Friends Don't Let Friends Tune Code

Jeffrey K. Hollingsworth
University of Maryland
hollings@cs.umd.edu

Ananta Tiwari (UMD)

About the Title

Why Automate Performance Tuning?

- **Too many parameters that impact performance.**
- **Optimal performance for a given system depends on:**
  - Details of the processor
  - Details of the inputs (workload)
  - Which nodes are assigned to the program
  - Other things running on the system
- **Parameters come from:**
  - User code
  - Libraries
  - Compiler choices

Automated Parameter tuning can be used for adaptive tuning in complex software.

Active Harmony

Automated Performance Tuning

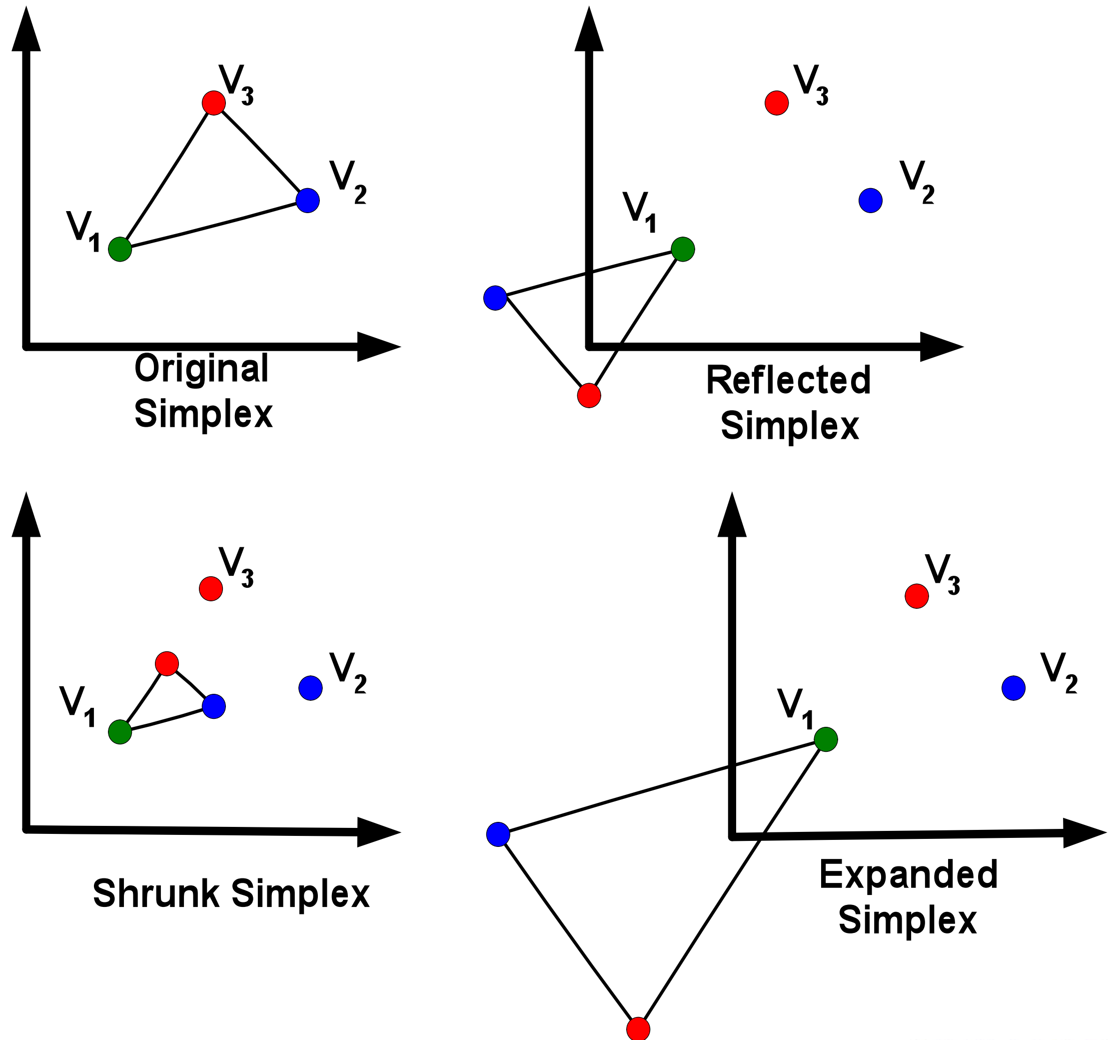
- **Goal: Maximize achieved performance**
- **Problems:**
  - Large number of parameters to tune
  - Shape of objective function unknown
  - Multiple libraries and coupled applications
  - Analytical model may not be available
- **Requirements:**
  - Runtime tuning for long running programs
  - Don’t try too many configurations
  - Avoid gradients

