Experience with Automated Performance Tuning Using Active Harmony

Jeffrey K. Hollingsworth

hollings@cs.umd.edu
Department of Computer Science
University of Maryland, College Park, MD 20742

Questions/ Position

- All layers of the software stack (e.g., OS, middleware, MPI, libraries, apps) will be "autotuned."
- We need to integrate these multiple layers!

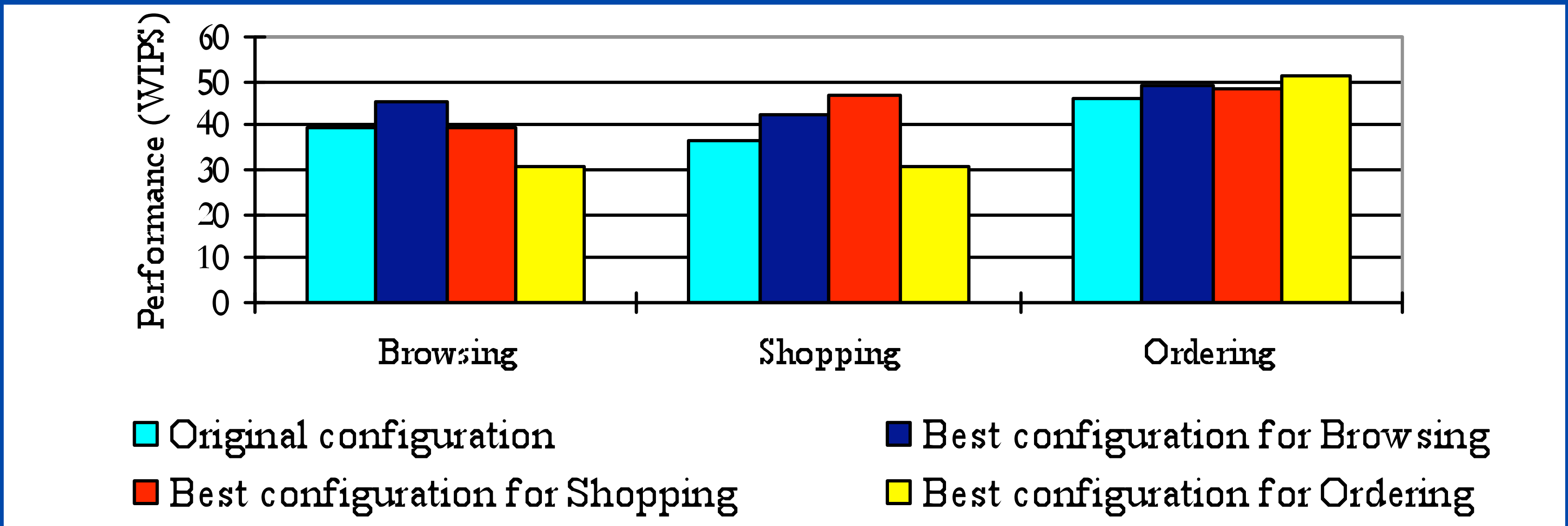
Active Harmony

- 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

Example: Cluster Based Web Server

- 3-tier system
- Harmony Provides
  - Parameter updates for DB, and App Severs
- TPC-W Benchmark
  - Transactional web benchmark
  - Mimic operations of an e-commerce site
  - Uses Java implementation from Univ. of Wisconsin
  - Performance metrics
    - Web Interaction Per Second (WIPS)

