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

$$Performance \cong f(No.Threads, No.CPUs)$$

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, synvhronization`**
- 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 (pset.max = k, pset.min=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

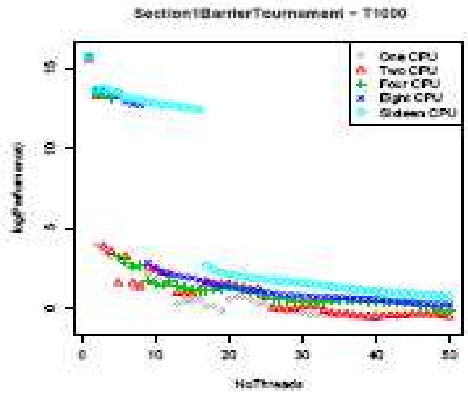
Experimental Procedure

Setup (con't)

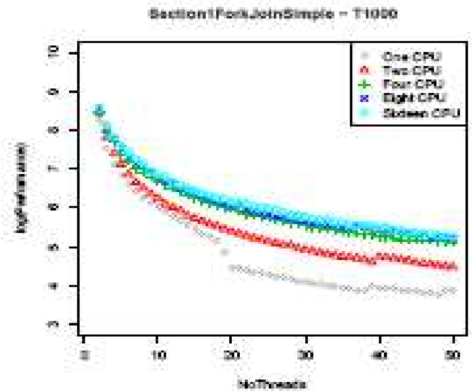
- Ran each benchmark for:
  - A set of threads ranging from 1 to 50
    - For each container on each machine
- Performance was measured
  - Given by the output of the benchmark used

