Coarse-grained Simulation Method for Performance Evaluation of a Shared Memory System

R. Kawahara, K. Nakamura, K. Ono, T. Nakada
(IBM Research – Tokyo, Japan)

Y. Sakamoto
(Global Business Services, IBM Japan, Japan)
Abstract

- **Purpose**
  - Performance evaluation at early stage of development

- **Subject**
  - Evaluation of memory access contention without knowing precise memory access timings

- **Our method**
  - Exploit a statistical approximation which assumes that memory accesses are random and uncorrelated

- **Comparison with preliminary experiment**
  - Error of order of 3% on the execution time
Introduction: Application-level performance estimation at early stage of development

UML is used to clarify software specification

It is difficult to evaluate application performance because it requires platform resources to be taken into account
Introduction: Memory Access Contention

- **Context**
  - Improvement of performance for next-generation product
- **Metric of performance**
  - execution time of application

![Diagram showing memory access contention between Old MPU1 and New MPU1, and MPU2, with synchronous access and contention at Memory]
Related works: UML-based simulation

Cortellessa et al. (2007)
- Evaluation of resource contention
- Executable UML model
  - Application
  - Platform
  - Parameters
  - Processing time of each step

Difficult to determine parameters

Simulation method of resources other than CPU is unknown

Ono et al. (2010)
- Parameters can be obtained by measurements of existing product

Our study (for memory bandwidth resource)
Related works: Simulation methods of memory access contention

<table>
<thead>
<tr>
<th>Dev. phase</th>
<th>Methods</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-level</td>
<td>Queuing</td>
<td>System requirements</td>
<td>Statistics</td>
</tr>
<tr>
<td>analysis</td>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML-level</td>
<td>??</td>
<td>Architecture (HW / SW), Step time</td>
<td>Execution time, Step trace</td>
</tr>
<tr>
<td>design</td>
<td>????</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESL-level</td>
<td>SystemC</td>
<td>Detailed design, Transaction timings</td>
<td>Execution time, Transaction trace</td>
</tr>
<tr>
<td>design</td>
<td>TLM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>ISS</td>
<td>Binary code</td>
<td>All</td>
</tr>
</tbody>
</table>

TLM: Transaction-Level Modeling, ISS: Instruction Set Simulator

Question: How to evaluate memory access contention without access timing information?
Method: Main Idea

Approximations on memory access timings:
- Random within a simulation step
- Uncorrelated between processors

Example:
- Bandwidth utilization:
  - \( U_1 = 5/10, \ U_2 = 5/10 \)
- Collision probability:
  - \( P_{12}(U_1, U_2) = U_1 \ U_2 = 0.25 \)
- For round-robin arbitration:
  - \( A = 1*(1/2) + 2*(1/2) = 1.5 \)
- Increase of step time:
  - \( T'/T = 1+ (A - 1)*P_{12} = 1.12 \)
Method: Model and Parameter

Input model (application behavior with parameters)

\[ U_i = \frac{N_{R_i}}{T_{iW_R}} + \frac{N_{W_i}}{T_{iW_W}} \]

- \( N_R \): Number of read access
- \( N_W \): Number of write access
- \( W_R \): Read throughput (times/s)
- \( W_W \): Write throughput (times/s)
Method: Overview of our method

Input model
(platform architecture)

\[
\frac{T_i'}{T_i} = 1 + \sum_{c=0}^{2^M-1} (A_{c,i} - 1)P_c(U_1, \ldots, U_M)
\]

\[
\frac{U_i'}{U_i} = \frac{T_i}{T_i'}
\]

\(T\) : step processing time (w/o contention)
\(M\) : number of processors \((i=1, \ldots, M)\)
\(T'\) : step processing time (w/ contention)
\(c\) : collision pattern index \((2^M\) patterns\)
\(A_{c,i}\) : ratio of access time at collision to original access time

\(P_c\) : collision probability

\(MPU1\) \(\rightarrow\) \(BHS\) \(\rightarrow\) \(MPU2\)
\(\{i = 1\}\)
\(\{i = 2\}\)

Memory
Method: Numerical characteristics

- Numerical improvements in calculation of $A_{c,i}$ and $P_c$
  - $A_{c,i} \rightarrow A_{c,i}(U_i)$,
    - $U$ dependence in round-robin arbitration
  - $P_c(U_1,\ldots,U_M) \rightarrow P_c(U_1^*,\ldots,U_M^*)$, where $U^* = \frac{(T'-T) + TU}{T'}$
    - Increase of bandwidth utilization due to arbitration

