Maximizing Speedup through Self-Tuning of Processor Allocation

Hyungmo Kim
SNUCSE, 2017-26932
12 Dec 2017
Contents

• Motivation

• Experimental Environments

• Self-Tuning Algorithm

• Multi-Phase Self-Tuning Algorithm

• Conclusion
Motivation

The diagram illustrates the relationship between speedup ($S(p)$) and the number of processors ($p$) for both an ideal case and a practical case. The ideal case shows a linear increase in speedup with the number of processors, while the practical case exhibits diminishing returns after a certain point, indicating the limits of scalability.
Motivation

- Speedup $S(p)$ is not **linear** with respect to processor number $p$
  - “many parallel applications achieve maximum speedup at some intermediate allocation”

- Dynamic measurements are needed
  - it varies over tasks (also time)
  - “No static allocation may be optimal for the entire execution lifetime of a job”
Experimental Environments

• Machine
  • **KSR-2 COMA** shared memory multiprocessor

• Parallelization
  • **KSR KAP** preprocessor
  • **KSR PRESTO** runtime system and **CThreads**

• Monitoring
  • H/W monitoring unit – *the event monitor*

• Benchmarks
  • `iteration { parallel region { do jobs } }`
Runtime Measurement

• **Core metric:** Efficiency $E(p)$ and Speedup $S(p)$

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

- System overhead
- Idleness
- Communication (= Processor stall)

\[
S(p) = p \times E(p)
\]
Contents

• Motivation

• Experimental Environments

• **Self-Tuning Algorithm**

• Multi-Phase Self-Tuning Algorithm

• Conclusion
Self-Tuning Algorithm

- A basic self-tuning algorithm using MGS

(\text{Target}) \ S(p) : [1, P] \rightarrow R

- First, narrow the range as below
- [1, P]
- \([S(P), P]\) practically, \((1 < S(P) < P)\)
- Then apply MGS manner optimization to the interval

![Diagram of Self-Tuning Algorithm](image)
Self-Tuning Algorithm

• But, speedup is also a function of time!

• A change-driven self-tuning algorithm
  • it reinitiate speedup when significant change in efficiency occurred

• A time-driven self-tuning algorithm
  • it reinitiate speedup periodically and when significant change occurred
Self-Tuning – Performance

No tuning
Basic self-tuning
Change-driven self-tuning
Time-driven self-tuning
Contents

• Motivation

• Experimental Environments

• Self-Tuning Algorithm

• Multi-Phase Self-Tuning Algorithm

• Conclusion
Multi-Phase Self-Tuning Algorithm

```plaintext
loop {
    parallel { // phase 1
        do job 1
    }
    parallel { // phase 2
        do job 2
    }
}
```
Multi-Phase Self-Tuning Algorithm

• Speedup $S(p)$ of each phase may be maximized in different processor number $p$

• Extension of the optimization problem
  • For each iteration, there are $N$ many phases
  • Find a processor allocation vector $(p_1, p_2, ..., p_N)$ which maximizes total speedup $S = \sum S_{phase}(p_{phase})$
Multi-Phase Self-Tuning Algorithm

• Independent multi-phase self-tuning algorithm
  • apply the basic self-tuning alg. to each phase independently
  • but, phases are dependent each other

• Inter-dependent multi-phase self-tuning algorithm
  • randomized approach

initial candidate vector \[\rightarrow\] \[\rightarrow\] \[\rightarrow\] \[\rightarrow\] \[\rightarrow\] accepted vector
Multi-Phase Self-Tuning – Performance

No tuning
Basic self-tuning
Independent multi-phase self-tuning
Inter-dependent multi-phase self-tuning
Contents

• Motivation

• Experimental Environments

• Self-Tuning Algorithm

• Multi-Phase Self-Tuning Algorithm

• Conclusion
Conclusion

• “proposed a technique to automatically regulate the number of processors used in the execution if a parallel program so as to maximize its speedup”

• “simple search procedures, guided by the runtime measurements, can automatically select appropriate numbers of processors”

• “self-tuning is especially promising for compiler-parallelized applications”