Flexible Architectural Support for Fine-Grain Scheduling

Daniel Sanchez
Richard M. Yoo
Christos Kozyrakis

March 16th 2010
Stanford University
Overview

• Our focus: User-level schedulers for parallel runtimes
  – Cilk, TBB, OpenMP, ...

• Trends:
  – More cores/chip → Need to exploit finer-grain parallelism
  – Deeper memory hierarchies
  – Costlier cache coherence
  → Communication through shared memory increasingly inefficient

• Existing fine-grain schedulers:
  – Software-only: Slow, do not scale
  – Hardware-only: Fast, but inflexible

• Our contribution: Hardware-aided approach
  – HW: Fast, asynchronous messages between threads (ADM)
  – SW: Scalable message-passing schedulers
  – ADM schedulers scale like HW, flexible like SW schedulers
Outline

• Introduction

• Asynchronous Direct Messages (ADM)

• ADM schedulers

• Evaluation
**Fine-grain parallelism**

- Fine-grain parallelism: Divide work in parallel phase in small **tasks** (~1K-10K instructions)

- Potential advantages:
  - Expose **more parallelism**
  - Reduce **load imbalance**
  - **Adapt** to a dynamic environment (e.g. changing # cores)

- Potential disadvantages:
  - Large scheduling **overheads**
  - Poor **locality** (if application has inter-task locality)
Task-stealing schedulers

- One task queue per thread
- Threads dequeue and enqueue tasks from queues
- When a thread runs out of work, it tries to steal tasks from another thread
Task-stealing: Components

1. Queues

2. Policies

3. Communication

- In software schedulers:
  - Queues and policies are cheap
  - Communication through shared memory increasingly expensive!
Hardware schedulers: Carbon

- Carbon [ISCA ‘07]: HW queues, policies, communication
  - One hardware LIFO task queue per core
  - Special instructions to enqueue/dequeue tasks

- Implementation:
  - Centralized queues for fast stealing (Global Task Unit)
  - One small task buffer per core to hide GTU latency (Local Task Units)

Graphs showing normalized execution time for different workloads and thread counts, illustrating the benefits and usefulness of carbon implementations depending on app matches with HW policies.
Approaches to fine-grain scheduling

Fine-grain scheduling

Software-only
- OpenMP
- Cilk
- ... (Other software-only options)
- SW queues & policies
- SW communication
- High-overhead
- Flexible
- No extra HW

Hardware-only
- TBB
- X10
- Carbon
- GPUs
- HW queues & policies
- HW communication
- Low-overhead
- Inflexible
- Special-purpose HW

Hardware-aided
- Asynchronous Direct Messages
- SW queues & policies
- HW communication
- Low-overhead
- Flexible
- General-purpose HW
Outline

• Introduction

• Asynchronous Direct Messages (ADM)

• ADM schedulers

• Evaluation
Asynchronous Direct Messages

• ADM: Messaging between threads tailored to scheduling and control needs:
  — Low-overhead
  — Short messages
  — Overlap communication and computation
  — General-purpose

Send from/receive to registers
Independent from coherence

Asynchronous messages with user-level interrupts

Generic interface
Allows reuse
• One ADM unit per core:
  – Receive buffer holds messages until dequeued by thread
  – Send buffer holds sent messages pending acknowledgement
  – Thread ID Translation Buffer translates TID → core ID on sends
  – Small structures (16-32 entries), don't grow with # cores
## ADM ISA

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adm_send r1, r2</td>
<td>Sends a message of (r1) words (0-6) to thread with ID (r2)</td>
</tr>
<tr>
<td>adm.peek r1, r2</td>
<td>Returns source and message length at head of rx buffer</td>
</tr>
<tr>
<td>adm_rx r1, r2</td>
<td>Dequeues message at head of rx buffer</td>
</tr>
<tr>
<td>adm_ei / adm_di</td>
<td>Enable / disable receive interrupts</td>
</tr>
</tbody>
</table>

- Send and receive are atomic (single instruction)
  - Send completes when message is copied to send buffer
  - Receive blocks if buffer is empty
  - Peek doesn't block, enables polling
- ADM unit generates an user-level interrupt on the running thread when a message is received
  - No stack switching, handler code partially saves context (used registers) → fast
  - Interrupts can be disabled to preserve atomicity w.r.t. message reception
Outline

• Introduction

• Asynchronous Direct Messages (ADM)

• ADM schedulers

• Evaluation
ADM Schedulers

- Message-passing schedulers
- Replace parallel runtime’s (e.g. TBB) scheduler
  - Application programmer is oblivious to this
- Threads can perform two roles:
  - **Worker**: Execute parallel phase, enqueue & dequeue tasks
  - **Manager**: Coordinate task stealing & parallel phase termination
- Centralized scheduler: Single manager coordinates all

![Diagram](image.png)
Centralized Scheduler: Updates

Manager keeps *approximate task counts* of each worker
Workers only notify manager at exponential thresholds
• Manager requests a steal from the worker with most tasks
Hierarchical Scheduler

- Centralized scheduler:
  - Does all communication through messages
  - Enables directed stealing, task prefetching
  - Does not scale beyond ~16 threads

- Solution: Hierarchical scheduler
  - Workers and managers form a tree
Hierarchical Scheduler: Steals

- Steals can span multiple levels
  - A single steal rebalances two partitions at once
  - Scales to hundreds of threads
Outline

• Introduction

• Asynchronous Direct Messages (ADM)

• ADM schedulers

• Evaluation
Evaluation

- Simulated machine: Tiled CMP
  - 32, 64, 128 in-order dual-thread SPARC cores (64 – 256 threads)
  - 3-level cache hierarchy, directory coherence

- Benchmarks:
  - Loop-parallel: canneal, cg, gtfold
  - Task-parallel: maxflow, mergesort, ced, hashjoin
  - Focus on representative subset of results, see paper for full set

64-core, 16-tile CMP
Results

- SW scalability limited by **scheduling overheads**
- **Carbon and ADM**: Small overheads that scale
- ADM matches Carbon → No need for HW scheduler
Flexible policies: gtfold case study

• In gtfold, FIFO queues allow tasks to clear critical dependences faster
  — FIFO queues trivial in SW and ADM
  — Carbon (HW) stuck with LIFO
• ADM achieves 40x speedup over Carbon
• Can’t implement all scheduling policies in HW!