Title

Abstract

I – Introduction and Background

1. GPU overview
   * GPU architecture, taken from a high level view, is composed of the GPU core and the off-chip memory
   * ordinarily, the GPU is dedicated to graphics rendering
   * graphics cards are constructed for massive parallelism, they can dwarf the calculation rate of even the most powerful CPUs for many parallel processing tasks
   * most modern GPUs use single instruction multiple data (SIMD) stream architecture -o
     + only one instruction can be run at any given point in time
     + single instruction is copied and ran across each core at the same time
   * fundamental advantage of SIMD is that data parallelism allows it to execute computations quickly (multiple processors doing the same thing) and efficiently (only one instruction unit).
   * running the same mathematical function over and over again at scale
   * core contains an array of Streaming Multiprocessors
   * each SM is composed of eight Scalar Processors, base computational cores of the GPU
   * Each SM executes following a model similar to SIMD which NVIDIA refers to as SIMT, or Single Instruction Multiple Threads
   * largest and slowest memory available in the GPU is the aforementioned off-chip memory, more commonly referred to as the “global” memory
   * No caching so resulting in these steep latency hits with every access
   * SM level there exists the “shared” memory. Access to shared memory is exclusive to thread block
2. Finance overview – data processing in finance (pricing, calculations,

II – Analysis of Problem

1. Complex financial calculations
   1. Overview of the types of problems / calculations
      1. rise in population and the number of financial products
      2. transactions per minute have also increased drastically
      3. store this data and its petabytes of structured and unstructured data to forecast client behavior and develop strategies
2. Quantitative trading
   1. demand has grown steadily for professionals who not only understand the complex mathematical models that price these securities, but who are able to enhance them to generate profits and reduce risk
      1. high demand for quants is driven by multiple trends
         1. The rapid growth of hedge funds and automated trading systems
         2. The increasing complexity of both liquid and illiquid securities
         3. The need to give traders, accountants and sales reps access to pricing and risk models
         4. The ongoing search for market-neutral investment strategies.
      2. Experts in:
         1. Calculus (including differential, integral and stochastic)
         2. Linear algebra and differential equations
         3. Probability and statistics
      3. Also
         1. Portfolio theory
         2. Equity and interest rate derivatives, including exotics
         3. Credit-risk products
3. Data Science
   1. supervised machine learning models
   2. sample tests on equity portfolios - typically tens of thousands of stocks
   3. incorporate unstructured data into their decision making; for instance Twitter data for market sentimen