Active Harmony

- **Runtime performance optimization**
  - Can also support training runs
- **Automatic library selection (code)**
  - Monitor library performance
  - Switch library if necessary
- **Automatic performance tuning (parameter)**
  - Monitor system performance
  - Adjust runtime parameters
- **Hooks for Compiler Frameworks**
  - Working to integrate USC/ISI Chill
  - Looking at others too

Parallel Rank Ordering Algorithm

- All, but the best point of simplex moves.
- Computations can be done in parallel.



Application Parameter Tuning: GS2

- **Physics application (DOE SciDAC project)**
- **Developed to study low-frequency turbulence in magnetized plasma**
- **Performance (execution time) improvement by changing layout and three parameters (negrid, ntheta, nodes)**
- **Data layout analysis**

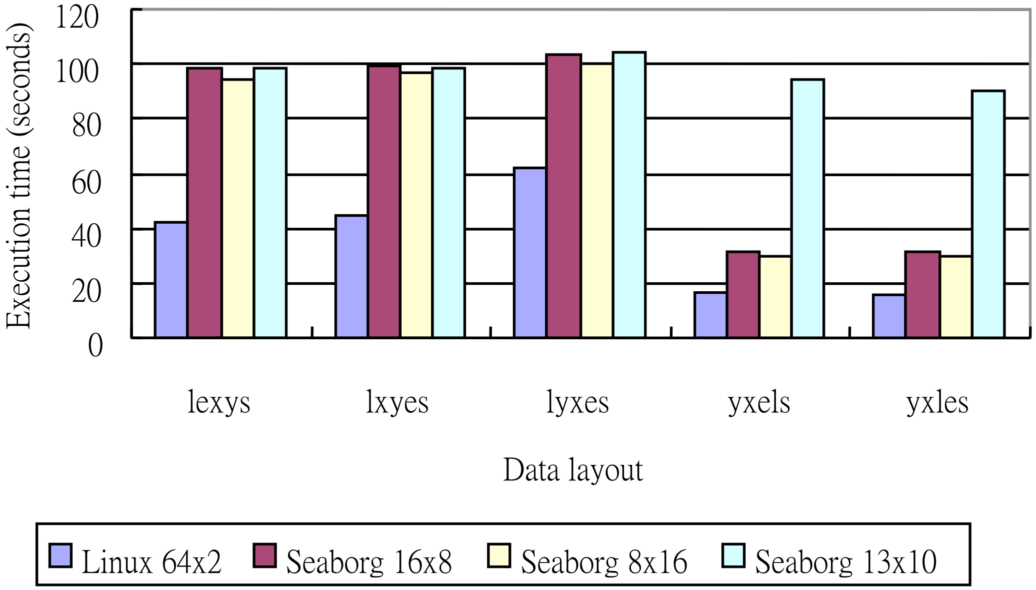
(benchmarking runs)

