Tasking with Out-of-Order Spawn in TLS Chip Multiprocessors: Microarchitecture and Compilation

Jose Renau
University of California at Santa Cruz
http://masc.soe.ucsc.edu

James Tuck, Wei Liu, Luis Ceze, Karin Strauss, and Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu
Technology Trends

- Wire delay is becoming dominant
- Power efficiency is decreasing → More power density
- Complexity is growing exponentially

Future Proposals must consider these trends
Wire Delay is Becoming Dominant

1 clock cycle reaches

From Semiconductor Industry Association (SIA) Roadmap

Micr
ocprocessor Chip

70 nm
100 nm
130 nm

Not a single monolithic processor
Power Efficiency is Decreasing

- Same work: faster ... but require much more power

From keynote presentation Micro 2003
Complexity is Growing Exponentially

Complexity growth rate (58%/Year)

Productivity growth rate (21%/Year)

From Sematech

$90 M
$160 M
$360 M
$750 M
$1160 M
Implications

- Wire delay:
  - Not a single monolithic processor

- Power:
  - Energy-efficient design (simple cores are efficient)

- Complexity:
  - Very large block reuse

Chip Multiprocessor?
Chip Multiprocessor (CMP)

- Has a natural advantage for parallel apps
- **No speedups** for sequential applications

**CMP with Thread Level Speculation**
Thread Level Speculation (TLS)

Sequential \[\Rightarrow\] 

\[
\text{for}(i=0; i<n; i++) \{
    X[Y[i]] = X[Z[i]]...
\}
\]

- Compilers cannot parallelize
- TLS: Assume no dependences, hardware verifies

TLS Task A \[\Rightarrow\] 

\[
\text{for}(i=0; i<n/2; i++) \{
    X[Y[i]] = X[Z[i]]...
\}
\]

TLS Task B \[\Rightarrow\] 

\[
\text{for}(i=n/2; i<n; i++) \{
    X[Y[i]] = X[Z[i]]...
\}
\]
Thread Level Speculation (TLS)

- TLS Hardware:
  - Tracks data accesses at run-time
  - Detects dependence violations
  - Kills and restarts tasks

**Sequential** | **TLS (no dep violations)** | **TLS (dep violation)**
--- | --- | ---
Time | A | A
Time | B | B
Time | A | B
Time | B
Architectures Compared

1 CPU 3-issue
Sequential apps only

1 CPU 6-issue

4 CPUs 3-issue with TLS
Sequential apps using ALL cores and parallel apps
My Contributions: Performance

Boost TLS performance through much more flexible task execution

- Aggressive out-of-order task spawn
- Novel TLS CMP micro-architecture
- Novel compiler algorithms
Main Results

**TLS: 26% faster and 26% more energy**

1 CPU 3-issue

1 CPU 6-issue

4 CPUs 3-issue with TLS

6-issue: 23% faster and 52% more energy
More Flexible Task Selection

- Current TLS CMP are restricted to in-order spawn:
  - Simpler hardware
  - More complex compiler

- Proposal: More flexible task selection
Task Generation

- Fully automated gcc pass that generates tasks on:

**Loop iterations**

- Time
- Dynamic Execution
- Loop
- Iteration 0
- Iteration 1
- Iteration 2

**Subroutines**

- Time
- Caller
- Subroutine
- Continuation
- Spawn
- Continuation
Out-of-Order Task Spawn

- High flexibility: Any subroutine or loop iteration
- Task nesting induces out-of-order spawn

Sequential

Sub S1{
  ...
  Spawn S2Cont
  S2();
}
S2Cont:

Sub S2{
  ...
  Spawn S3Cont
  S3();
}
S3Cont:

Time

1 2 3 4 5

More Speculative

Spawn

S2Cont

Non-spec Task

Out-of-order Spawn

Time

1 2 3 4 5

S2Cont

S3Cont

S3
Challenges in Out-of-Order Spawn

- Hard-to-predict dynamic task hierarchy
- How to maintain task order dynamically?
  - Communication between tasks requires relative order
- How to manage resources dynamically?
  - Highly spec tasks can clog resources

In-Order

Out-of-Order

More Speculative

Time

A B C D E
Task Ordering: Version Intervals

- In-order spawn:
  - Each task has a Version ID
  - On spawn: Parent task gives Version ID + 1 to child

- Proposal for out-of-order spawn:
  - Each task is assigned an interval of versions \([\text{Base}, \text{Limit})\]
  - Communicating tasks use \text{Base} to identify order
  - On spawn: Parent task gives upper half interval to child
Version Intervals Example

Sub S1{
  ...
  Spawn S2Cont
  S2();
}

S2Cont:
  ....

Sub S2{
  ...
  Spawn S3Cont
  S3();
}

S3Cont:
  ....

