HPCS Toolkit: Extending High Performance to High Productivity

David Klepacki
Manager, Advanced Computing Technology
T.J. Watson Research Center

July 2006
Outline

- Vision for Performance Tools
- Roadmap for HPCS Toolkit
- New Technologies Needed for HPCS
- Core Productivity Technology: pSigma
- Current Technology Review: HPC Toolkit
- Productivity Inhibitors of Current Technology
- HPCS Toolkit Deliverables
- Summary
Sanity Check on System Evolution

- Device Scaling imposing **fundamental constraints on system**
  Power dissipation and energy consumption
  Physical size / packaging
- Pressure to **re-think system architecture**
  Blue Gene: low power devices, embedded (small)
  Cell: Attached (embedded) co-processing engine
- Systems become inherently **more complex**
  Connectivity / hierarchical topology (torus, intra-cell)
  Memory constraints (less per processor)
  Additional technology “boosts” (hyper-threading, SMT)
- This poses **new challenge to application programming**
  New programming paradigm? (not on horizon - legacy codes, ISV apps, etc.)
- Conclusion: **New software tools essential** to mitigate evolving system complexity
Projected Roadmap for a HPCS Toolkit

- **Phase I**
  - Core Productivity Infrastructure (pSigma)
  - New Tools to further leverage pSigma capability
    - Cache simulator
    - Performance prediction assistant
    - I/O Profiling tools
  
  *Abstracts Developer from the Source Code Instrumentation*

- **Phase II**
  - Automation of Performance-Tuning Cycle
    - Intelligent agents to collect performance metrics, mine the information, determine/characterize likely bottlenecks and propose/implement an optimization
    - Investigation of real-time decision system for run-time optimizations
  
  *Abstracts Developer from the System*

*Programmer can be removed from the Performance-Tuning Cycle!*
Innovative Technologies for HPCS

- Completely Binary Approach (pSigma)
  Programmable and dynamic, yet without the need for source code modification.

- Data-Centric Analysis (DCA)
  For HPCS systems, new tools are needed to provide detailed information on the impact of an application’s data structures in relation to the underlying hardware.

- Alternate Scenario Prediction (ASP)
  Data structure layout (“what if my array A was dimensioned like …”)
  Order of a parallel computation, scheduling of threads, etc.

- User-Controlled Automation (autoPerf)
  Productivity is controlled by degree of automation chosen by programmer.
  Can be fully automated if desired.
Core Productivity Technology: pSigma

- pSigma technology was created in response to HPCS challenge. It is the foundation that makes automation possible. Automation is the key to productivity.

- pSigma is a general method to insert 'data-collection points' into a binary application non intrusively (i.e., without source code modification).

- pSigma provides an interface between the user and a measurement system such that the user can interact with it at a higher conceptual level. pSigma translates the user’s higher-level input into lower-level instructions (sort of a "compiler" for performance analysis).
The IBM HPCS Toolkit

Binary Application

HPCS GUI

- Communication Profiler
- Memory Profiler
- CPU Profiler
- I/O Profiler
- Shared-Memory Profiler

Visualization
- Query
- Analysis

pSigma

Binary instrumentation

Instrumented Binary

Binary execution
DCA Interactive Cache Tool (Prototype)
## Alternate Scenario Prediction Technology (ASP)

### Memory Profile for a1, a2, a3

<table>
<thead>
<tr>
<th>Data: a1</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>96%</td>
<td>99%</td>
<td>7%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>16%</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: a2</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>97%</td>
<td>50%</td>
<td>7%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>16%</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: a3</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>96%</td>
<td>59%</td>
<td>0.1%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>17%</td>
<td>-</td>
</tr>
</tbody>
</table>

### Query Repository:

- parameter (nx=512, ny=512)
- real a1(nx+1,ny), a2(nx+1,ny), a3(nx+1,ny)

### Predicted Memory Profile for a1, a2, a3

<table>
<thead>
<tr>
<th>Data: a1</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>96%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>16%</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: a2</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>97%</td>
<td>50%</td>
<td>98%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>16%</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: a3</th>
<th>L1</th>
<th>L2</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Hit Ratio:</td>
<td>96%</td>
<td>59%</td>
<td>96%</td>
</tr>
<tr>
<td>Store Hit Ratio:</td>
<td>99%</td>
<td>17%</td>
<td>-</td>
</tr>
</tbody>
</table>
autoPerf Technology (Automation)