Cluster-Based Web Service Tuning

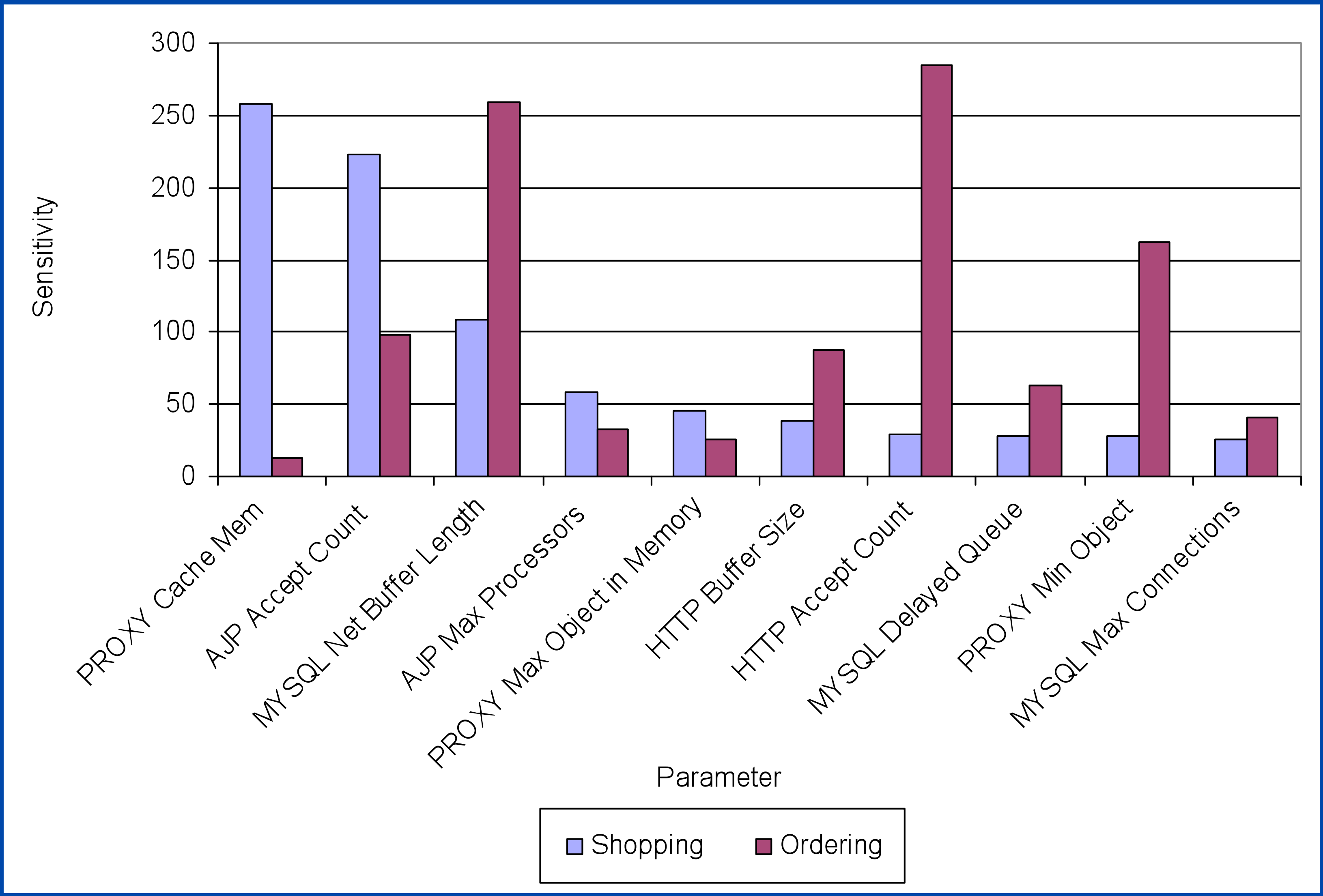


	Best configuration after 200 iterations		
	Browsing	Shopping	Ordering
Improvements compared to the default configuration	15%	16%	5%

