Physical Experimentation with Prefetching Helper Threads on Intel’s Hyper-Threaded Processors

Dongkeun Kim, Steve Shih-wei Liao, Perry H. Wang, Juan del Cuvillo
Xinmin Tian, Xiang Zou, Hong Wang, Donald Yeung, Milind Girkar
and John P. Shen

Microarchitecture Research Lab
Intel Compiler Labs, Desktop Platforms Group
Intel Corporation

Department of ECE
Institute for Advanced Computer Studies
University of Maryland at College Park
Prefetching Helper Threads

Main Thread

time
Prefetching Helper Threads

- Helper Thread
- Main Thread

Run simultaneously
Execute slices

time
Prefetching Helper Threads

- Run simultaneously
- Execute slices
- Target last-level cache misses
- Trigger cache misses early
Prefetching Helper Threads

.Helper Thread
.Main Thread
.Cache
.Miss
.Cache Hit!

✿ Affect only performance not correctness
✿ How to generate effective helper threads?
Previous Studies

- Moshovos, ICS '01
- Collins, MICRO '01
- Roth, HPCA '01
- Zilles, ISCA '01
- Liao, PLDI '02
- Luk, ISCA '01
- Kim, ASPLOS '02

Simulation-based evaluation

CGO'04
March 22, 2004
Motivations

- Do helper threads really work on real silicon?
- Wall-clock speedup in real physical system
- What are the obstacles to speedup?
  - Resource contention in hyper-threaded processor
  - Large thread synchronization overhead
- How to address the problems?
- What is the potential of helper threads?
Outline

- Background and Motivation
- Physical System Configuration
- Dynamic Performance Monitoring
- Synchronization of Helper Threads
- Experimental Evaluation
- Conclusion
Research Compiler Framework

C++
front-end

FORTRAN90
front-end

Profiler

Interprocedural
analysis & optimizations

AutoHelper

Global scalar optimizations

IA-32 back-end

Helper thread optimization module in Intel compiler infrastructure
Research Compiler Framework

- C++ front-end
- FORTRAN90 front-end
- Profiler
- Interprocedural analysis & optimizations
- AutoHelper
- Global scalar optimizations
- IA-32 back-end

Helper thread optimization module in Intel compiler infrastructure

- Delinquent load identification
- Loop selection, Slicing, Live-in variable analysis
- Helper thread code generation
**System Configuration**

**Processor**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2.66GHz Intel Pentium 4 with HT technology</td>
</tr>
<tr>
<td>L1 trace cache</td>
<td>12K uops, 8-way, 6 uops per line</td>
</tr>
<tr>
<td>L1 data cache</td>
<td>16KB, 4-way, 64B line, 4-cycle latency</td>
</tr>
<tr>
<td>L2 unified cache</td>
<td>512KB, 8-way, 64B line, 7-cycle latency</td>
</tr>
<tr>
<td>Reorder buffer</td>
<td>128</td>
</tr>
<tr>
<td>Load buffer</td>
<td>48</td>
</tr>
</tbody>
</table>

**Operating System**

- Windows XP Professional

**Benchmark**

- MCF, BZIP2, ART (SPEC CPU2000)
- MST, EM3D (Olden)
Resource Contention

- HW resource management in hyper-threaded processor

<table>
<thead>
<tr>
<th>Shared</th>
<th>Trace cache, L1 D-cache, L2 cache, Execution units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Microcode ROM, Instruction fetch logic, Allocator, DTLB</td>
</tr>
<tr>
<td></td>
<td>IA-32 instruction decode, Global history array</td>
</tr>
<tr>
<td></td>
<td>Instruction scheduler, Uop retirement logic</td>
</tr>
</tbody>
</table>

| Partitioned                    | Reorder buffer, Memory instruction queue             |
|                                | General instruction queue, Load/Store buffers, Uop queue |

- HW resource contention
  - Unhelpful helper threads potentially degrade performance
  - Dynamic mode transition between ST and MT modes
Dynamic Program Behavior

L2 cache behavior for entire program execution

Do not invoke helper thread if no cache miss
Dynamic Performance Monitoring

- Monitor dynamic program behavior
  - Fine-grain chronology
  - Low overhead

EmonLite

- User-level library routines
- Monitor microarchitectural events
  - e.g. cycles, cache misses
EmonLite Code Instrumentation

```
main(int argc, char *argv[])
{
......
emonlite_begin();
......
}
while (arcin) {
    /* emonlite_sample() */
    if (!((num_iter++ % SAMPLE_PERIOD))) {
        current_val = readpmc(16);
        L2miss[num_sample++] = current_val - prev_val;
        prev_val = current_val;
    }
    tail = arcin->tail;
    if (tail->time + arcin->org_cost > latest) {
        arcin = (arc_t *)tail->mark;
        continue;
    }
    ......
}
```

✔️ Adjustable profiling interval & overhead
✔️ Prototyped in Intel research compiler infrastructure
Two helper threading scenarios

To invoke and synchronize helper threads

while (arcin) {

    tail = arcin->tail;
    if (tail->time + arcin->org_cost > latest) {
        arcin = (arc_t *)tail->mark;
        continue;
    }

    ...
}

CGO’04
March 22, 2004

MRL-SSG-DPG, Intel
Univ. of Maryland
Helper Threading Scenarios

Two helper threading scenarios

To invoke and synchronize helper threads

```c
helper_invoke();
while (arcin) {
  tail = arcin->tail;
  if (tail->time + arcin->org_cost > latest) {
    arcin = (arc_t *)tail->mark;
    continue;
  }
  ...
}
```

+ Low thread synchronization overhead
- Lack of synchronization
Helper Threading Scenarios

Two helper threading scenarios
To invoke and synchronize helper threads

while (arcin) {
    if(!(num_iter++ % SAMPLE_PERIOD))
        helper_invoke();
    tail = arcin->tail;
    if (tail->time + arcin->org_cost > latest) {
        arcin = (arc_t *)tail->mark;
        continue;
    }
    ...
}

+ Avoid run-away helper thread
- Thread synchronization and execution overhead
Thread Synchronization Cost

Two mechanisms to invoke & suspend threads

1) Win32 API: `SetEvent()` & `WaitForSingleObject()`
   - Jump to OS scheduler
   - Non-deterministic transition time: 10K~30K cycles

2) Light-weight hardware mechanism
   - Deterministic transition time: ~1,500 cycles
Do Helper Threads Really Help?

- Wall-clock speedup on real silicon
- More speedup with light-weight HW mechanism
- Synchronization is important

CGO’04
March 22, 2004
Large cache miss reduction by helper threads
More headroom for higher miss coverage
Dynamic Effect of Helper Threads

Cache miss reduction = speedup

BZIP2

Sample ID

# L2 Cache Misses

Cycles (x1000)

Baseline

w/o Helper Thread

w/ Helper Thread
Dynamic Effect of Helper Threads

- Cache miss reduction?
- Speedup
- Need for dynamic throttling mechanism

![Graph showing the dynamic effect of helper threads.](image-url)
Potential of Dynamic Throttling

- Additional speedup with dynamic throttling
- Multiple factors determine throttling mechanism
Contributions

- First work experimenting with helper threads on physical system with hyper-threaded processor
- Compiler, processor, OS, and real workloads
- Helper thread should adapt to dynamic behavior
- Dynamic performance monitoring
- Light-weight thread synchronization is crucial
Conclusion

- Wall-clock speedup on real silicon
- Judicious invocation of helper threads
- More optimized helper thread construction
- Lighter-weight synchronization mechanism
- Much potential for dynamic throttling
- Develop practical dynamic throttling framework