Working Set Model for Multithreaded Programs
(short paper)

TRIOS'14

Kishore Kumar Pusukuri
Oracle Inc.

Motivation

- Working Set Size (WSS):
  #pages referenced over a time interval * page size
- Effectively consolidating Java fusion applications on Oracle cloud servers
- Approximating WSS: simulations, program traces (exiting techniques, single threaded programs)
- Solaris on SPARC T4-4 (PARSEC, SPEC OMP 2012):
  Measuring overhead is high: 80 msec to 2.5 hours
- Dynamically optimizing resources?

Modeling Working Set Size

- Several factors affect WSS
- Varies application to application (4 MB to 21 GB)

- Identifying factors that correlate to WSS
- Using statistical models based on machine learning

Outline

- Identifying Factors Correlate with Working Set Size
- Developing Models for Working Set Size
- Working Set Aware Scheduling (Future Work)

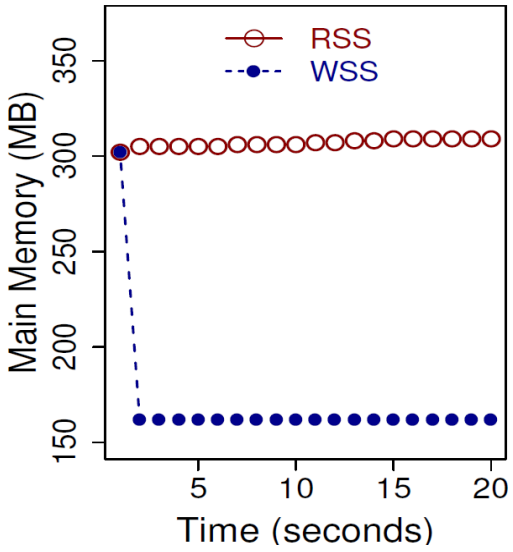
Experimental Setup

- 20 multithreaded programs from PARSEC & SPEC OMP 2012
- pthreads, stress CPU and Main Memory

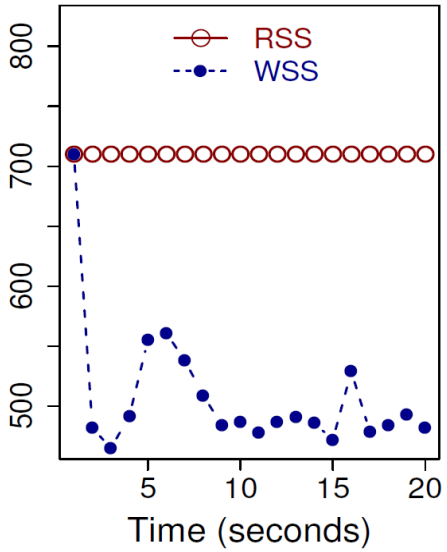
SPARC T4-4: 4 × SPARC T4 Processors @3.0 GHz;
Total 256 vCPUs (i.e., 4 × 64 vCPUS);
L3: 4 MB; RAM: 512 GB; OS: Oracle Solaris 11™

Identifying Factors

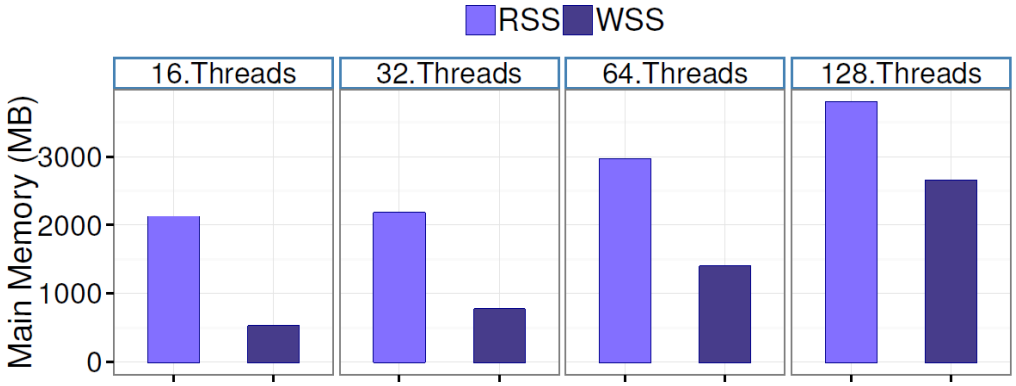
Resident Set Size (RSS)