More Speculative

Non-spec Task

Time

[1,2) [2,3) [3,4)
Dynamically Managing Resources

- In-order spawn:
  - Easy to manage resources: More spec tasks spawned later

- Out-of-order spawn:
  - Hard to manage resources:
    - Example: highly spec tasks can clog resources
Dynamic Task Merging

Proposal: “Fuse” tasks at run time

- **MergeNext:**
  - Parent skips spawn instruction: Fuses parent & child

- **MergeLast:**
  - Task kills most spec task: Fuses two most spec tasks
Dynamic Task Merging

**Without Task Merge**
More Speculative

**MergeNext**
More Speculative

**MergeLast**
More Speculative
Task Merging Trade-offs

- **MergeNext:**
  - Our first choice:
    - Reduces overheads
    - Increases instructions per cycle (IPC)
    - No task kill
  - Activated when there are no free CPUs

- **MergeLast:**
  - Last resort: Kills potentially useful work
  - Activated when number of tasks over threshold
Simulation Environment

- 3issue
  - 1 CPU 3-issue

- 6issue
  - 1 CPU 6-issue

- TLS4
  - 4 CPUs 3-issue with TLS

- 70nm @ 5GHz
- All processors have same pipeline depth
- 16KB L1 cache (1 cycle slower in TLS due to versioning)
- 1MB L2 cache on-chip
TLS Compilation Infrastructure

- Fully automatic TLS pass using TreeSSA from gcc 3.5
- Builds tasks and inserts spawn & commit instructions

Software value prediction:
- Subroutine return value
- Loop induction variables

Automated, simple, and fast profiling pass
- Use SPECint train input set
OutOrder is 20% faster than InOrder

TLS4 OutOrder is slightly faster than 6issue
TLS4 OutOrder is Promising

- Outperforms:
  - 3issue (26% speedup), 6issue (3% speedup)

- Consumes 15% less power than 6issue

---

... for hard to parallelize applications

CMP has a natural advantage for parallel apps!
Questions?
TLS CMP: Architecture Requirements

- Minor modifications in the processor core
- Versioned L1 cache
  - Cache lines are associated to tasks
  - Ability to discard per-task cached state (in a task kill)
- TLS aware cache coherence protocol
  - Data for speculative tasks cannot be displaced from L1
  - Detects cross-task dependence violations
Task Ordering: Timestamps Intervals

- A task is assigned a range of timestamps $[Base, Limit)$
- Tasks compare $Base$ to identify order
- On spawn: Parent task gives upper half interval to child

Parent Task before spawn

Parent Task after spawn

Child Task

$[B, \frac{L}{2})$

$[B, L)$

$[B + \frac{L}{2}, L)$
More Technology Trends

Range of a Wire in One Clock Cycle

- From the SIA Roadmap

- *Keynote presentation Micro 2003

*SIA Roadmap
Task Squash Optimizations

- Stall task after second restart
  - Diminishing returns in performance (cache, branches) with more restarts

- Energy-aware task pruning by profiling
  - Deselect tasks that are expected to give minor speedups at high energy cost
Storage & Logic Optimizations

- Avoid eagerly walking the cache tags
  - TLS requires group ops on cache tags: task commit/squash
  - We perform group ops lazily in background

- Low-energy data reuse on task restart
  - On task restart, keep same Version ID, add version offset
  - Reduces the number of checks to reuse a clean cache line
Applications

- SPECint 2000
  - Run with ref input set
  - Profile with train input set

- Multiple binaries: (High quality compiler opt)
  - Sequential binary (No TLS instrumentation): Base
  - TLS binaries: InOrder and OutOrder

- Simulate more than 500 million inst after initialization
TLS Compilation Infrastructure

• Fully automatic TLS pass using TreeSSA from GCC3.5

• Generates tasks for:
  • Iterations of multiple loops in a nest
  • Subroutines and functions (any nesting level)

• Software value prediction:
  • Subroutine return value
  • Loop induction variables

• Inserts spawn & commit instructions

• Spills live-ins from registers across tasks
Compiler Phases

Selection
- Program Structure
- Dependence Based Passes

Spawn Hoisting
- Task Graph Analysis
- Value Prediction
- Valid Locations
- Dependence Restriction
- Placement

Pruning
- Spawn Distance
- Small Tasks
- High Dependence
- Profiling
- In-Order?

Live-ins
- Task Graph Analysis
- Restores Set
- Lazy Code Motion
- Placement
- Finalize Tasks

Profiler

Stored Binaries
In-Order Task Generation

- In-order spawn requires a more complex compiler:
  - Additional check in task pruning pass to guarantee in-order
  - Added IPA (Inter-Procedural Analysis)
  - Use perfect IPA in libraries
Profiling Infrastructure

- Automated, simple and fast profiling pass
  - Sequential execution
  - Ideal time-less simulation
  - Use SPECint train input set

- Profiler prunes a task when:
  - Too small...
  - Too many estimated dependences...
  - Little expected task overlap...
  - ... unless it has a high estimated L2 cache miss rate
SESC Code Size

Number of lines vs Date

Date:
- 07/01
- 10/01
- 01/02
- 04/02
- 07/02
- 10/02
- 01/03
- 04/03
- 07/03
- 10/03
- 01/04

Number of lines:
- 0
- 10000
- 20000
- 30000
- 40000
- 50000
- 60000
- 70000
- 80000