- 55.06s →16.25s

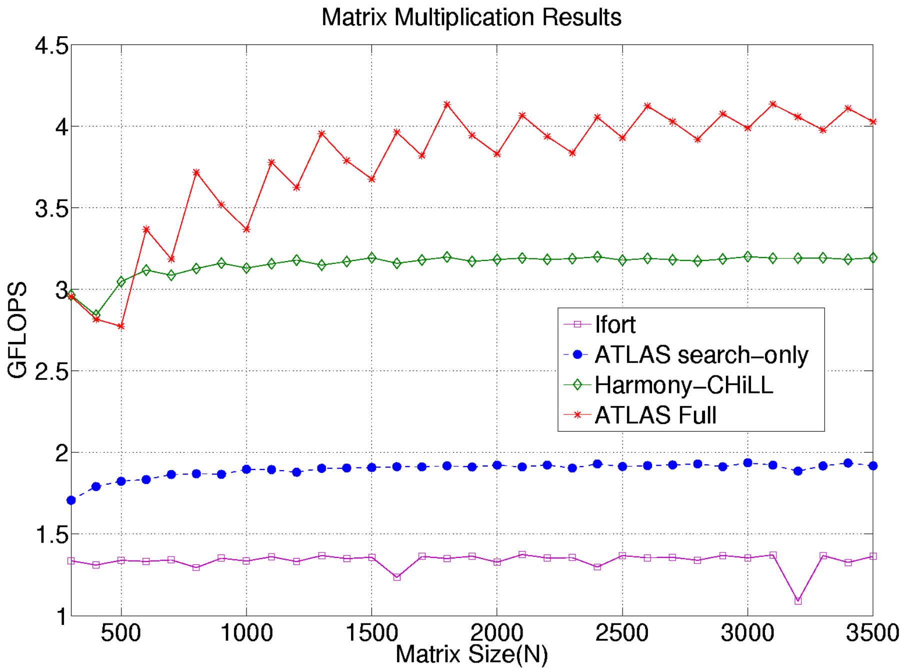
(3.4x faster, W/O collision)

- 71.08s → 31.55s

(2.3x faster, W collision)



Tool Integration: CHiLL + Active Harmony



Generate and evaluate different optimizations that would have been prohibitively time consuming for a programmer to explore manually.

Ananta Tiwari, Chun Chen, Jacqueline Chame, Mary Hall, Jeffrey K. Hollingsworth, “A Scalable Auto-tuning Framework for Compiler Optimization,” IPDPS 2009, Rome, May 2009.

