Fast Knowledge Discovery in Time Series with GPGPU on Genetic Programming

Sungjoo Ha, Byung-Ro Moon

Optimization Lab
Seoul National University
Computer Science
GECCO 2015

July 13th, 2015

Outline

- Motivation
  - Knowledge discovery in time series data
- Knowledge discovery engine
  - Discovery engine
  - Query engine
    - CUDA execution/memory model
- Details of parallel framework
- Experiments & results
- Conclusion

Knowledge Discovery

- A process of finding interesting patterns or structures from the given data

Time Series Data

- A sequence of records each containing a set of attributes and a timestamp
  - May be collected from multiple sources that share common characteristics
- Examples of time series data
  - Stock markets
  - Scientific computing
  - Monitoring metrics
  - IoT

Knowledge Discovery in Time Series

- Finding precursors to some events of interest
  - If *the closing price of yesterday is less than the five-day moving average of yesterday* than it is going to be a *bull market* tomorrow
  - If *the volume of certain hashtag doubles in 10 minutes* than that hashtag becomes a *new trend* in twitter.

Knowledge Discovery Engine

- Discovery engine
  - Propose candidate pattern
  - Genetic programming
- Query engine
  - Match pattern against data
  - GPGPU

Pattern

- Pattern is a conjunction of Boolean expressions
  - Expression is a combination of constants, comparison, arithmetic, logical operators, and attributes
  - May contain domain specific functions
  - $1.1 \times p_o(-1) < p_c(0) \wedge MA_{20}(0) > MA_{60}(0)$
- Find patterns such that they indicate an occurrence of some event
- A pattern may not yield the same result all the time
  - Deal with the uncertainties by taking the expectation of the events

Discovery Engine

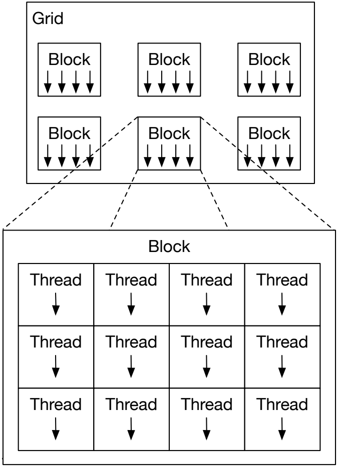
- Standard GP
  - Any additional features that are appropriate for the domain may be included
- Fitness of a pattern should reflect how interesting the events associated with the pattern are
  - Profitability of a technical pattern r
  - Expected earning rate of r after k trading days
  - $E_k[r] = \frac{1}{|R(r)|} \sum_{(i,j) \in R(r)} \frac{p_c(i,j+k)}{p_c(i,j)}$
  - $R(r) = \{(i,j) | r$ matches company i on trading day $j\}$
  - $p_c(i,j)$ is the closing price of company i at trading day j

Parallel GP

- Only parallelize fitness evaluation
- Low complexity
  - Simpler code
  - Flexible GP
- Fitness evaluation is most time consuming
  - Especially for hybrid GP

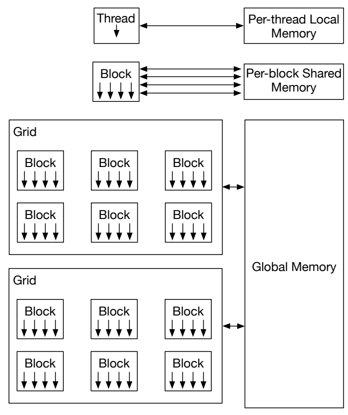
CUDA Execution Model

- Kernel executes in parallel by multiple threads
- Threads form blocks, blocks form grids



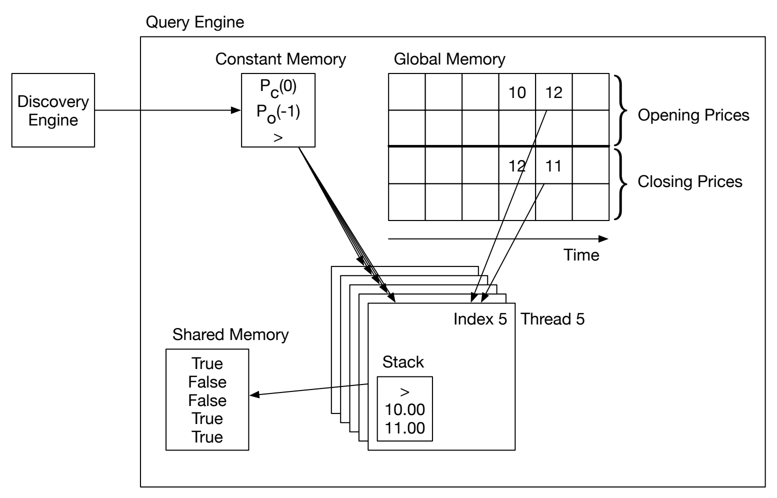
CUDA Memory Model

- Per-thread local memory
  - Local memory (global memory, slow)
  - Register (fast)
- Threads within a block can communicate using shared memory
  - For global synchronization, launch multiple kernels
- Global memory
  - Constant memory



