Helper Threading: Improving Single-Thread Performance Using Additional Execution Contexts

Jeff Brown, UCSD/CSE
2003/07/28
Introduction to Helper Threading
Introduction

- An ever-present goal in computer architecture: to run faster.
- Processors fall far short of performance potential:
  - Waiting for input values
  - Waiting for memory
  - Speculating uselessly
- Due in part to imperative control-driven execution models.
- Recent designs with multiple execution contexts (SMT, CMP) allow processors to exploit parallelism in control-independent instruction streams.
- We'll explore helper threading: techniques for using these additional contexts to speed up single-thread workloads.
Enabling Technologies: SMT & CMP
Helper Techniques: Prediction, Speculation

- Software-guided prefetching
- Early branch resolution (supplements BTB, predictor)
- Speculative optimization, re-optimization
- Task-level speculation
- Microarchitectural optimization

- Distinct from (but related to) automatic parallelization:
  - May be hardware-speculative
  - Finer granularity
  - Hardware-aware
Example: Prefetching
Example: Prefetching (2)
Example: Prefetching (3)
Outline

- Introduction

- Slice-based Helper Threads
  - Basic Helper Threading Scheme
  - Design of Slice-based Helpers
  - Execution-based Prediction Using Speculative Slices
  - Speculative Data-Driven Multithreading
  - Dynamic Speculative Precomputation

- Beyond Slices

- Wrap-up
Slice-based Helper Threads: Prefetchers, Predictors
Example: Problematic Load

... 

P0: LD R2, 0 (SP) 
P1: BEQ R2, ... (Bias: NT) 
P2: ADD ... 
P3: ST ... 
P4: LD R3, 0 (R2) 
P5: LD R4, 0 (R2) 
P6: CMPE R5, R4, R3 
P7: BEQ R5, ... (Bias: NT) 
P8: LD R3, 8 (R3) (Likely L2 miss) 
P9: ST R3, 0 (R2) 
... 

Profile information
Example: Problematic Load (2)

P0: LD R2, 0 (SP)
P1: BEQ R2, ...
P2: ADD ...
P3: ST ...
P4: LD R3, 0 (R2)
P5: LD R4, 0 (R2)
P6: CMPE R5, R4, R3
P7: BEQ R5, ...
(Bias: NT)
P8: LD R3, 8 (R3)  (Likely L2 miss)
P9: ST R3, 0 (R2)
...

Full dependences
Hot-path dependences
Profile information
Basic Helper Threading Scheme

Main

Original Code
P0: LD R2, 0(SP)
P1: BEQ R2, ...
P2: ADD ...
P3: ST ...
P4: LD R3, 0(R2)
P5: LD R4, 0(R2)
P6: CMPIE R5, R4, R3
P7: BEQ R5, ...
P8: LD R3, 8(R3)
P9: ST R3, 0(R2)
...

Program State
IP  P2
R2  0x12345678
...
...

Static Helper Info
P2  R2  P4,8

Helper

Original Code
P4: LD R3, 0(R2)
P8: LD R3, 8(R3)
HALT

Program State
IP  P2
R2  0x12345678
...

3. Helper Construction
1. Trigger Match
2. Live-in Copy

Trigger
Target
Design of Slice-based Helpers

- Identification / classification of suitable targets:
  - Online vs. offline analysis
  - Static vs. dynamic instructions

- Construction of helpers:
  - Instructions to include (leanness vs. ability)
  - Trigger selection (lead-time vs. utility)
  - Representation of helpers

- Usage of helper threads:
  - Trigger conditions
  - Helper fetch policy
  - Communication between threads
  - Memory synchronization
  - Helper looping/forking, termination policy
  - Hiding state changes (stores, exceptions)
"Program Slicing": A Common Technique

- Helper prefetchers/predictors often a small number of loads/branches:
  - Usually, most of benefit comes from the top few targets.
  - Overhead from helpers must be mitigated.

- Backwards program "slices" are a natural way to construct helpers:
  - Starting with target (slice "criterion"), do a transitive closure in reverse DFG/CFG.
  - Defined in both static and dynamic instruction streams.
  - Bounded by size slice size, window size, or instruction types.

