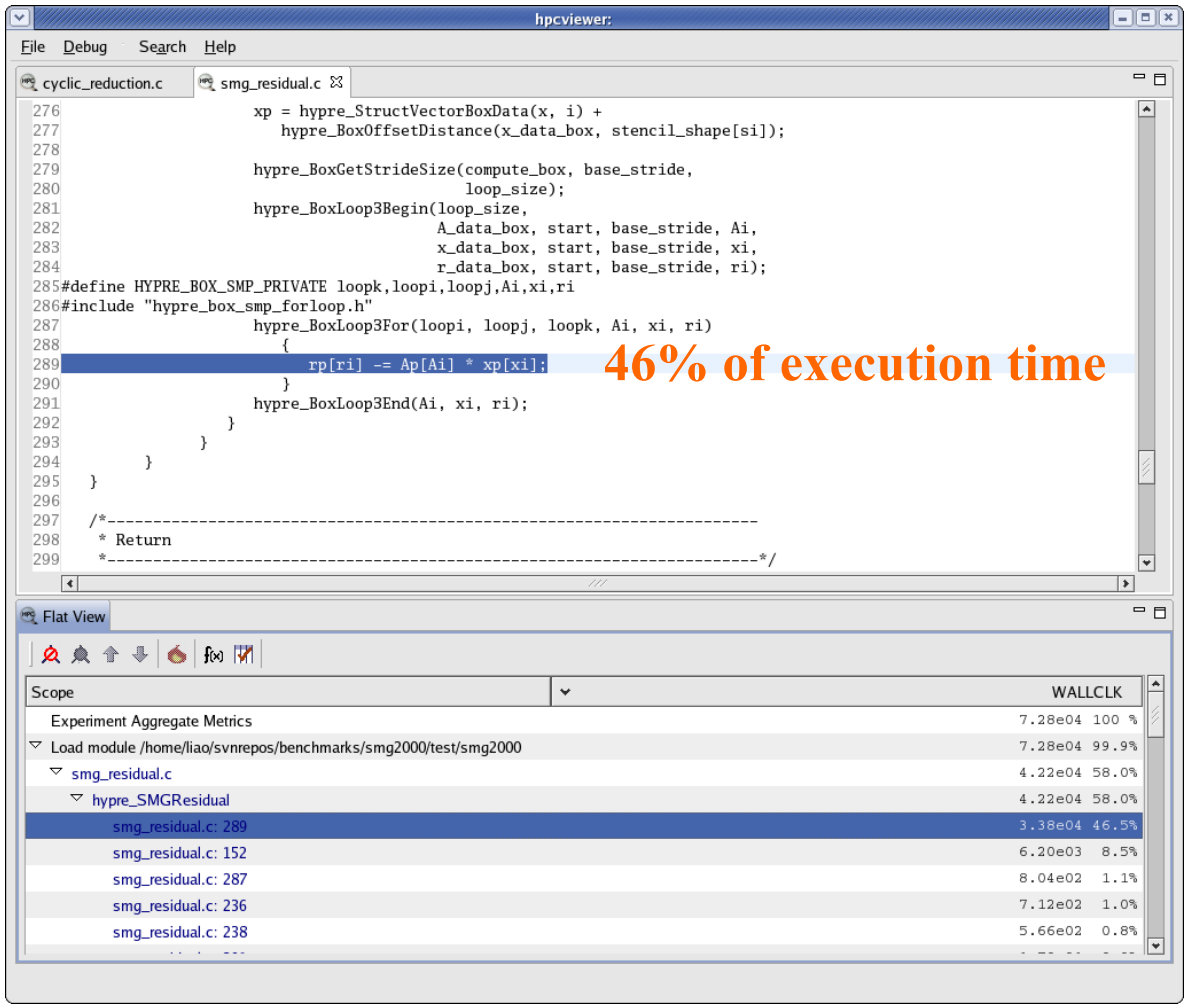
2D 6-point Stencil



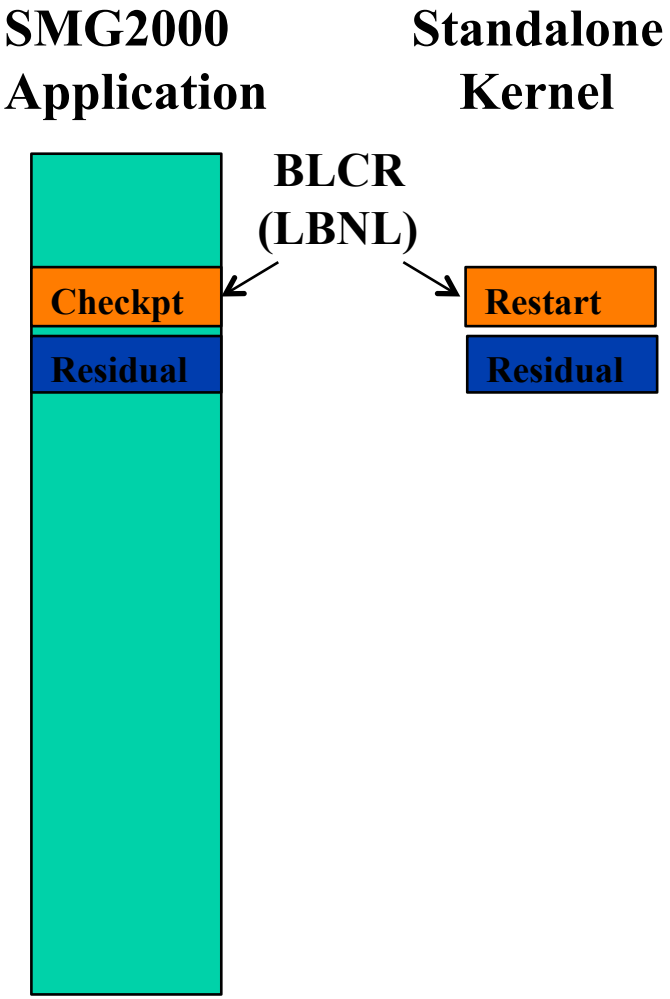
$S = \{(-1,1),(-1,0),(-1,-1),(0,1),(0,0),(0,-1)\}$

2. SMG2000: Triage Step Identifies Key Computation, Isolates into Standalone Executable

HPCToolkit



ROSE Outliner



THE UNIVERSITY OF UTAH

2. SMG2000 Kernel has Huge Optimization Search Space!

Outlined Code (from ROSE outliner)

```
for (si = 0; si < stencil_size; si++)
  for (kk = 0; kk < hypre__mz; kk++)
    for (jj = 0; jj < hypre__my; jj++)
      for (ii = 0; ii < hypre__mx; ii++)