Data Analysis

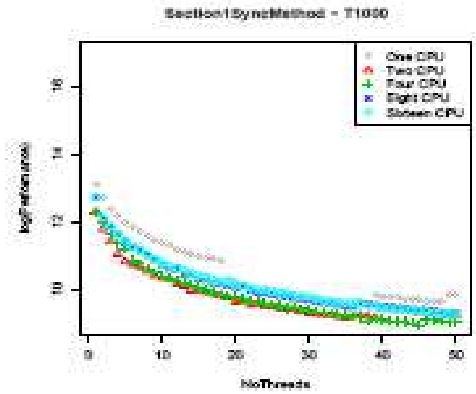
Visualization



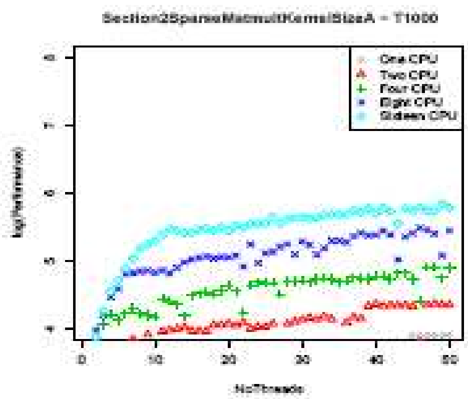
(a) Section1-BarrierTournament



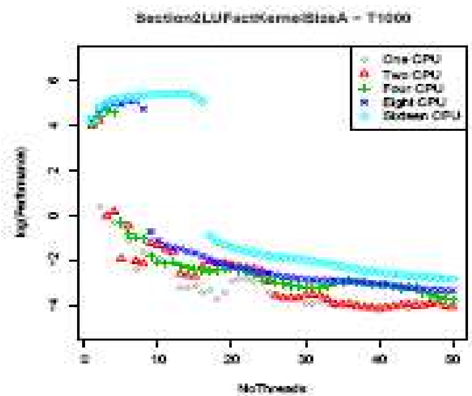
(b) Section1-ForkJoinSimple



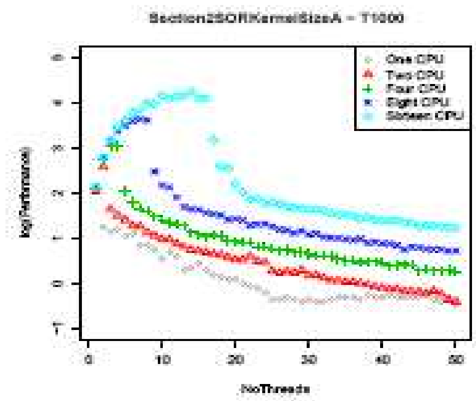
(c) Section1-SyncMethod



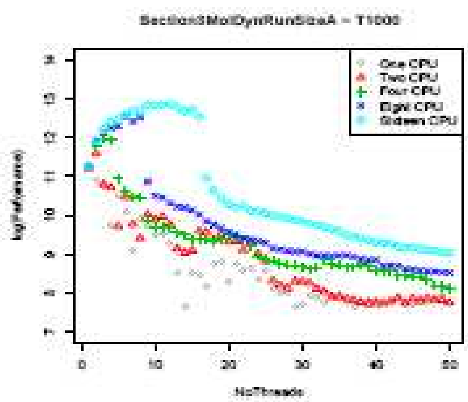
(d) Section2-SparseMatmultKernelSizeA



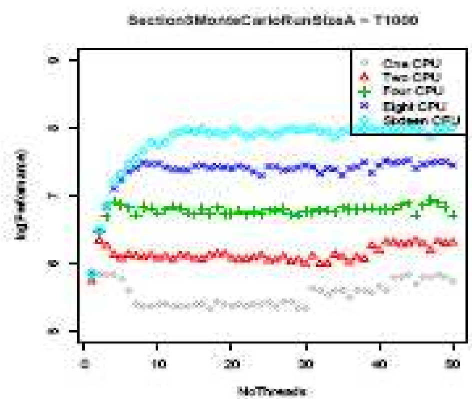
(e) Section2-LUFactKernelSizeA



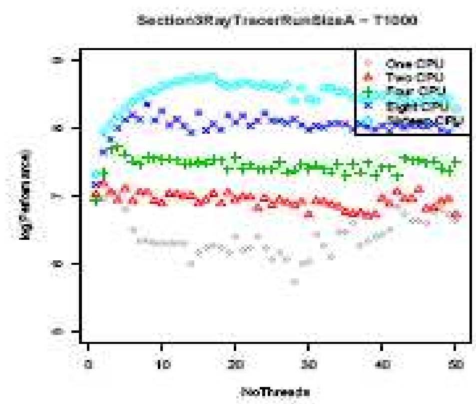
(f) Section2-SORKernelSizeA



(g) Section3-MolDynRunSizeA



(h) Section3-MonteCarloRunSizeA



(i) Section3RayTracerRunSizeA

Data Analysis

Multiple Linear Regressions

- Fitting various models of the form:

$$Y = C_0 + C_1.X_1 + C_2.X_2 + \ldots + C_n.X_n + \varepsilon$$

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 cab 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:

$$Performance = C_0 + C_1.(\#CPU)$$

$$Performance = C_0 + C_1.(\#threads)$$

$$Performance = C_0 + C_1.\log(\#CPU)$$

$$Performance = C_0 + C_1.\log(\#threads)$$

$$\log(Performance) = C_0 + C_1.\log(\#CPU)$$

$$\log(Performance) = C_0 + C_1.\log(\#threads)$$

$$Performance = C_0 + C_1.(\#CPU) + C_2.(\#threads)$$

$$Performance = C_0 + C_1.\log(\#CPU) + C_2.(\#threads)$$

$$Performance = C_0 + C_1.(\#CPU) + C_2.\log(\#threads)$$

$$Performance = C_0 + C_1.\log(\#CPU) + C_2.\log(\#threads)$$

...

$$\log(Performance) = C_0 + C_1.\log(\#CPU) + C_2.\log(\#threads)$$

Data Analysis

Multiple Linear Regressions

- The best model found:

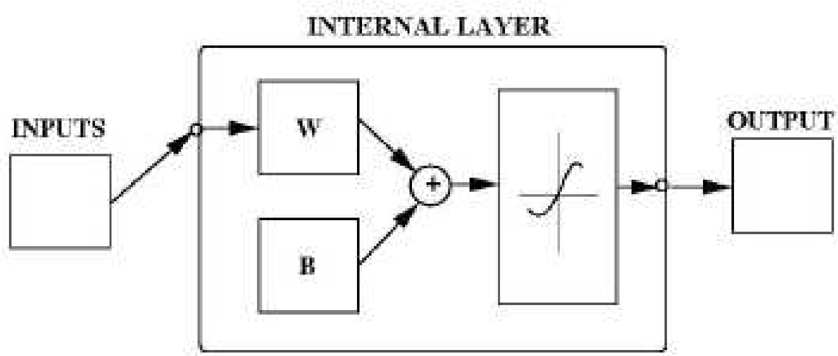
$$\log(Performance) = C_0 + C_1.\log(\#CPU) + C_2.\log(\#threads)$$

Benchmark Programs	T1000					M3000				
	C_0	C_1	C_2	R^2	MSE	C_0	C_1	C_2	R^2	MSE
Section1:BarrierSimple	11.460	0.149	-1.356	0.905	0.151	12.973	0.360	-1.366	0.900	0.164
Section1:BarrierTournament	11.457	1.554	-3.772	0.718	5.234	9.817	0.817	-3.071	0.651	4.024
Section1:ForkJoinSimple	9.958	0.519	-1.620	0.742	0.776	9.951	0.899	-1.212	0.977	0.026
Section1:SyncMethod	12.846	-0.036	-0.915	0.894	0.076	15.384	-0.489	-1.033	0.681	0.650
Section1:SyncObject	12.819	-0.040	-0.907	0.891	0.078	15.435	-0.450	-1.052	0.677	0.439
Section2:SeriesKernelSizeA	5.424	0.892	0.184	0.944	0.047	7.507	0.962	0.037	0.950	0.015
Section2:LUFactKernelSizeA	3.602	1.098	-2.331	0.753	1.763	3.454	0.595	-2.168	0.790	1.000
Section2:CryptKernelSizeA	7.483	0.787	-0.053	0.741	0.208	8.769	0.887	-0.023	0.713	0.101
Section2:SORKernelSizeA	2.179	0.799	-0.748	0.841	0.198	3.305	0.711	-1.062	0.866	0.160
Section2:SparseMatmultKernelSizeA	2.452	0.710	0.352	0.936	0.039	5.238	0.943	0.132	0.874	0.207
Section3:MolDynRunSizeA	11.758	0.735	-1.139	0.838	0.294	13.129	0.412	-1.451	0.907	0.172
Section3:MolDynTotalSizeA	-2.317	0.735	-1.133	0.842	0.283	-0.930	0.414	-1.448	0.909	0.168
Section3:MonteCarloRunSizeA	5.112	0.828	0.153	0.938	0.044	7.184	0.944	0.064	0.938	0.019
Section3:MonteCarloTotalSizeA	-4.08	0.742	0.131	0.936	0.036	-2.141	0.925	0.066	0.931	0.020
Section3:RayTracerInitSizeA	8.851	0.240	-0.809	0.499	0.561	9.865	0.187	-0.547	0.194	0.958
Section3:RayTracerRunSizeA	6.382	0.759	0.008	0.933	0.039	9.324	0.856	-0.445	0.847	0.070
Section3:RayTracerTotalSizeA	-3.568	0.741	-0.027	0.931	0.039	-0.601	0.790	-0.516	0.861	0.065