IV – Summary of Papers

1. Efficiently Running SQL Queries on GPU
   1. potential of GPU's accelerating database systems by developing kernels for the SQL aggregation functions. Using the CUDA enabled GPU's as a coprocessor
   2. Background
      1. Database use CPUs because most of their workload is scheduling, giving access to memory buffers
      2. main drawback is heat, also they still provide complex mechanisms to increase the number of cores
      3. paper focuses on speeding up select queries with aggregation functions (SUM, MAX, MIN)
      4. developed efficient algorithms to execute database operations on the GPU, including predicates, Boolean combinations and aggregation queries.
      5. using a GPU for efficiently accelerating database operations by offloading SELECT queries with integer values on to the GPU
      6. costs of data transfer between the host and device memory
      7. translate high-level data-parallel operations into efficient GPU programs. The data-parallel operations are provided as a data type in a conventional imperative language, making them easily accessible to programmers.
      8. hybrid query processing to accelerate the execution of relational operators with GPU. The proposed approach automatically parses input queries splitting them into query parts as a way to manage the distribution to be executed either on the CPU, on the GPU, or on both
      9. MapD [8] is a GPU data analytical system that systematically exploits both multi-GPU architectures and multi-node architectures
   3. Implemantation
      1. propose a simple system where the GPU's kernel reads a. csv file with data from the database table that it is needed to speed up the simple aggregation functions in the SQL query
      2. architecture consists of a database implemented on the GPU and would use the GPU's memory for shorter data transfer latency and throughput
      3. Generated one million rows of random data -id and number
      4. Tested SUM, MAX, MIN functions
      5. using a function for basically reading every single row in the. csv and put in an array.
      6. Using cudaMemcpy function that array is migrated on the GPU using the cudaMemcpyDeviceToHost
      7. executed basic SQL aggregation functions in such a way that separate kernels are coded for every SQL aggregation function optimizing them with the CUDA reduce algorithm
      8. The kernels orient around the idea to get a chunk of data and put it in the shared memory to speed up the computation and synchronize the threads on every computation.
      9. A program was written to generate the block size and grid size, considering that every block taken by the streaming multiprocessor runs in warps of 32 threads
      10. limited the size of the blocks to the numbers divisible by 32 with the maximum size of 1024 (hardware limitation), which is the grid size multiplied with the block size
   4. Results – show images
      1. The fastest time for execution of the MAX aggregation function kernel was 3.306496 milliseconds with the thread combination of: gridDIm.x = 31250, gridDim.y = 1, blockDim.x = 4, blockDim.y = 8. MySQL's average performance was 251.6 milliseconds
      2. MIN the fastest time of 3.489888 milliseconds with a speedup of 78 compared to the MySQL's average performance of 273.5 milliseconds
      3. on an average of approximately 97 times,
2. Option Pricing on the GPU - we study the pricing of the American put lookback option, due to the exotic nature of this product and to study its suitability with GPU
   1. Options overview
      1. Option but not obligation to buy or sell something at a future time
      2. Call option gives the owner (seller) the right (obligation) to buy (sell) a specific number of shares of the underlying stock at a specific price (strike price) by a predetermined date
      3. Put options give the owner (seller) the right (obligation) to sell (buy) a specific number of shares of the underlying stock at a specific price by a specific date
      4. European - contract allows the holder to exercise the option only at the expiration time of the contract period
      5. American option - can be exercised any time prior to the expiration period
      6. Long vs short explanation
      7. pricing the options quickly is advantageous to the market players and, hence, advanced algorithms and computing platforms are always in demand for real time pricing
      8. Regular algorithms are algorithms that exhibit structured, predictable data access patterns
   2. CUDA - Compute Unified Device Architecture
      1. framework for developing parallel general purpose applications on a GPU.
      2. Using GTX 260 in paper
      3. Threading model
         1. At the highest level there exists the thread grid
         2. Groups of threads within the grid form thread blocks.
         3. CUDA threading model exposes a series of thread groupings
         4. Threads within blocks are assigned a thread ID unique only among threads in the same block.
         5. threads within each block are ordered into 32-thread warps.
         6. threads within a warp must execute the same instruction or no instruction at all.
   3. Binomial and Trinomial model
      1. The binomial lattice approximating the asset price movements that enables pricing of an option.
      2. essentially a binary tree with the root node existing at time zero, and the leaf nodes at the maturity date of the option.
      3. assumes that the price of the underlying asset follows some random walk through the lattice, from the root node to a leaf node.
      4. binomial/trinomial methods exhibit regular properties, suitable for GPU
         1. Each time step is computed synchronously, and the data required for the computations at each node in the lattice are well-defined and predictable.
   4. Implementation
      1. Trinomial Lattice
         1. implement two distinct GPU algorithms for the trinomial lattice method of option pricing
            1. Naïve - does not make use of shared memory to cache values that will be used by multiple threads during the kernel.
            2. Coalesced - uses shared memory and global memory coalescing in order to improve performance of the kernel
      2. Pricing of Lookback Options
         1. parallel algorithm - Trinomial GPU Kernel with Shared Memory Use
         2. Hybrid - After a certain threshold is met, calculations are performed on the CPU rather than the GPU
   5. Results – - pricing European options using trinomial lattice and American lookback options using binomial lattice; Both the GPU trinomial lattice and lookback option pricing applications were developed using CUDA and executed on a GTX 260 graphics card with a core clock speed of 576Mhz, a shader clock speed of 1242Mhz and a memory clock of 1000Mh
      1. trinomial lattice - simple implementation that allows for all threads to collaborate for a single option performs better when only a single option is to be priced
      2. Binomial - significant speedup over the sequential CPU implementation for large time step counts
      3. <https://ieeexplore.ieee.org/document/5581462/footnotes#footnotes-id-fn1>
      4. A picture containing text, line, screenshot, plot

         Description automatically generated
3. A High-Performance Multi-user Service System for Financial Analytics Based on Web Service and GPU Computation
   1. in finance, securities, such as stocks, funds, warrants and bonds, are actively traded in financial markets. there are millions of securities in the financial markets. Pricing them is very time consuming
   2. Abundance of market data and accurate estimation of the expected value of a security (also called pricing of a security
   3. develop a pricing and data/information service system for financial analytics of securities with the following goals:
      1. supporting fast pricing and data/information services for massive multiple users,
      2. saving cost in hardware equipments and reducing energy consumption, and
      3. easy maintenance of the system, easy expansion of a service component, and fast deployment of new services in the system
   4. Performance comparison between the PC/GPU-paired cluster and the CPU-based multi-core server cluster –
      1. vector processing capabilites of CPUs makes them especially well-suited to financial computing
   5. Proposal - Financial Analytics Service System based on Web Service and Service-Oriented Architecture
      1. Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications. Each service provides a business capability, and services can also communicate with each other across platforms and languages. Web services
         1. Interoperability - description documents that specify the functionality of the service
         2. Loose coupling - having as little dependency as possible on external resources such as data models or information systems.
         3. Abstraction - Clients or service users in SOA need not know the service's code logic or implementation details
         4. Granularity - appropriate size and scope, ideally packing one discrete business function per service