**Instrumented data:**
Memory, MPI, IO, openMP, HPM

1. Performance Data Collection via **pSigma**.

2. Intelligent Mining of Data to Determine Bottlenecks via **DCA**.

3. Characterization of Bottlenecks and Solution Determination via **ASP** and **autoPerf**.

4. Automatic Source Code Patch of Solution via **autoPerf**.

- FPU stalls
- L2 misses
- MPI

**e.g., Communication imbalance: Array A**

**Automatic performance tuning**

**Knowledge base**

**e.g., Block cyclic distribution of A**
Current Technology: The IBM HPC Toolkit

- IBM Corporate strategy for a common application performance analysis environment across all of its HPC platforms:
  - pSeries (POWER, PowerPC) – AIX, Linux
  - xSeries (Intel, AMD) - Linux
  - Blue Gene Systems
  - Standalone or use within Eclipse environment

- Unified Performance Analysis Environment:
  - Hardware Performance Monitor (HPM)
  - MPI Profiler and Tracer
  - Cray SHMEM Porting Library and Profiler
  - OpenMP Profiler
  - GUI framework w/ source code traceback
Unified GUI with Source Code Traceback

Optional Eclipse Environment

Simultaneous Performance Analysis within a Single Framework!

HPM  MPI / SHMEM  OpenMP

PeekPerf GUI
Advanced Computing Technology Center

HPCS Toolkit

IBM ACTC Perf/Perf: Main Window

File Tools Options Action

HPM | MPI | SIGMA | DPOMP |

Name

# Probe

- MPI IProbe
- MPI Probe

#Recv

- MPI Irecv
- MPI Rrecv

Send

- Blocking Send
  - MPI Bsend
  - MPI Rsend
  - MPI Send
  - MPI Isend
- Nonblocking Send
  - MPI Ibsend
  - MPI Irsend
  - MPI Isend
  - SendRecv
- Test
  - MPI Test
  - MPI Testall
  - MPI Testaux

swim_omp

<table>
<thead>
<tr>
<th>Label</th>
<th>Count</th>
<th>ExecSec</th>
<th>IncSec</th>
</tr>
</thead>
<tbody>
<tr>
<td>- All</td>
<td>1</td>
<td>0.006</td>
<td>0.062</td>
</tr>
<tr>
<td>Calc1</td>
<td>102</td>
<td>0.068</td>
<td>0.283</td>
</tr>
<tr>
<td>Calc2</td>
<td>102</td>
<td>0.072</td>
<td>0.373</td>
</tr>
<tr>
<td>Calc3</td>
<td>100</td>
<td>0.096</td>
<td>0.379</td>
</tr>
<tr>
<td>Initial</td>
<td>1</td>
<td>0.042</td>
<td>0.042</td>
</tr>
<tr>
<td>Loop 100</td>
<td>102</td>
<td>0.119</td>
<td>0.119</td>
</tr>
<tr>
<td>Loop 200</td>
<td>102</td>
<td>0.178</td>
<td>0.178</td>
</tr>
<tr>
<td>Loop 300</td>
<td>100</td>
<td>0.209</td>
<td>0.209</td>
</tr>
<tr>
<td>MPI Calc1 end</td>
<td>102</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>MPI Calc1 start</td>
<td>102</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>MPI Calc2 end</td>
<td>102</td>
<td>0.068</td>
<td>0.068</td>
</tr>
<tr>
<td>MPI Calc2 start</td>
<td>102</td>
<td>0.055</td>
<td>0.055</td>
</tr>
<tr>
<td>MPI in Calc2</td>
<td>100</td>
<td>0.072</td>
<td>0.072</td>
</tr>
<tr>
<td>loop 110</td>
<td>102</td>
<td>0.009</td>
<td>0.009</td>
</tr>
</tbody>
</table>

calc1.f | calc2.f | calc3.f | swim_omp.f | swim_omp.f | calc1.f | calc2.f | calc3.f |