        rp[((ri+ii)+(jj*hypre__sy3))+(kk*hypre__sz3)] -=
          ((Ap_0[((ii+(jj*hypre__sy1))+ (kk*hypre__sz1))+
           (((A->data_indices)[i])[si])])*
           (xp_0[((ii+(jj*hypre__sy2))+(kk*hypre__sz2))+(( *dxp_s)[si])]));
```

CHiLL Transformation Recipe

```
permute([2,3,1,4])
tile(0,4,TI)
tile(0,3,TJ)
tile(0,3,TK)
unroll(0,6,US)
unroll(0,7,UI)
```

Constraints on Search

$0 \leq TI, TJ, TK \leq 122$

$0 \leq UI \leq 16$

$0 \leq US \leq 10$

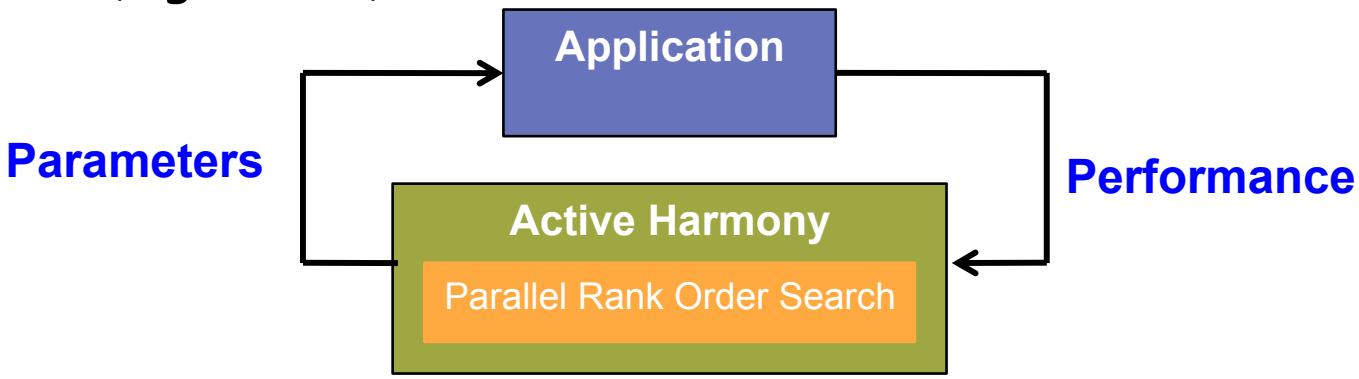
compilers $\in$ {gcc, icc}

Search space:

$122^3 \times 16 \times 10 \times 2 = 581M$ points

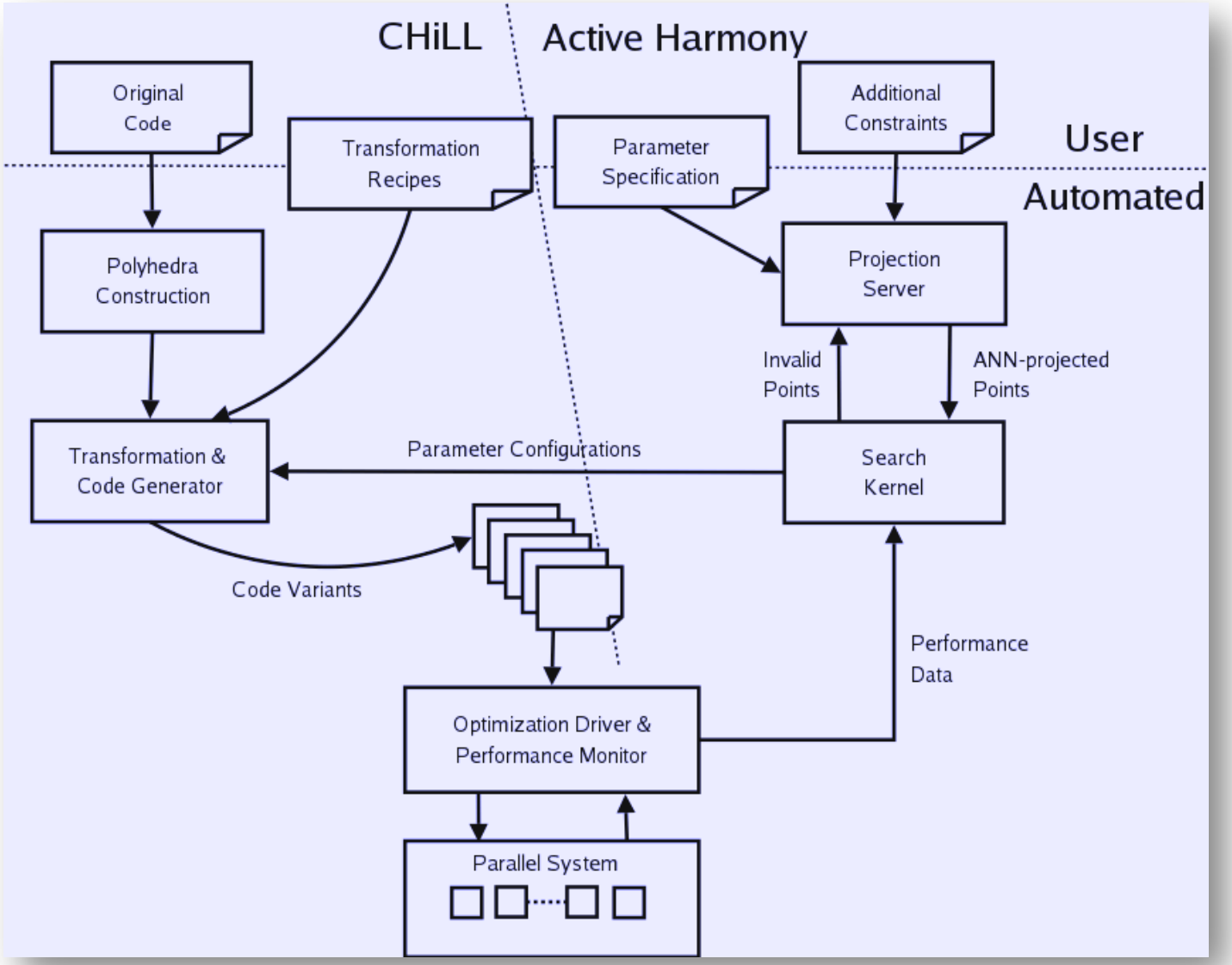
2. From Lecture 1: Application-level tuning using Active Harmony

- Search-based collaborative approach
  - Simultaneously explore different tunable parameters to search a large space defined by the user
    - e.g., Loop blocking and unrolling factors, number of OpenMP threads, data distribution algorithms, granularity controls, …
  - Supports both online and offline tuning
  - Central controller monitors performance, adjusts parameters using search algorithms, repeats until converges
  - Can also generate code on-demand for tunable parameters that need new code (e.g. unroll factors) using code transformation frameworks (e.g. CHiLL)



