Performance Analysis of Computer Systems

Monitoring Techniques

Holger Brunst (holger.brunst@tu-dresden.de)
Matthias S. Mueller (matthias.mueller@tu-dresden.de)
Summary of Previous Lecture

Holger Brunst (holger.brunst@tu-dresden.de)
Matthias S. Mueller (matthias.mueller@tu-dresden.de)
Stream Benchmark for Memory Bandwidth

- Author: John McCalpin (“Mr Bandwidth“)

STREAM: measure memory bandwidth with the operations:
- Copy: \( a(i) = b(i) \)
- Scale: \( a(i) = s * b(i) \)
- Add: \( a(i) = b(i) + c(i) \)
- Triad: \( a(i) = b(i) + s * c(i) \)

STREAM2: measures memory hierarchy bandwidth with the operations:
- Fill: \( a(i) = 0 \)
- Copy: \( a(i) = b(i) \)
- Daxpy: \( a(i) = a(i) + q * b(i) \)
- Sum: \( \text{sum} += a(i) \)
## Stream 2 properties

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Code</th>
<th>Bytes/iter read</th>
<th>Bytes/iter written</th>
<th>FLOPS/iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL</td>
<td>$a(i) = q$</td>
<td>0 (+8)</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>COPY</td>
<td>$a(i) = b(i)$</td>
<td>8 (+8)</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>DAXPY</td>
<td>$a(i) = a(i) + q*b(i)$</td>
<td>16</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>SUM</td>
<td>sum = sum + a(i)</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Stream 2 Results

\[ a(i) = b(i) + \alpha c(i) \]

- NEC_Azusa_Intel_Itanium_azusa_efc
- Pentium4_1400MHz_loan1_ifc

log_2(loop length)
What is LINPACK NxN

- LINPACK NxN benchmark
  - Solves system of linear equations by some method
  - Allows the vendors to choose size of problem for benchmark
  - Measures execution time for each size problem

- LINPACK NxN report
  - Nmax – the size of the chosen problem run on a machine
  - Rmax – the performance in Gflop/s for the chosen size problem run on the machine
  - N1/2 – the size where half the Rmax execution rate is achieved
  - Rpeak – the theoretical peak performance Gflop/s for the machine

- LINPACK NxN is used to rank TOP500 fastest computers in the world
HPCS Performance Targets

- HPCC was developed by HPCS to assist in testing new HEC systems.
- Each benchmark focuses on a different part of the memory hierarchy.
- HPCS performance targets attempt to:
  - Flatten the memory hierarchy
  - Improve real application performance
  - Make programming easier

**HPC Challenge Performance Targets**

<table>
<thead>
<tr>
<th>Max</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Pflop/s</td>
<td>8x</td>
</tr>
<tr>
<td>6.5 Pbyte/s</td>
<td>40x</td>
</tr>
<tr>
<td>0.5 Pflop/s</td>
<td>200x</td>
</tr>
<tr>
<td>64000 GUPS</td>
<td>2000x</td>
</tr>
</tbody>
</table>

**Examples of Benchmarks**

- **HPL**: linear system solve
  \[ Ax = b \]

- **STREAM**: vector operations
  \[ A = B + s \cdot C \]

- **FFT**: 1D Fast Fourier Transform
  \[ Z = \text{fft}(X) \]

- **RandomAccess**: integer update
  \[ T[i] = \text{XOR}(T[i], \text{rand}) \]
HPC Challenge Benchmark

- Consists of basically 7 benchmarks;
  - Think of it as a framework or harness for adding benchmarks of interest.
- HPL (LINPACK) — MPI Global \((Ax = b)\)
- STREAM — Local; single CPU
  * STREAM — Embarrassingly parallel
- PTRANS \((A \ A + BT)\) — MPI Global
- RandomAccess — Local; single CPU
  * RandomAccess — Embarrassingly parallel
- RandomAccess — MPI Global
- BW and Latency — MPI
- FFT - Global, single CPU, and EP
- Matrix Multiply — single CPU and EP
Tests on Single Processor and System

- **Local** - only a single processor is performing computations.

- **Embarrassingly Parallel** - each processor in the entire system is performing computations but they do no communicate with each other explicitly.

