A Big Data Approach to Analyzing Market Volatility

Kesheng Wu
E. Wes Bethel
Ming Gu
David Leinweber
Oliver Rübel

Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720

A Big Data Approach to Analyzing Market Volatility

Kesheng Wu, E. Wes Bethel, Ming Gu, David Leinweber, Oliver Rübel

June 5, 2013

Abstract

Understanding the microstructure of the financial market requires the processing of a vast amount of data related to individual trades, and sometimes even multiple levels of quotes. Analyzing such a large volume of data requires tremendous computing power that is not easily available to financial academics and regulators. Fortunately, public funded High Performance Computing (HPC) power is widely available at the National Laboratories in the US. In this paper we demonstrate that the HPC resource and the techniques for data-intensive sciences can be used to greatly accelerate the computation of an early warning indicator called Volume-synchronized Probability of Informed trading (VPIN). The test data used in this study contains five and a half year's worth of trading data for about 100 most liquid futures contracts, includes about 3 billion trades, and takes 140GB as text files. By using (1) a more efficient file format for storing the trading records, (2) more effective data structures and algorithms, and (3) parallelizing the computations, we are able to explore 16,000 different ways of computing VPIN in less than 20 hours on a 32-core IBM DataPlex machine. Our test demonstrates that a modest computer is sufficient to monitor a vast number of trading activities in real-time – an ability that could be valuable to regulators.

Our test results also confirm that VPIN is a strong predictor of liquidity-induced volatility. With appropriate parameter choices, the false positive rates are about 7% averaged over all the futures contracts in the test data set. More specifically, when VPIN values rise above a threshold (CDF > 0.99), the volatility in the subsequent time windows is higher than the average in 93% of the cases.

Keywords: high-performance computing, market microstructure, probability of informed trading, VPIN, liquidity, flow toxicity, volume imbalance, flash crash.

(JEL codes: C02, D52, D53, G14, G23.)

1 Introduction

A number of factors have contributed to the explosion in the amount of financial data over the last few years: the migration of flow from pits to electronic exchanges, decimalization of stock prices, direct market access, etc. More than half of the trades in the E-mini S&P500 futures during the year 2012 were for a size of 1 unit. This extreme order slicing has created a great demand for extracting usable information from the mountains of data generated by ever-larger market databases. The major trading firms typically have their own well-funded computing efforts to handle the big data problem. However, the academic researchers and government regulators are lagging behind in computing power and software tools to work with the big data[1]. Markets need contributions of regulators and academic researchers to understand the

[1]For example, a private organization like NANEX publishes frequent analyses on market malfunctions, very often bringing them to the attention of regulators. We believe that a better understanding of HPC techniques could help regulators and academics to identify these situations as efficiently as NANEX.

implications of the rapid trading processes and mechanisms commonly called high-frequency trading. A better understanding of the high-frequency trading and better regulations might have prevented events like the Flash Crash of May 6, 2010 [15, 32]. Conversely, a lack of understanding of the market may mislead financial academics and regulators, resulting in flawed conclusions and counter-productive regulations. To empower the academic researchers and regulators, we introduce a set of techniques used by scientific data management community [25, 36] and demonstrate their effectiveness using a large set of futures trading data.

Scientific research activities such as the Large Hadron Collider at CERN and climate modeling produce and analyze petabytes (10^{15} bytes) of data. The data rates from such activities are in fact multiple orders of magnitude higher than those from financial markets. In most cases, scientists are conducting their analysis tasks not on the expensive data warehousing systems, but using open-source software tools augmented with specialized data structures and analysis algorithms [36, 7]. We believe that these tools and techniques could significantly improve the handling of financial data in a relatively inexpensive way, and help academics and regulators understand and respond to the challenges posed by high-frequency data.

To demonstrate the effectiveness of the techniques to be used, we will process a large data set to compute a popular liquidity indicator called Volume-synchronized Probability of Informed Trading (VPIN) [8, 17]. When the values of VPIN exceed a certain event threshold, say 2 standard deviations larger than the average, we declare a "VPIN event." Within a time window immediately following a VPIN event, we expect that the market would be more volatile than usual. To verify this hypothesis, we compute a realized volatility measure named the Maximum Intermediate Return (MIR), which essentially measures the largest range of price fluctuation in a time window [13]. We compare the MIR values following VPIN events against the average MIR value sampled at random time periods. If the MIR following a VPIN event is equal or less than the average MIR of random time periods, the VPIN event is a false positive. The functions for computing VPIN and MIR are controlled by six parameters including the size of the support window for computing VPIN and the time window for computing MIR. These parameters will be explained in Sections 3 - 7 where we discuss the details of the algorithm for computing VPIN and MIR. For each set of these parameters, we compute an average false positive rate over the different types of trading instruments. In later discussions, we use this average false positive rate as the measure of the quality of VPIN.

