Extending High Performance to High Productivity

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

- Roadmap for a Productivity Toolkit (HPCS)
- New Technologies Needed for Productivity
- Current Technology Review: HPC Toolkit
- Productivity Inhibitors of Current Technology
- HPCS Toolkit Proposed Deliverables
- Summary
Sanity Check on System Evolution

- Device Scaling imposing fundamental constraints on system power dissipation and energy consumption. This imposes constraints on physical size and packaging.
- Pressure to re-think system architecture. Examples include Blue Gene with low power devices, and Cell with an attached co-processing engine.
- Systems become inherently more complex due to connectivity and hierarchical topology (e.g., torus, intra-cell). Memory constraints (less per processor) also contribute to complexity.
- Additional technology "boosts" like hyper-threading and SMT introduce further complexity.
- This poses new challenges to application programming. There is no new programming language on the horizon, but legacy codes, ISV apps, etc., must be adapted.
- Conclusion: New software tools essential to mitigate evolving system complexity.
Projected Roadmap for a Productivity-Performance Toolkit

- **Phase I - Abstract Developer from the Source Code Instrumentation**
  - Core Productivity Infrastructure (pSigma)
  - New Tools to further leverage pSigma capability
    - Performance Prediction
    - Memory Simulation
    - I/O Profiling

- **Phase II - Abstracts Developer from the System**
  - 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

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

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

- **Data-Centric Analysis (DCA)**
  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.
The Proposed IBM HPCS Toolkit

- Binary Executable
  - pSigma
  - Communication Profiler
  - Memory Profiler
  - CPU Profiler
  - I/O Profiler
  - Shared-Memory Profiler
  - Instrumented Binary
  - Visualization
    - Query
    - Analysis

Binary instrumentation

Instrumented Binary
Data-Centric Analysis Technology (DCA)
autoPerf Technology (Automation)

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

Automatic performance analysis
- HPM
- FPU stalls
- L2 misses
- MPI

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.


- e.g., Communication imbalance: Array A
- 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
  Cell Systems
  Standalone or use within Eclipse environment

- Unified Performance Analysis Environment:
  Hardware Performance Monitor (HPM)
  MPI Profiler and Tracer
  SHMEM Profiler
  OpenMP Profiler
  Integrated GUI framework w/ source code traceback
Productivity Limitations of (All) 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 (All) 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)
    - Eliminates library dependency and corresponding limitations
    - Can dynamically respond to user-defined events to select instrumentation point
  - DCA technology (data centric)
  - ASP technology (prediction)
  - autoPerf technology (automation)

- **Enhancements and New Tools**
  - Support for UPC
  - Memory profiling and analysis
  - Cache visualization
  - I/O analysis
Thank you!
Supplemental
### MPI Profiler Visualization

**PeekPerf Main Window**

<table>
<thead>
<tr>
<th>Label</th>
<th>Call Count [Max]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Allreduce_807</td>
<td>16</td>
</tr>
<tr>
<td>MPI_Allreduce_868</td>
<td>38</td>
</tr>
<tr>
<td>MPI_Barrier_1496</td>
<td>1</td>
</tr>
<tr>
<td>MPI_Barrier_248</td>
<td>1</td>
</tr>
<tr>
<td>MPI_Barrier_765</td>
<td>4</td>
</tr>
<tr>
<td>MPI_Bcast_924</td>
<td>285</td>
</tr>
<tr>
<td>MPI_Comm_rank_973</td>
<td>5</td>
</tr>
<tr>
<td>MPI_Comm_size_972</td>
<td>5</td>
</tr>
<tr>
<td>MPI_Iprobe_1160</td>
<td></td>
</tr>
<tr>
<td>MPI_Recv_1027</td>
<td></td>
</tr>
<tr>
<td>MPI_Recv_1135</td>
<td></td>
</tr>
<tr>
<td>MPI_Ssend_1002</td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Allreduce_807</td>
<td></td>
</tr>
<tr>
<td>Summary_MPI_Allreduce_868</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Barrier_1496)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Barrier_248)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Barrier_765)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Bcast_924)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Comm_rank_973)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Comm_size_972)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Iprobe_1160)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Recv_1027)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Recv_1135)</td>
<td></td>
</tr>
<tr>
<td>Summary(MPI_Ssend_1002)</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 [Max]</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>1.6908e+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>1.0993e+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>0.00773</td>
<td>16</td>
<td>16</td>
<td>0.00773</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>1.6908e+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>1.0993e+07</td>
</tr>
</tbody>
</table>
MPI Tracer Visualization using PeekPerf – Original
MPI Tracer Visualization using PeekPerf – Tuned
OpenMP Profiler Visualization

---

The image displays a screenshot of a software interface with a focus on OpenMP Profiler Visualization. The interface includes various tables and code snippets, likely demonstrating performance metrics and code analysis. The tables show data on task execution times, thread times, and imbalance, while the code snippets illustrate the syntax and output related to OpenMP directives and functions.
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!