- **Global** - all processors in the system are performing computations and they explicitly communicate with each other.
Computational Resources and HPC Challenge

- HPL Matrix Multiply
- CPU computational speed
- Computational resources
- Memory bandwidth
- Node Interconnect bandwidth
- Random & Natural Ring Bandwidth & Latency

STREAM

ZIH Center for Information Services & High Performance Computing
Memory Access Patterns

- **STREAM (EP & SP)**
- **PTRANS (G)**
- **HPL Linpack (G)**
- **Matrix Mult (EP & SP)**

**Applications:***

- **Computational Fluid Dynamics**
- **Radar Cross Section**
- **Traveling Sales Person**
- **Digital Signal Processing**
- **Zoom-FFT Algorithm**

**Figure 1.** Concept of Radar Cross Section

**Spatial Locality**

**Temporal Locality**
## Condensed Results - Base Runs Only - 106 Systems - Generated on Mon Jun 26 09:17:02 2006

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Atipa Conquest cluster AMD Opteron</td>
<td>1.4GHz 1 128 1 128</td>
<td>0.252610</td>
<td>3.2471</td>
<td>208.525</td>
<td>1.6261</td>
<td>0.05627</td>
<td>23.66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustersvision BV Beesley AMD Opteron</td>
<td>2.4GHz</td>
<td>32 1 32</td>
<td>0.103764</td>
<td>0.8159</td>
<td>0.0000235</td>
<td>2.1470</td>
<td>106.951</td>
<td>3.3422</td>
<td>4.1943</td>
<td>0.02668</td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 64 1 64</td>
<td>0.521560</td>
<td>3.2288</td>
<td>959.334</td>
<td>14.9566</td>
<td>0.94074</td>
<td>20.34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 60 1 60</td>
<td>0.577790</td>
<td>20.4312</td>
<td>898.446</td>
<td>14.9741</td>
<td>1.02291</td>
<td>20.62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 120 1 120</td>
<td>1.068790</td>
<td>2.4603</td>
<td>1019.519</td>
<td>8.4860</td>
<td>0.83014</td>
<td>20.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray T3E Alpha 21164</td>
<td>0.6GHz 1024 1 1024</td>
<td>0.0481693</td>
<td>10.2765</td>
<td>529.242</td>
<td>0.5160</td>
<td>0.03314</td>
<td>12.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 252 1 252</td>
<td>2.384290</td>
<td>97.4076</td>
<td>3729.404</td>
<td>14.9413</td>
<td>0.42899</td>
<td>22.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 124 1 124</td>
<td>1.2054200</td>
<td>39.3232</td>
<td>1856.964</td>
<td>14.9771</td>
<td>0.70857</td>
<td>20.17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 60 1 60</td>
<td>0.5058740</td>
<td>1.6342</td>
<td>0.0020750</td>
<td>2.1444</td>
<td>694.114</td>
<td>14.9019</td>
<td>10.91520</td>
<td>1.16779</td>
<td>14.66</td>
</tr>
<tr>
<td>Cray T3E Alpha 21164</td>
<td>0.67GHz 512 1 512</td>
<td>0.233180</td>
<td>9.7741</td>
<td>0.0288644</td>
<td>15.4774</td>
<td>272.166</td>
<td>0.5316</td>
<td>0.66077</td>
<td>0.03571</td>
<td>8.14</td>
</tr>
<tr>
<td>Cray X1I AMD Opteron</td>
<td>2.26GHz 64 1 64</td>
<td>0.223890</td>
<td>10.5924</td>
<td>0.0223566</td>
<td>16.8011</td>
<td>159.359</td>
<td>2.6555</td>
<td>4.03375</td>
<td>0.22697</td>
<td>1.63</td>
</tr>
<tr>
<td>Cray X1 MSP</td>
<td>0.8GHz 32 1 32</td>
<td>0.2767140</td>
<td>32.6836</td>
<td>0.0166620</td>