The primary goal of this paper is to introduce a set of scientific data management techniques for the analysis of financial data. These techniques include

1. an efficient file organization for storing the trading records,
2. efficient data structures and algorithms for computing VPIN and MIR,
3. parallelization of computational tasks to make full uses of the computing resources.

The bulk of this paper is on the data structures and algorithms used for computing VPIN and MIR. For example, we have developed a new algorithm for computing MIR in Section 7, introduced a new way to modify the computation of VPIN in Section 3, and applied a number of new ways implement the computation procedures.

To demonstrate their effectiveness, we apply our programs on a large set of trading records of futures contracts. Our test data contains all trading records of nearly 100 most liquid futures contracts spanning from the beginning of 2007 through the middle of 2012 including about 3 billion trading records. This data set has a large variety of trading instruments including stock indices, precise metal, currency and so on. Without compression, it takes up 140 GB of disk space. This data set is much larger than any previous datasets used for VPIN computations. To our knowledge, no study on market microstructure in research literature has this large volume of data or this variety of trading instruments [2, 33, 40].

The computational task we have chosen for this study is to evaluate the effectiveness of VPIN. Our efficient computer algorithms allow us to tune the handful of free parameters of VPIN for the 100 or so different types of futures contracts. We are able to determine a set of recommendations for choosing these parameters to achieve a low average false positive rate over the variety of futures contracts.

There are many market indicators that can be computed in real-time to monitor the health of the market [11]. Further research might reveal better ways of tuning VPIN or computing better indicators, however, it is clearly useful to compute some indicators in real-time on the vast amount of high-frequency trading data. By demonstrating that we can perform this computation on a modest computer, we show that the technology required to monitor real-time liquidity conditions over all markets already exists, and can be deployed with a modest investment.

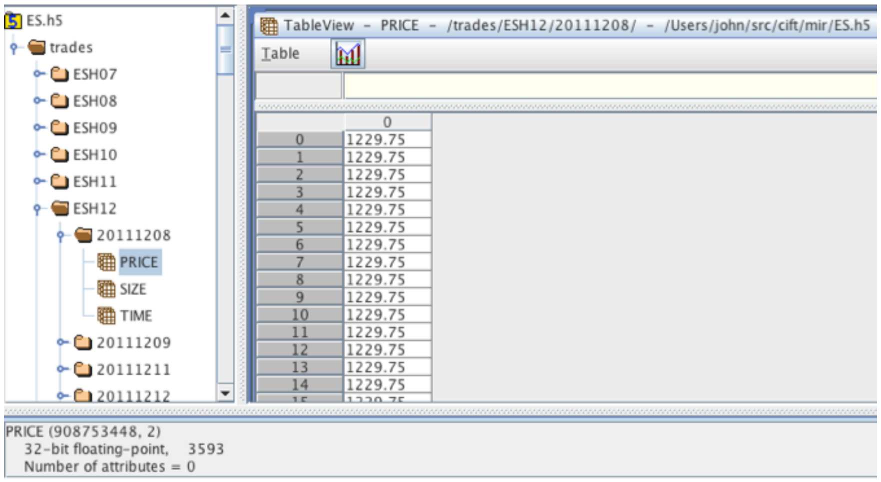
In the following we first discuss the dataset used in this study (Section 2) and afterwards describe the computation of VPIN and MIR (Section 3-7). One key technology that allows us to compute VPIN values efficiently is the HDF5 file format we use to store the trades. The structure of the HDF5 files is described in Section 2. The computation of VPIN and MIR is performed in several steps. First, the data is organized in volume bars to enable sampling of the market signal at regular intervals and to allow application of standard statistics (Section 3). VPIN evaluates the balance between buy and sell volumes in the market. Based on the volume bars, the trades are then classified as either seller-initiated or buyer initiated, or simply sell or buy (Section 4). Finally, the volume bars are organized into buckets used to compute the imbalances between buy and sell (Section 5). The procedure to compute VPIN is given in Section 6, where we also describe how VPIN values are normalized for easy comparisons. We give a careful definition of MIR in Section 7 and describe how to compute its value effectively. In Section 8, we discuss the effort to parallelize the computation. In Section 9, we describe our tests on computing VPIN and MIR using the sample data, and discuss how the false positive rates are affected by the six parameters controlling the computation procedures. We summarize the paper in Section 10.

