Query Co-Processing on Commodity Processors

Anastassia Ailamaki Naga K. Govindaraju Stavros Harizopoulos Dinesh Manocha
natassa@cmu.edu, {naga,dm}@cs.unc.edu, stavros@csail.mit.edu

The rapid increase in the data volumes for the past few decades has intensified the need for high processing power for database and data mining applications. Researchers have actively sought to design and develop new architectures for improving the performance. Recent research shows that the performance can be significantly improved using either (a) effective utilization of architectural features and memory hierarchies used by the conventional processors, or (b) the high computational power and memory bandwidth in commodity hardware such as network processing units (NPUs), Cell processors and graphics processing units (GPUs). This tutorial will survey the micro-architectural and architectural differences across these processors with data management in mind, and will present previous work and future opportunities for expanding query processing algorithms to other hardware than general-purpose processors. In addition to the database community, we intend to increase awareness in the computer architecture scene about opportunities to construct heterogeneous chips.

Conventional Processors: Traditionally, database and data mining systems have been optimized for conventional processors such as CPUs. In the recent years, due to the faster I/O storage systems, the performance bottleneck of many of these data- and computation-intensive applications is shifting to the memory hierarchy used by the conventional processors. Furthermore, due to the increasing gap between the processor and memory speeds, analysis of memory and processor behaviors has become important. In this tutorial, we will briefly survey the computer architecture and database literature on evaluating database application performance on conventional processors. We will describe strategies to reduce memory and resource stalls using data parallel algorithms, cache-coherent data structures, instruction buffering algorithms, and better storage models.

Network co-Processors: Unlike most scientific applications, database operations exhibit sequences of dependent memory accesses, which limit the opportunity for speculation and out- of-order execution to issue parallel memory accesses in a single-threaded micro-architecture. Instead of focusing on instruction-level parallelism in a single thread, recent proposals show that thread-level parallelism can significantly improve database performance by increasing the aggregate number of pending cache misses. Indeed, several database operations boast reuse and therefore use efficiently the caches, whereas others (such as file scan or hash join) would rather benefit from more threads and fewer cache levels. However, the overhead of supporting multiple contexts on an aggressive micro-architecture limits the expansion of conventional multithreaded architectures beyond four or eight threads. Network processors, which are used to handle network traffic and routing, are actually well-suited for executing many relational operators. In this tutorial we will present the tradeoffs and will survey results from studies that show that mapping certain common database operators to the network processor architectures can prove extremely rewarding.

Graphics and Cell Processors: High performance graphics processors (GPUs) are as ubiquitous as CPUs. They are now a part of every personal computer, console and even cell phones. GPUs can perform 10× higher operations per second and have 10× more memory bandwidth than CPUs – therefore, they can be used to accelerate many traditional algorithms by an order of magnitude as compared to CPU-based implementations. Moreover, GPUs are becoming increasingly programmable, and their computational power is increasing at a rate faster than the Moore's law for CPUs. We describe how many of the essential computational components for database and data mining algorithms such as relational database operations, stream data mining, linear algebra and sorting operations can be efficiently implemented on the GPUs and cell processors. In many cases, they outperform the fastest CPU-based implementations.

Proceedings of the 32nd VLDB Conference, Seoul, Korea, 2006