Performance Complexity:
A New Execution Time Metric

Erich Strohmaier
Future Technology Group
EStrohmaier@lbl.gov

HPEC 2006, Sep. 21, 2006
• The **Programming Complexity** of our systems is of growing concern.
  – At least with respect to performance.
  – Performance transparency is sinking!

• **Can we develop a measurement based metric** for PC?
  – The values in this metric will depend on the codes we use for measuring.
    • Just like performance does.
• **Performance Models** help us to learn about and show how well we understand:
  – algorithms,
  – Systems, and
  – programming environments.

• Can we use them to give us a measure for:
  – How complex our systems are?
  – How difficult they are to program (from a performance point of view)?
If performance of a system is **predictable**, it is easy controllable and ‘programmable’.

**Sum of Squared Errors (SSE)** measures how well we model our performance data.

Can we use SSE to compare performance transparency/programming complexity?
• Use a defined "benchmark set".
• Measure performance of these benchmarks.
• Predict performance with performance models.
• Use a metric derive from SSE for the whole set to compare systems.
  – Do sound statistics (log-transformations).
• Define two execution time metrics:
  – Performance (P) and
  – Performance Complexity (PC) in same dimensions.
### P and PC Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute PC</td>
<td>$PC_a = \exp\left(\sqrt{\text{SSE}(\log P, \log M)}\right) - 1$</td>
<td>[ops/cycle]</td>
</tr>
<tr>
<td>Relative PC</td>
<td>$PC_r = \exp\left(\sqrt{\frac{\text{SSE}}{\text{SS}'}}\right) - 1$</td>
<td>[]</td>
</tr>
</tbody>
</table>

“Geometric mean” and “geometric standard deviation”
As first exercise we use a synthetic parameterized benchmark - APEX-Map!

- Each parameter set represents a benchmark.
- Only one set of performance models.
- Still spans a nice range of execution requirements.

Our metric is “cycles/memory access”.
• Local and global memory access is the dominant performance bottleneck for many scientific applications.
• Synthetic benchmarks are suited to investigate details of data access as they are easier to handle and understand than application benchmarks.
• Dependency of performance on spatial and temporal data locality features is of great interest.
• Parameterized benchmarks allow to explore the dependency of performance of a whole range of descriptive parameters by generating multi-dimensional performance maps.
• Characterize a data access stream quantitatively with as few parameters for spatial and temporal locality as possible.
  – Use the same concepts for temporal and process locality for parallel execution.
  – Simulate temporal locality by more frequent accesses to certain memory locations.
  – Ignore other effects such as sharing of variables or process affinity.
• Create a parameterized synthetic benchmark, which generates a synthetic stream of references, to investigate data access performance.
• Data set size: $M$
• Spatial Locality ($L$):
  – Blocked access to $L$ contiguous data elements.
  – $L$ is also the innermost loop length!
• Temporal Locality ($\alpha$):
  – Achieve more frequent access to certain memory locations by using non-uniform random starting addresses of blocks distributed according to a power law.
  – Characterize temporal locality with the exponent $\alpha$ of the power law.
• Use the Power distribution as non-uniform random address generator.
  - Self-similar and thus scale invariant.
  - Single descriptive parameter.
  - Exponent $\alpha$ in $[0,1]$
    - $\alpha=1$ : Uniform random access.
    - $\alpha=0$ : Access to a single vector only.
Effect of Alpha
Local:Remote Memory Ratio of 1:256
**Sequential:**

Repeat N Times
- Generate Index Array()
- CLOCK(start)
- For each Index $i$ in the Array
  - Compute()
- CLOCK(end)
- RunningTime += end - start

**Parallel:**

Repeat N Times
- Generate Index Array()
- CLOCK(start)
- For each Index $i$ in the Array
  - If (remote data)
    - GetRemoteData()
  - End If
  - Compute()
  - CLOCK(end)
- RunningTime += end - start

**Depends on the chosen parallel programming model**
Parallel APEX-Map Design

- Same parameters as sequential version ($\alpha$, $L$, $M*P$).
- Data evenly distributed among processes.
- $X$ distributed across all processes.
- Each remote access results in a message with length $L$ (for the message passing paradigm).
Parallel Performance Surfaces
256 Processors - MPI