<td>2.9849</td>
<td>475.846</td>
<td>14.8702</td>
<td>8.28248</td>
<td>1.41289</td>
<td>14.94</td>
</tr>
<tr>
<td>Cray XT3 AMD Opteron</td>
<td>2.6GHz 1100 1 1100</td>
<td>4.762340</td>
<td>217.9230</td>
<td>0.1370020</td>
<td>266.6600</td>
<td>5274.588</td>
<td>4.7952</td>
<td>4.61050</td>
<td>0.26637</td>
<td>25.94</td>
</tr>
<tr>
<td>Cray X1I AMD Opteron</td>
<td>2.4GHz 128 1 128</td>
<td>0.5030760</td>
<td>10.8515</td>
<td>0.0666722</td>
<td>35.5152</td>
<td>500.069</td>
<td>2.9068</td>
<td>4.23425</td>
<td>0.25919</td>
<td>2.06</td>
</tr>
<tr>
<td>Cray X1E X1E MSP</td>
<td>1.13GHz 252 1 252</td>
<td>3.1945908</td>
<td>85.2040</td>
<td>0.1486884</td>
<td>15.3353</td>
<td>2439.985</td>
<td>9.6835</td>
<td>14.18470</td>
<td>0.36024</td>
<td>14.93</td>
</tr>
<tr>
<td>Cray XT3 AMD Opteron</td>
<td>2.4GHz 8744 1 8744</td>
<td>14.7402000</td>
<td>608.3060</td>
<td>0.2202950</td>
<td>417.1720</td>
<td>16146.382</td>
<td>4.0460</td>
<td>4.413200</td>
<td>0.16154</td>
<td>25.32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XT3 AMD Opteron</td>
<td>2.4GHz 5200 1 5200</td>
<td>20.5270000</td>
<td>674.6990</td>
<td>0.2095000</td>
<td>644.7200</td>
<td>260.2000</td>
<td>5.0040</td>
<td>4.29353</td>
<td>0.14622</td>
<td>25.80</td>
</tr>
<tr>
<td>Cray xt3 AMD Opteron</td>
<td>2.4GHz 32 1 32</td>
<td>0.1387810</td>
<td>7.3764</td>
<td>0.0326000</td>
<td>9.3983</td>
<td>158.424</td>
<td>4.8883</td>
<td>4.77641</td>
<td>0.57281</td>
<td>7.84</td>
</tr>
<tr>
<td>Cray X1E</td>
<td>1.13GHz 4070 1 8600</td>
<td>16.3750000</td>
<td>202.9790</td>
<td>0.5320720</td>
<td>905.5690</td>
<td>2065.456</td>
<td>5.0421</td>
<td>4.78166</td>
<td>0.18965</td>
<td>9.44</td>
</tr>
<tr>
<td>Cray XT2 AMD Opteron</td>
<td>2.6GHz 1100 1 1100</td>
<td>4.7279660</td>
<td>233.3460</td>
<td>0.358608</td>
<td>328.8260</td>
<td>5161.134</td>
<td>4.6919</td>
<td>4.77440</td>
<td>0.3967</td>
<td>7.29</td>
</tr>
<tr>
<td>Cray Inc XT3 AMD Opteron</td>
<td>2.4GHz 2520 1 2520</td>
<td>20.4006000</td>
<td>944.2270</td>
<td>0.6724120</td>
<td>751.7290</td>
<td>2426.447</td>
<td>4.6590</td>
<td>4.41173</td>
<td>0.20636</td>
<td>9.20</td>
</tr>
<tr>
<td>Cray Inc XT3 AMD Opteron</td>
<td>2GHz 10130 1 10130</td>
<td>35.2985000</td>
<td>1813.0800</td>
<td>1.0176500</td>
<td>1118.2900</td>
<td>43581.780</td>
<td>4.2108</td>
<td>3.86719</td>
<td>0.15188</td>
<td>10.32</td>
</tr>
<tr>
<td>Cray Inc 1X Cray E</td>
<td>1.13GHz 1008 1 1008</td>
<td>12.0263000</td>
<td>106.0190</td>
<td>0.0651190</td>
<td>82.2604</td>
<td>15522.031</td>
<td>15.0599</td>
<td>14.50000</td>
<td>0.15667</td>
<td>16.30</td>
</tr>
<tr>
<td>Cray Inc XT3 AMD Opteron</td>
<td>2.6GHz 4128 1 4128</td>
<td>16.6421000</td>
<td>674.7860</td>
<td>0.6767490</td>
<td>821.6770</td>
<td>19295.876</td>
<td>4.6743</td>
<td>4.72946</td>
<td>0.22249</td>
<td>8.27</td>
</tr>
<tr>
<td>Daloce Opteron/Quillot Linux Cluster AMD Opteron</td>
<td>2.2GHz 64 1 64</td>
<td>0.2180410</td>
<td>6.3195</td>
<td>0.0047003</td>
<td>13.5481</td>
<td>133.394</td>
<td>2.3968</td>
<td>3.87865</td>
<td>0.17003</td>
<td>11.46</td>
</tr>
</tbody>
</table>
NPC Challenge Benchmark