- Program slicing: late 1970s debugging aid; [SLICE, Zilles] studies it from a helper-threading perspective:
  - "Correct" slices approach full-program size.
  - Speculation can reduce to <10% of that.
Common Concerns For Helper Schemes

- Implementation feasibility, complexity
- ISA changes
- Hardware costs
- Pipeline length, timing impact
- Impact on future designs
Execution Based Prediction ... [EBP, Zilles]

- **Software-driven, SMT-based helper threading:**
  - Targets performance-degrading L1 cache misses and branch mispredictions.
  - Uses static helper threads, hand-built from backwards (static) program slices.

- **Helper thread actions:**
  - Spawned at trigger fetch; fetch favors main thread
  - Helper loops, but bounded with an iteration count limit
  - Targeted branches in helper are specially flagged, and outcomes passed to hardware correlator
EBP Overview
EBP Analysis

- Achieved 13% mean speedup on a subset of SPECINT2000.
- Helper-generated branch predictions were 99% accurate.
- New hardware:
  - Associative structures: slice table, PGI table (total <512B)
  - Register map copy pathway, stealing rename ports
- Live-in copy through physical register re-use
  - They imply modeling rename port contention
  - Assumes reference-counting physical reg. allocation
- Pipeline cycle time is likely unaffected.
Speculative Data-Driven Multithreading [DDMT, Roth]

- Software-driven, SMT-based helper threading:
  - Targets L1 misses and branch mispredictions.
  - Offline analysis, static helpers.
  - Helper construction guided by a novel cost metric: Fetch-Constrained Dataflow Height (FCDH).
  - Performs induction unrolling as part of candidate generation.

- Helper thread actions:
  - Spawned at trigger rename
  - One-shot execution; control instructions do not affect helper sequencing, helpers cannot spawn helpers
  - Dedicated hardware "cloaking table" handles helper stores
  - "Register integration" hardware allows threads to share results (SMT only)
  - Branch outcomes passed via integration
DDMT Overview

Main

Helpers

Integration

(Unrolled)

Integration

(Arbitrary value re-use)

(More insts)
Register Integration: Tradeoffs

- Introduced in [RI, Roth] for squash re-use
  - Modifies logical -> physical register mapping
  - Memoizes (PC, PhysIn1, PhysIn2) -> PhysOut values
  - On a (PC, PhysIn1, PhysIn2) match, PhysOut is re-used

- Allows for fine-grained, bidirectional communication

- Helper dataflow severely constrained
DDMT Analysis

● Achieved 14% mean speedup on memory-targeted workloads, 6% on branch-targeted.

● Significant new concept: FCDH

● New hardware
  ◆ Assumes preexisting Integration Table
  ◆ Associative structures: DDTC, CT
  ◆ New data-path: register map copy (1:n)

● One-shot helpers: missed opportunity?

● Register integration tightly constrains helpers

● Cycle time effects
  ◆ Cloaking Table: likely negligible
  ◆ Integration
  ◆ Register map copying
Dynamic Speculative Precomputation [DSP, Collins]

- Hardware-only, SMT-based sequel to [SP, Collins]:
  - Targets off-chip (L2) misses with hardware-constructed helpers.
  - Identifies "delinquent" loads at run-time, those blocking commit.
  - Builds helpers with backwards (dynamic) program slices from targeted loads.
  - Analyzes, constructs, & optimizes helpers in back-end of pipeline.

- Helper thread actions:
  - Spawned at trigger rename; fetch favors helpers
  - Live-in values copied with explicit moves
  - Usually, short load-add-load sequences with few live-ins

- Optimizations:
  - Induction Unrolling
  - Chaining: use the same slice repeatedly to capture loops; throttle based on iteration/recursion run-ahead
DSP Overview

Main

Helpers

(Looping)

(Branches not targeted)

(Unrolled)
DSP Analysis

- Achieved 33% mean speedup on "memory-bound" workloads.
  - ( Classified on speedup from perfect L2 cache.)
- Helper benefits are all from cache effects.
- New hardware
  - Associative structures: DLIT, SIT (per-cyc), SC, SAT
  - Analysis engines: RIB (high latency), SIT
- Doesn't perform higher-level optimizations
  - Misses out on drastic size reductions reported in [SLICE, Zilles].
- Pipeline cycle time is likely unaffected.
Outline

- Introduction
- Slice-based Helper Threads
- Beyond Slices
  - A Study of Slipstream Processors
  - Master/Slave Speculative Parallelization
  - Simultaneous Subordinate Multithreading
- Wrap-up
Beyond Slices: Other Helper Threading Schemes
Slipstreaming [SLIP, Purser]

- Hardware-based, two-thread system:
  - Augments original program (R-stream) with a single, speculative helper (A-stream).
  - Dynamically constructs and refines the helper in hardware.
  - Uses helper for branch and value predictions.