Tuning Results for Different Workloads

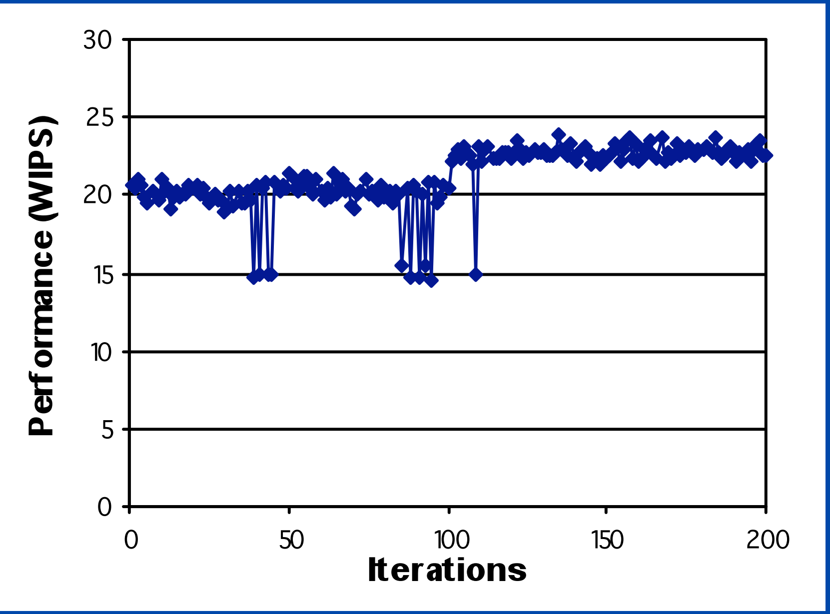
Tunable parameters	Default config.	Best configuration after 200 iterations		
		Browsing	Shopping	Ordering
Proxy Server				
cache_mem	8	13	17	21
minimum_object_size	0	0	50	306
maximum_object _size_in_memory	8	6	256	2,560
HTTP & App. Server				
Min Threads	**5**	**1**	**16**	**102**
Max Threads	**20**	**11**	**16**	**131**
Queue Size	**10**	**6**	**21**	**136**
Buffer Size	**2,048**	**2,049**	**3,585**	**6,657**
AJPminProcessors	5	6	26	136
AJPmaxProcessors	20	86	296	161
AJPacceptCount	10	76	306	671
Database Server				
binlog_cache_size	32,768	63,488	153,600	284,672
max_connections	100	201	451	701

