AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

BOLT Stocks Data Analysis

1 Overview

1.1 Location

`$<APPSDKSamplesInstallPath>\samples\bolt\`

1.2 How to Run

See the *Getting Started* guide for how to build samples. You first must compile the sample.

Use the command line to change to the directory where the executable is located. The pre-compiled sample executable is at `$<APPSDKSamplesInstallPath>\samples\bolt\bin\x86\` for 32-bit builds, and `$<APPSDKSamplesInstallPath>\samples\bolt\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `StocksDataAnalysis`
   This command runs the program with the default options.
2. `StocksDataAnalysis -h`
   This command prints the help file.
3. `StocksDataAnalysis -h`
   This command generates a build with the multiCoreCpu path (the Thread Building Block library), enabled.

1.3 Command Line Options

Table 1 lists, and briefly describes, the command line options.

Table 1 Command Line Options

Short Form	Long Form	Description
`-h`	`--help`	Shows all command options and their respective meaning.
	`--device`	Explicit device selection for Bolt [auto/openCL/multiCoreCpu/SerialCpu]
`-q`	`--quiet`	Quiet mode. Suppresses most text output.
`-e`	`--verify`	Verify results against reference implementation.
`-t`	`--timing`	Print timing.
`-v`	`--version`	AMD BOLT library and run-time version string.
`-i`	`--iterations`	Number of iterations for kernel execution.
`-f`	`--file`	File containing the stocks data

Note: The `--device multiCoreCpu` option becomes available when the sample is compiled with `ENABLE_TBB` defined. Microsoft Visual Studio build configurations `Debug_TBB` and `Release_TBB` are created for this purpose. These configurations have `ENABLE_TBB` defined to enable the TBB path (multiCoreCpu) for all the AMD BOLT functions used in the sample.

2 Introduction

This sample demonstrates how Bolt APIs can be used for performing analytics computations on stocks data. The sample uses fabricated stocks data.

The sample expects an input .csv file in the following format:

The first line of the .csv file must be a number that indicates the number of stock quotes, that is, the number of rows. The second line must specify the following comma-separated field names (this line is skipped during the file parsing step):
Company, Date, Open, High, Low, Close, Volume

This line must be followed by the comma-separated stock quotes values corresponding to the field names. The following example shows a sample .csv file.

```
3,
Company, Date, Open, High, Low, Close, Volume
AAA, 30/12/2011, 186.28, 186.74, 182.17, 185.62, 3774100
AAA, 29/12/2011, 183.19, 187.25, 182.62, 186.76, 3112300
AAA, 28/12/2011, 183.75, 186.71, 183.30, 186.63, 2731300
```

3 Implementation Details

The stocks data is stored in an array of structure `Quote`.

```
struct Quote
{
         // integer representation of company name
        unsigned int tickerSymbol;

         // integer representation of date
        unsigned int date;

        float open;
        float high;
        float low;
        float close;
        unsigned int volume;
    };
```

This sample performs 2 operations on the stock data:

1. Average closing price of the stock in N days: For every single day, the sample calculates the N-day closing price average of the closing prices on the current day and the (N-1) previous days.