2 Test data and data file formats

In this work, we use a comprehensive set of liquid futures trading data to illustrate the techniques to be introduced. More specifically, we will use 67 months worth of tick data of the most liquid futures traded on all asset classes. The data comes to us in the form of 94 CSV files, one for each futures contract traded. The source of our data is TickWrite, a data vendor that normalizes the data into a common structure after acquiring it directly from the relevant exchanges. The total size of the CSV (Coma Separated Values) files is about 140GB. They contain about 3 billion trades spanning from beginning of January 2007 through end of July 2012[2]. Five of the most heavily traded futures contracts have more than 100 million trades during this five and half year period. The most heavily traded futures, the file containing E-mini S&P500 futures, symbol ES, has about 500 million trades involving a total number of about 3 billion contracts. The second most heavily traded futures is Euro exchange rates, symbol EC, which 188 million trades. The next three are Nasdaq 100 (NQ), 173 million trades; light crude oil (CL), 165 million trades; and E-mini Dow Jones (YM), 110 million trades. The complete list of the futures contracts used in our study is shown in A.

To improve the efficiency of data handling, we convert the CSF files into HDF5 files [24]. In a published study on a similar task of computing the VPIN values on a months worth of stock trades, it took 142 seconds

[2]The end date is the time when the data files are collected and passed to the hands of this group of researchers for this study.



TableView - PRICE - /trades/ESH12/20111208/ - /Users/john/src/cift/mir/ES.h5

	0
0	1229.75
1	1229.75
2	1229.75
3	1229.75
4	1229.75
5	1229.75
6	1229.75
7	1229.75
8	1229.75
9	1229.75
10	1229.75
11	1229.75
12	1229.75
13	1229.75
14	1229.75

```
PRICE (908753448, 2)
  32-bit floating-point,   3593
  Number of attributes = 0
```

Figure 1: A portion of the HDF5 file structure for ES as shown by the tool h5view.

using the CSV file while it only took 0.4 seconds with the corresponding HDF5 file [8]. One reason for this efficiency is that the HDF5 files store data in the binary form, which require less storage space. For the set of test data used in this work, the total size of all HDF5 files is about 41GB, while the CSV files are 140 GB in size. The total size of HDF5 files is about 29% of that of CSV files. It is possible to compress the CSV files to reduce the storage requirement. In fact, it takes only 21GB to store the gzip compressed CSV files. However, compressed CSV files have to be decompressed before their contents could be used. This decompression process typically takes more time than reading the uncompressed data.

Since a CSV file takes up more bytes than the corresponding HDF5 file, the latter requires less time to read into memory. Furthermore, after reading the CSV values, the program has to convert the strings into numerical values, while the values read from a HDF5 file can be directly used by the application code without further interpretation. Furthermore, HDF5 allows for better organization of the data, e.g., by day and stock symbol, so that the data required for a given calculation can be located and loaded directly in HDF5 without the need for time consuming search operations needed to located data in CSV files.

We are not the first to note the potential gain from using HDF5 for financial data. For example, StratBox from PuppetMaster Trading stores its data as HDF5 files at the backend since 2008 [3]; and professor Henning Bunzel of Aarhus University mentioned that HDF5 is a part of the set of software packages available at his organization [4]. However, the only peer-reviewed documentation of the effectiveness of HDF5 in handling financial data is in a 2012 article by Bethel et al [8]. In these cases, HDF5 was used to store TAQ data. In this work, we use the HDF5 for trades, i.e., level-1 information of futures trading tick data.

One difference between trading data about stocks and futures is that the latter have expiration dates that occur at a certain frequency. As a result, contracts on a single instrument with different expiration dates have different symbols, and need to be rolled before expiration. For example, while the basic symbol for E-MINI is ES, the contract that expires in March 2007 is labeled as ESH07, and the contract that expires in June 2010 would be ESM10. The details of the rolling rules are also given in A.