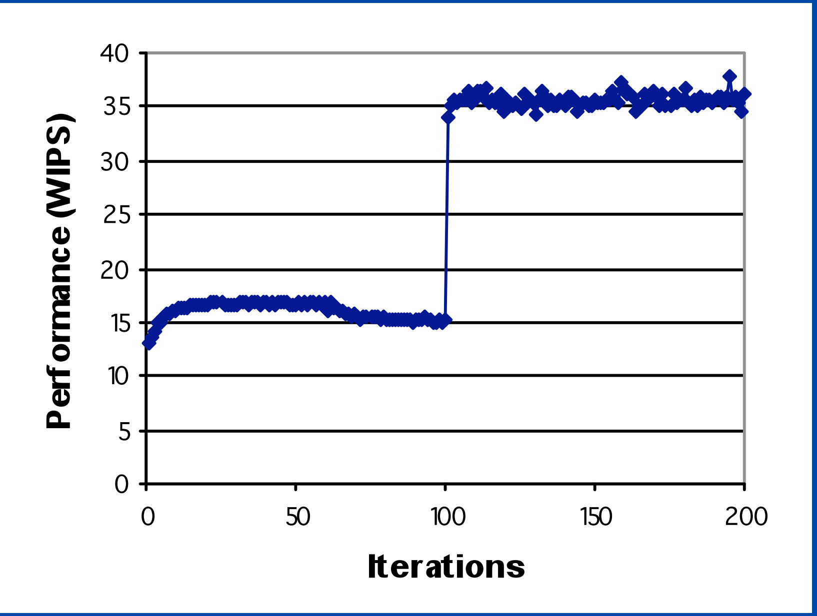
Importance of various parameters



Bigger Changes Often Matter Most

- External tuning - reconfiguration



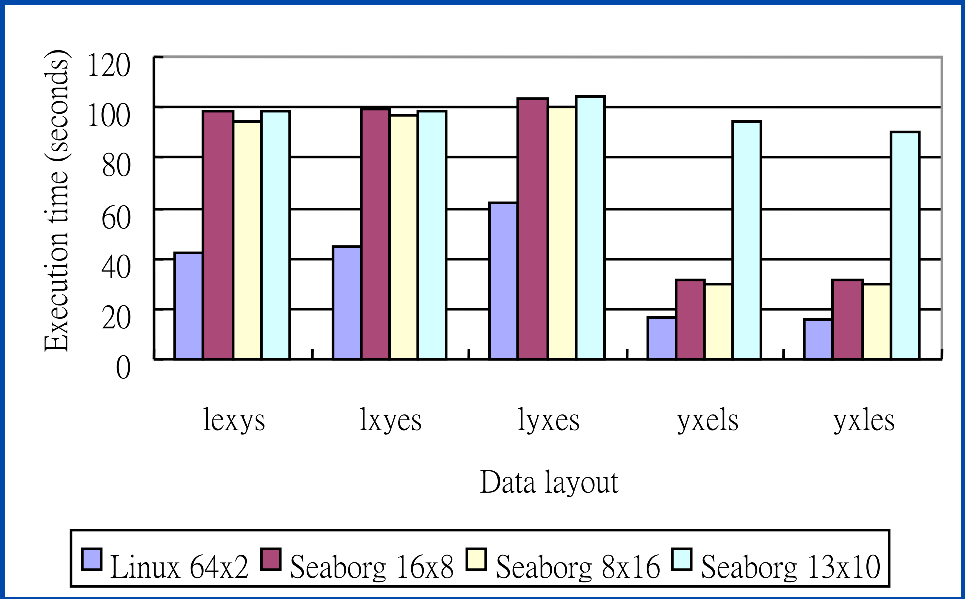


(a) One node moved from the proxy server tier to the application server tier

(b) One node moved from the application server tier to the proxy server tier

Example 2: 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)