Parallel Framework

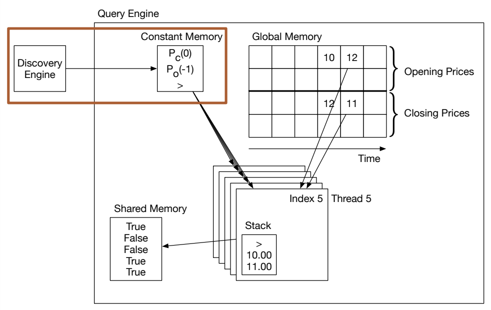
Big Picture



Parallel Framework

Storing Pattern

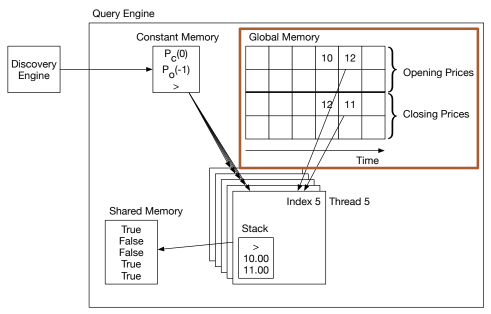
- Discovery engine generates candidate pattern
- Pattern is stored on constant memory in postfix notation



Parallel Framework

Storing Time Series

- Time series data is stored on global memory



Memory Utilization

- Each time series as 1-D array per field arranged in time
  - Threads jump along the time axis and process data
  - Better memory access pattern
- Example
  - Field 1/2 : opening/closing prices
  - First index : company
  - Second index : day index

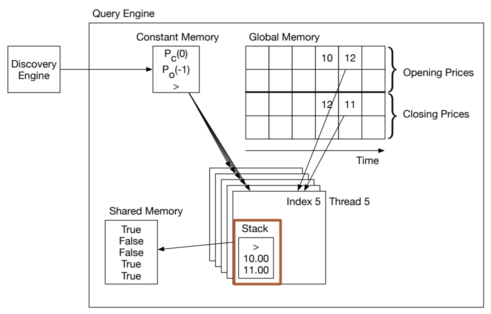
	Time			
Attribute 1	Field1[0][0]	Field1[0][1]	Field1[0][2]	Field1[0][3]
	Field1[1][0]	Field1[1][1]	Field1[1][2]	Field1[1][3]
	Field1[2][0]	Field1[2][1]	Field1[2][2]	Field1[2][3]
Attribute 2	Field2[0][0]	Field2[0][1]	Field2[0][2]	Field2[0][3]
	Field2[1][0]	Field2[1][1]	Field2[1][2]	Field2[1][3]
	Field2[2][0]	Field2[2][1]	Field2[2][2]	Field2[2][3]

	Time			
Opening Price	IBM[0]	IBM[1]	IBM[2]	IBM[3]
	APPL[0]	APPL[1]	APPL[2]	APPL[3]
	MSFT[0]	MSFT[1]	MSFT[2]	MSFT[3]
Closing Price	IBM[0]	IBM[1]	IBM[2]	IBM[3]
	APPL[0]	APPL[1]	APPL[2]	APPL[3]
	MSFT[0]	MSFT[1]	MSFT[2]	MSFT[3]

Parallel Framework

Evaluation of Pattern

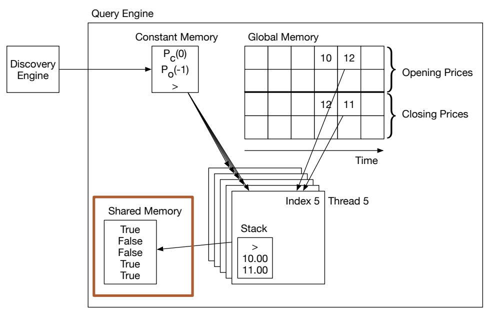
- Evaluation of postfix pattern using stack
  - Shared memory vs local memory
  - Local may lead to better performance (even though shared is much faster)