Following the organization of the CSV files, we create a HDF5 file for each CSV file, and place contracts with the same symbols in a top-level HDF5 group. We further divide the trades into groups of days. This

[3]`http://www.puppetmastertrading.com/blog/2009/01/04/managing-tick-data-with-hdf5/`

[4]`http://creates.au.dk/fileadmin/site_files/filer_oekonomi/Research_centres/CREATES/Data_and_Computational_Resources.pdf`

allows us to only record the time of the day for each trade. Effectively we only store three 4-byte words for each trade, which is the key reason for the compact sizes of the HDF5 files. A depiction of the actual organization is shown in Figure 1 using the tool named h5view from HDF5 library.

3 Bars

One key characteristic of high-frequency data is that it arrives at irregular frequency. This complicates the application of standard statistical techniques [3, 6]. Practitioners typically circumvent this problem by sampling market data at regular time or volume intervals, which generate time or volume bars [9, 20]. This type of time-scale transformation is common in physical sciences in order to recover data invariance. Aggregating the individual trades in bars could significantly improve the effectiveness of microstructure models. Part of the reason for this is that the estimation of those models typically relies on the sequential price changes and it is difficult to reconstruct the precise order of the execution of trades with (nearly) identical time stamps. In this work, we decide to follow the recommendation from recent publications and construct volume bars [15].

All trades involving the same instrument, such as S&P 500 E-mini (ES), are processed together. In the previous section, we noted an important feature of futures contract trading to be that the symbols are rolled periodically. When this happens, we drop the last few trades that do not fill up a bar. Other than this feature, the process of constructing bars for futures contracts is the same as for stocks and commodities reported in literature [14, 16].

A volume bar is a number of trades that are close to each other in time and having a prescribed total volume. Based on the published literature [18, 16], we determine the volume of a bar as a fraction of the average daily volume over the entire data set. This average daily volume is computed by dividing the total volume of all trades in the data set by the number of trading days. Most of the commodities are traded after hours as well as during the normal weekday business hours. If we simply count the number of distinct days that appear in the data records, we typically see six trading days per week, Monday through Friday plus an hour or so of trading before Midnight on Sunday. Most traders view these trades on Sunday night as part of the Monday trading; therefore we have decided to do the same. In general, we have decided to absorb days with less than two hours of trading to the following day. This allows us to have five effective trading days for the purpose of computing daily volume. The resulting average daily volumes are close to what are reported in the financial press. Therefore, we believe this to be a reasonable choice.

In later processing steps, each bar is treated as a single trade with a time stamp, a volume and a price. Clearly, the volume is the user specified constant, which does not need to be recorded in the data structure for the bar. Therefore, the basic information for each bar includes its nominal time and its nominal price. Following the convention used in the published works, we declare the time stamp of the last trade in the bar as the time stamp for the bar [14, 16]. The time stamps of bars are not used in any important operations in this work.

The second piece of information in a bar is its nominal price. In earlier work, the nominal price for a volume bar is defined as the price of the last trade in the bar. We say that the price of the volume bar is represented by the closing price [14, 16]. Typically, the volume of a bar is relatively small and the trades in the bar have nearly the same price, however, there are cases where the prices of the trades are different. Therefore, it might be worthwhile to consider different ways of computing the nominal price for a bar. Here are a few choices we consider in this work: (1) closing prices, (2) unweighted average prices, (3) median prices, (4) volume-weighted average prices, and (5) volume-weighted median prices.

In our implementation, we use a stable sorting procedure known as “shell sort” to order the trades in

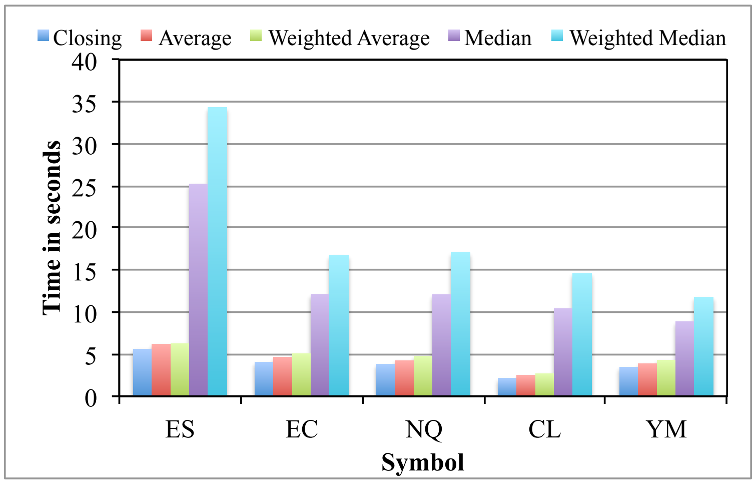


Figure 2: Time (seconds) needed to construct volume bars with different nominal prices.