SMG2000 Optimization

Outlined Code

```
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

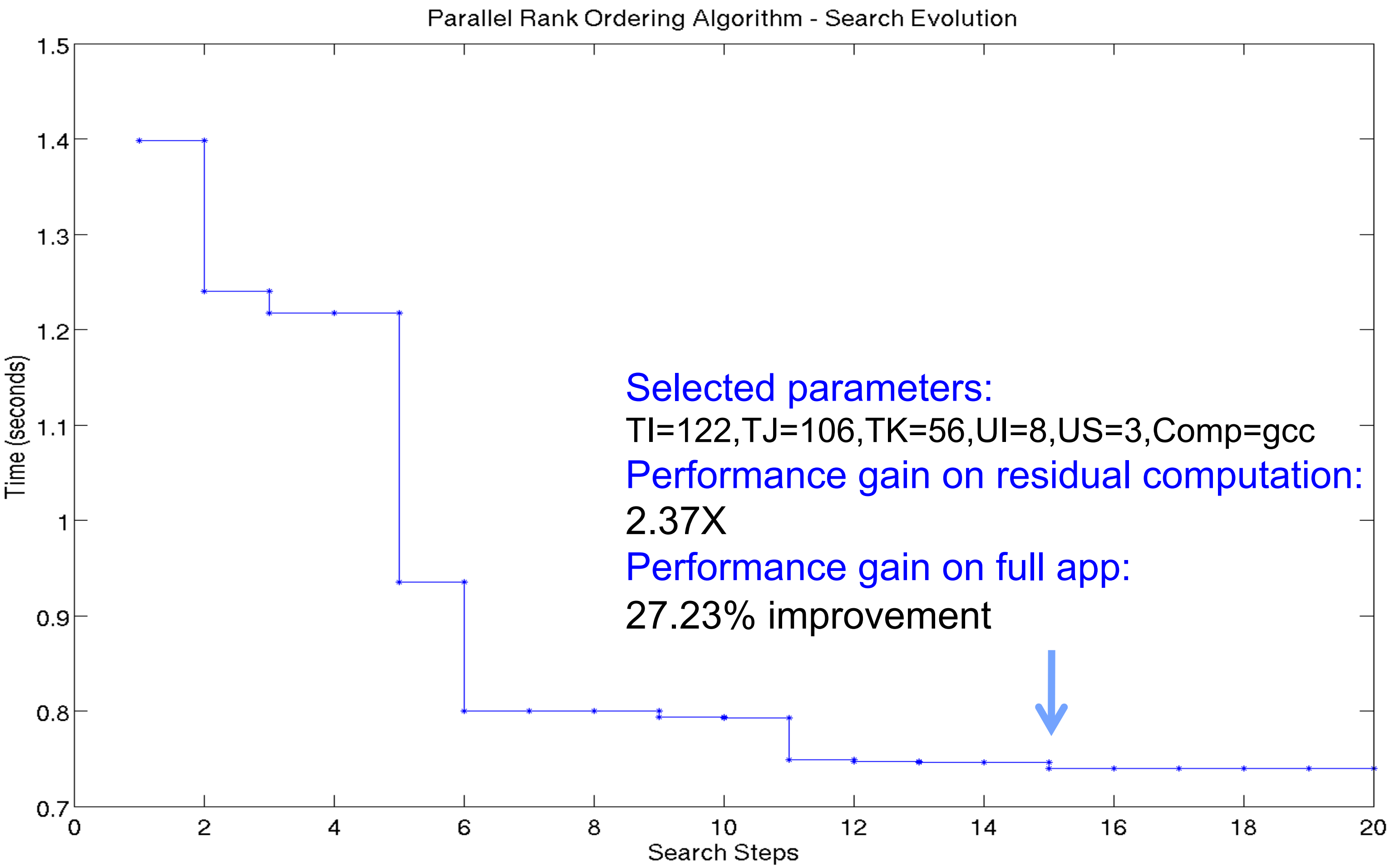
$0 \leq TI, TJ, TK \leq 122$
$0 \leq UI \leq 16$
$0 \leq US \leq 10$
compilers ∈ {gcc, icc}

Search space:

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

SMG2000 Search and Results

Parallel search evaluates 490 points and converges in 20 steps



Selected parameters:
TI=122,TJ=106,TK=56,UI=8,US=3,Comp=gcc

Performance gain on residual computation:
2.37X

Performance gain on full app:
27.23% improvement

Auto Tuning For Different Platforms

- **Fixed parameters:**
  - Code: PMLB
  - Processors: 64
- **Study how parameters differ for the two systems**
- **Use harmony determined parameters from one system**
  - Run a post-line (fix parameters for entire run) run on another

Problem Size	Speedup (post-line) run on UMD Cluster		Speedup (post-line) run on Carver Cluster	
	UMD Best Config	Carver Best Config	Carver Best Config	UMD Best Config
384^3	1.44	1.19	1.32	1.30
448^3	1.42	1.13	1.51	1.38
512^3	1.30	1.26	1.34	1.30
576^3	1.38	1.16	1.42	1.39

Autotuning PFloTran (Trisolve)

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 > 14)
known(bs < 16)
unroll(1,2,u1)
unroll(1,3,u2)
```