Benchmarks normalize to the show the highest performance with a value of 1

- **PP-HPL**
- **PP-PTAHM**
- **PP-RandomAccess**
- **SN-STREAM Triad**
- **RandomRing Latency**
- **RandomRing Bandwidth**

**Legend:**
- Gray XL 1: Fujitsu-Siemens - 32 procs - 0.8 GHz
- 1 thread/MPI process (32) - Cray modified 2-0 Torus - 11-22-2004
- NEC SX-6 32 procs - 0.5 GHz
- 1 thread/MPI process (32) - InfiniBand Crossbar Switch - 11-04-2004
- SGI Altix 3700 8x2 Intel Itanium 2 32 procs - 1.6 GHz
- 1 thread/MPI process (32) N/A 03-15-2005
- Dell PowerEdge 2650 Cluster Intel Xeon 32 procs - 2.4 GHz
- 1 thread/MPI process (32) Gigabit Ethernet, PowerConnect 5224 switch - 02-18-2005

Differences in the benchmark results between computers, even of the same model, can be a result of the number of processors used, the number of threads used, the processor interconnect, the amount of memory allocated for the run, the version of the BLAS and MPI, and other factors. A complete listing of the environment for each benchmark run can be found at: [http://icl.cs.utk.edu/mpc/export/mpc,xhtml](http://icl.cs.utk.edu/mpc/export/mpc,xhtml)
Monitoring Techniques

Holger Brunst (holger.brunst@tu-dresden.de)
Matthias S. Mueller (matthias.mueller@tu-dresden.de)
When the only tool you have is a hammer,

every problem begins to resemble a nail.

Abraham Maslow
Outline

- Introduction
- General: Terminology and classification
- Trigger Mechanisms
- Interval Timers
- Program Execution Monitors
- Instrumentation
- Tools and their Formats
**Introduction**

- A monitor is a tool to observe the activities on a system

- Reasons to monitor a system:
  - System programmer:
    - Find frequently used segments of a program and optimize their performance
  - System manager:
    - Measure resource utilization and find performance bottleneck(s)
    - Tune the system by adjusting system parameters
  - System Analyst:
    - Use monitor data to characterize the workload for capacity planning
    - Find model parameters, validate models, and find model inputs
General: Monitor Terminology

- **Event**: A change in the system’s state
  - Measuring any kind of values typically based around the idea of events
  - Examples: Memory reference, disk access, network communication operation, change in a processor’s internal state, pattern or a combination of other sub-events

- **Trace**: A log of timed events including event type and important parameters

- **Overhead**: The perturbation to the system induced by the monitor

- **Domain**: The set of observable activities forms the domain of a monitor

- **Input Rate**: Maximum frequency of events a monitor can correctly observe

- **Resolution**: The coarseness of the information observed

- **Input Width**: The size of information recorded per event
General: Monitor Classification

- System level at which monitor is implemented:
  - Software monitor
  - Hardware monitor
  - Firmware monitor
  - Hybrid monitor

- Trigger mechanisms:
  - Event-driven
  - Timer-driven

- Recording:
  - Profiling
  - Tracing

- Displaying ability:
  - On-line
  - Batch/Post mortem
Trigger Mechanisms: Event-driven

- Measure performance whenever the pre-selected event occurs
- Simplest type: event counter
- Perturbation to the system might be small if the event occurs infrequently
- High-frequency events: great deal of overhead may be introduced
- Can significantly alter program behavior
- Inter-event time can be highly variable and completely unpredictable
- Perturbation assessment not easy
Trigger Mechanisms: Timer-driven

- Measures at fixed time intervals the portion of the system state
- Overhead due to this strategy is independent of the number of times a specific event occurs
- Is instead function of the sampling frequency
- Determined by the resolution necessary
- Not every occurrence of the events will be measured
- Sampling produces statistical view on the overall behavior of the system
- Events that occur infrequently may be completely missed
- Each run of a sampling-experiment is likely to produce a different result
- Exact behavior may differ, statistical behavior should remain approximately the same
Interval Timers

- Most fundamental measuring tool in computer-system performance analysis
- Used to measure execution time
- Can provide time basis for sampling measurement tool
- Are based on the idea of counting the number of clock pulses
- Hardware timers:
  - Counter typically initialized with power up
  - Difference gives time interval
- Software timers:
  - Software interrupt based interval timer
  - Hardware counter used to initiate interrupt
  - Value is a count of the number of interrupts
- Time rollover:
  - Important is the number of bits available for counting
### Interval Timers: Rollover Time