- Helper thread actions:
  - Executes a reduced copy of main thread
  - Runs strictly ahead of main thread
  - Passes all outcomes through a FIFO
  - Stores to non-shared memory; misspeculation recovery done with explicit copies from main thread
Slipstreaming Overview

Main

(No triggers)

(Arbitrary value re-use)

(Runs continuously)

Helper

(More insts)

Result FIFO
Slipstreaming Pipeline ([SLIP, Purser], figure 3)
Slipstreaming Analysis

- Achieved 12% mean speedup on a two-way CMP for a subset of SPECINT95.

- Helper benefits: cache effects, branch predictions, and value predictions.

- New hardware
  - Associative structures: IR-predictor
  - Analysis engines: IR-detector (computes R-DFG)

- Delay buffer: feasible on a CMP?
  - Reversible, high-bandwidth path between cores
  - Can be pipelined for cycle time

- Pipeline cycle time is likely unaffected.
Master-Slave Spec. Parallelization [MSSP, Zilles]

- Speculative task-level parallelization
  - Descended from the Multiscalar execution paradigm [MP, Sohi]
  - Parallelized off-line, with a binary-to-binary translator

- Slave threads are much more than slices:
  - Tasks: entire loops, functions, etc.
  - Tasks are started speculatively; task changes are buffered.

- Master thread executes a "distilled" program:
  - Specialized for typical executions, based on profiling.
  - Speculatively optimized (e.g., with biased branches promoted and likely-useless instructions removed).

- Speculation takes place at higher levels:
  - Distillation: instruction selection, optimization
  - Task triggering: existence, explicit next-task prediction
  - Task initiation: explicit live-in prediction

- The original code (in slaves) is used as a checker.
MSSP Overview

Original

Master

Slaves

(Time saved)

Verify/Commit
MSSP Analysis

● Achieved 26% mean speedup on SPECINT2000.

● Benefits stem from parallelization:
  ◆ One thread focuses only on time-critical task sequencing.
  ◆ Other threads start work (original code) sooner.

● Inter-thread communication:
  ◆ Only at task boundaries
  ◆ Only between slaves and master
  ◆ Changes apply a task at a time

● New hardware
  ◆ Associative: Checkpoint Buffer, PC map, (Task Summaries?)
  ◆ Analysis engine: Verify/Commit unit

● Cycle time effects unclear
Simultaneous Subordinate Multithreading [SSMT, Chappell]

- Software-driven helper threading:
  - Runs statically-built "microthread" helpers.
  - Helpers can prefetch, do branch prediction, manage hardware resources.

- Helper thread actions:
  - Spawned by explicit spawn instructions, or through event triggers (cache misses, timer expirations)
  - Reacts to program execution as it unfolds
  - Manipulates the hardware below the traditional ISA level
  - Can perform sophisticated hardware management
  - Need not be tightly coupled with original application

- As proof-of-concept, they implemented a 16MB software branch predictor
SSMT Overview

Main

Helpers

(Trigger event)

Prediction Cache
SSMT Analysis

- Achieved 23% speedup on *one* SPECINT95 benchmark.
  - Not a stunning proof-of-concept.
- Demonstrated a predictor for predesignated branches.
  - 16 MB of table storage provides long, conflict-free local histories
  - Prediction latency and prediction cache correlation not explored
- New hardware
  - Associative structure: Prediction Cache
- Pipeline is substantially similar to SMT
Outline

- Introduction
- Slice-based Helper Threads
- Beyond Slices
- Wrap-up
Wrap-up
## Helper Schemes