- Complexity: $O(M^2 2^M)$

- Effects of correlation
  - Positive correlation $\rightarrow$ upper bound
  - No correlation $\rightarrow$ our method
  - Negative correlation $\rightarrow$ lower bound
Method: Comparison with MC Simulation

Monte-Carlo (MC) simulation of Round-robin arbiter
Assumptions:
• Memory access occurs randomly
• No correlations between processors

Simulation Conditions

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request of utilization</td>
<td>$U_1 = [0, 1]$, $U_2 = 0.1$, $U_3 = 0.5$</td>
</tr>
<tr>
<td>MC iterations</td>
<td>1000</td>
</tr>
</tbody>
</table>
Experiment: Overview

Execute two processes
Memory access contention makes the execution time longer

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel® Xeon™ 3.60GHz x 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main mem.</td>
<td>PC2-5300 DDR2 SDRAM, 2GB</td>
</tr>
<tr>
<td>L1 cache</td>
<td>16KB, 32byte block (I), 16KB, 64byte block (D)</td>
</tr>
<tr>
<td>L2 cache</td>
<td>1MB x 2, 64byte block</td>
</tr>
</tbody>
</table>
Experiment: Model

- Benchmarks are two image processing commands in ImageMagick software
  - Resize: enlarge a JPEG file by 150%
  - Rotate: rotate a JPEG image by 45 degrees
Experiment: Parameters

<table>
<thead>
<tr>
<th>Program</th>
<th>NR (x10^6)</th>
<th>NW (x10^6)</th>
<th>T (s)</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>-resize</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>0.001</td>
<td>1.252</td>
<td>0.56 +- 0.02</td>
<td>0.255</td>
</tr>
<tr>
<td>Resize</td>
<td>23.542</td>
<td>7.936</td>
<td>6.49 +- 0.04</td>
<td>0.324</td>
</tr>
<tr>
<td>Write</td>
<td>4.124</td>
<td>1.409</td>
<td>2.26 +- 0.03</td>
<td>0.166</td>
</tr>
<tr>
<td>-rotate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>0.001</td>
<td>1.252</td>
<td>0.56 +- 0.02</td>
<td>0.255</td>
</tr>
<tr>
<td>Border</td>
<td>1.248</td>
<td>10.214</td>
<td>1.19 +- 0.04</td>
<td>1.03</td>
</tr>
<tr>
<td>Rotate</td>
<td>15.645</td>
<td>6.157</td>
<td>16.5 +- 0.2</td>
<td>0.091</td>
</tr>
<tr>
<td>Write</td>
<td>3.828</td>
<td>1.284</td>
<td>1.61 +- 0.02</td>
<td>0.212</td>
</tr>
</tbody>
</table>

- Throughputs: read=19.56+-0.07, write=8.76+-0.03 (x 10^6/s)
- Number of memory accesses is measured with Valgrind tool
  - Cache hits are subtracted from the above values of Ns
## Experiment: Result

<table>
<thead>
<tr>
<th>Program</th>
<th>Inc. (%)</th>
<th>T'(Exp.) (s)</th>
<th>T'(Est.) (s)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-resize</td>
<td>Read</td>
<td>-0.3</td>
<td>0.56+-0.02</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Resize</td>
<td>11.7</td>
<td>7.25+-0.2</td>
<td>7.00</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>3.2</td>
<td>2.33+-0.05</td>
<td>2.32</td>
</tr>
<tr>
<td></td>
<td>(Total)</td>
<td>8.3</td>
<td>10.14+-0.3</td>
<td>9.91</td>
</tr>
<tr>
<td>-rotate</td>
<td>Read</td>
<td>2.7</td>
<td>0.57+-0.01</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Border</td>
<td>8.2</td>
<td>1.28+-0.05</td>
<td>1.71</td>
</tr>
<tr>
<td></td>
<td>Rotate</td>
<td>7.3</td>
<td>17.73+-0.2</td>
<td>16.74</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>-0.2</td>
<td>1.61+-0.02</td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td>(Total)</td>
<td>6.8</td>
<td>21.31+-0.2</td>
<td>20.66</td>
</tr>
</tbody>
</table>
Discussion