Facesim
(PARSEC)



Fluidanimate
(PARSEC)



#Threads

Imagick
(SPEC OMP2012)

Identifying Factors (cont...)

TLB Miss Rate & Number of Threads

- TLB reach: (#TLB entries * page size)
- TLB reach is critical for the performance
- Multithreaded Programs (data parallelism): higher the #threads, smaller the TLB miss rate

Last-level Cache (LLC) Miss Rate

- Thread Migrations → Load Balancing
- Thread Migrations → Data Locality → LLC miss rate

The Factors

- Resident Set Size (RSS)
- TLB Miss Rate
- Number of Threads
- LLC Miss Rate

Developing Models

Data Collection (training data)

- Run the 20 programs with 16, 32, 64, 128 threads
- Predictors: *RSS, #threads, TLB misses per instruction, TLB misses per second, LLC misses per instruction, LLC misses per second*
- WSS is the target parameter
- 80 data points, 7-tuple data point,

Developing Models (cont...)

Finding Important Factors

- Prediction Accuracy vs Cost of Approximation
- Forward and Backward Input Selection Techniques (AIC - Akaike Information Criterion)
- Over-fitting problem
- Multicollinearity Problem
- More robust model

RSS & TLB Misses per Instruction

The Models

- Linear Regression (LR)
- K Nearest Neighbour (KNN)
- Regression Tree (RT)

Model Selection

- 10-fold cross-validation (CV) test
- Normalized Root Mean Squared Error (NRMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} |F_i - A_i|^2}$$

$$NRMSE = \frac{RMSE}{A_{max} - A_{min}} * 100$$

$\mathbf{F_i}$ → Forecast Value

$\mathbf{A_i}$ → Actual Value

Prediction Accuracy = (100 - NRMSE)

Model	NRMSE	Prediction Accuracy (%)
LR	13.8	86.2
DT	7.9	92.1
KNN	4.3	95.7

The KNN model as the WSS model

Further Evaluation

- Prediction accuracies of the WSS model for memcached (Data Cache of Cloud Suite) and SPECjbb 2005 are 93% and 88%.

Overhead of the WSS model

- 24 microseconds
- The WSS model vs the existing techniques: (microseconds vs hours)

Overall Approach

1. Develop a linear regression model with all reasonable predictors of WSS using Training Data.
2. Select important predictors among the initial predictors using Akaike Information Criterion.
3. Develop different models using the important predictors selected in step 2.
4. Use cross-validation tests (on both Training Data and Test Data) to select the best model

Outline

- Identifying Factors Correlate with Working Set Size
- Developing Models for Working Set Size
- Working Set Aware Scheduling (Future Work)

Working Set Aware Scheduling

- Grouping vs Spreading: running the threads on a few multicore sockets compared to spreading threads uniformly across all the sockets.
- Grouping → Programs with small WSS
  e.g., 31% performance improvement for bodytrack (PARSEC)
- Spreading → Programs with large WSS
  e.g., 24% performance improvement for streamcluster

Consider WSS and other factors such as lock contention

Conclusions

- Identifies important factors that correlate with WSS of multithreaded applications
- Demonstrates the usage of machine learning for developing simple models for approximating WSS
- Overhead is negligible and can be used for dynamically optimizing resources

Acknowledgments

Dave Dice, Darryl Gove, Blake Jones, Tim Farkas, Darrin Johnson

Prof. Lorenzo Alvisi