

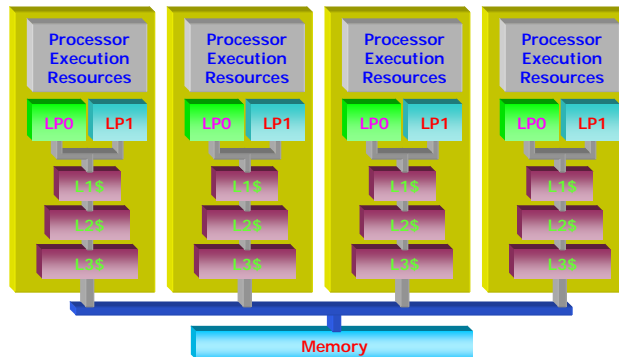
Adaptive Execution Techniques for SMT Multiprocessor Systems

Changhee Jung ETRI
Jaejin Lee Seoul National University

Outline

- SMT multiprocessor architecture
- Motivation
- Adaptive execution strategies
- Performance evaluation
- Conclusions

SMT Multiprocessor Architecture

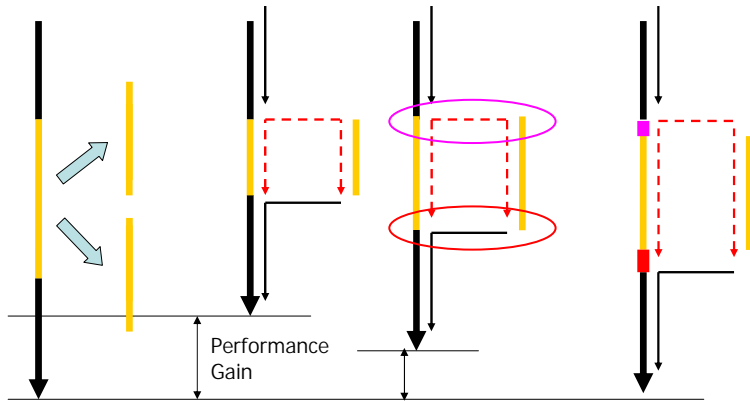


- Instructions from different threads execute in parallel
- Identical instruction streams from multiple threads may degrade performance
 - Contend for the same functional unit
- Commercially available: Intel Xeon SMP with hyper-threading

Our Goal & Approach

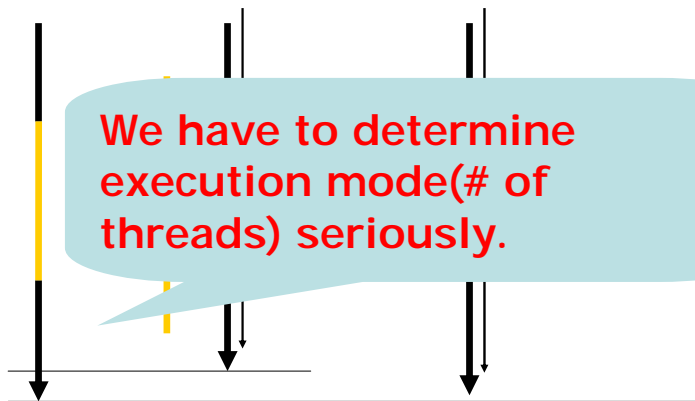
- Improve the performance of applications that contain conventional loop-level parallelism
- Avoid executing some parallel loops in parallel
- Dynamically change the number of threads to run the parallel loops

Parallel Loop Overheads



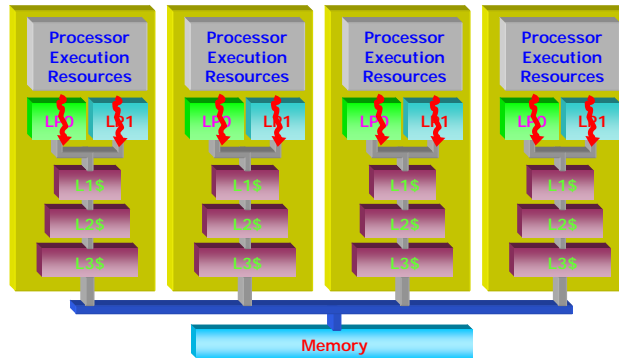
Serialize the parallel loop that contains small work.