Slide source: Ananta Tiwari

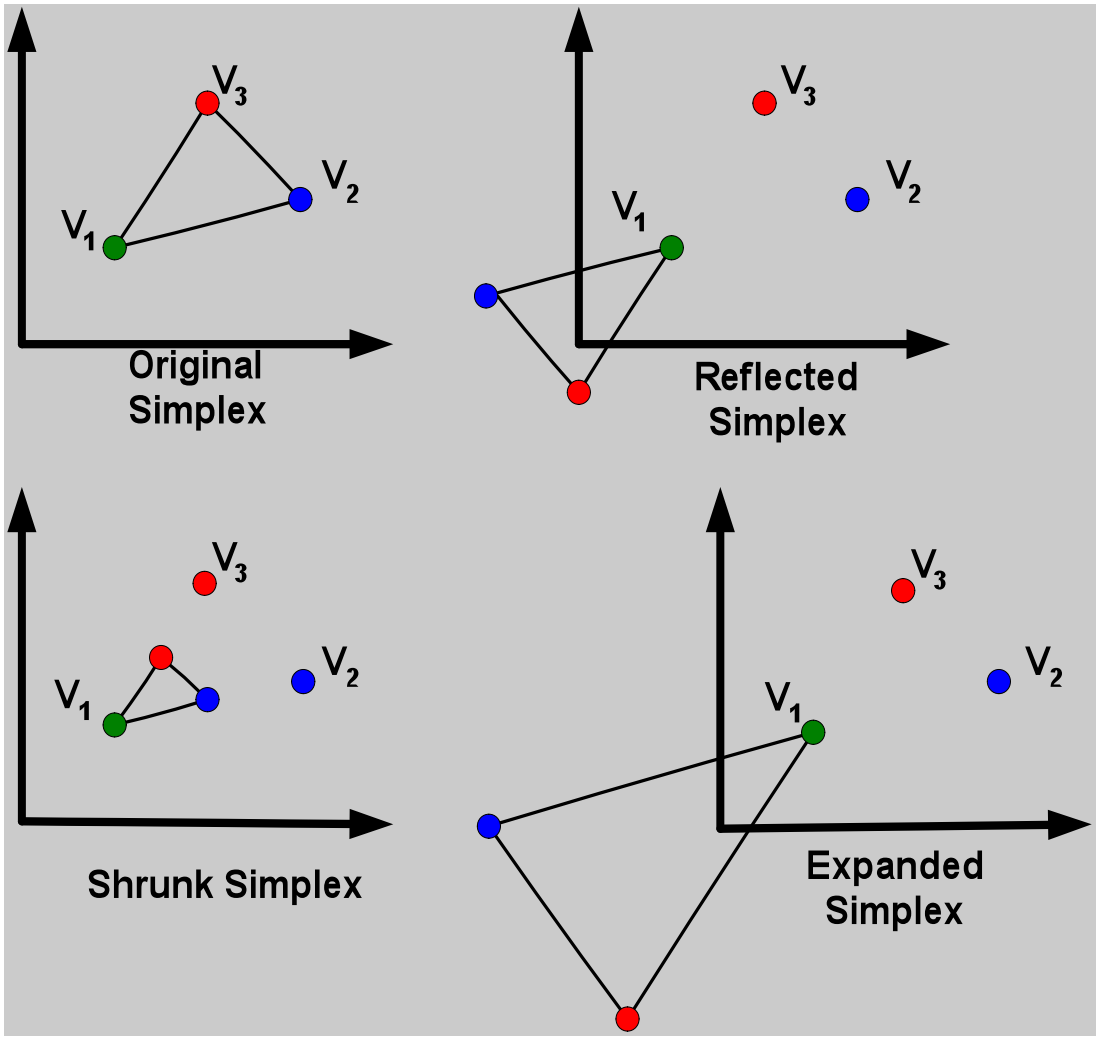
2. SMG2000 Workflow and Integration



Slide source: Ananta Tiwari

2. Recall Active Harmony Parallel Rank Order Algorithm

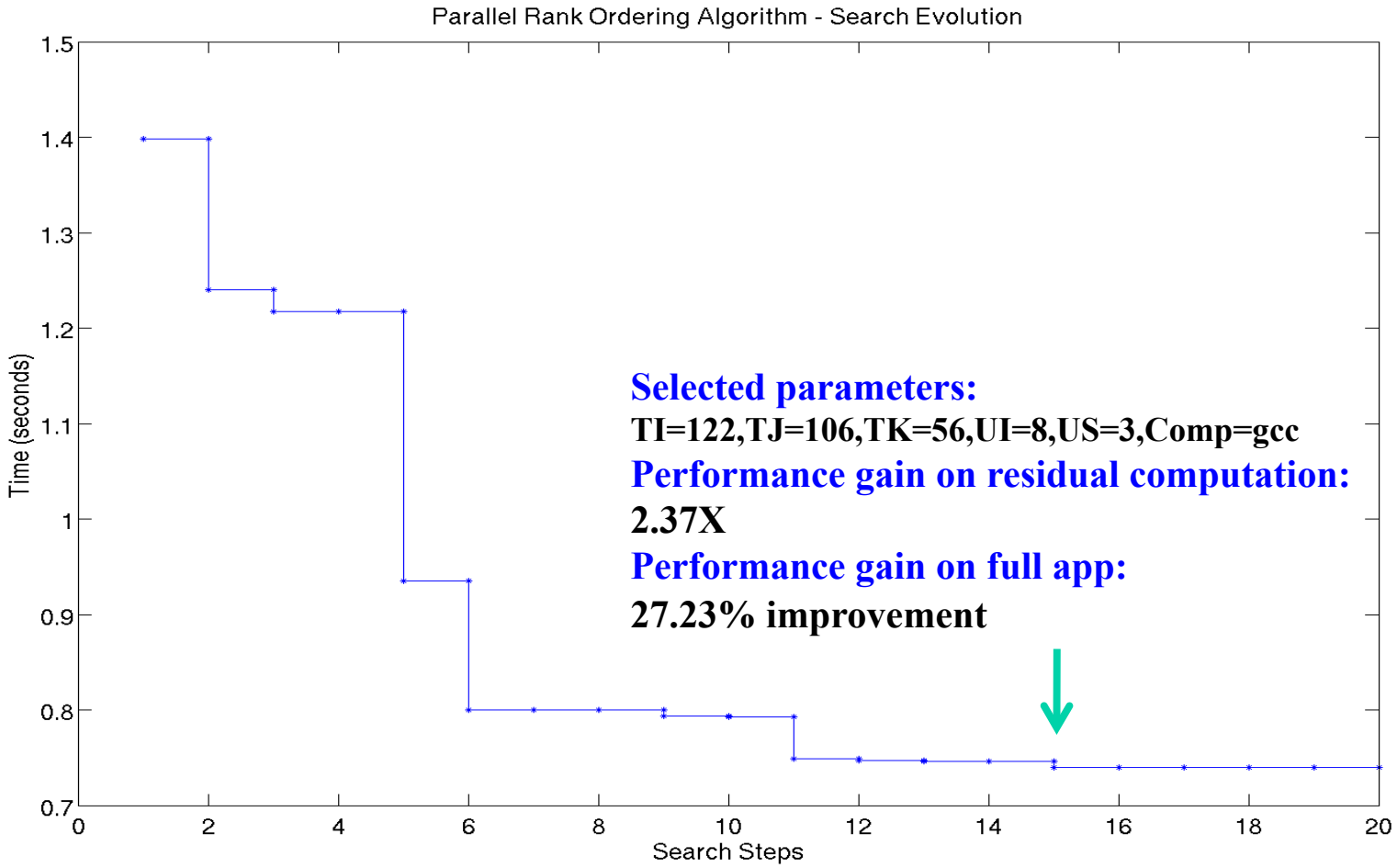
- All, but the best point of simplex moves
- Computations can be done in parallel
- N parallel evaluations for N+1 point simplex



Slide source: Ananta Tiwari

Parallel Heuristic Search Converges Rapidly for SMG2000

Parallel heuristic search (Active Harmony) evaluates 490 points and converges in 20 steps.



Slide source: Ananta Tiwari

2. Extending to On-line Tuning

- On-line tuning that combines CHiLL and Active Harmony is described in the following recent paper:

  "Online Adaptive Code Generation and Tuning," Ananta Tiwari and Jeff Hollingsworth, Proceedings of the International Parallel and Distributed Processing Symposium (Best Paper, Software), May 2011.

- CHiLL scripts are instantiated dynamically, triggering dynamic code generation and linking
- Active Harmony monitors availability of new code variants, identifies improved versions, and patches into running program when appropriate
- Tunes just one computation in a program at a time

On-line tuning profitable, even though overheads are high

3. PFloTran: Optimizing PETSc Sparse Linear Algebra

- PFloTran models Multiscale-Multiphase-Multicomponent Subsurface Reactive Flows (groundwater modeling)
- PETSc routines comprise 25% of execution time and achieve 4% of peak on Jaguar
  - Example: **MatSolve_SeqBAIJ_N**
  - Represents sparse matrix as collection of dense blocks
  - Large number of different implementations specialized for different block sizes

3. PFloTran: Optimizing Triangular Solve

Outlined Code

```
#define SIZE 15
void forward_solve_kernel( … )  {
  ….
  for (cntr = SIZE - 1; cntr >= 0; cntr--) {
        x[cntr] = t + bs * (*vi ++);
        for (j=0; j<bs; j++)
           for (k=0; k<bs; k++)
                s[k]-= v[cntr][bs* j+k] * x[cntr][j];
   }
}
```

CHiLL Transformation Recipe

```
original()
known(bs = 15)
unroll(1,2,u1)
unroll(1,3,u2)
```