- The accuracy can be improved by choosing an appropriate throughput parameter
  - DRAM throughput of accesses to sequential addresses is higher than that of accesses to random addresses
  - We adopt throughputs of random access in the simulation
  - However ‘Border’ step consists of sequential accesses
- Improved estimate: $T' = 1.284\text{s}$ (error: 0.3%, $U=0.316$)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>NR ($\times10^6$)</th>
<th>NW ($\times10^6$)</th>
<th>T (s)</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rotate</td>
<td>Border</td>
<td>1.248</td>
<td>10.214</td>
<td>1.19 +/- 0.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>Inc. (%)</th>
<th>T'(Exp.) (s)</th>
<th>T'(Est.) (s)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rotate</td>
<td>Border</td>
<td>8.2</td>
<td>1.28+-0.05</td>
<td>1.71</td>
</tr>
</tbody>
</table>
Summary

- We propose a method for the evaluation of the memory access contention
  - Access timings are approximated as random and uncorrelated
  - The method can be used in event-driven simulations
- Error of order of 3% is found in comparison of our estimate with experimental result
  - Error becomes larger if there are sequential accesses
- Future works
  - More comparison with actual embedded systems
  - Support for cache memory and out-of-order execution
Logos and trademarks

- Following terms are registered trademarks of International Business Machines Corporation in United States, other countries, or both:
  - IBM, Rational, and Rhapsody.
- Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.
# Table of Contents

- **Introduction:**
  - Application-level performance estimation at early stage of development
  - Memory Access Contention

- **Related works:**
  - simulation methods of memory access contention
  - UML-based simulation

- **Method:**
  - Main Idea
  - Model and parameter
  - Waiting time by arbitration

- **Method: (continued)**
  - Estimation of collision probability
  - Comparison with MC Simulation

- **Experiment:**
  - Overview
  - Model
  - Parameters
  - Result

- **Discussion**

- **Summary**
Introduction: V-Model and Modeling

UML is used to clarify software specification.

It is difficult to evaluate application performance because it requires hardware resources to be taken into account.

System Requirements Definition
System Architecture Design
Software Requirements Definition
Software Architecture Design
Software Detail Design
Implementation
Unit Test
Software Integration Test
Software System Test
System Integration Test
System Test
Re-work
Method: waiting time by arbitration

Generalization:

\[ A_{c,i}(U_i) = \sum_{n=0}^{m(c)-2} E_n r_n(U_i) + E_{m(c)-1}(1-U_i)^{m(c)-1}, \]

\[
\begin{align*}
    m(c) &= \text{# of simultaneous access in pattern } c \\
    r_n(U) &= (1-U)^n U \\
    E_n &= (m(c)-1) \frac{m(c)-n-1}{m(c)} + (n+1) \frac{2m(c)-n}{2m(c)} \\
    \text{Prob. of access N cycles after current cycle} \\
    \text{Relative access time at collision and successive access after n cycles}
\end{align*}
\]
Method: Correlation of memory access

- Error due to correlation has upper and lower bounds

**Relative increase of execution time \( T'/T \)**

- Periodic access
- Random access
- Cluster access

**Utilization request U**

**Negative correlation**

**Positive correlation**
Method: waiting time by arbitration

(a) Without successive access

(b) With successive access

Average access time:

\[ A = \frac{3}{2} (1 - U_1) + 2U_1 \]

(for round-robin arbitration case)
Method: Estimation of collision probability

- Simple estimation underestimates the increase of collisions due to arbitration
  \[ P_c(U_1, \cdots U_M) = \prod_{q=1}^{M} \left(1 - U_q\right)^{n(q,c)} U_q^{b(q,c)} \]
  \( b(q,c) = 1 \) if \( q \)-th processor accesses in pattern \( c \)
  \( n(q,c) = 1 - b(q,c) \)

- Improved estimation uses iterative calculation \( (O(M^2 2^M)) \)
  \[ U^* = \{(T' - T) + TU\} / T' \] : Bandwidth utilization taking into account the accesses waiting for arbitration
  \[ P_c(U_1, \cdots U_M) = \prod_{q=1}^{M} \left(1 - U_q^*\right)^{n(q,c)} U_q^{* b(q,c)} \]
  \[ T_i' / T_i = 1 + \sum_{c=0}^{2^M-1} (A_{c,i} - 1) P_c(U_1, \cdots, U_M) \]