Database Techniques for Multi Cores and Big Memory

Xiongpai Qin
Renmin University of China, China

Biao Qin
Renmin University of China, China

Cuiping Li
Renmin University of China, China

Hong Chen
Renmin University of China, China

Xiaoyong Du
Renmin University of China, China

Shan Wang
Renmin University of China, China

INTRODUCTION

In recent years, computer hardware undergoes a big progress (Ailamaki, 2004). CPU (Central Processing Unit) is moving from multi cores to many cores. The number of cores in a GPGPU (General Purpose Graphics Processing Unit) is even larger. The parallelism of GPU is exploited not only for computing intensive tasks, but also for data intensive tasks. The capacity of main memory is increasing, and the price is decreasing dramatically. Database software should be carefully designed to utilize the advantages of new hardware, and evade hardware's shortcomings. The chapter introduces some efforts of database community in this aspect.

BACKROUND

This section gives a brief introduction to some new hardware technologies that database systems could leverage.

Multi Core CPU

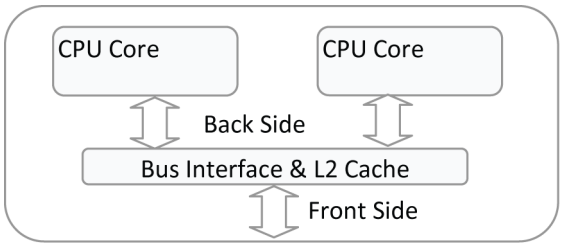
Improving the performance of CPU through increasing its clock frequency becomes more and more difficult. Researchers and engineers are seeking new ways to improve the performance of CPU, and they bring forth multi core technology.

In a typical multi core CPU, 2, 4, 8 or more cores are integrated on a chip. The cores have their own private caches (L (Level) 1 Cache), and share some larger but slower caches (L2 Cache). These cores access the shared main memory for parallel data processing. Putting several cores on a die allows for higher communication speeds between the cores, which will benefit many computing tasks.

The amount of performance gained by using multi core CPUs depends on the problem to be solved and the algorithms used. Many applications such as database systems are basically multi-threaded, and they can benefit from multi core CPUs at once without any modification to exist software. However, for fully utilizing the cores to boost software performance, there is still much work to do.

DOI: 10.4018/978-1-4666-5202-6.ch062

Figure 1. Diagram of a generic dual core CPU (Wikipedia-a, 2013)



GPGPU

GPU is traditionally used to accelerate the specific task of graphic rendering. GPU venders have integrated many computing units in a single die, and optimized the bandwidth for processing large volume of graphic data.

The huge computation power of GPU is exploited to perform other tasks as well, and GPU has become GPGPU (General Purpose GPU). GPU vendors have recognized the value of that. NVIDIA, one of major GPU manufacturers, has provided CUDA (Compute Unified Device Architecture), a SDK for easy programming of GPU for general tasks. Since GPU is designed primarily for graphic processing instead of general tasks, the architecture of a GPU is rather different from CPU. Taking NVIDIA CUDA as an example, it has its own unique thread hierarchy and memory hierarchy.

The thread hierarchy consists of four levels. (a) *Grid* is the first level of thread hierarchy, which is a group of one or more blocks. A grid is created for each CUDA kernel function. (b) The next level of thread hierarchy is *Block*, which is a user defined group of 1 to 512 threads. Each block is identified by a blockIdx. (c) The third level of thread hierarchy is *Warp* - scheduling unit of up to 32 threads. (d) Final level of the hierarchy is *Thread*, which is distributed by the CUDA runtime and identified by a threadIdx.

The CUDA platform has three primary memory types in the memory hierarchy. (a) *Local Memory* is per-thread memory for local variables and register spilling. (b) *Shared Memory* is per-block low latency memory to allow for intra-block data sharing and synchronization. (c) *Global Memory* is the device level memory that may be shared between blocks or grids. CUDA has also constant cache and texture cache for fast access of some specific data. The caches are read only and have faster access than shared memory. In CUDA data can be copied from one memory type to another, and to and from main memory, for GPU to process the data.

Big Memory Capacity

The price of memory is going down, and engineers can install more memory in a single machine now. It is rather common that a single server possesses a memory capacity as large as 64GB, 128GB, 512GB, and up to terabytes. For small to medium-size applications, it is possible to load the whole dataset into memory for fast access. In that case, memory becomes traditional hard disk, and the CPU cache becomes traditional memory. The optimization focus has shifted to how to efficiently exchange data between main memory and CPU cache for faster processing.

Authors of (Ailamaki, DeWitt, Hill, & Wood, 1999) have suggested some methods to improve database system from this aspect, including (a) optimizing data layout for last level cache, but not L1 cache to reduce data cache miss, (b) optimizing instruction layout to reduce first level instruction cache stalls, (c) other measures such as branching elimination should also be taken to decrease overall execution time.

MAIN FOCUS OF THE CHAPTER

This section introduce various database techniques for multi core CPU, GPU, and big memory, including storage models, compression techniques, index schemes, query processing techniques, concurrency control methods, and recovery techniques.