Parallel Loop Overheads (contd.)

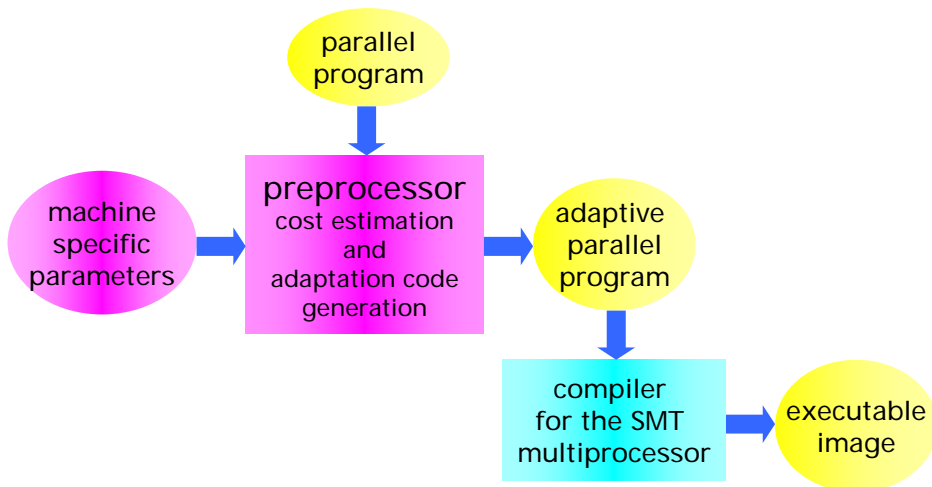


Execution Environment

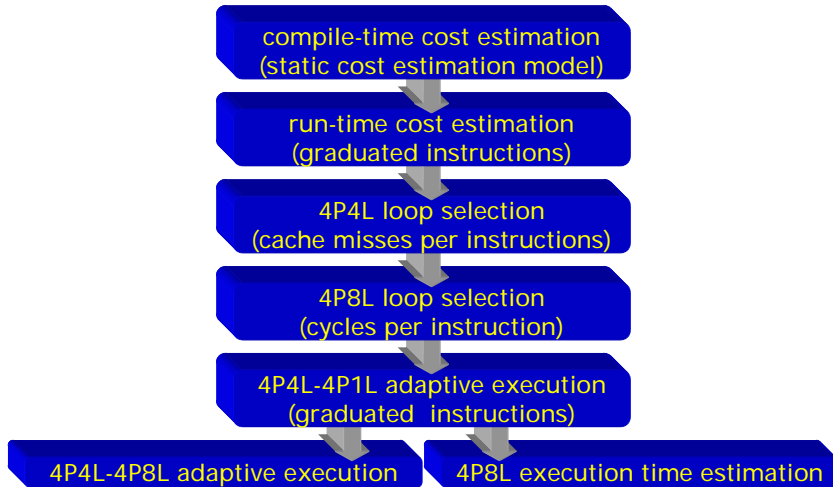
4P8L Execution Mode



Adaptive Execution Framework



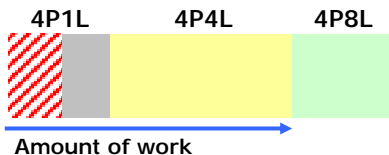
Adaptation Schemes



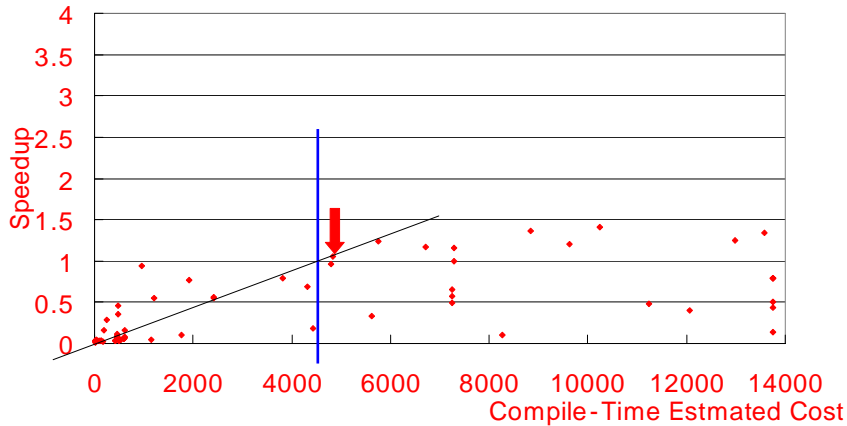
Static Cost Estimation

compile-time cost estimation
(static cost estimation model)

