An Infrastructure for Efficient Synchronization of Asymmetric Threads on Hyperthreaded Processors

Nikos Anastopoulos, Nectarios Koziris

(ananost, nkoziris)@csibl.ece.ntua.gr

Computing Systems Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens

Motivation

The privileged instructions MONITOR/MWAIT introduced with Intel Prescott core, offer a “performance-optimized” way for threads involved in synchronization events to wait on a condition. So far, they have been used mostly for inter-thread synchronization in operating systems code. In this work, we explore the potential of using these instructions for synchronizing application threads on hyper-threaded processors which are characterized by workload imbalance.

Implementing basic primitives with MONITOR/MWAIT

Where to allocate the region to be monitored?

- in user-space: fast notification, but requires copying the contents of monitored region to kernel-space on each condition check
- in kernel-space: requires additional system call to enable update of monitored memory from user-space
- in kernel-space, but map it to user-space for direct access

How should synchronization be implemented?

- low resource consumption (idle periods may dominate execution time of helper)
- worker: fast notification of helper (low call overhead)
- helper: fast resumption (high responsiveness)

Options for synchronization

- spin-wait loops: they provide high responsiveness, but consume significant resources. Even if we loosen the spinning of helper using PAUSE, the worker still experiences notable interference (15-20% on the spinning of helper using PAUSE, the worker still suspensionlatency.
-不可能-20% on the spinning thread halts, the worker halts as well.
- Possible directions of our work:
  - “low-resource” vs. “high-resource” synchronization
  - “spin-loops” vs. “MWAIT”

Experimental evaluation

System configuration:

- Intel Xeon@2.8GHz, 2 hyper-threads
- Linux 2.6.13, gcc-4.12, glibc-2.5

Case 1: Barriers – raw performance

<table>
<thead>
<tr>
<th>Monitor/Mwait</th>
<th>User-space</th>
<th>Kernel-space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (seconds)</td>
<td>Total (cycles)</td>
<td>Total (cycles)</td>
</tr>
<tr>
<td>MONITOR/MWAIT</td>
<td>0.286</td>
<td>0.286</td>
</tr>
<tr>
<td>spin-loops</td>
<td>0.295</td>
<td>0.295</td>
</tr>
<tr>
<td>MWAIT</td>
<td>0.275</td>
<td>0.275</td>
</tr>
</tbody>
</table>

Conclusions – Future work

MONITOR/MWAIT-based primitives make the best compromise between low resource waste and low call and wakeup latency for our considered model.

Possible directions of our work:

- “monitor”-like hierarchical schemes in multi-SMT systems (e.g. tree barriers)
- other “producer-consumer” models (disk/network I/O applications, MPI programs, etc.)
- multithreaded applications with irregular parallelism

References

- Facilitating Efficient Synchronization of Asymmetric Threads on Hyper-threaded Processors. N. Anastopoulos and N. Koziris. In 3rd Workshop on Multithreaded Architectures and Applications (MTAAP 2008), Miami, FL.