Accelerating SQL Database Operations on a GPU with CUDA

Peter Bakkum and Kevin Skadron
Department of Computer Science
University of Virginia, Charlottesville, VA 22904
{pbb7c, skadron}@virginia.edu

ABSTRACT

Prior work has shown dramatic acceleration for various database operations on GPUs, but only using primitives that are not part of conventional database languages such as SQL. This paper implements a subset of the SQLite command processor directly on the GPU. This dramatically reduces the effort required to achieve GPU acceleration by avoiding the need for database programmers to use new programming languages such as CUDA or modify their programs to use non-SQL libraries.

This paper focuses on accelerating SELECT queries and describes the considerations in an efficient GPU implementation of the SQLite command processor. Results on an NVIDIA Tesla C1060 achieve speedups of 20-70X depending on the size of the result set.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel Programming; H.2.4 [**Database Management**]: Parallel Databases

Keywords

GPGPU, CUDA, Databases, SQL

1. INTRODUCTION

GPUs, known colloquially as video cards, are the means by which computers render graphical information on a screen. The modern GPU's parallel architecture gives it very high throughput on certain problems, and its near universal use in desktop computers means that it is a cheap and ubiquitous source of processing power. There is a growing interest in applying this power to more general non-graphical problems through frameworks such as NVIDIA's CUDA, an application programming interface developed to give programmers a simple and standard way to execute general purpose logic on NVIDIA GPUs. Programmers often use CUDA and similar interfaces to accelerate computationally intensive data processing operations, often executing them fifty times faster on the GPU [2]. Many of these operations have direct parallels to classic database queries [4, 9].

The GPU's complex architecture makes it difficult for unfamiliar programmers to fully exploit. A productive CUDA programmer must have an understanding of six different memory spaces, a model of how CUDA threads and threadblocks are mapped to GPU hardware, an understanding of CUDA interthread communication, etc. CUDA has brought GPU development closer to the mainstream but programmers must still write a low-level CUDA kernel for each data processing operation they perform on the GPU, a time-intensive task that frequently duplicates work.

SQL is an industry-standard generic declarative language used to manipulate and query databases. Capable of performing very complex joins and aggregations of data sets, SQL is used as the bridge between procedural programs and structured tables of data. An acceleration of SQL queries would enable programmers to increase the speed of their data processing operations with little or no change to their source code. Despite the demand for GPU program acceleration, no implementation of SQL is capable of automatically accessing a GPU, even though SQL queries have been closely emulated on the GPU to prove the parallel architecture's adaptability to such execution patterns [5, 6, 9].

There exist limitations to current GPU technology that affect the potential users of such a GPU SQL implementation. The two most relevant technical limitations are the GPU memory size and the host to GPU device memory transfer time. Though future graphics cards will almost certainly have greater memory, current NVIDIA cards have a maximum of 4 gigabytes, a fraction of the size of many databases. Transferring memory blocks between the CPU and the GPU remains costly. Consequently, staging data rows to the GPU and staging result rows back requires significant overhead. Despite these constraints, the actual query execution can be run concurrently over the GPU's highly parallel organization, thus outperforming CPU query execution.

There are a number of applications that fit into the domain of this project, despite the limitations described above. Many databases, such as those used for research, modify data infrequently and experience their heaviest loads during read queries. Another set of applications care much more about the latency of a particular query than strict adherence to presenting the latest data, an example being Internet search engines. Many queries over a large-size dataset only address a subset of the total data, thus inviting staging this subset into GPU memory. Additionally, though the finite memory size of the GPU is a significant limitation, allocat-

GPGPU-3 March 14, 2010, Pittsburg, PA, USA