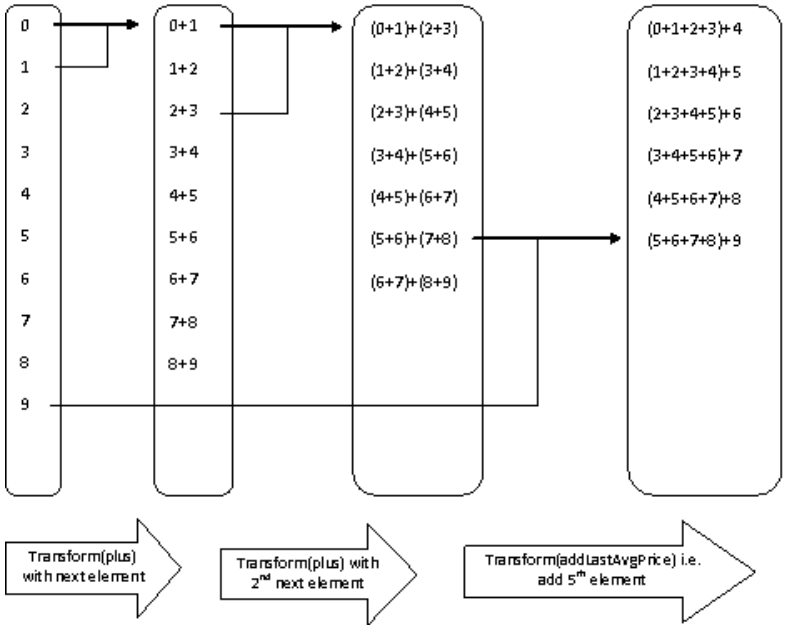
2. Daily volume of trade: For every single day, the sample calculates the total volume of trade from all the companies.

3.1 Average closing price of stock in N days

The average closing price of stock is implemented in two methods using two different BOLT APIs.

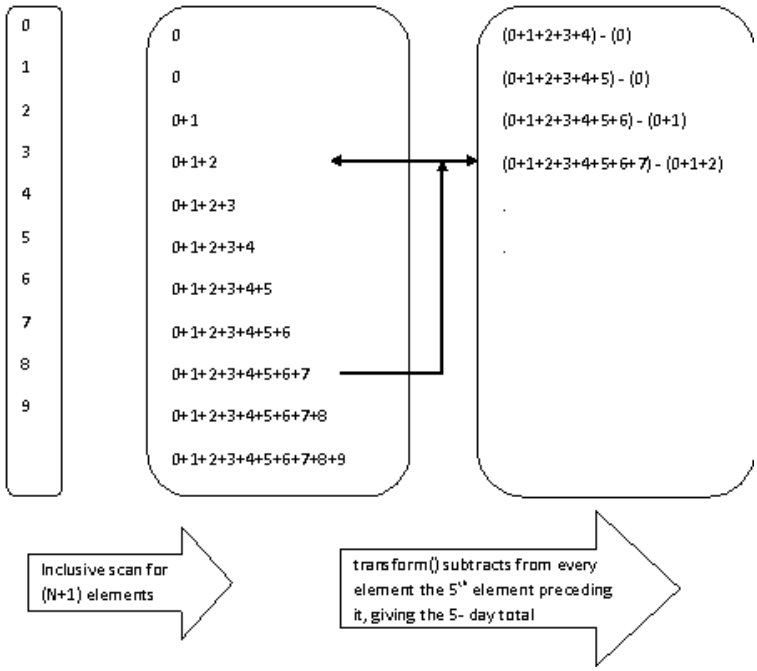
3.1.1 Using `bolt::transform()`

Successive transforms can be applied to arrive at the N-day closing price average. The sample computes a 5-day average, for illustration. To arrive at the 5-day average, a set of 3 transforms is applied, as shown in the following figure.



3.1.2 Using `bolt::inclusive_scan()`

The `inclusive_scan()` operation calculates a running sum over a range of values, inclusive of the current value. In this sample, `inclusive_scan()` calculates the 30-day closing price average of the stocks data. The following figure shows illustrates the computation of a 5-day closing point average.



3.2 Daily volume of trade

3.2.1 Using `bolt::transform()`

The stocks data is sorted according to date. The sorting yields a list in which the closing-prices are grouped according to date. A transform on this list yields the daily volume of trade for a given day.

3.2.2 Using `bolt::reduce_by_key()`

The method creates a key-value pair of (`date`, `volume`) for every quote in the stocks data. The list is sorted with `date` as the key. The sorting yields a list which is sorted by `date`, with the corresponding volume. A `reduce_by_key()` on this list, with `date` as the key, will accumulate the volume for a particular day.

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:
URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Support: developer.amd.com/appsdksupport
Forum: developer.amd.com/openclforum