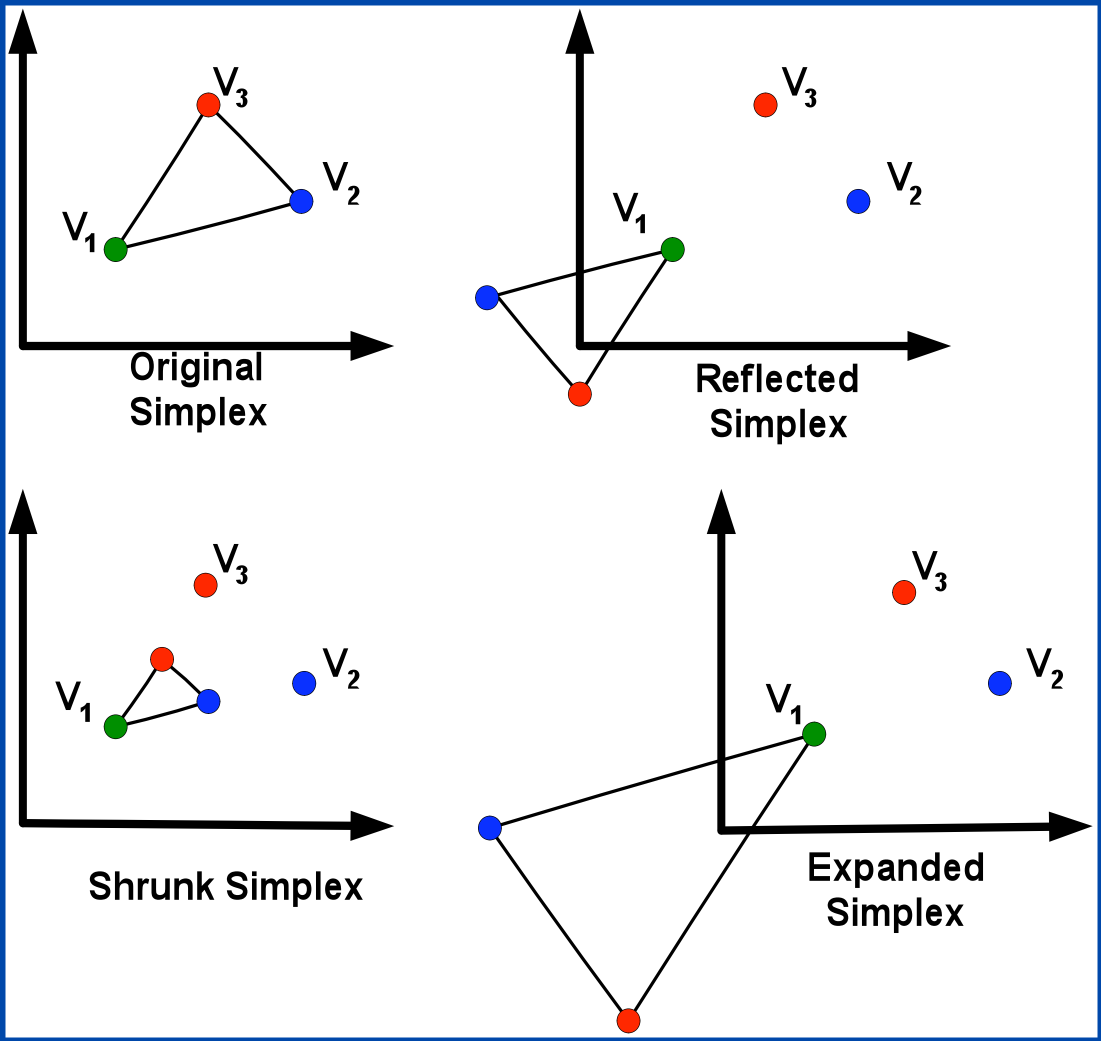


A Bit More About Harmony Search

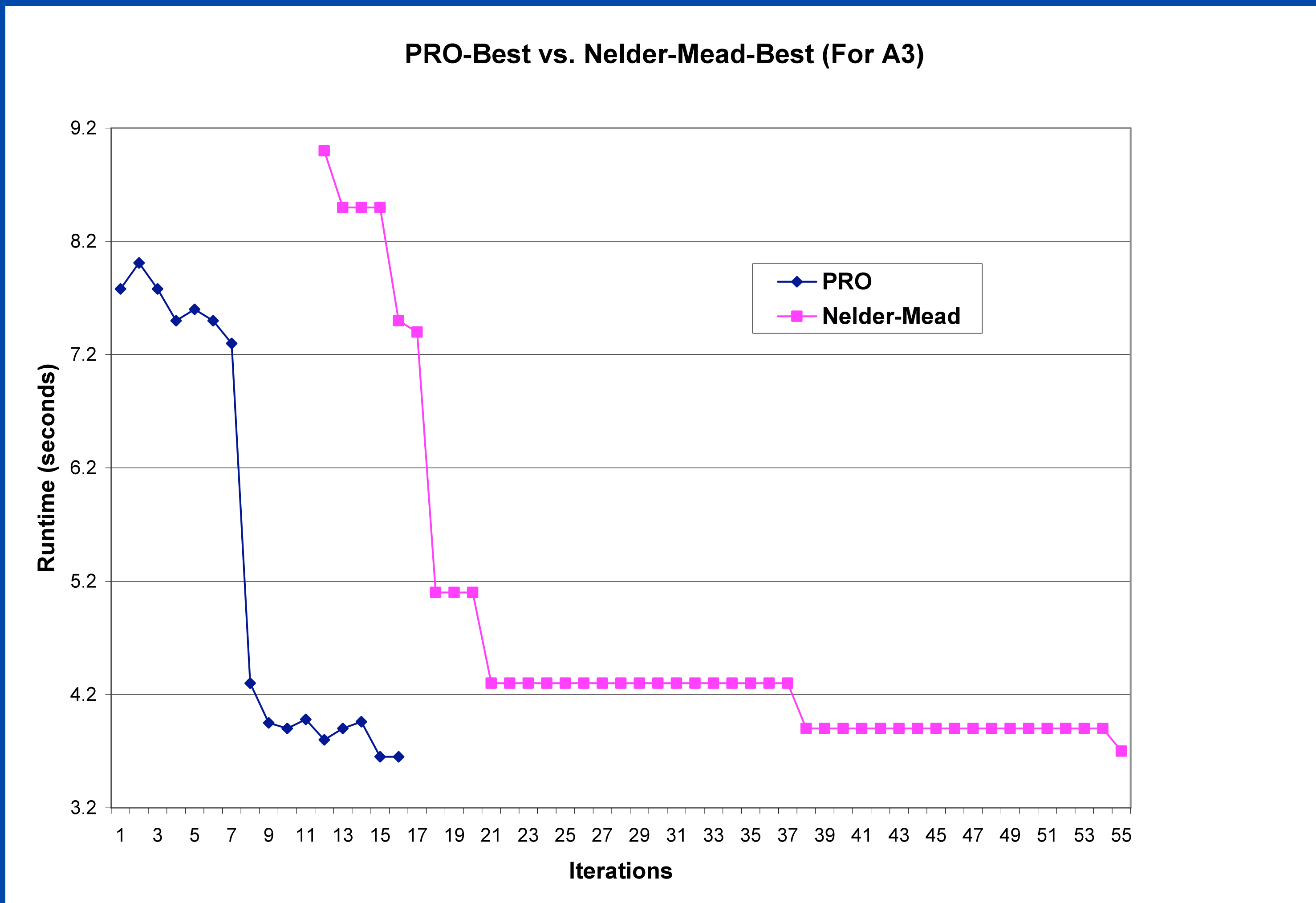
- Pre-execution
  - Sensitivity Discovery Phase
  - Used to **Order** not **Eliminate** search dimensions
- Online
  - Use Parallel Rank Order Search
    - Different configurations on different nodes

