Predicting Multi-Core Performance
A Case Study Using Solaris Containers

Akbar Siami Namin
AdVanced Empirical Software Testing and Analysis (AVESTA)
Department of Computer Science
Texas Tech University, USA
Akbar.namin@ttu.edu

Mohan Sridharan
Department of Computer Science
Texas Tech University, USA
Mohan.sridharan@ttu.edu

Pulkit Tomar
AdVanced Empirical Software Testing and Analysis (AVESTA)
Department of Computer Science
Texas Tech University, USA
Pulkit.tomar@ttu.edu

International Workshop on Multi-core Software Engineering, IWMSE 2010, Cape Town, South Africa
May 2010
Outline

- Motivation
- Related work
- Experimental procedure
- Data analysis
- Discussion
- Conclusion and research direction
Motivation

Problem Statement

- Utilization of the multi-core technology
- Auto-tuning - Development of proper techniques for
  - Creating an optimum number of threads
  - Allocating threads to an optimum number of CPUs
- Handled by the resource manager provided by the operating system
Motivation

Research Question and Our Approach

- Research question:
  - Investigate the effect of two parameters on performance:
    - The number of threads
    - The number of CPUs
  - Modeling using linear regression and neural networks

\[
\text{Performance} \cong f(\text{No.Threads, No.CPUs})
\]
Related work

Java Benchmarks

- Java Grande Benchmark (Bull et al., 2000)
  - Three sections with inputs for the size of the data
    1. Low level operations
    2. Kernels computation
    3. Large scale applications

- Sequential converted to parallel (Smith et al, 2001)
  - Using threads, Barrier, fork, join, synchronization

- DeCapo (Blackburn et al., 2006)
  - Three inputs: small, default, and large

- Tak Benchmark, Java Generic Library (JGL), RMI, JavaWorld
Related work

Auto-Tuning Performance

- Dynamic allocation of threads and CPUs
- Identifying the near optimum configuration of tuning parameters from a search space (Werner-Kytl and Tichy, 2000)
- Reducing the search space using the characteristic information of parameterized parallel patterns (Schaefer, 2009)
  - Number of threads, load per worker, number of worker threads, etc.
- Dynamic approach of increasing and decreasing the number of threads (Hal et al., 1997)
  - Adaptive thread management
Experimental Procedure

Goal and Approach

- **Goal** - Study relationships among performance, number of threads, and number of CPUs
- **Approach**
  - **Modeling**
    - Multiple linear regressions
    - Neural networks
  - **Run a selected benchmark**
    - Observe: performance while number of threads and CPUs are controlled
  - **Apply linear regressions and neural networks:**
    - Independent variables “number of threads” and “number of CPUs”
    - Dependent variable “performance”
Experimental Procedure

Generation of Solaris Containers

- Introduced by Solaris 10
- Resource management for applications using *projects*
  - Workload control
  - Security control by restricting access
- Generation
  1. \( k = \) number of CPUs
  2. For \( k \) in 1, 2, 4, 6, 8, 16
  3. create \((\text{pset.max} = k, \text{pset.min} = \text{pset.max})\)
- Monitor using `mpstat` command
Experimental Procedure

Machines Used

- **Sun Fire T1000**
  - UltraSPARC T1 processor 1.2 GHz, 32 GB memory
  - Supporting 32 concurrent hardware threads
  - Suitable for:
    - Tightly coupled multi-threaded applications
    - Computational less expensive threads: serving more threads

- **Sun SPARC Enterprise M3000**
  - SPARC64 VII processor 2.75 GHz, 64 GB memory
  - Supporting eight concurrent hardware threads
  - Suitable for:
    - Single threaded workloads
Experimental Procedure

Benchmarks Used

- Java Grande benchmark
  - Section one: low level computations
    - ForkJoin: Forking and joining threads
    - Barrier: Barrier synchronization
    - Syn: Synchronization of blocks
  - Section two: kernel processes
    - Fourier coefficient analysis
    - LU factorization
    - Over-relaxation
    - IDEA encryption
    - Sparse matrix multiplication
  - Section three: large scale applications
    - Molecular simulation
    - Monte Carlo simulation
    - 3D ray tracer
Experimental Procedure

Setup

- For T1000 machine:
  - Created 5 containers (projects)
    - One-CPU, Two-CPU, Four-CPU, Eight-CPU, Sixteen-CPU

- For M3000 machine:
  - Created 3 containers
    - One-CPU, Two-CPU, Four-CPU

- Commands used:
  - poolcfg: To create pools and processor sets
  - projadd: To create projects
  - mpstate: to monitor the assignment and utilization
Experimental Procedure

Setup (con’t)

- Ran each benchmark for:
  - A set of threads ranging from 1 to 50
    - For each container on each machine
  - Performance was measured
    - Given by the output of the benchmark used
Data Analysis

Visualization
Data Analysis

*Multiple Linear Regressions*

- Fitting various models of the form:

  \[ Y = C_0 + C_1X_1 + C_2X_2 + ... + C_nX_n + \varepsilon \]

  - \( C_0 \): Intercept
  - \( C_{i \neq 0} \): Coefficients regression
  - \( X_i \): Explanatory variables
  - \( Y \): Response variable

- Goodness of fit:

  R-squared: how much of variation of one variable can be explained by another one.