101: call mpi_recv(p+1, n+1, mpi_comm_world, req(4), ierr)
102: call mpi_send(vold+1, n+1, mpi_comm_world, req(4), ierr)
103: call mpi_recv(p+1, n+1, mpi_comm_world, req(4), ierr)
104: call mpi_send(vold+1, n+1, mpi_comm_world, req(4), ierr)
105: call mpi_recv(p+1, n+1, mpi_comm_world, req(4), ierr)
106: call mpi_send(vold+1, n+1, mpi_comm_world, req(4), ierr)
107: call mpi_recv(p+1, n+1, mpi_comm_world, req(4), ierr)
108: call mpi_send(vold+1, n+1, mpi_comm_world, req(4), ierr)
109: call mpi_recv(p+1, n+1, mpi_comm_world, req(4), ierr)
110: endif
111: if(taskid.eq.0) then
112: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
113: if(numtasks-1.eq.1, mpi_comm_world, req(4), ierr)
114: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
115: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
116: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
117: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
118: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
119: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
120: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
121: if(numtasks-1.eq.1, mpi_comm_world, req(4), ierr)
122: call mpi_send(v(1,1), mpi_comm_world, req(4), ierr)
123: endif
124: endif
125: if(taskid.eq.0 or taskid.eq.numtasks-1) then
126: call mpi_waitall(12, req, istat, ierr)
127: endif
128: call f_prostop(17)
129: RETURN
130: END
131: SUBROUTINE CALC3Z
132: TIME.SmoothforFirstIteration
133: PARAMETER(N1=513, N2=513)
134: INCLUDE "mpif.h"
135: COMMON/decomp/js, je, taskid, numtasks, req(16),
136: istat(MPI_STATUS_SI)
137: integer taskid, req
138: COMMON U(N1,N2), v(N1,N2), p(N1,N2),
139: u(N1,N2), v(N1,N2), p(N1,N2),
140: n(N1,N2), v(N1,N2), p(N1,N2),
141: n(N1,N2), v(N1,N2), p(N1,N2),
142: n(N1,N2), v(N1,N2), p(N1,N2),
143: n(N1,N2), v(N1,N2), p(N1,N2),
144: n(N1,N2), v(N1,N2), p(N1,N2),
145: n(N1,N2), v(N1,N2), p(N1,N2),
146: n(N1,N2), v(N1,N2), p(N1,N2),
147: n(N1,N2), v(N1,N2), p(N1,N2),

© 2006 IBM Corporation
The IBM High Performance Computing Toolkit is a suite of performance-related tools and libraries to assist in application tuning. This toolkit is an integrated environment for performance analysis of sequential and parallel applications using the MPI and OpenMP paradigms. It provides a common framework for IBM’s mid-range server offerings, including pSeries and eSeries servers on Linux.

The HPC Toolkit is available from Absoft as a standalone product or as a component of the HPC SDK Enhanced Edition for Linux on POWER™.

### Key Benefits

#### Hardware Performance Monitor

Provides comprehensive reports of events that are critical to performance on IBM systems. HPM is able to gather the usual timing information, as well as critical hardware performance metrics, such as the number of misses on all cache levels, the number of floating point instructions executed, and the number of instruction loads that cause TLB misses, which all help the algorithm designer or programmer identify and eliminate performance bottlenecks.

#### MPI Tracer/Profiler

Consists of a set of libraries that collect profiling and tracing data for MPI programs. Performance metrics, such as the time used by MPI function calls and message sizes, are reported. MPI tracer works with visualization tools to help users identify performance bottlenecks.高峰maps performance metrics back to the

---

### Related Information

- Compatibility & Companion Products
- Technical Support
- Order Information
- Terms and Conditions
- Additional Information / Notices

### Purchase Options

- Academic
- Government
- Commercial
Productivity Limitations of Current Technology

- Automation not possible
  - Lack of a programmable dynamic binary approach.
- Dependency on instrumentation libraries limits HPCS needs for memory/data-movement optimization
  - Library-based methods are not symbolic:
    - If a measurement is invoked on a memory operation (e.g., L2 cache miss), they cannot provide information about which data structure the memory operation refers.
    - They cannot use symbolic entities of the application source program to describe the measurement event (e.g., “Count L1 cache misses every time the array A is touched”).
  - Library-based methods are not performance-oriented:
    - They cannot provide information about effects on the memory subsystem (e.g., What is the impact to L1 cache performance when loading array A in function foo?).
    - They cannot provide conditional control of the measurement process (e.g., “Count cycles only on each L1 cache miss of loading array A while in function foo…”).
Productivity Limitations of Current Technology

- Dynamic Probe Class Library (DPCL/DynInst) approaches (as used in the HPC Toolkit) are considered state-of-the-art today but have inherent productivity limitations:
  - System daemons running in the background are required
    - Acceptable in most cases, but does not optimize productivity needs.
  - Cannot perform instruction-level instrumentation
    - The DPCL approach is limited to function-boundary instrumentation only, which meets most HPC needs…but not those of HPCS.
  - Cannot dynamically respond to user-defined (or automated machine-generated) events and criteria
    - Requests from the user or intelligent agent cannot be filtered to decide whether to interrupt the execution of the program and invoke an event handler, which is also an automation inhibitor for HPCS.
  - Not data-centric
    - Execution cannot be intercepted when something happens to some selected data structure.
HPCS PERCS Proposed Deliverables