<table>
<thead>
<tr>
<th>Resolution</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>48</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,00E-09</td>
<td>6,5536E-05</td>
<td>1,6777E-02</td>
<td>4,2950E+00</td>
<td>2,8147E+05</td>
<td>1,8447E+10</td>
</tr>
<tr>
<td>1,00E-08</td>
<td>6,5536E-04</td>
<td>1,6777E-01</td>
<td>4,2950E+01</td>
<td>2,8147E+06</td>
<td>1,8447E+11</td>
</tr>
<tr>
<td>1,00E-07</td>
<td>6,5536E-03</td>
<td>1,6777E+00</td>
<td>4,2950E+02</td>
<td>2,8147E+07</td>
<td>1,8447E+12</td>
</tr>
<tr>
<td>1,00E-06</td>
<td>6,5536E-02</td>
<td>1,6777E+01</td>
<td>4,2950E+03</td>
<td>2,8147E+08</td>
<td>1,8447E+13</td>
</tr>
<tr>
<td>1,00E-05</td>
<td>6,5536E-01</td>
<td>1,6777E+02</td>
<td>4,2950E+04</td>
<td>2,8147E+09</td>
<td>1,8447E+14</td>
</tr>
<tr>
<td>1,00E-04</td>
<td>5536E+00</td>
<td>1,6777E+03</td>
<td>4,2950E+05</td>
<td>2,8147E+10</td>
<td>1,8447E+15</td>
</tr>
</tbody>
</table>

| RTC         |     |     |     |     |     |
| 4 Seconds   |     |     |     |     |     |

Often bad: gettimeofday, or MPI_Wtime
Interval Timers: Overhead

- Time overhead
  - xstart = read_timer()
  - <event being timed>
  - x_end = read_timer()
  - elapsed_time = (x_end – x_start) * t_cycle

Time we measure includes more than the time required for the event
- Timer may require operating system call
- If the interval being measured is substantially larger than time overhead: no problem
- Alternatively, overhead can also be subtracted
- But! Overhead subtraction impossible for concurrent processes depending on each other
Interval Timers: Measuring Short Intervals

- Based on quantization effects, we cannot directly measure events whose durations are less than the resolution of the timer.
- We can however make many measurements of a short duration event to obtain a statistical estimate of the event's duration.
- Problem: Events need to take place within the timers resolution from time to time.
- Problem 2: Only average values.
Use of inaccurately synchronized timers results in an erroneous representation of the program trace data:

- Q1 Qualitative error: Violation of the logical order of distributed events.
- Q2 Quantitative error: Distorted time measurement of distributed activities. Leads to skewed performance values.
Synchronization of Multiple Timers: Standard

- **Hardware synchronization:**
  - Tight synchronization, cost-intensive, not portable

- **Software synchronization:**
  - Asymmetric:
    - no load-balancing, reference timer can be false ticker and is bottleneck
  - Symmetric: needs $O(n^2)$ messages, not scalable to thousands of processes
  - Controlled logical clock: corrects violation of logical order (Q1), no correction of skewed performance values (Q2)
Synchronization of Multiple Timers: Goals

Need for a novel timer synchronization with respect to the requirements of parallel event tracing:

- Load-balanced, low synchronization overhead,
- Portable, scalable and robust synchronization algorithm,
- Restore the relationship of concurrent events,
- Accurate mapping of the event flow for an enhanced performance analysis.
Synchronization of Multiple Timers: Solution

- Two parts of the synchronization scheme:
  - Recording synchronization information during runtime
  - Subsequent correction, i.e. Transformation of asynchronous local time stamps to synchronous global time stamps with a linear interpolation

- Due to small fluctuations in the timer drift the synchronization error will be accumulated over long intervals

- Linear begin-to-end correction insufficient for long trace runs

- Synchronize the timers frequently and piecewise interpolate the timer parameters between the synchronization phases
Program Monitors: PC Sampling

- General statistical measurement technique in which a subset (i.e. a sample) of the members of a population being examined is selected at random

- Information of interest is gathered from the subset of the total population

- Assumption: Since the samples were chosen completely at random, the characteristics of the overall population will approximately follow the same proportion as do the characteristics of the subset actually measured

- Profile: Samples are taken at fixed times

- Interrupt service routine examines the return address stack to find the address of the instruction/function that was executed
Program Monitors: PC Sampling