Rank Ordering Algorithm

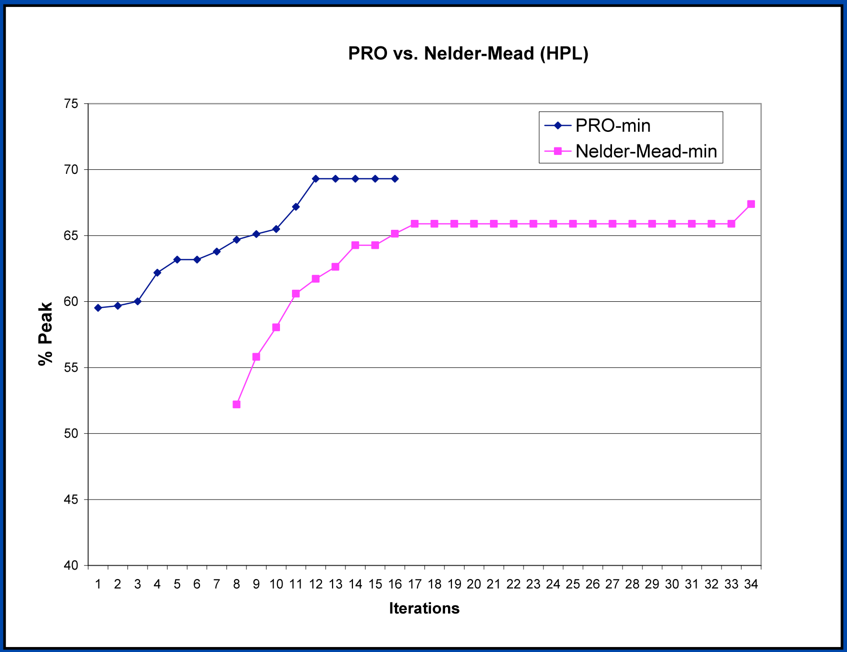
- All, but the best point of simplex moves.
- Computations can be done in parallel.

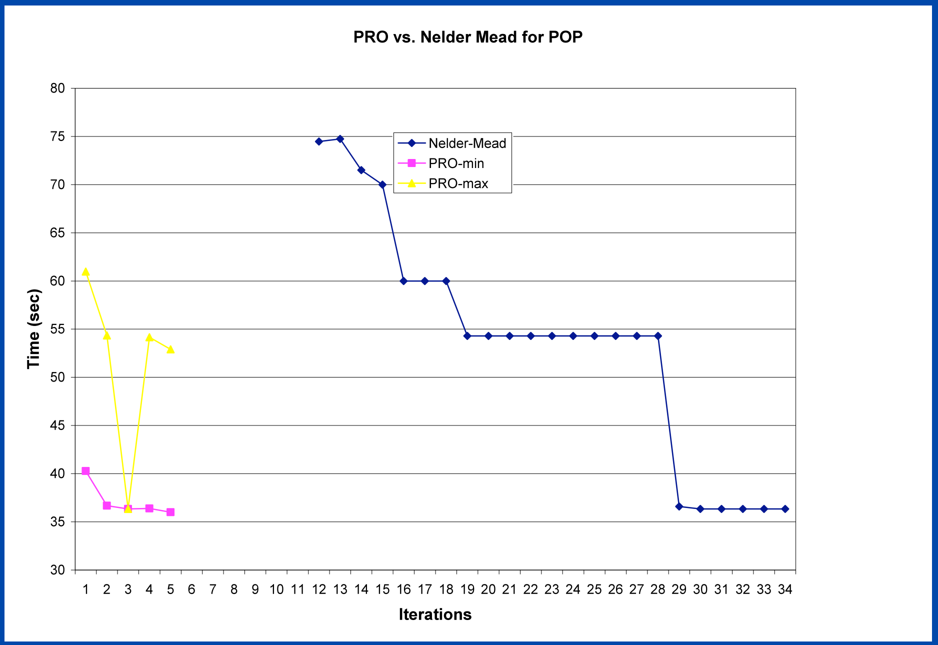


Gains When Using PRO Search Algorithm



Additional PRO Results





- Performance for Two Programs
  - High Performance Linpack
  - POP Ocean Code

Must Coordinate Auto Tuners

- Problem: Warring auto tuning systems
  - Multiple components "auto tuning" at once
  - Tuning based on multiple changes at once
- Solution:
  - Need some level of coordination
  - Possible Answer:
    - Exposing different tuning systems
  - Part of PERI Auto-tuning Effort