- **HPC Toolkit**
  - Integrated HPM, MPI, SHMEM, and OpenMP tools
  - Support for Fortran/95 and C/C++
  - Already established and widely available on multiple platforms

- **Modified to optimize productivity**
  - pSigma technology (binary approach)
  - DCA technology (data centric)
  - ASP technology (prediction)
  - autoPerf technology (automation)

- **Enhancements and New Tools**
  - Support for X10 and UPC and within Eclipse environment
  - Memory profiling and analysis
  - Cache visualization
  - I/O analysis
HPC Toolkit Versus HPCS Toolkit

- **HPC Toolkit**
  - End of life-cycle for new technology and research
  - Will be relegated to commercial product with support for IBM, Intel, AMD, etc. as defined by demand and economy.

- **HPCS Toolkit**
  - Potential to add “quantum leap” in productivity to application development
  - Complex/non-trivial to implement (labor-intensive, expensive)
  - Only possible with DARPA backing
Summary of Productivity Impact

- “End-to-End” Optimization of Application Performance-Tuning Cycle
  - **Intelligent Automation** (selective removal of programmer from tuning cycle)
    - Eliminates the traditional manual process of collecting performance information, trying to make sense of it, and then doing something about it.
    - Reduces complexity of the system to the application developer.
  - **Intelligent Prediction**
    - Alternate scenario evaluation capability provides fast-path to testing effects of altering data structures and/or code structure, without having to change any source code.
    - Automated usage provides solution validation component of intelligent automation.

- **Data Centric Approach**
  - Performance information is directly related to the application data structures.
  - Critical to understanding data movement and memory-related performance of shared memory hierarchical systems intended for HPCS.

- **Source Code Secure**
  - Eliminates user-injected bugs and any unintentional changes to source code.
HPCS Toolkit Community Activity

- **Conference Participation**
  - EuroPar 2005 (Sbaraglia, Eknath, et al.)
  - SC 2005 (Sbaraglia, Eknath, et al.)
  - IPDPS 2006 (Chung, Walkup, et al.)
  - ICS 2006 (Chung; Sbaraglia, Eknath, et al.)

- **Watson Petascale Tools Workshop**
  - SC PIC Supported
  - Participants: IBM, Academia, Government
  - May 3, 4 2005
  - May 16, 17 2006
Thank you!
Supplemental
### MPI Profiler Visualization

#### PeakPerf Main Window

<table>
<thead>
<tr>
<th>Label</th>
<th>Call Count</th>
<th>[Ma]</th>
</tr>
</thead>
<tbody>
<tr>
<td>-MPI_Allreduce_807</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>-MPI_Allreduce_868</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>-MPI_Barrier_1496</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-MPI_Barrier_248</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-MPI_Barrier_765</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>-MPI_Bcast_924</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>-MPI_Comm_rank_973</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-MPI_Comm_size_972</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-MPI_Iprobe_1160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-MPI_Recv_1027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-MPI_Recv_1135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-MPI_Ssend_1002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Allreduce_807</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Allreduce_868</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Barrier_1496</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Bcast_924</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Comm_rank_973</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Comm_size_972</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Iprobe_1160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Recv_1027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Ssend_1002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Metric Browser: MPI_Bcast_924