- Highly inefficient parallel loops
- A simple model
 - $W_{\text{body}} = \Sigma(\text{\# of each op} * \text{op_cost})$
 - $W = n * W_{\text{body}}$
- If $W < W_{\text{threshold}}$, run it sequentially
- Use heuristics to determine $W_{\text{threshold}}$



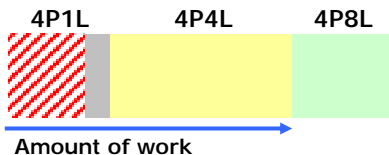
Determining $W_{\text{threshold}}$



Run-Time Cost Estimation

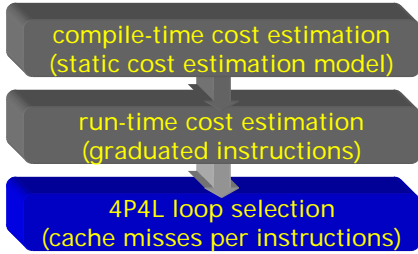
compile-time cost estimation
(static cost estimation model)

run-time cost estimation
(graduated instructions)



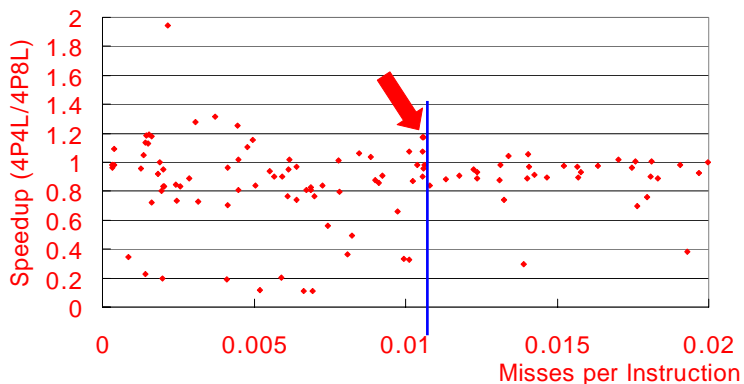
- The # of instructions executed is proportional to the workload
- W = the average number of instructions executed in each iteration in the first invocation
- If $W < W_{\text{threshold}}$, run it sequentially in the following invocations
- Inefficient loops that cannot be handled by

4P4L Loop Selection

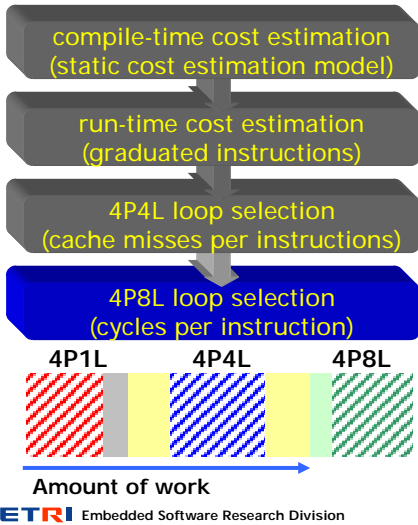


- Loops that are heavily effected by the cache conflicts between two logical processors
- L2/L3 cache misses per instruction (MPI)
- High MPI in 4P4L
high MPI in 4P8L
- If $MPI > MPI_{threshold}$, run it in 4P4L

Determining $MPI_{threshold}$



4P8L Loop Selection



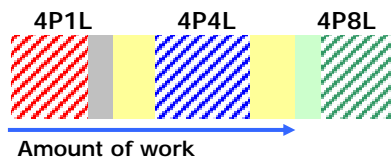
- Some loops may perform better in 4P8L mode
- Cycles per instruction (CPI)
 - Partially dependent upon MPI
 - The bigger the CPI, the higher the interference in a single physical processor
 - Small CPI less intra-thread contention
- The decision run is in 4P4L
- If the CPI in 4P4L $< \text{CPI}_{\text{threshold}}$ value, run it in 4P8L

