Complementing User-Level Coarse-Grain Parallelism with Implicit Speculative Parallelism

Nikolas Ioannou, Marcelo Cintra

School of Informatics
University of Edinburgh
Introduction

- Multi-cores and many-cores here to stay

Source: Intel
Introduction

- Multi-cores and many-cores are here to stay
- Parallel programming is essential to realize potential
- Focus on coarse-grain parallelism
- Weak or no scaling of some parallel applications
- Can we exploit under-utilized cores to complement coarse-grain parallelism?
  - Nested parallelism in multi-threaded applications
  - Exploit it using implicit speculative parallelism
Contributions

- Evaluation of implicit speculative parallelism on top of explicit parallelism to improve scalability:
  - Improve scalability by 40% on avg.
  - Same energy consumption
- Detailed analysis of multithreaded scalability:
  - Performance bottlenecks
  - Behavior on different input datasets
- Auto-tuning to dynamically select the number of explicit and implicit threads
Outline

- Introduction
- Motivation
- Proposal
- Evaluation Methodology
- Results
- Conclusions
Bottlenecks: Large Critical Sections

$T_0$ $T_1$ $T_2$ $T_3$

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

Time

T₀ T₁ T₂ T₃

Integer Sort (IS)
NASPB
Bottlenecks: Large Critical Sections

Time

T₀  T₁  T₂  T₃

Integer Sort (IS)

NASPB
Bottlenecks: Large Critical Sections

Time

Integer Sort (IS)  NASPB

T_0  T_1  T_2  T_3

Graph showing Speedup over Cores:

- Busy
- Lock
- Barrier

Graph showing Normalized Execution Time over Cores:
Bottlenecks: Load Imbalance

$T_0 \ T_1 \ T_2 \ T_3$

Time

RADIOSITY

SPLASH 2
Bottlenecks: Load Imbalance

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

T₀  T₁  T₂  T₃

Time

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

RADIosity
SPLASH 2
Bottlenecks: Load Imbalance

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

RADIOSITY

SPLASH 2
Bottlenecks: Load Imbalance

Time

T₀  T₁  T₂  T₃

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

RADIOSITY
SPLASH 2
Bottlenecks: Load Imbalance

RADIOSITY

SPLASH 2
Bottlenecks: Load Imbalance

Time

T₀ T₁ T₂ T₃

RADIOSITY

SPLASH 2

Speedup

Cores

Busy

Lock

Barrier

Norm. Execution Time

Cores

Intl. Symp. on Microarchitecture - December 2011
Bottlenecks: Load Imbalance

Can we use these cores to accelerate this app?

RADIOSITY
SPLASH 2
Outline

- Introduction
- Motivation
- Proposal
- Evaluation Methodology
- Results
- Low power nested parallelism
- Conclusions
Proposal

- **Programming:**
  - Users explicitly parallelize code
  - Tradeoff development time for performance gains

- **Architecture and Compiler:**
  - Exploit fine-grain parallelism on top of user threads
  - Thread-Level Speculation (TLS) within each user thread

- **Hardware:**
  - Support both explicit and implicit threads simultaneously in a nested fashion
#pragma omp parallel for
for(j = 0; j < M; ++j) {
  ...
  for(i = 0; i < N; ++i)
    {
      ... = A[L[i]] + ...
      ...
      A[K[i]] = ...
    }
  ...
}
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        A[L[i]] = ...
        ...
    }
}
#pragma omp parallel for
for (j = 0; j < M; ++j) {
    ...
    for (i = 0; i < N; ++i) {
        ...
        A[L[i]] = ...
    }
    ...
}
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        A[L[i]] = ...
    }
    ...
}
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
    ...
}
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
    ...
}
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        A[L[i]] = ...  
        ...
    }
}
Proposal: Many-core Architecture

- Many-core partitioned in clusters (tiles)
- Coherence (MESI)
  - Snooping coherence within cluster
  - Directory coherence across clusters
- Support for TLS only within cluster
  - Snooping TLS protocol
  - Speculative buffering in L1 data caches
Proposal: Many-core Architecture

Intl. Symp. on Microarchitecture - December 2011
Proposal: Many-core Architecture
Complementing Coarse-Grain Parallelism
Complementing Coarse-Grain Parallelism

Time

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

2x Explicit Threads
Complementing Coarse-Grain Parallelism

Time

$T_0$ $T_1$ $T_2$ $T_3$

$T_0$ $T_1$ $T_2$ $T_3$ $T_4$ $T_5$ $T_6$ $T_7$

2x Explicit Threads
Complementing Coarse-Grain Parallelism
Complementing Coarse-Grain Parallelism