Parallel Framework

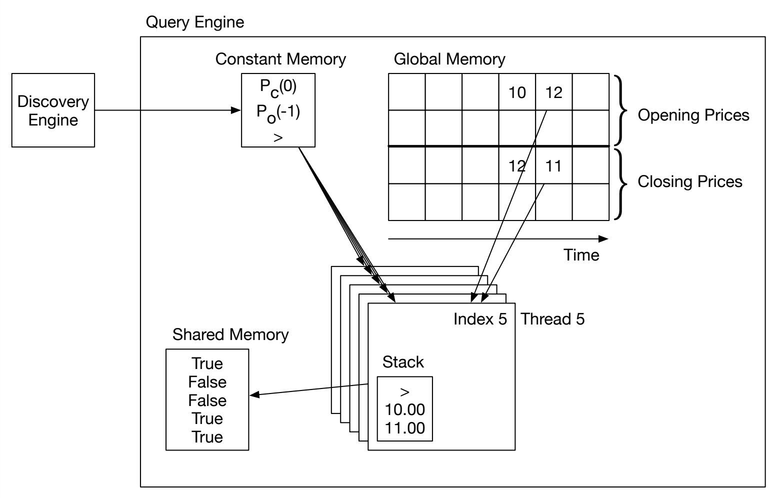
Per-Block Partial Results

- Per-block partial results
  - Shared memory
  - Parallel reduction
- If multiple input sources are needed to calculate the partial results
  - Accumulate temporary values in global memory
  - Launch a separate kernel for reduction



Parallel Framework

Recap



Experiments

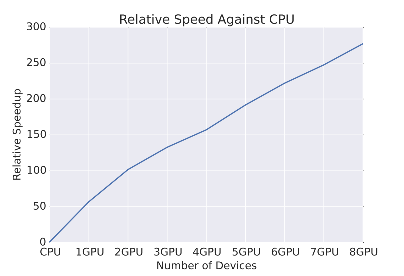
- Stock market technical pattern mining
  - Find patterns with high profitability
- Korean stock market data from 2000 to 2014
- Genetic programming
  - steady state
  - 500 individuals
  - Random initialization
  - Roulette wheel selection
  - Swap randomly chosen subtrees for crossover
  - Random subtree mutation
  - Traverse each nodes and change the contants for local optimization

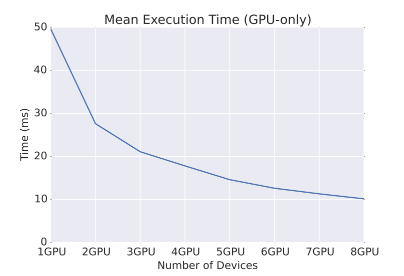
Results

- A typical execution of GP for 50 generations using 8 CUDA devices take 120 to 180 seconds
- Roughly 17,000 to 21,000 fitness evaluations are performed
- Local optimization takes more than 97% of the time

Results

Single Fitness Evaluation





- Strong linear relationship between the relative speed and the number of GPU devices
- Roughly yields 80% gain per additional device

Result

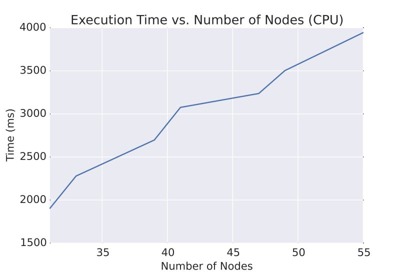
Local vs Shared Memory for Stack

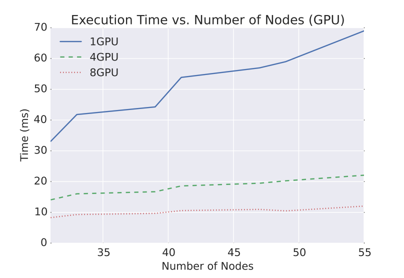
# Devices	Relative Speed			
	Local		Shared	
	μ	σ	μ	σ
CPU	1.0	0	1.0	0
1 GPU	56.9	3.8	35.8	2.8
2 GPU	101.7	7.2	65.0	5.5
3 GPU	132.7	11.1	88.0	8.1
4 GPU	157.1	15.5	107.2	10.7
5 GPU	191.7	19.9	130.3	13.7
6 GPU	222.0	24.5	151.5	16.6
7 GPU	247.5	29.3	172.2	19.5
8 GPU	277.0	36.8	188.7	22.4

- Local memory for stack performs better than shared for this task
  - Freeing shared memory lead to high occupancy of the multiprocessors
  - High latency of the global memory is hidden

Result

Tree Sizes





- Parallelization allows us to build increasingly complex trees
  - For CPU, time nearly doubles as tree grows from 31 to 55 nodes
  - Using many devices leads to a slower increase in execution time

Conclusion

- Proposed knowledge discovery framework using genetic programming and GPGPU
  - GP based discovery engine
  - GPGPU based query engine
- Performs well on a real world task
  - Strong linear relationship