<table>
<thead>
<tr>
<th>Slices</th>
<th>Targets</th>
<th>Innovations</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBP</td>
<td>Pf, Br</td>
<td>Slice optimizations</td>
</tr>
<tr>
<td>DDMT</td>
<td>Pf, Br</td>
<td>Bidir comm/reuse with R.I., FCDH, Store-cloaking h/w (one-shot)</td>
</tr>
<tr>
<td>DSP</td>
<td>Pf</td>
<td>Hardware-only, on-line FDDO</td>
</tr>
<tr>
<td>SLIP</td>
<td>Hot-path Br, values</td>
<td>Hardware-only, Outcome FIFO</td>
</tr>
<tr>
<td>MSSP</td>
<td>Task-level control, live-in</td>
<td>Shortens true dep. path, no inter-helper comms</td>
</tr>
<tr>
<td>SSMT</td>
<td>Unbiased Br</td>
<td>Uses s/w as custom h/w, Opens microarch to helpers</td>
</tr>
</tbody>
</table>
Future Work

- Main benefit of slice-based schemes is memory parallelism
  - [RMC, Pai] targets this explicitly, as compiler optimization
  - ISA changes to expose it as part of the code?

- SSMT's specific example wasn't convincing
  - Custom helper front-end: only a detail
  - Promising, in a more latency-tolerant role?

- New idea: opening architecture to helpers
  - Vanilla main program, target-tuned helpers
  - OS/runtime-supplied helpers replacing h/w, s/w
  - Helper management of complex internal structures
    - Trace Cache filling, translation, optimization
    - Pipeline tuning for instruction types, data widths
Conclusion

- We've examined six related helper threading schemes.
- Common themes: finding parallelism, hiding latency
- Looking forward
  - Applying SSMT ideas to SMT
  - Pushing hardware / microcode into threads

- Thanks for your time!
Results, Etc.
EBP Results ([EBP, Zilles], figure 11)
## DDMT Results ([DDMT, Roth], table 5)

<table>
<thead>
<tr>
<th></th>
<th>Cache Misses</th>
<th>Branch Mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mcf</td>
<td>vpr</td>
</tr>
<tr>
<td><strong>Base</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>529.39</td>
<td>1304.95</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>12.43</td>
<td>3.86</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>24.57</td>
<td>47.48</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td><strong>9.2</strong></td>
<td><strong>12.0</strong></td>
</tr>
<tr>
<td><strong>Base + DDMT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>574.96</td>
<td>1400.95</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>7.45</td>
<td>3.36</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>19.44</td>
<td>39.02</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td><strong>9.2</strong></td>
<td><strong>12.0</strong></td>
</tr>
<tr>
<td><strong>Base + DDMT - critical scheduling</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>529.42</td>
<td>1305.15</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>12.41</td>
<td>3.80</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>24.64</td>
<td>39.16</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td><strong>0.0</strong></td>
<td><strong>-0.1</strong></td>
</tr>
<tr>
<td><strong>Base + DDMT - Integration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructions fetched (M)</td>
<td>605.93</td>
<td>1406.29</td>
</tr>
<tr>
<td>Load latency (cycle)</td>
<td>9.45</td>
<td>3.65</td>
</tr>
<tr>
<td>Resolution latency (cycle)</td>
<td>22.56</td>
<td>41.29</td>
</tr>
<tr>
<td>Execution time saved (%)</td>
<td><strong>1.7</strong></td>
<td><strong>8.5</strong></td>
</tr>
</tbody>
</table>
DSP Results ([DSP, Collins], figure 7)
(With instant analysis and free live-in copies.)

- Prefetch Conversion
- Improved Trigger
- Induction Unrolling
- Chaining

Graph showing speedup over no SP for various benchmarks with different techniques applied.
Slipstream Results ([SLIP, Purser], figure 6)
MSSP Execution ([MSSP, Zilles], figure 2a)
MSSP Results ([MSSP, Zilles], figure 12c)
SSMT Pipeline ([SSMT, Chappell], figure 2)
SSMT Results ([SSMT, Chappell], figure 5)
Administrativa
Research Exam Purpose
(Copied from department web site)

● To assess my preparedness for continuing in the Ph.D. program.

● To demonstrate my ability to reason about a research area:
  ♦ Comprehension
  ♦ Synthesis
  ♦ Presentation
  ♦ Directions for future research

● Focus is ideas more than implementation details.

● *Not* a thesis proposal.
Primary Paper List

- [DDMT, Roth]: Speculative Data-Driven Multithreading, Roth, Sohi (2001)
- [SLIP, Purser]: A Study of Slipstream Processors, Purser, Sundaramoorthy, Rotenberg (2000)
- [SSMT, Chappell]: Simultaneous Subordinate Microthreading (SSMT), Chappell, Stark, Kim, Reinhardt, Patt (1999)
Related Paper List