Cheetah – IBM SP Power4

Phoenix – Cray X1
Limitations

- Biases of this study:
  - Benchmark selection (Only one code)
  - Parameter range selection
    - \( L \) and \( \alpha \)
    - \( M = 512 \) MB/process fixed
    - \( l=1024 \)
  - Power of 2 \( L \) parameter scaling
  - Model selection
• Use **four performance models**:
  
  – **Ideal, flat memory**
    
    *T = const*
  
  – **Two level memory hierarchy (c and m)**
    
    *T = P(c/m)*a + (1-P(c/m))*b*
  
  – **Linear access timing to single level memory**
    
    *T = [a +(L-1)*b]/L*
  
  – **Combined**: Linear timing for two levels
    
    *T = [P(c/m)*(a+b*(L-1)) + (1-P(c/m))*(c+d*(L-1)) ]/L*

  
• **We back-fit effective values for parameters.**
PC Values for 3 simple Models

Performance Complexity - Model 0, 1, 2

- Flat memory
- naïve cache
- Vector unit

<table>
<thead>
<tr>
<th>Model</th>
<th>Power3</th>
<th>Power4</th>
<th>Power5</th>
<th>Xeon</th>
<th>Itanium</th>
<th>Opteron</th>
<th>X1 MSP</th>
<th>X1 SSP</th>
<th>SX8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG/L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xeon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Itanium</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opteron</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1 MSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1 SSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SX8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PC Values for 4 simple Models

Performance Complexity - Model 0, 1, 2, 3

- Flat memory
- naïve cache
- Vector unit
- cache and vector

Models:
- BG/L
- Power3
- Power4
- Power5
- Xeon
- Itanium
- Opteron
- X1 MSP
- X1 SSP
- SX8
Residual Error - Sequential

Residual Errors - Itanium

![Graph showing residual errors for Itanium processors at different values of L and SE.](image-url)
Residual Error - Sequential

Residual Error - Opteron

0.0000 0.0050 0.0100 0.0150 0.0200 0.0250 0.0300 0.0350 0.0400 0.0450 0.0500 0.0550

0.000 0.001 0.010 0.100 1.000

a

L
Latency

<table>
<thead>
<tr>
<th>System</th>
<th>Local Latency</th>
<th>Global Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG/L</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Power3</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>Power4</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>Power5</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>Xeon</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>Itanium</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>Opteron</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>X1 MSP</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>X1 SSP</td>
<td>100</td>
<td>1,000</td>
</tr>
<tr>
<td>SX8</td>
<td>100</td>
<td>1,000</td>
</tr>
</tbody>
</table>
Serial Perf. Model Parameters

Lower is better
P-C Maps for all Four Performance Models for Serial Execution

Model 0 - Flat Memory

Model 1 - Naïve Cache

Model 2 - Vector Unit

Model 3 - Cache and Vector
Performance Complexity - Model 0, 1, 2

- **Flat memory**
- **2 Level Memory**
- **Linear Timing**

PC metrics for different models:
- BG/L
- SP Power3
- SP Power4
- SP Power5
- Itanium CI
- Opteron CI
- X1 MPI
- X1 shmem
- X1 UPC
- X1 UPC blocked
- X1E UPC
- X1E CAF
- NEC SX6
Residual Error - Parallel

Residual Error - BG/L

- Parameters: L, SE
- Values: 1, 4, 16, 64, 256, 1024, 4096, 16384, 65536
- Error Values: 0.001, 0.010, 0.100, 1.000

Graph showing the residual error with varying values of L and SE.
Residual Error - Parallel

Residual Error - Opteron Infiniband

SE

L

1
4
16
64
256
1024
4096
16384
65536
P-C Maps for all Four Performance Models for Parallel Execution

Model 0 - Flat Memory

Model 1 - Two Level Memory

Model 2 - Linear Time Model

Model 3 - Two Level Memory with Linear Timing Model
Conclusions

• The combination of benchmarks and performance models allows to characterize the performance complexity of systems quantitatively.
• Two dimensional performance complexity maps enable a high-level comparison of different systems.
• Performance models allow to analyze system behavior through residual errors in great detail.
• APEX-Map is a parameterized synthetic data access benchmark, which allows to map performance for a large range of temporal and spatial localities systematically.
• These performance maps reflect the performance impact of different system and memory features.