Time

Sample

Task 1

Task 2

Task 3

Sample Monitors: PC Sampling
Program Monitors: Basic Block Counting

- Produces an exact execution profile by counting the number of times each basic block is executed.
- Basic block is sequence of processor instructions that has no branches into or out of the sequence.
- Additional instructions simply count the number of times the block is executed.
- After termination: values form a histogram.
- Show how often each block is executed.
- Complete instruction execution frequency counts can also be obtained from these counts.
- Key difference between basic block profiling and PC sampling: basic block profiling gives the exact execution frequencies of all instructions.
- Can add substantial amount of runtime overhead! Average number of instructions varies between three and 20!
- Overhead: More instructions, different memory behavior!
Program Monitors: Indirect Strategies

- Indirect strategy has to be used if metric is not directly accessible.
- Try to deduce and derive the desired performance metric from related event which can be measured.
- Development of an appropriate indirect measurement strategy which minimizes the overhead: difficult, needs experience.
- Impossible to make any general statements about a measurement tool that makes use of an indirect strategy.
- Key: match the characteristics of the desired metric with the appropriate measurement strategies.
Program Monitors: Tracing

- Not only simply recording the fact that the event has happened
- Stores some portion of the system state
- Instead of keeping just the number of page faults, a tracing record strategy may store the addresses that caused the page fault.
- Requires significantly more storage
- Time required to save the state can significantly alter program behavior being measured
Program Monitors: Tracing

1. Application
   Monitor

2. Application
   Monitor

3. Application
   Monitor

4. Application
   Monitor

. . .

10,000

Trace Data

Performance Visualization

Enable Scalability

LARS: Monitoring – Slide 35
Program Monitors: Tracing

- Profiling: provides summary information
- Profiling does not provide any information about the order in which the instructions were executed

- Trace:
  - Dynamic list of the events generated by the program as it executes
  - Time ordered list of:
    - all of the instructions executed
    - sequences of memory addresses accessed by a program
    - sequences of disk blocks referenced by the file system
    - sizes and destination of all messages sent over a network
Program Monitors: Tracing

- Several difficulties
  - Execution-time slowdown
  - Other program perturbations by the execution of the additional tracing information
  - Volume of data
  - Disk speed, organization of the whole process

- Advantages:
  - Very detailed
  - Summarized information can be computed for arbitrary time intervals
  - Useful for both performance tuning and debugging
  - Easy identification of synchronization issues
Instrumentation

- Source-code modification
- Software exceptions
- Emulation
- Microcode modification
- Library approach
- Compiler modification
Instrumentation: Source Code Modification

- Programmer may add additional tracing statements to the source code manually
- Additional program statements will be executed after compilation
- Programmer can determine which parts he wants to instrument
- Disadvantage:
  - Manual approach
  - Time consuming
  - Error prone
  - Programmers mostly believe that they have a clear understanding of the program execution and instrument only small code areas
Some processors support software exceptions just before the execution of each instruction.

Exception routine can decode the instruction to determine its operands.

Accurate but:
- Slowed down execution time by a factor of about 1000.
- By far too detailed in most cases.
Instrumentation: Emulation

- Emulator is a program that makes the system on which it executes appear to the outside as if it were something completely different.

- Java Virtual Machine executes application programs written in the Java programming language by emulating the operation of a processor that implements Java byte-code instructions.

- Tracing then straightforward, but:
  - slows down execution significantly
  - Not clear how to implement selective tracing
Parallel programs most often use communication libraries
These libraries can be instrumented easily
Communication is two sided in many cases
Merging results is quite a challenge
Gives quite a good overview about the program behavior
Instrumentation: Compiler Modification

- Modify the executable code produced by the compiler
- Similar to basic block profiling
- Details about the content of the basic block can be obtained from the compiler
- Two versions:
  - Compilation option
  - Post-compilation software tool
Several parallel trace formats exist

- Different trace formats for different performance systems
  - VTF (Vampir), EPILOG (Kojak), SLOG2 (JumpShot-4), TAU
  - All public domain
- No real common format or one with special emphasis on scalability

Community has no portable scalable tracing system

- How to support open source and cross-platform tracing tools?
- Mainly concerned with robust analysis and visualization
- Target an open scalable trace format and get community support
Summary

- Monitor terminology and classification
- Trigger mechanisms
- Interval timers
- Program execution monitors
- Instrumentation
- Tools and their formats