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]

4ETs + 4ISTs
Complementing Coarse-Grain Parallelism

4ETs + 4ISTs
Complementing Coarse-Grain Parallelism

Time

\[ T_0 \quad T_1 \quad T_2 \quad T_3 \]
Complementing Coarse-Grain Parallelism

Time

T₀ T₁ T₂ T₃

2x Explicit Threads
Complementing Coarse-Grain Parallelism

Time

2x Explicit Threads
Complementing Coarse-Grain Parallelism
Complementing Coarse-Grain Parallelism

Time

T₀ T₁ T₂ T₃

4ETs + 4ISTs
Complementing Coarse-Grain Parallelism

4ETs + 4ISTs

Time
Expected Speedup Behavior

- 4-way TLS speedup region
- 2-way TLS speedup region
- Baseline speedup region

A
B
C

Baseline
4-way TLS
2-way TLS
Proposal: Auto-Tuning the Thread Count

- Find the scalability tipping point dynamically
- Choose whether to employ implicit threads
- Simple hill climbing approach
- Applicable to OpenMP applications that are amenable to Dynamic Concurrency Throttling (DCT [Curtis-Maury PACT’08] )
- Developed a prototype in the Omni OpenMP System
Auto-tuning example

```c
... #pragma omp parallel for
for(j = 0; j < M; ++j) {
  ...
  for(i = 0; i < N; ++i) {
    ... = A[L[i]] + ...
    ...
    A[K[i]] = ...
  }
} ...
```
Auto-tuning example

```c
... #pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
        }
    ...
}```
Auto-tuning example

...#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i)
    {
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
}...

omp parallel region $i$ detected:

First time:
Can we compute iteration count statically and is less than max core count?
Auto-tuning example

Learning

#pragma omp parallel for
for (j = 0; j < M; ++j) {
    for (i = 0; i < N; ++i) {
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
...
Auto-tuning example

```
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    for(i = 0; i < N; ++i)
    {
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
}
```

Omp parallel region i detected:
First time:
Can we compute iteration count statically and is less than max core count?
Yes -> set Initial Tcount to 32
Auto-tuning example

Learning

omp parallel region $i$ detected:
First time:
Can we compute iteration count statically and is less than max core count?
Yes -> set Initial $Tcount$ to 32
Measure execution time $t_j$
Auto-tuning example

```
... #pragma omp parallel for
for(j = 0; j < M; ++j) {
  ...
  for(i = 0; i < N; ++i) {
    ...
    ... = A[L[i]] + ...
    ...
    A[K[i]] = ...
  }
  ...
}
...```

Learning → --- i
Auto-tuning example

```
... #pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i)
    {
        ...
        A[L[i]] = ...
        ...
        A[K[i]] = ...
    }
    ...
}
...```

Learning → i --- i
Auto-tuning example

```
#pragma omp parallel for
for(j = 0; j < M; ++j) {
  ...
  for(i = 0; i < N; ++i) {
    ...
    A[L[i]] = ...
    ...
    A[K[i]] = ...
  }
  ...
}
```

omp parallel region \( i \) detected:

Set \( T\text{count} \) to next value (16)

Measure execution time \( t_i^2 \)

\( t_i^2 < t_i^1 \) → continue exploration
Auto-tuning example

```c
... #pragma omp parallel for
     for(j = 0; j < M; ++j) {
         ...
         for(i = 0; i < N; ++i) {
             ...
             = A[L[i]] + ...
             ...
             = A[K[i]] = ...
             }
         ...
     }
...```
Auto-tuning example

```
...#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        ... = A[L[i]] + ...
        ...
        A[K[i]] = ...
    }
    ...
}
...```

Auto-tuning example

... #pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i)
    {
        ...
        A[L[i]] = ...
        ...
        A[K[i]] = ...
    }
    ...
...
Auto-tuning example

\[
\begin{align*}
\#pragma \text{omp parallel for} \\
\text{for}(j = 0; j < M; ++j) \{ \\
\quad \text{for}(i = 0; i < N; ++i) \{ \\
\quad \quad \ldots = A[L[i]] + \ldots \\
\quad \quad \ldots \\
\quad \quad A[K[i]] = \ldots \\
\quad \} \\
\} \\
\ldots 
\end{align*}
\]
Auto-tuning example

```
#pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i) {
        ...
        \[ A[L[i]] \] + ...
        ...
        \[ A[K[i]] \] = ...
    }
    ...
}
...```
Auto-tuning example