Constraints on Search

0 <= u1 <= 16

0 <= u2 <= 16

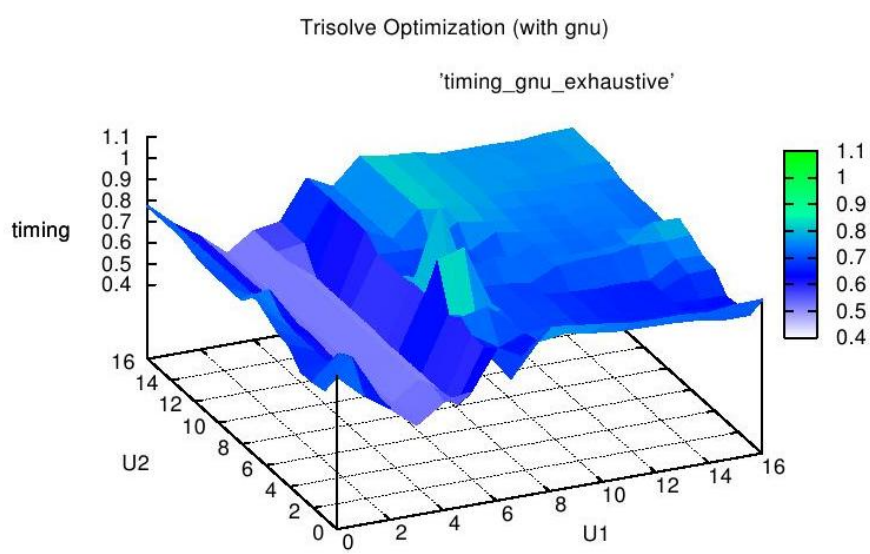
compilers ∈ {gnu, pathscale, cray, pgi}

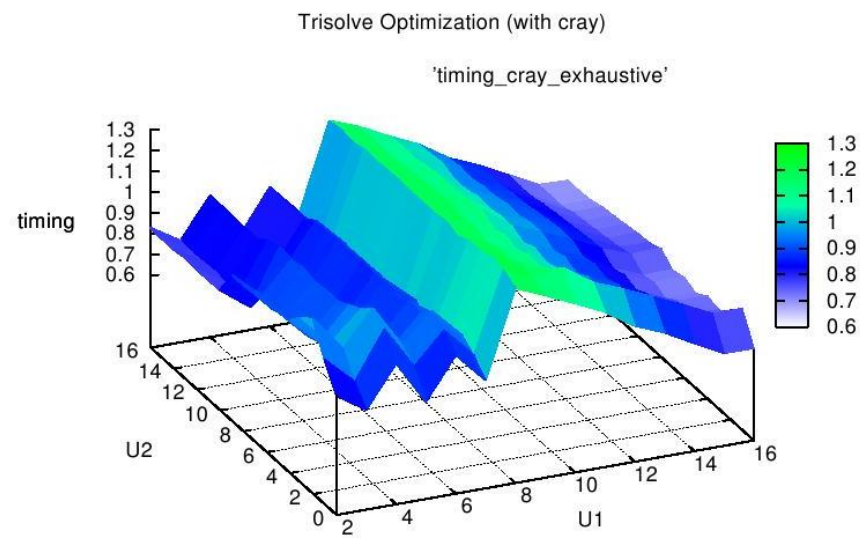
Search space:

17x17x4 = 1156 points

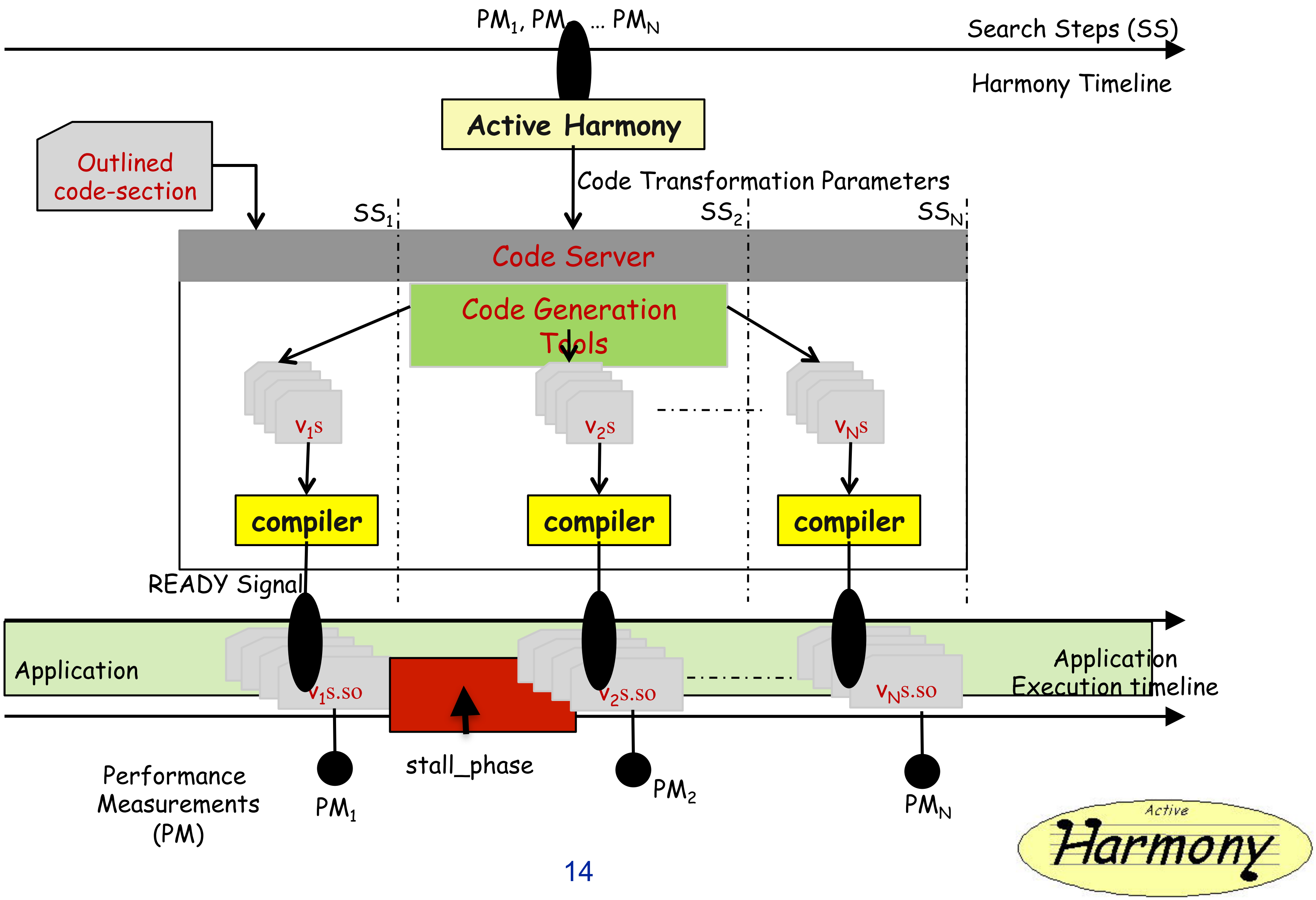
PFloTran: Trisolve Results

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**





Compiling New Code Variants at Runtime



Online Code Generation Results

- **Two platforms**
  - umd-cluster (64 nodes, Intel Xeon dual-core nodes) – myrinet interconnect
  - Carver (1120 compute nodes, Intel Nehalem. two quad core processors) – infiniband interconnect
- **Code servers**
  - UMD-cluster – local idle machines
  - Carver – outsourced to a machine at umd
- **Codes**
  - Poisson Solver
  - PMLB Parallel Multi-block Lattice Boltzman
  - SMG2000

How Many Nodes to Generate Code?

- **Fixed parameters:**
  - Code: poission solver
  - problem-size (1024^3)
  - number of processors (128)
- **Up to 128 new variants are generated at each search step**

Code Servers	Search Steps[+]	Stalled steps[+]	Variations evaluated[+]	Speedup[+]
1	6*	46	502	0.75
2	17*	13	710	0.97
4	27	7.2	928	1.04
8	23	4.5	818	1.23
12	22	4.1	833	1.21
16	26	3.6	931	1.24

*** Search did not complete before application terminated**

+ Mean of 5 runs

Conclusions and Future Work

- **Ongoing Work**
  - More end-to-end Application Studies
  - Continued Evaluation of Online Code Generation
- **Conclusions**
  - Auto tuning can be done at many levels
    - Offline – using training runs (choices fixed for an entire run)
      - Compiler options
      - Programmer supplied per-run tunable parameters
      - Compiler Transformations
    - Online –training or production (choices change during execution)
      - Programmer supplied per-timestep parameters
      - Compiler Transformations
  - It Works!
    - Real programs run faster

Active Harmony