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

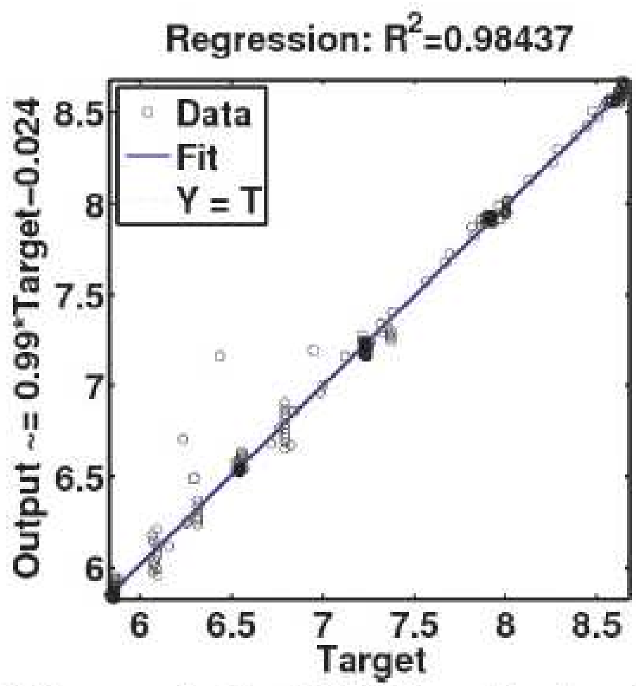
Neural Networks.

Benchmark Programs	T1000		M3000	
	R^2	MSE	R^2	MSE
S1:BarrierSimple	0.991	0.051	0.908	0.003
S1:BarrierTournament	0.924	0.651	0.961	0.174
S1:ForkJoinSimple	0.996	0.034	0.995	0.004
S1:SyncMethod	0.992	0.002	0.963	0.043
S1:SyncObject	0.994	0.002	0.937	0.042
S2:SeriesKernelSizeA	0.931	0.101	0.851	0.087
S2:LUFactKernelSizeA	0.982	0.036	0.961	0.193
S2:CryptKernelSizeA	0.994	0.004	0.902	0.020
S2:SORKernelSizeA	0.984	0.002	0.963	0.011
S2:SparseMatmultKernel-	0.971	0.017	0.923	0.035
S3:MolDynRunSizeA	0.968	0.057	0.938	0.048
S3:MolDynTotalSizeA	0.967	0.052	0.935	0.034
S3:MonteCarloRunSizeA	0.990	0.003	0.978	0.013
S3:MonteCarloTotalSizeA	0.992	0.022	0.943	0.008
S3:RayTracerInitSizeA	0.612	0.385	0.595	0.496
S3:RayTracerRunSizeA	0.986	0.007	0.937	0.045
S3:RayTracerTotalSizeA	0.985	0.006	0.938	0.056

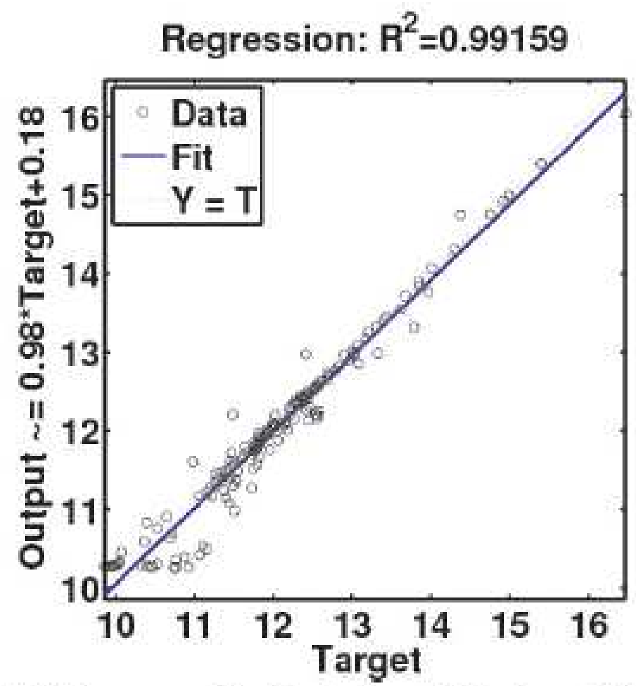
Data Analysis

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