... #pragma omp parallel for
for(j = 0; j < M; ++j) {
    ...
    for(i = 0; i < N; ++i)
    {
        ...
        A[L[i]] = ...
        ...
        A[K[i]] = ...
    }
...

omp parallel region i detected:
Use Tcount = 16, no further exploration
Set TLS to 4-way
Outline

- Introduction
- Motivation
- Proposal
- Evaluation Methodology
- Results
- Conclusions
Evaluation Methodology

- SESC simulator - extended to model our scheme

Architecture:
- Core:
  - 4-issue OoO superscalar, 96-entry ROB, 3GHz
  - 32KB, 4-way, DL1 $ - 32KB, 2-way, IL1 $
  - 16Kbit Hybrid Branch Predictor
- Tile/System:
  - 128 cores partitioned in 2-way or 4-way tiles (evaluate both)
  - Shared L2 cache, 8MB, 8-way, 64MSHRs
  - Directory: Full-bit vector sharer list
  - Interconnect: Grid, 64B links - 48GB/s to main memory
Evaluation Methodology

- **Benchmarks:**
  - 12 workloads from PARSEC 2.1, SPLASH2, NASPB
  - Simulate parallel region to completion

- **Compilation:**
  - MIPS binaries generated using GCC 3.4.4
  - Speculation added automatically through source-to-source compiler
  - Selection of speculation regions through manual profiling

- **Power:**
  - CACTI 4.2 and Wattch
Evaluation Methodology

- Alternative schemes compared against:
  - Core Fusion [Ipek ISCA’07]:
    - Dynamic combination of cores to deal with lowly-threaded apps
    - Approximated through wide 8-issue cores with all the core resources doubled without latency increase => upper bound
  - Frequency Boost:
    - Inspired by Turbo Boost [Intel’08]
    - For each idle core one other core gains a frequency boost of 800MHz with a 200mV increase in voltage (same power cap)

- All these schemes shift resources to a subset of cores in order to improve performance
Outline

- Introduction
- Motivation
- Proposal
- Evaluation Methodology
- Results
- Conclusions
Bottom Line

- Speedup over best scalability point

![Speedup Chart](chart.png)
Bottom Line

- Speedup over best scalability point

```
Benchmark  TLS-2  TLS-4  CFusion  FBoost
bodytrack  0.8    1.0    1.2    1.4    1.6    1.8    2.0
streamcluster  41% avg
```

TLS-4: 41% avg
TLS-2: 27% avg
Energy

- Showing best performing point for each scheme

![Normalized Energy Graph]

- Benchmark: bodytrack, canneal, streamcluster, swaptions, cholesky, ocean-ncp, radiosity, water-nsquared, ep, ft, ls, sp, average

- Color-coded comparisons: 2TLS, 4TLS, CFusion, FBoost

International Symposium on Microarchitecture - December 2011
Energy

- Showing best performing point for each scheme

Energy consumption slightly *lower* on avg
Energy

- Showing best performing point for each scheme
Energy

- Showing best performing point for each scheme

- Spending less time in busy synchronization
Energy

- Showing best performing point for each scheme

![Bar chart showing normalized energy for different benchmarks and schemes.](chart.png)
Energy

- Showing best performing point for each scheme

High mispeculation: Higher energy

Normalized Energy

Benchmark

- bodytrack
- canneal
- streamcluster
- swaptions
- cholesky
- ocean-ncp
- radiosity
- water-nsquared
- ep
- ft
- is
- sp
- average

2TLS
4TLS
CFusion
FBoost

Normalized Energy
Energy

- Showing best performing point for each scheme
**Energy**

- Showing best performing point for each scheme

Little synchronization: Higher energy
Serial/Critical Sections

- is NASPB
Load Imbalance

- radiosity
  SPLASH2

![Graph showing Load Imbalance](image)

- Graph 1: Speedup vs. Cores for various techniques.
  - base
  - TLS-2
  - TLS-4
  - FBoost
  - CFusion

- Graph 2: Norm. Execution Time vs. Cores for different activities.
  - Busy
  - Lock
  - Barrier
Synchronization Heavy

- ocean
- SPLASH2
Coarse-Grain Partitioning

- swaptions
- PARSEC
Poor Static Partitioning

- sp
- NASPB
Effect of Dataset size

- Unchanged behavior: **cholesky**
- Also: canneal, ocean, ft, is, sp

![Graph showing the effect of dataset size on speedup for different core counts and configurations. The graph includes lines for base, baseL, TLS-2, TLS-2L, TLS-4, and TLS-4L, highlighting the speedup across different core counts.](image)
Effect of Dataset size