<https://ieeexplore.ieee.org/document/5634351>

A picture containing text, screenshot, diagram, design

Description automatically generated

* + 1. High-Performance Financial Analytics with GPUs
       1. Example: Convertible bond is a bond that the holders can convert it into shares of common stock at a pre-determined rate.
          1. Goldman Sachs model price evolution can be computed with a binomial tree
          2. find the value of the convertible bond, we first compute the convertible bond price at maturity (at time step n) and initialize the conversion probability.
          3. To obtain accurate pricing result, it requires at least 10,000 time steps; that is, 10,000 levels in the binomial tree.
          4. distribute the convertible bond to each CPU evenly, allocate one block to each bond, and use the total threads in each block to access data at each time step
          5. Simultaneous global memory access by threads of a half-warp size can be coalesced into a single memory transaction
          6. code achieves speedup factors of 511 for 256 bonds and 668 for 1000 bonds
    2. Web services
       1. built a convertible bond information system which can be used by front, middle, or back offices of financial institutions

III – Research – Proposed Solutions

GPU oriented architecture to address all of the problems:

CUDA parallel computing platform

1. Nvidia's popular Compute Unified Device Architecture (CUDA) parallel computing platform
2. API that enables developers to optimize how GPU resources are used -- without the need for specialized graphics programming knowledge
3. fundamental components of the hierarchy
   1. threads
      1. CUDA core -- is a parallel processor that computes floating point math calculations in an Nvidia GPU
      2. data processed by a GPU is processed via a CUDA core.
      3. typical GPU has over 4,000 processing cores and computation-heavy tasks can be designed to run in parallel on these processors
   2. thread blocks
      1. grouping of CUDA cores (threads) that can be executed together in series or parallel
      2. Thread blocks share memory on a per-block basis
      3. Every thread in a given CUDA block can access the same shared memory
   3. kernel grid
      1. groupings of thread blocks on the same kernel.
      2. Grids can be used to perform larger computations in parallel
4. CUDA memory hierarchy –
   1. Registers
      1. Registers are the memory that gets allocated to individual threads
      2. registers exist in “on-chip” memory and are dedicated to individual threads, the data stored in a register can be processed faster than any other data.
   2. Read-only memory
      1. on-chip memory on GPU; fetching data from read-only memory can be faster and more efficient than using global memory
   3. L1 Cache/shared memory
      1. on-chip memory that is shared within thread blocks (CUDA blocks)
      2. shared memory usage is controlled via software while L1 cache is controlled by hardware
   4. L2 Cache
      1. Layer 2 cache can be accessed by all threads in all CUDA blocks.
      2. L2 cache stores both global and local memory
   5. Global memory
      1. PU analogy, global memory is comparable to RAM.
      2. Fetching data from global memory is inherently slower than fetching it from L2 cache

<https://en.wikipedia.org/wiki/CUDA>

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

1. Complex financial calculations – ie Black Scholes
   1. Numba Cuda
      1. Overview
      2. Research article on loops / black scholes
         1. first widely used mathematical method to calculate the theoretical value of an option contract, using current stock prices, expected dividends, the option’s strike price, expected interest rates, time to expiration, and expected volatility
         2. developed in 1973 by Fischer Black, Robert Merton, and Myron Scholes. Scholes and Merton won the Nobel Memorial Prize in Economic Sciences in 1997
      3. A picture containing font, typography, text, calligraphy

         Description automatically generated
         1.   time to maturity
         2.  risk free rate
         3.  spot price of the underlying asset
         4.  strike price
         5.  volatility of returns of the underlying asset
         6.  cumulative distribution function of the standard normal distribution
      4. Code example w/ decorator; explain how it works under the hood
         1. derive 4 million independent results based on feeding the function with random stock prices, option strike prices, and times to maturity.
   2. Benchmark – cpu vs gpu
      1. A picture containing text, font, screenshot, line

         Description automatically generated
2. SQL performance
   1. BlazingSQL
      1. GPU-accelerated SQL engine built on top of the RAPIDS ecosystem.
      2. standard SQL queries to be distributed across GPU clusters, and the results to be fed directly into GPU-accelerated visualization and machine learning libraries
      3. Used CUDA based on the Apache Arrow columnar memory format
      4. BlazingSQL draws on Dask, which is an open source tool that can scale Python packages to multiple machines
      5. create a relational algebra plan from a SQL string; brains of the engine
   2. Benchmark BlazingSQL vs Apache Spark
      1. 20M rows of Netflow data two times
      2. BlazingSQL was 71X faster than Apache Spark
3. Data science analytics
   1. Dataframe
      1. 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns
      2. Popularized by Python Pandas
   2. cuDF is a GPU DataFrame library
      1. Python-based GPU DataFrame library for working with data including loading, joining, aggregating, and filtering data
      2. cuDF’s API is a mirror of Pandas’s and in most cases can be used as a direct replacement
      3. Pandas is only running on the CPU
      4. Modern datasets can have as much as millions, billions, or even trillions of data points that need processing
   3. Benchmark – cuDF vs Pandas
      1. DataFrame has over 100 Million points
      2. calculate the mean value of the ‘a’ variable in our data in Pandas vs. cuDF.
      3. A screenshot of a computer program

         Description automatically generated with medium confidence
      4. Merge
      5. A screenshot of a computer code

         Description automatically generated with medium confidence
4. System architecture – necessary?

IV – Proof of concept application

* Flask APP?
* WPF with Flask back end?
* Demonstrate performance difference - cpu vs gpu?