PREDICTIVE ACCELERATION OF CRITICAL SECTION EXECUTION WITH ASYMMETRIC MULTICORE ARCHITECTURE

Padmanabham Patki  Paul Kennedy  Yifei Yao
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, USA

INTRODUCTION

We extend this idea and propose to predict the critical region well in advance so as to accelerate the thread reaching the critical region earliest. This would reduce the contention between different threads contending for the same critical section even further by increasing the temporal gap between the occurrence of critical sections in different threads, thus improving performance.

MECHANISM

Part I: The Input: Flagging the critical section:

a. The compiler identifies instructions triggering the beginning of critical section execution.
b. The compiler then analyses all possible paths that may lead to the critical section. On these paths, it inserts instructions (“flags”) indicating that the thread is about to enter a critical region and also inserts a rough estimate (based on profiling) as to how many instructions later the critical section would be reached.
c. Further, the compiler analyses the paths and control flow that might cause it to skip the critical region despite having executed the flagging instructions. On these paths, it inserts instructions that indicate that the previously predicted critical region will no longer be reached.

In the current model, the role of the compiler to insert the prediction and release flags is performed manually

Part II: The Scheduler: Dynamic prioritization of threads being scheduled on the faster core depending on the thread and system state.

The threads are scheduled on the faster core in the following order for an aggressive predictive accelerating mechanism:

i. Thread holding lock having most number of threads waiting upon it.
ii. Thread holding lock having active predictions in most threads.
iii. Thread holding a lock
iv. Threads being closest to highly contended critical sections
v. Threads predicting a critical section
vi. Threads performing normal work.

Part III: Dynamic Prediction Improvement: Adapting to run-time environment.

a. The system maintains a prediction table and updates the predicted distances based on the past history and current number of instructions executed between the prediction flag and the start of critical section.
b. The system also maintains a confidence flag. If a flag mispredicts a critical section, then the confidence of the particular flag is decreased. When below threshold, the predicted state of the thread for the corresponding low confidence predictor is not considered while scheduling the thread on the faster core.

c. Although, performance of Aggressive ACS (AACS) is comparable to Aggressive PACS (APACS), APACS is expected to be more immune to migration cost penalties than AACS.

RESULTS

<table>
<thead>
<tr>
<th>Environment</th>
<th>Application</th>
<th>Critical region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>OLTP</td>
<td>Speed - up on Faster Core</td>
</tr>
<tr>
<td>Global</td>
<td>complex</td>
<td>Speed - up on Faster Core</td>
</tr>
<tr>
<td>Global</td>
<td>basic</td>
<td>Speed - up on Faster Core</td>
</tr>
</tbody>
</table>

CONCLUSION

a. The results support the proposition that predicting the critical sections and accelerating the thread in advance would help reduce the contention between the critical sections and improve performance.
b. In some cases, the wait-cycles increase upon prediction, which may be attributed to either pattern of critical section occurrence or mispredictions.

c. Although, performance of Aggressive ACS (AACS) is comparable to Aggressive PACS (APACS), APACS is expected to be more immune to migration cost penalties than AACS.

FUTURE WORK

Future work could comprise of looking into the following aspects of the Predictive Acceleration of Critical Sections:

a. Impact on Power Consumption due to higher utilization of faster core
b. Cost of thread migration and its impact on performance
c. Design a compiler to insert the prediction and release flags.

ACKNOWLEDGEMENTS

We would like to thank Prof. Onur Mutlu and the TAs, Yoongu Kim and Justin Meza, for their critical suggestions and help. Their constant motivation has been the driving factor which enabled us to complete the project.