Faster Work—Stealing With Return Barriers

Vivek Kumar, Stephen M Blackburn

The Australian National University
The New Era of Computing

• Commodity processors with parallel execution abilities

• A fundamental turn toward concurrency in software
The Challenge

- Modern hardware requires s/w parallelism
- Software parallelism difficult to identify, expose
  - Hard coded optimizations may get you there…
- Hard to realize potential of modern processors

Goal: performance *and* productivity
Background

Options?

• Language based features to expose parallelism
  – Dynamic task parallelism
  – Work-stealing scheduler

• A runtime to hide the hardware complexities
Contributions

✔ In-depth analysis
  Of overheads associated with stealing tasks

✔ A new design
  Simple extension to JVM re-using old idea

✔ Detailed performance study
  Using standard work-stealing benchmarks

✔ Results
  That show we can significantly reduce the tasks stealing overhead
Understanding Work–Stealing
Work—Stealing

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Work–Stealing

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Work–Stealing

Initiation

Termination

State Management

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Our Prior Work
Work-Stealing Without The Baggage*

Vivek Kumar†, Daniel Frampton§, Stephen M. Blackburn†, David Grove‡, Olivier Tardieu‡

†Australian National University ‡Microsoft ‡IBM T.J. Watson Research

Abstract
Work-stealing is a promising approach for effectively exploiting software parallelism on parallel hardware. A programmer who uses work-stealing explicitly identifies potential parallelism and the runtime then schedules work, keeping otherwise idle hardware busy while relieving overloaded hardware of its burden. Prior work has demonstrated that work-stealing is very effective in practice. However, work-stealing comes with a substantial overhead: as much as 2× to 12× slowdown over orthodox sequential code.

In this paper we identify the key sources of overhead in work-stealing schedulers and present two significant

1. Introduction
Today and in the foreseeable future, performance will be delivered principally in terms of increased hardware parallelism. This fact is an apparently unavoidable consequence of wire delay and the breakdown of Dennard scaling, which together have put a stop to hardware delivering ever faster sequential performance. Unfortunately, software parallelism is often difficult to identify and expose, which means it is often hard to realize the performance potential of modern processors. Work-stealing [3, 9, 12, 18] is a framework for allowing programmers to explicitly expose potential parallelism. A work-stealing scheduler within the underlying lan...
Eliminating Sequential Overheads

- Sequential overheads
  - Initiation
  - State management
  - Code restructuring

- Exploit existing JVM mechanisms
  - **Initiation**: Execution stack for steal initiation
  - **State management**: Extract state from stack & registers
  - **Code restructuring**: Try–catch blocks for control flow

- Eliminated most sequential overheads **12%**
Motivating Analysis
Methodology

• Benchmarks
  – Jacobi
  – LU Decomposition
  – Heat Diffusion

• High steal ratio

• Hardware Platform
  – 2 Intel Xeon E7530
    • 6 cores each

• Software Platform
  – Jikes RVM (3.1.2)
Steals:Task Ratio

Motivating Analysis

---

Measured using JavaWS (Try–Catch)
Motivating Analysis

Steal Rate

Steals/second

Threads

Measured using JavaWS (Try–Catch)
Insight

- Steal ratio and steal rate not correlated
Steal Overhead

Measured using JavaWS (Try–Catch)
Insights

• Steal ratio and steal rate not correlated

• Higher steal \( \text{rate} \) correlates to high steal overhead
Our Approach
Reducing Stealing Overhead

- Two pronged attack
  - Reduce the cost of each steal
    - Return barriers
  - Reduce total number of steal events
    - Steal more than one continuation at a time
Implementation
Return Barrier

- Allows runtime to intercept a common event
- Hijack a return and bridge to some other method
- Register and stack state preserved
Thief Installs Return Barrier

Stack Growth Direction

BASE

A

B

C

D

E

TOP

Yieldpoint Mechanism
Victim Moves The Return Barrier

Stack Growth Direction

TOP

D

C

B

A

BASE
Robbing A Victim With Return Barrier

Stack Growth Direction

TOP

Yieldpoint Mechanism

BASE

D

C

B

A

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Performance Evaluation
Evaluation

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Evaluation

Victim wait time CPU cycles (%)

Speedup Over Sequential

Threads

2 3 4 5 6 7 8 9 10 11 12

Threads

2 3 4 5 6 7 8 9 10 11 12

- No Return Barrier
- With Return Barrier

Faster Work-Stealing With Return Barriers | Kumar & Blackburn | VMIL’12
Summary

• Steal overhead dominated by steal rate
• Two pronged attack
  – Reduce the cost of each steal
    • Return barriers
  – Reduce total number of steals
    • Steal more than one
• No change in speedup

58%
Future Work

- Steal overhead dominated by steal rate
- Two pronged attack
  - Reduce the cost of each steal
    - Return barriers
  - Reduce total number of steals
    - Steal more than one
- No change in speedup?
- Merge both the techniques