**HelperCore\textsubscript{DB}: Exploiting Multicore Technology for Databases**

Kostas Papadopoulos, Kyriakos Stavrou and Pedro Trancoso
Department of Computer Science, University of Cyprus, Cyprus
Email: \{csp5kp1,tsik,pedro\}@cs.ucy.ac.cy
http://www.cs.ucy.ac.cy/carch/casper

The current trend in microprocessor design is to keep the design simple while packing more processors on the same die in order to increase the performance. These chips are called Chip Multiprocessors (CMP) or Multicores. While currently the parallelism provided by the multiple cores is used to achieve higher throughput, it is possible to split a single application into multiple tasks and execute them in parallel.

We identify two scenarios where the use of the multicore technology will not result in the expected benefit for the database workloads. The first scenario is for the case where we are able to parallelize the Database Management System in order to issue queries in parallel. Due to the synchronization required by the different query executions in order to update database metadata structures and keep shared structures consistent, the parallelization will not be beneficial when the number of execution cores increases beyond a certain point. Therefore, we predict that for large scale multicore chips that will be available in the future, the benefit from adding extra cores will become diminishing. The second scenario is where simpler Database Management Systems are used such as the publicly available PostgreSQL and MySQL. Such systems do not easily support query parallelism. As such, the use of multicores is limited to exploiting them for higher throughput.

In this work we propose to exploit the cores of a multicore architecture in an alternative way. Instead of executing application code on all cores, some cores execute code that will indirectly result in a performance benefit for the application. We call these the Helper Cores and for this work we focus on the implementation of HelperCore\textsubscript{DB}, a Helper Core for improving the performance of database workloads. The proposed implementation focuses on solving one of the most serious problems in memory-resident database applications, the memory latency penalty, by performing efficient data prefetching. As the prefetching is done by the Helper Core and the data is used in a different core, the only hardware requirement is that the helper and application cores share some level of the memory hierarchy.

HelperCore\textsubscript{DB} requires minimal changes in the original Database Management System code, it is effective even when on top of an architecture with hardware prefetching as it covers a wider range of memory access patterns, it is a full-software solution therefore executing on any commodity multicore processor, and it efficiently utilizes the available cores to improve the application’s performance.

Database systems organize data in blocks. A block is a construct of a specified fixed size that generally holds tuples of data and control information. Blocks reside on permanent storage as pages and in main memory as buffers. In the database applications, whenever a buffer is accessed most of the data on that buffer is likely to be used at that time. Taking advantage of this fact, we designed HelperCore\textsubscript{DB} to prefetch at the buffer granularity. Consequently, the worker thread (the thread executing the database application) needs to communicate to the HelperCore\textsubscript{DB} only the next block to be processed by the query plan. The synchronization between the HelperCore\textsubscript{DB} and worker threads is implemented with a busy-wait loop, which is efficient as the Helper Core code executes on a separate core, therefore not consuming resources that could be used for the application.

The methodology for implementing the proposed technique is general enough to be applied to any database system. First the database system is profiled in order to identify the functions that cause the most L2 misses. Tracing the execution leads to the function where the first reference to a block is obtained. The block is addressed by a simple identifier. By assigning this address to globally accessible variables, we complete the modifications of the database system. The next step is to create the HelperCore\textsubscript{DB} code, which is a simple loop that waits for a new block address and fetches the data for future use by the worker thread.

To validate our proposal, we have made an implementation for the HelperCore\textsubscript{DB} code for a commodity Database Management System (PostgreSQL) executing on a commodity dual core processor with shared L2 (Intel Core 2 Duo). The evaluation was performed using the standard TPC-H benchmark. The experimental results showed a significant improvement in the execution time (up to 20%) therefore justifying the proposed technique.