SIGPL 2005

15

Most Recent with Timing(MRT)

- Uses the recent past behavior of a loop to predict its future behavior



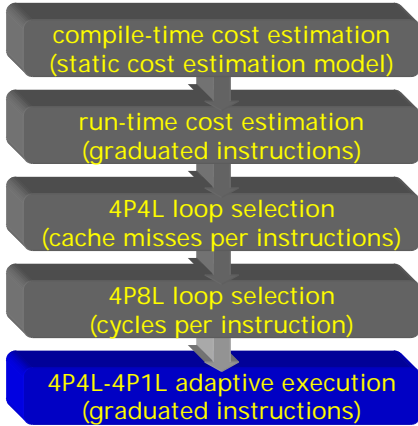
- If # of graduated instructions $< N_{\text{threshold}}$
 - 4P1L-4P4L MRT
 - Otherwise, 4P4L-4P8L MRT

ETRI Embedded Software Research Division

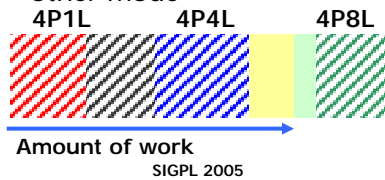
SIGPL 2005

16

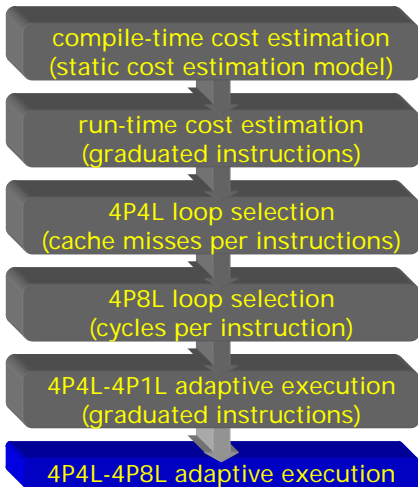
4P4L-4P1L MRT



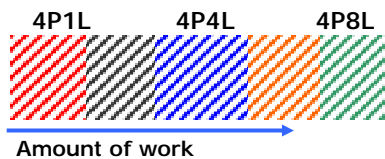
- On its second invocation, time it in 4P1L
- Comparing the two measurements, determine its mode in the next invocation
- For the remaining invocations, time it again and compare it to its most recent execution time in the other mode



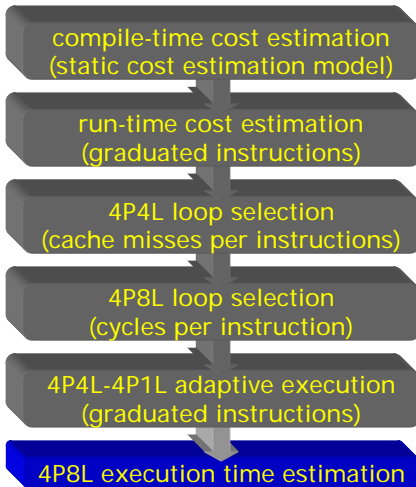
4P4L-4P8L MRT



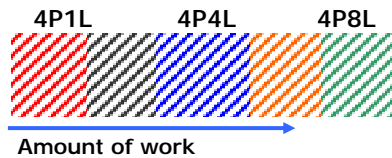
- On its second invocation, it is executed in 4P8L and timed again
- The rest is similar to 4P4L-4P1L MRT



4P8L Execution Time Estimation



- The larger the number of graduated instructions, the higher the interference between the two logical processors
- L2 and L3 cache misses affect the performance significantly
- Estimate the time in 4P8L with the measurements in 4P4L and determine its mode