<table>
<thead>
<tr>
<th>Task</th>
<th>Message Size</th>
<th>WallClock</th>
<th>Count</th>
<th>Call Count</th>
<th>WallClock [Max]</th>
<th>Transferred Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(2) 5 ... 16</td>
<td>0.049632</td>
<td>133</td>
<td>133</td>
<td>0.049632</td>
<td>1064</td>
</tr>
<tr>
<td>0</td>
<td>(3) 17 ... 64</td>
<td>0.000144</td>
<td>2</td>
<td>2</td>
<td>0.000144</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>(4) 65 ... 256</td>
<td>0.001124</td>
<td>16</td>
<td>16</td>
<td>0.001124</td>
<td>1408</td>
</tr>
<tr>
<td>0</td>
<td>(5) 257 ... 1K</td>
<td>0.030647</td>
<td>163</td>
<td>163</td>
<td>0.030647</td>
<td>47408</td>
</tr>
<tr>
<td>0</td>
<td>(6) 1K ... 4K</td>
<td>0.011084</td>
<td>134</td>
<td>134</td>
<td>0.011084</td>
<td>198472</td>
</tr>
<tr>
<td>0</td>
<td>(7) 4K ... 16K</td>
<td>0.024244</td>
<td>285</td>
<td>285</td>
<td>0.024244</td>
<td>1.69084e+06</td>
</tr>
<tr>
<td>0</td>
<td>(8) 16K ... 64K</td>
<td>0.001328</td>
<td>16</td>
<td>16</td>
<td>0.001328</td>
<td>227200</td>
</tr>
<tr>
<td>0</td>
<td>(9) 64K ... 256K</td>
<td>0.078051</td>
<td>121</td>
<td>121</td>
<td>0.078051</td>
<td>1.09938e+07</td>
</tr>
<tr>
<td>1</td>
<td>(2) 5 ... 16</td>
<td>0.185036</td>
<td>133</td>
<td>133</td>
<td>0.185036</td>
<td>1064</td>
</tr>
<tr>
<td>1</td>
<td>(3) 17 ... 64</td>
<td>0.000183</td>
<td>2</td>
<td>2</td>
<td>0.000183</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>(4) 65 ... 256</td>
<td>0.009773</td>
<td>16</td>
<td>16</td>
<td>0.009773</td>
<td>1408</td>
</tr>
<tr>
<td>1</td>
<td>(5) 257 ... 1K</td>
<td>0.018897</td>
<td>163</td>
<td>163</td>
<td>0.018897</td>
<td>47408</td>
</tr>
<tr>
<td>1</td>
<td>(6) 1K ... 4K</td>
<td>0.019423</td>
<td>134</td>
<td>134</td>
<td>0.019423</td>
<td>198472</td>
</tr>
<tr>
<td>1</td>
<td>(7) 4K ... 16K</td>
<td>0.251916</td>
<td>285</td>
<td>285</td>
<td>0.251916</td>
<td>1.69084e+06</td>
</tr>
<tr>
<td>1</td>
<td>(8) 16K ... 64K</td>
<td>0.406296</td>
<td>16</td>
<td>16</td>
<td>0.406296</td>
<td>227200</td>
</tr>
<tr>
<td>1</td>
<td>(9) 64K ... 256K</td>
<td>0.087307</td>
<td>121</td>
<td>121</td>
<td>0.087307</td>
<td>1.09938e+07</td>
</tr>
</tbody>
</table>
MPI Tracer Visualization using PeekPerf – Original
MPI Tracer Visualization using PeekPerf – Tuned
OpenMP Profiler Visualization

```
...profile data...

!$omp do schedule(static)
    do i=1,nchunk
        !profile data...
    end do
!$omp end do
...
```

Metric Browser: loop_1125

<table>
<thead>
<tr>
<th>Task</th>
<th>Thread</th>
<th>Time in Master</th>
<th>T-T: Thread Time</th>
<th>C-T: Computation Time</th>
<th>%Imbalance</th>
<th>TO = TT - CT</th>
<th>%TO (Barter)</th>
<th>%TO (RTL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>13.3219</td>
<td>13.3219</td>
<td>13.3211</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15.8615</td>
<td>15.8615</td>
<td>15.8615</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>21.0235</td>
<td>21.0235</td>
<td>21.0235</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>24.4325</td>
<td>24.4325</td>
<td>24.4325</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>24.1006</td>
<td>24.1006</td>
<td>24.1006</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>25.0957</td>
<td>25.0957</td>
<td>25.0957</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>25.1062</td>
<td>25.1062</td>
<td>25.1062</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>23.5659</td>
<td>23.5659</td>
<td>23.5659</td>
<td>0</td>
<td>10.0073</td>
<td>0.000275</td>
<td>0.000510</td>
</tr>
</tbody>
</table>
Modular I/O Performance Tool

- **I/O Analysis**
  - Trace module
  - Summary of File I/O Activity + Binary Events File
  - Low CPU overhead

- **I/O Performance Enhancement Library**
  - Prefetch module (optimizes asynchronous prefetch and write-behind)
  - System Buffer Bypass capability
  - User controlled pages (size and number)
Example: MSC.Nastran V2001 using Modular I/O

Benchmark:
SOL 111, 1.7M DOF, 1578 modes, 146 frequencies, residual flexibility and acoustics. 120 GB of disk space.

Machine:
4-way, 1.3 GHz p655, 32 GB with 16 GB large pages, JFS striped on 16 SCSI disks.

MSC.Nastran:
V2001.0.9 with large pages, dmp=2 parallel=2 mem=700mb
The run with MIO used mio=1000mb

6.8 TB of I/O in 26666 seconds is an average of about 250 MB/sec

NO source code modifications needed!