Constraints on Search

0 <= u1 <= 16

0 <= u2 <= 16

compilers ∈ {gnu, pathscale, cray, pgi}

Search space:

17x17x4 = 1156 points

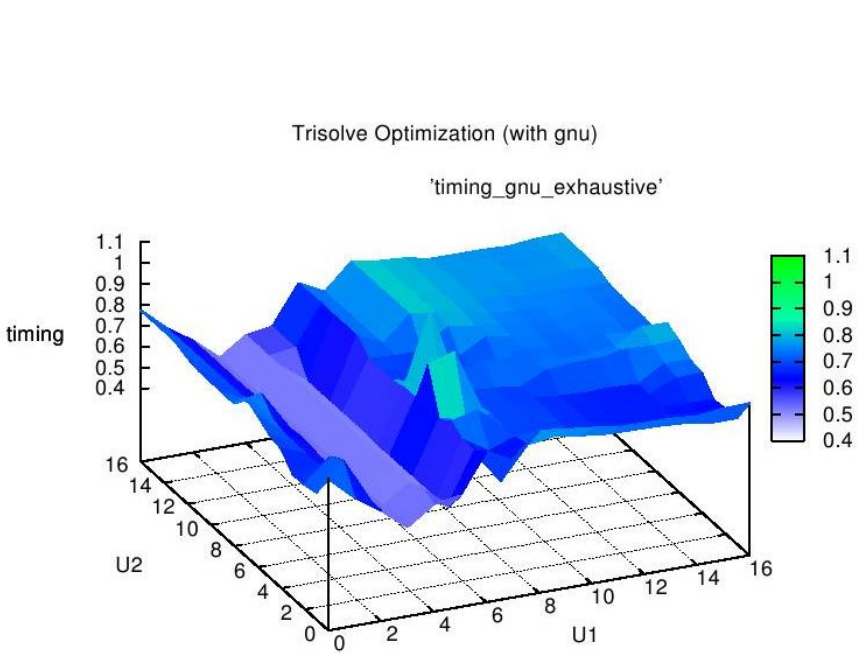
Slide source: Jeff Hollingsworth, SciDAC '10

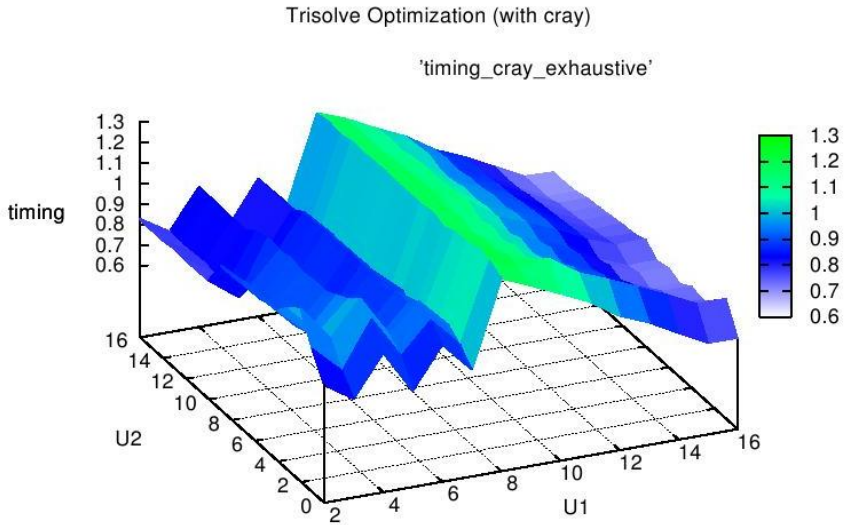
3. PFloTran: A Few Words on Strategy for Optimizing Triangular Solve

- Data is organized into set of dense blocks (blocked sparse row)
- Zeros may be computed inside block
- Allows for very fast computation within a block (looks like a dense computation) at the cost of additional storage
- Optimization strategy similar to dense linear algebra for small matrices

3. PFloTran Triangular Solve Results (Active Harmony + CHiLL)

Compiler	Original	Active Harmony			Exhaustive		
	Time	Time	(u1,u2)	Speedup	Time	(u1,u2)	Speedup
pathscale	0.58	0.32	(3,11)	1.81	0.30	(3,15)	1.93
gnu	0.71	0.47	(5,13)	1.51	0.46	(5,7)	1.54
pgi	0.90	0.53	(5,3)	1.70	0.53	(5,3)	1.70
cray	1.13	0.70	(15,5)	1.61	0.69	(15,15)	1.63





Slide source: Jeff Hollingsworth, SciDAC '10

4. Getting to Exascale: DARPA Exascale Technology Reports

Georgia Institute of Technology

See http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECS_reports.htm

Exascale Computing Study Report

Beginning in mid-2007, DARPA/IPTO, contracting through AFRL, has sponsored a series of studies intended to understand the future course of mainstream computing technology and determine whether or not it would allow a 1,000X increase in the computational capabilities of computing systems by the 2015 time frame. Where current technology trends were deemed incapable of achieving such increases, the study was also charged with identifying the major challenges and the areas where additional targeted research could lay the groundwork for overcoming them.