Regression Analysis

$$T_{4P8L} = a \cdot N_{4P8L}^{grad} + b \cdot N_{4P8L}^{L2} + c \cdot N_{4P8L}^{L3} + d$$

$$N_{4P8L}^{grad} = a^{grad} \cdot N_{4P4L}^{grad} + b^{grad}$$

$$N_{4P8L}^{L2} = a^{L2} \cdot N_{4P4L}^{L2} + b^{L2}$$

$$N_{4P8L}^{L3} = a^{L3} \cdot N_{4P4L}^{L3} + b^{L3}$$

$$\begin{aligned}
 T_{4P8L} &= a \cdot a^{grad} \cdot N_{4P4L}^{grad} + b \cdot a^{L2} \cdot N_{4P4L}^{L2} \\
 &\quad + c \cdot a^{L3} \cdot N_{4P4L}^{L3} + a \cdot b^{grad} + b \cdot b^{L2} + c \cdot b^{L3} + d \\
 &= a' \cdot N_{4P4L}^{grad} + b' \cdot N_{4P4L}^{L2} + c' \cdot N_{4P4L}^{L3} + d'
 \end{aligned}$$

The Result of Regression Analysis

Formulas	R^2
$T_{4P8L} = 0.773 \cdot N_{4P8L}^{grad} - 23.258 \cdot N_{4P8L}^{L2} + 393.006 \cdot N_{4P8L}^{L3} + 2293277$	0.9994
$N_{4P8L}^{grad} = 1.000 \cdot N_{4P4L}^{grad} - 760536$	0.9999
$N_{4P8L}^{L2} = 3.206 \cdot N_{4P4L}^{L2} - 150422$	0.9253
$N_{4P8L}^{L3} = 1.623 \cdot N_{4P4L}^{L3} - 15684$	0.8905
$T_{4P8L} = 0.773 \cdot N_{4P4L}^{grad} - 74.573 \cdot N_{4P4L}^{L2} + 637.781 \cdot N_{4P4L}^{L3} - 960399$	-

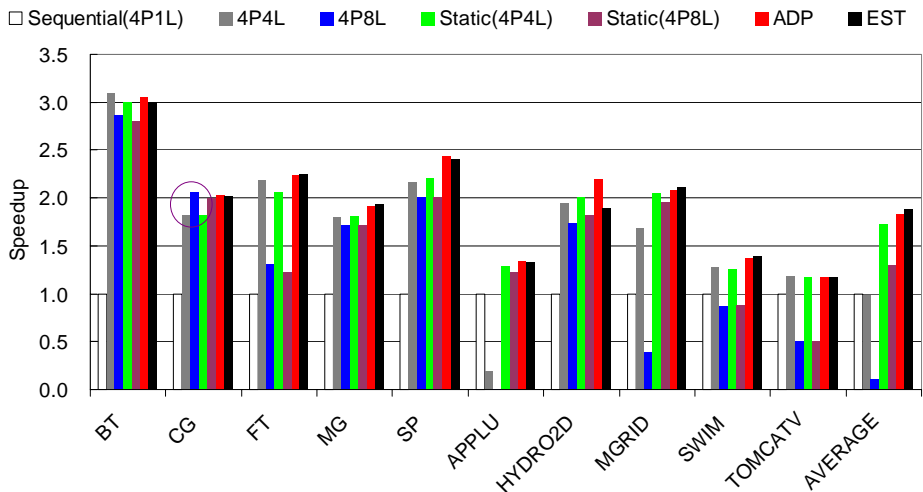
Evaluation Environment

- Implemented in a compiler preprocessor written in Perl
- Applications
 - 10 highly parallel NAS and Spec2K benchmarks (FORTRAN 77)
 - Parallelization information from Polaris
- Compilers
 - Intel Fortran compiler for Windows v8.0
- OS
 - Windows 2003 Server
- Architecture
 - 4 way Intel Xeon MP hyper-threading (1.5 GHz)
- Performance counter library
 - Our own implementation using windows system calls
- Changing the # of threads
 - OpenMP directives
 - Windows system call to map threads to logical processors

Adaptation Schemes



Speedup



Conclusions

- Presented performance estimation models and techniques for generating adaptive execution code for SMT multiprocessor architectures.
- Our code is about twice and eighteen times faster on average than the original code executed on 4 and 8 logical processors, respectively
- Adaptive execution techniques are promising and effective at speeding up shared-memory parallel programs for SMT multiprocessors