  Mean Square Error (MSE): mean of least squared error
Data Analysis
Multiple Linear Regressions

- Fitting various models of the form:

  \[ \text{Performance} = C_0 + C_1.(\#CPU) \]

  \[ \text{Performance} = C_0 + C_1.(\#threads) \]

  \[ \text{Performance} = C_0 + C_1.\log(\#CPU) \]

  \[ \text{Performance} = C_0 + C_1.\log(\#threads) \]

  \[ \log(\text{Performance}) = C_0 + C_1.\log(\#CPU) \]

  \[ \log(\text{Performance}) = C_0 + C_1.\log(\#threads) \]

  \[ \text{Performance} = C_0 + C_1.(\#CPU) + C_2.(\#threads) \]

  \[ \text{Performance} = C_0 + C_1.\log(\#CPU) + C_2.(\#threads) \]

  \[ \text{Performance} = C_0 + C_1.(\#CPU) + C_2.\log(\#threads) \]

  \[ \text{Performance} = C_0 + C_1.\log(\#CPU) + C_2.\log(\#threads) \]

  \[ \ldots \]

  \[ \log(\text{Performance}) = C_0 + C_1.\log(\#CPU) + C_2.\log(\#threads) \]
Data Analysis

Multiple Linear Regressions

- The best model found:

\[
\log(\text{Performance}) = C_0 + C_1 \cdot \log(\text{CPU}) + C_2 \cdot \log(\text{threads})
\]
Data Analysis

*Neural Network*

- A machine language technique for classification and regression problems
  - Nodes: Variables
    - Inputs: (log(#CPU), log(#threads))
    - Output: log(performance)
  - Connections: The relationships between variables
  - Internal layers:
    - W and B: Matrices of weights and bias values (tuning)
    - Some other variables (15 in our case)
Data Analysis

**Neural Network**

- A 60-20-20 split was used
  - 60% for training the model and coefficients
  - 20% for tuning the model
  - 20% for testing the model

<table>
<thead>
<tr>
<th>Programs</th>
<th>T1000</th>
<th>M3000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$R^2$</td>
<td>MSE</td>
</tr>
<tr>
<td>S1:BarrierSimple</td>
<td>0.991</td>
<td>0.051</td>
</tr>
<tr>
<td>S1:BarrierTournament</td>
<td>0.924</td>
<td>0.651</td>
</tr>
<tr>
<td>S1:ForkJoinSimple</td>
<td>0.996</td>
<td>0.034</td>
</tr>
<tr>
<td>S1:SyncMethod</td>
<td>0.992</td>
<td>0.002</td>
</tr>
<tr>
<td>S1:SyncObject</td>
<td>0.994</td>
<td>0.002</td>
</tr>
<tr>
<td>S2:SeriesKernelSizeA</td>
<td>0.931</td>
<td>0.101</td>
</tr>
<tr>
<td>S2:LUFactKernelSizeA</td>
<td>0.982</td>
<td>0.036</td>
</tr>
<tr>
<td>S2:CryptKernelSizeA</td>
<td>0.994</td>
<td>0.004</td>
</tr>
<tr>
<td>S2:SORKernelSizeA</td>
<td>0.984</td>
<td>0.002</td>
</tr>
<tr>
<td>S2:SparseMatmulKernelSizeA</td>
<td>0.971</td>
<td>0.017</td>
</tr>
<tr>
<td>S3:MolDynRunSizeA</td>
<td>0.968</td>
<td>0.057</td>
</tr>
<tr>
<td>S3:MolDynTotalSizeA</td>
<td>0.967</td>
<td>0.052</td>
</tr>
<tr>
<td>S3:MonteCarloRunSizeA</td>
<td>0.990</td>
<td>0.003</td>
</tr>
<tr>
<td>S3:MonteCarloTotalSizeA</td>
<td>0.992</td>
<td>0.022</td>
</tr>
<tr>
<td>S3:RayTracerInitSizeA</td>
<td>0.612</td>
<td>0.385</td>
</tr>
<tr>
<td>S3:RayTracerRunSizeA</td>
<td>0.986</td>
<td>0.007</td>
</tr>
<tr>
<td>S3:RayTracerTotalSizeA</td>
<td>0.985</td>
<td>0.006</td>
</tr>
</tbody>
</table>
Data Analysis

*Neural Network*

- Compared to linear regression model
  - Similar model obtained with different coefficients
  - Better precision
    - Higher R-squared, Lower MSE

(a) Program Section2:SORKernelSizeA on T1000.
(b) Program Section1:SyncObject on M3000.
Discussion

Limitations and Generalization

- Middle-size programs
- Simultaneous execution of programs in different containers
  - Only one physical CPU for both T1000 and M3000
- Java versions
  - 1.5 on T1000
  - 1.6 on M3000
- The model developed still was the best
- #CPU and #threads not the only parameters influencing the performance
Conclusion & Research Directions

- A model developed for estimating the performance of multi-cores systems
  - Similar to the practical models developed intuitively
- The optimal performance
  - One-to-one thread to CPU assignment
- The work part of a project concerning auto-tuning
- Comparing sequential programs to the paralleled versions
- Adaptive testing and auto-tuning for multi-core systems
Thank You