Maximizing Speedup through Self-Tuning of Processor Allocation

Thu D. Nguyen, Raj Vaswani, and John Zahorjan
University of Washington

Presenter: Minsu Kim
17 Dec 2015
Introduction

Ideal vs Reality

- Parallelization overhead
- System overhead
- Idleness
- Communication

Dynamically

- Maximize Speedup (Self-tuning!)
- Static allocation may not be optimal for entire job
Overview

Introduction

Experimental Environments

Self-Tuning Algorithms
  ➢ Basic
  ➢ Change-driven
  ➢ Time-driven

Multi-Phase Self-Tuning Algorithms
  ➢ Independent (IMPST)
  ➢ Inter-dependent (DMPST)

Conclusion
Experimental Environments

KSR-2 COMA shared memory multiprocessor
➤ OS: OSF/1 (UNIX variant)

H/W monitoring unit available on each node of the KSR-2
➤ Performing runtime measurements
➤ Event monitor: cache misses, processor stall time, etc.

Ten parallel applications
➤ Five Hand-coded parallel applications
➤ Five Complier-parallelized sequential programs
➤ Only consider iterative parallel applications
Runtime Measurement

Chosen runtime metric: Efficiency

Three critical HW counters (system overhead, processor stall time)
- Elapsed wall-clock time
- Elapsed user-mode execution time
- Accumulated processor stall time
- Reading these three register at the beginning and end of each iter.

Measuring idleness
- Instrument all thread synchronization code to track elapsed idle time
- Assume all application synchronization are invoked by calls to certain libraries.
Runtime Measurement

**Efficiency = 1 - Loss**

\[
E(p) = 1 - \frac{WT(p) - UT(p)}{WT(p)} - \frac{IT(P)}{WT(p)} - \frac{PST(p)}{WT(p)}
\]

- Parallelization overhead  Typically Small
- System overhead
- Idleness
- Processor Stall

**Speedup = (# of processors) x efficiency**

\[
S(p) = p \times E(p)
\]
Self-Tuning Algorithms

**MGB**
- Searches for the maximum
- Iteration 1: Search domain
  
  \[ [1, P] \Rightarrow (\text{reduced}) \ [S(P), P] \]
- Keep this iterative while narrowing the interval

**Extension to non-unimodal S function**
- Heuristic: a Greedy Algorithm
- Simply continue the search in the largest subinterval where the largest \( S \) has found so far.
- May not correct yet quite practical.
Advanced Self-Tuning Algorithms

**Change-driven self-tuning Algorithm**
- Continuously monitors job efficiency and re-initiates the search procedure whenever it notices a significant change in efficiency.

**Time-driven self-tuning Algorithm**
- Includes change-driven self-tuning
- Will also rerun the search procedure periodically regardless of change in job efficiency
- Considering the possibility that job efficiency changes in the middle of a change-driven self-tuning search
Performance

- Self-tuning imposes very little overhead
- Basic self-tuning can significantly improve performance over no-tuning
Performance

- Time-driven self-tuning is not useful for the programs here
- The performance benefit of self-tuning can be limited by the cost of probes
Multi-phase Self-tuning

**One Iteration = Multiple Parallel Phases**
- Phase: a specific piece of code
- Speedup of each phase may be maximized in different # of processors!

**Extension of the problem**
- In an iteration there are N phases
- Find a processor allocation vector \((p_1, p_2, \ldots, p_N)\) which maximizes \(S\)
- Actually, basic self-tuning algorithm = Find \((p, p, \ldots, p)\)
Multi-phase Self-tuning

**Independent multi-phase self-tuning (IMPST)**
- Merely apply basic self-tuning to each phase independently
- Just a naïve, simple extension
- Problem: Performance of each phase ALSO depends on the allocations for other phases.

**Inter-dependent multi-phase self-tuning (DMPST)**
- Randomized search technique
  - Choosing an initial candidate allocation vector
  - Selecting a new candidate vector (random multiplied)
  - Evaluating and accepting new candidate vectors until steady state
  - Terminating the search
Performance

- Multi-phase techniques are able to achieve performance not realizable by any fixed allocation
- Inter-dependent self-tuning yields better performance than any other.
Conclusion

Maximizing application speedup through runtime, self-selection of an appropriate number of processors on which to run

- Based on ability to measure program inefficiencies (HW support)
- Peak speedups are data or time dependent

Simple search procedures can automatically select appropriate numbers of processors

- Relieves the user from the burden of determining the precise number of processors to use for each input data set
- Potential to outperform any static allocation