- Improved scalability, but TLS boost remains: swaptions
- Also: bodytrack, radiosity, ep
Effect of Dataset size

- Improved scalability, lessened TLS boost: streamcluster
Effect of Dataset size

- Worse scalability, even better TLS boost: \textit{water}
Outline

- Introduction
- Motivation
- Proposal
- Evaluation Methodology
- Results
- Conclusions
Conclusions

- Multicores and many-cores are here to stay
  - Parallel programming essential to exploit new hardware
  - Some coarse-grain parallel programs do not scale
  - Enough nested parallelism to improve scalability
- Proposed speculative parallelization through implicit speculative threads on top of explicit threads:
  - Significant scalability improvement of 40% on avg
  - No increase in total energy consumptions
  - Presented an auto-tuning mechanism to dynamically choose the number of threads that performs within 6% of the oracle
Complementing User-Level Coarse-Grain Parallelism with Implicit Speculative Parallelism

Nikolas Ioannou, Marcelo Cintra

School of Informatics
University of Edinburgh
Related Work

- [von Praun PPoPP’07] Implicit ordered transactions
- [Kim Micro’10] Speculative Parallel-stage Decoupled Software Pipelining
- [Ooi ICS’01] Multiplex
- [Madriles ISCA’09] Anaphase
- [Rajwar MICRO’01],[Martinez ASPLOS’02] Speculative Lock Elision
- [Moravan ASPLOS’06], etc., Nested transactional memory
Bibliography

- [Ipek ISCA’07] Ipek et al. Core fusion: Accommodating software diversity in chip multiprocessors
- [Kim Micro’10] Scalable speculative parallelization in commodity clusters, MICRO, 2010
Bibliography

- [Moravan ASPLOS’06] Supporting nested transactional memory in logtm. ASPLOS 2006
- [Curtis-Maury PACT’08] Prediction models for multi-dimensional power-performance optimization on many-cores.
## Benchmark details

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Input Sizes</th>
<th>Coverage of Speculative Regions(^a)</th>
<th>Types of Speculation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Normal</td>
<td>Large</td>
<td></td>
</tr>
<tr>
<td><strong>PARSEC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td>Computer Vision</td>
<td>sequenceB(_1)</td>
<td>sequenceB(_2)</td>
<td>59%</td>
</tr>
<tr>
<td>canneal</td>
<td>Chip Design</td>
<td>100000.nets</td>
<td>200000.nets</td>
<td>99%</td>
</tr>
<tr>
<td>streamcluster</td>
<td>Data Mining</td>
<td>4K</td>
<td>8K</td>
<td>92%</td>
</tr>
<tr>
<td>swaptions</td>
<td>Financial Analysis</td>
<td>16</td>
<td>32</td>
<td>80%</td>
</tr>
<tr>
<td><strong>SPLASH2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cholefsky</td>
<td>Sparse Matrix Multiplication</td>
<td>tk15</td>
<td>tk29</td>
<td>81%</td>
</tr>
<tr>
<td>ocean</td>
<td>Ocean Current Simulation</td>
<td>130</td>
<td>258</td>
<td>87%</td>
</tr>
<tr>
<td>radiosity</td>
<td>Graphics Rendering</td>
<td>test</td>
<td>room</td>
<td>69%</td>
</tr>
<tr>
<td>water</td>
<td>Molecular Dynamics</td>
<td>512</td>
<td>1000</td>
<td>99%</td>
</tr>
<tr>
<td><strong>NAS OpenMP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ep</td>
<td>Random Number Generator</td>
<td>1M</td>
<td>4M</td>
<td>100%</td>
</tr>
<tr>
<td>ft</td>
<td>3D FFT PDE</td>
<td>128K</td>
<td>512K</td>
<td>42%</td>
</tr>
<tr>
<td>is</td>
<td>Integer Sort</td>
<td>65K</td>
<td>1M</td>
<td>4%</td>
</tr>
<tr>
<td>sp</td>
<td>3D Fluid Dynamics</td>
<td>36</td>
<td>64</td>
<td>88%</td>
</tr>
</tbody>
</table>
Fetched Instructions

Benchmark

TLS-2
TLS-4
FBoost
CFusion

bodytrack
canneal
streamcluster
swaptions
cholesky
ocean-ncp
radiosity
water-nsquared
ep
ft
is
sp
average

Norm. Total Fetched Ins.
Failed Speculation

Norm. Execution Time

Restart
Busy

TLS-2
TLS-4
bodytrack
canneal
streamcluster
swaptions
cholesky
ocean-ncp
radiosity
water-nsquared
ep
ft
is
sp

Benchmark