The following report of the first Exascale Computing Study has been publicly released. Reports of additional exascale computing studies will be made available here as they are also publicly released.

- ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems (September 28, 2008)

  Approved for Public Release, Distribution Unlimited

  The material in this document reflects the collective views, ideas, opinions and findings of the study participants only, and not those of any of the universities, corporations, or other institutions with which they are affiliated. Furthermore, the material in this document does not reflect the official views, ideas, opinions and/or findings of DARPA, the Department of Defense, or of the United States government.

- ExaScale Computing Software Study: Software Challenges in Extreme Scale Systems (September 14, 2009)

  Approved for Public Release, Distribution Unlimited

  The material in this document reflects the collective views, ideas, opinions and findings of the study participants only, and not those of any of the universities, corporations, or other institutions with which they are affiliated. Furthermore, the material in this document does not reflect the official views, ideas, opinions and/or findings of DARPA, the Department of Defense, or of the United States government.

---

4. What's Different about Exascale Computing

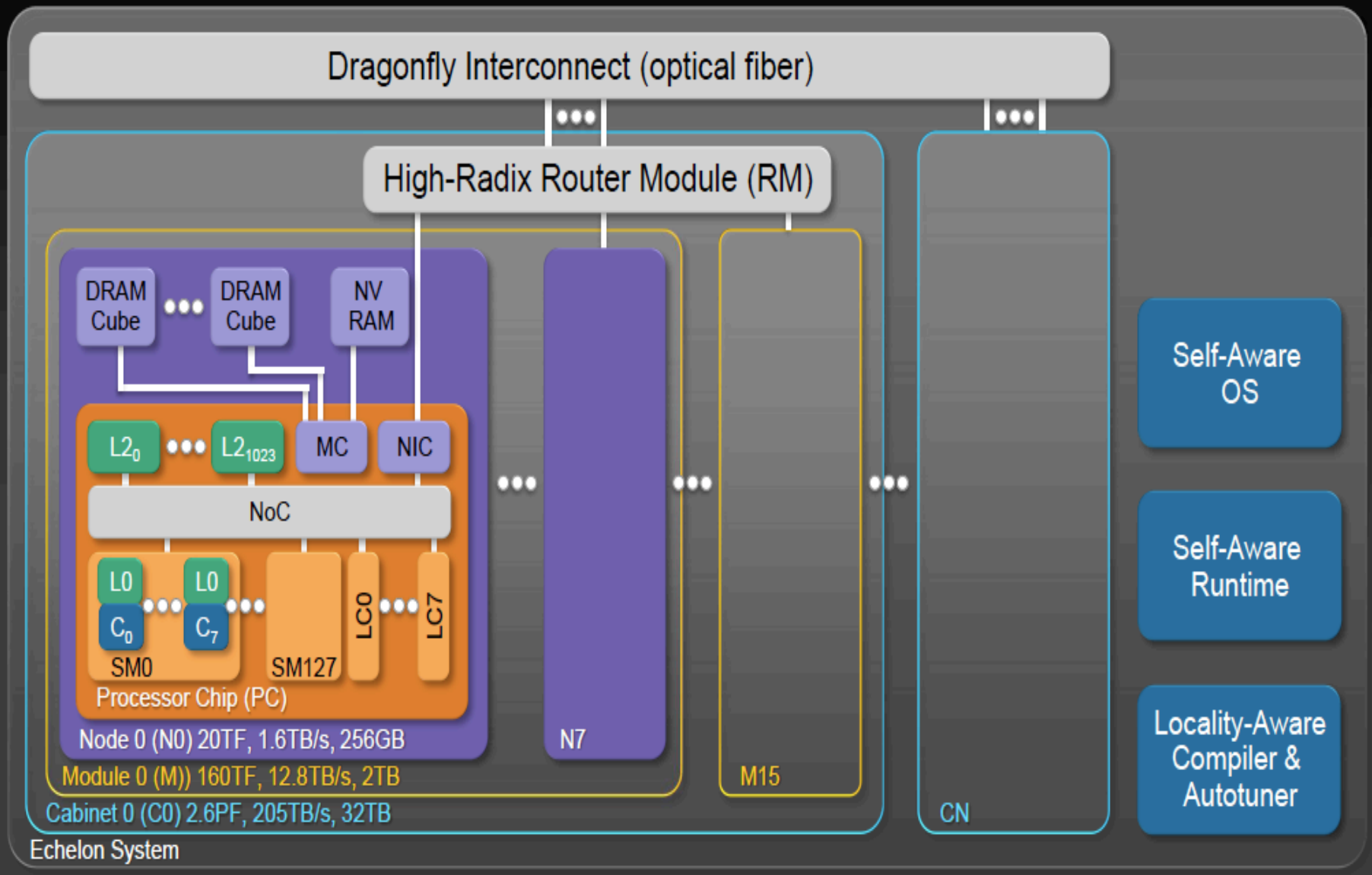
- Lower memory-computation ratio due to system cost
  - Terascale ~ 1 byte/FLOP
  - Petascale ~ .1 byte/FLOP
  - Exascale ~ 0.01 bytes/FLOP (projected)
- Therefore, weak scaling won't deliver 1000x increase in concurrency
- 1000x must come from strong scaling and software improvements
  - Reduce task granularity by 1000x
  - Reduce synchronization granularity by 1000x
  - Reduce communication overhead by 100x
  - Reduce sequential bottlenecks by 1000x