Parameter Specification Language – Requirements

- Define the search space:
  - Represent the search space symbolically
  - Specify parameter types (integer vs. float)
  - Represent parameter domain (range, step etc.)
- Represent constraints from:
  - tools
  - applications (via automated analysis)
  - programmers
- Provide support for arbitrary expression and function evaluation

Requirements …

- Express search hints:
  - Ordering/ranking parameters (*unroll* before *tiling*)
  - Group parameters, code regions and/or constraints into sets
  - Represent data from static modeling, historical runs
- Support for mapping language constructs
  - Identify where in the source code (e.g. what loop) the optimization is taking place
- Specify when and how to gather objective function value (compile-time vs. application launch-time)

Draft Specification Language

- Six main components:
  - Code Region Declaration
  - Region Set Declaration
  - Parameter Declaration
  - Constraint Declaration
  - Constraint Specification
  - Ordering Info
- Provides a rich expression syntax

What we might specify? Ex. #1

```
parameter space simple_example
  {
          parameter x int {
              range [1:1:3];
                   default 3;
               }

          parameter y int {
                   range [1:1:3];
                   default 2;
               }

          parameter z int {
                   range [1:1:3];
                   default 1;
               }
```

```
# And then the constraints.
          constraint c1 {
                   x≥z;
          }

          constraint c2 {
                   y>z;
           }

          # Constraint specification.
           specification  {
                   c1 AND c2;
           }

           # Ordering information is
              optional.
  }
```

What we might specify? Ex. #2

```
parameter space tiling {
    code_region loopI;
    code_region loopJ;
    region_set loop [loopI, loopJ];
    # declare tile_size parameter
    parameter tile_size int {
            range [2:2:256]
            default 32;
            region loop;
        }

    # Arbitrary constraint
    constraint c1 {
        (loopI.tile_size *
         loopJ.tile_size * 3 * 4) ≤
          2048;
    }

    # rectangular tiles better.
    constraint c2 {
        loopI.tile_size > loopJ.tile_size;
    }

    constraint c3 {
         loopJ.tile_size > loopI.tile_size;
     }

    specification {
         (c1 AND c2) OR (c1 AND c3);
     }
}
```

What we might specify? Ex. #3

```
parameter space pstswm {
    ...
    # FTopt determines what FFT algorithm to use.
    parameter FTopt enum {
        enumeration [distributed, single_transpose, double_transpose];
        default distributed;
    }
    # LTopt determines which LT algorithm to use.
    parameter LTopt enum {
        enumeration {distributed,   transpose_based};
        default distributed;
    }
    constraint pq {
        (p*q) == 16;
    }
    # When FTopt is ‘double_transpose’, LTopt has to be ‘transpose_based’
    constraint ftLT {
        (FTopt.value=double_transpose) IMPLIES (LTopt.value=distributed);
    }
    specification {
        pq AND ftLT;
    }
}
```

Language Syntax and Implementation

- Looking into GNU-MathProg modeling language
  - Can this language address all the requirements that we have discussed?
  - May need to add syntactic sugar on top

- Looking into Python Constraint Module
  - No support for "on-demand" derivation of search points

Search API

- Needed functionality
  - Evaluate point
    - Run code at a point in search space
    - Likely to be a-sync to allow parallel search
  - Store/Read values for point in search space
    - Will include point in space, value, context (data set/machine info)
  - Query Spec
    - Learn about parameters, constraints
      - May use existing Math Prog API
    - Query Search Strategy Info

Search API

- Related Questions
  - Migrate ordering and grouping info to search API?
  - How can we use historical data?
    - Incorporating information from perf-db
  - Representation of the states
    - Types of iterators
    - "On Demand" evaluation needed to prevent space representation explosion

Conclusions

- First step towards integrating search-based auto-tuning frameworks
- Once a decision on parameter space language is made, search API will be rolled out

Acknowledgements

- Coding and Experiments
  - I-Hsin Chung (IBM Watson)
  - Vahid Tabatabaee
  - Ananta Tiwari
- PERI Search Coordination Effort
  - USC ISI, Univ. Tenn, Rice, SDSC, LLNL, UNC, ORNL, ANL, LBL
- Funding
  - DOE – PERC/PERI
  - NSF
  - LTS (DoD)