4. More on Getting to Exascale

- Exascale architectures will be fundamentally different
  - Power management THE fundamental issue
  - Reliability (hardware and software) increasingly a concern
  - Memory reduction to .01 bytes/flop
  - Hierarchical, heterogeneous
- Basic rethinking of the software "stack"
  - Ability to express and manage locality and parallelism for ~billion threads will require fundamental change
  - Support applications that are forward scalable and portable (over provision parallelism and map to h/w)
  - Managing power (although managing locality is a big part) and resilience requirements

Sarkar, Harrod and Snavely, "Software Challenges in Extreme Scale Systems," SciDAC 2009, June, 2009. Summary of results from a DARPA study entitled, "Exascale Software Study," June 2008 through Feb. 2009.

Echelon System Sketch
from "GPU Computing To Exascale and Beyond", Bill Dally, SC10



4. Getting to Exascale: Growing Importance of Autotuning

- Increasingly onerous for programmer to manage mapping to hardware, ***even for single socket***
- Many existing and future systems will include ***heterogeneous processors***, requiring different mapping strategies
- At extreme scale, anticipating dynamically varying behavior even for "regular" computations
  - Billions of threads
  - Active energy and reliability management
- Optimization must navigate performance, energy and reliability goals

Key Idea: Programming system can automate portions of this tuning process

Summary of Lecture

- Autotuning applied to applications
  - Building application-specific libraries
  - Triage to extract key computational kernels
  - Navigating very large search spaces
- Getting to exascale
  - Many concerns beyond performance (energy and reliability)
  - Heterogeneous architectures will change how we develop programs
  - Autotuning even more important

Concluding Remarks

Compiler-based collaborative auto-tuning:

- Close collaboration with architects and application developers
- Track manual tuning research and try to replicate (semi-) automatically
- Portable code generation, and start on heterogeneous support

Three core technical ideas

- **Compiler technology:** Modular compilers, systematic approach to optimization, empirical search, ***goal is hand-tuned performance.***
- **User Tools:** Access to transformation system, express parameters for automatic search, express expected problem size, express alternative implementations
- **Systematic:** Compose optimized applications in context. Express/derive parameters for search

References

Papers related to these experiments

D. H. Bailey, J. Chame, C. Chen, J. Dongarra, M. Hall, J. K. Hollingsworth, P. Hovland, S. Moore, K. Seymour, J. Shin, A. Tiwari, S. Williams, H. You, "PERI Auto-Tuning,'' Journal of Physics: Conference Series, Vol. 125, 2008.

J. Chame, C. Chen, M. Hall, J. K. Hollingsworth[3], Kumar Mahinthakumar[4], Gabriel Marin[5], Shreyas Ramalingam[2], Sarat Sreepathi[4], Vamsi Sreepathi, Ananta Tiwari, "PERI Autotuning of PFLOTRAN", Journal of Physics: Conference Series, 2011 (to appear).

M.W. Hall and J. Chame, "Languages and Compilers for Autotuning," In Performance Tuning of Scientific Applications, edited by David Bailey, Robert F. Lucas and Sam Williams. Taylor and Francis publishers, Nov. 2010.

J. Shin, M. W. Hall, J. Chame, C. Chen, P. F. Fischer, and P. D. Hovland. 2010. Speeding up Nek5000 with autotuning and specialization. In Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10).

A. Tiwari, C. Chen, C. Liao, J. Chame, J. Hollingsworth, M. Hall and D. Quinlan, "Auto-tuning Full Applications: A Case Study'', International Journal of High Performance Computing Applications, 2011 (to appear).