a bar in order to compute the median prices [34]. This sorting procedure is very simple and requires only a few words of working space. Let N_b denote the number of trades in a bar. On average, we expect that each trade is to be compared $\log_2 N_b$ times in order to compute the median prices. In contrast, the closing price can be obtained by simply taking the price of the last trade, and the average price can be computed with one pass through all the trades in a bar. Overall, we expect the cost of computing the closing prices to be less than the cost of computing the average prices and less than the cost of computing the median prices. Furthermore, computing the weighted average or weighted median would take slightly more time than computing their unweighted versions. Figure 2 shows the median time used to form the volume bars with different nominal prices. These tests generated about 6,000 bars per trading day and about 9,000,000 bars for each commodity type. From these timing values, we see that computing the median prices requires much more time than computing the closing prices or the average prices.

In constructing the volume bars, we have two free parameters to choose: how to compute the nominal prices and the size of the volume bars. We denote the pricing strategy with π in the remaining of this paper. In this study, we will consider five differ pricing strategies for a volume bar mentioned earlier. The size of a bar is further discussed in the next section.

4 Bulk volume classification

The computation of VPIN, like many other techniques for analyzing market microstructures, requires us to determine directions of trades, which is classifying each trade as either as buyer-initiated or seller-initiated [17], or simply as a *buy* or a *sell* [29, 28]. A common method used in the market microstructure literature is the tick rule or more formally the Lee-Ready trade classification algorithm [29, 38]. The basic idea of the tick rule is to assign a trade as buy if its price is higher than the preceding trade, as sell if its price is lower than the preceding trade, and the same type (buy or sell) as the preceding trade if there is no change in price.

This classification is heavily dependent on the sequential order of trades. Typically, the order of trades can be determined from the time stamps of the trades. However, in our data as in many other sources of data, there are frequently many trades with the same time stamp due to high-frequency trading. Another more serious source of timing issue is that the time stamps on trades executed at multiple matching engines may not be synchronized to the precision used to record the time stamps. Furthermore, the use of Bulk Volume Classification (BVC) significantly reduces the computational cost [10]. For these reasons, we follow the recommendation from a recent publication from Easley et al [18, 16] and use Bulk Volume Classification on trades.

BVC assigns a fraction of the volume to buys and the remainder to be sells based on the normalized sequential price change [18]. On a set of volume bars, each bar is used as a single trade with the nominal price of a bar as the price used for classifying the fractions of the volume as buy or sell. In the earlier publications [18, 16], the closing price of a bar is always used as the nominal price. In this work, we will also examine different choices for the bars as discussed in the previous section.

BVC is known to produce different buy and sell volumes than the tick rule, a strategy used earlier for classifying buy and sell volumes. For example, the differences reported by Chakrabarty et al. are between 5% and 16% on stocks [10] and the differences reported by Easley et al are around 10% on futures contracts [14]. Since the tick rule does not provide perfect classification anyway, and this particular study is carried out on futures, we follow that recommendation of Easley et al [16].

Given a sequence of volume bars with prices, P_0, $P_1, \ldots$, the sequential price changes are $\delta_1 \equiv P_1 - P_0$, $\delta_2 \equiv P_2 - P_1$, $\delta_3 \equiv P_3 - P_2$, and so on. The Bulk Volume Classification (BVC) method assumes these sequential price changes follow a distribution such as the Normal distribution or the Student t-distribution, and assign a fraction of the volume in a bar to be buy and the remaining to be sell. Let V_j^b denote the buy volume for bar j, and the volume of bar to be V_j, we can compute V_j^b from V_j as follows [18],

$$V_j^b = V_j Z(\delta_j / \varsigma), \tag{1}$$

where Z is the cumulative distribution function of either the Normal distribution or the Student t-distibution. The parameter ς is the standard deviation of the sequential price changes. The rest of the volume in the bar is considered as sells,

$$V_j^s = V_j - V_j^b.$$

The expression δ_j/σ could be considered a way to normalize the price changes. The common practice is to subtract the average price change first before dividing by the standard deviation. However, when working with the high-frequency data, the average is much smaller than the standard deviation as shown later in Table 6. Following the recommendation from earlier publications [18, 16], our implementation of the BVC method always use zero as the center of the Normal distribution and the Student t-distribution.

The fourth free parameter in computing VPIN is the parameter to control the assumed distribution for sequential price changes. We use either the Normal distribution or the Student t-distribution. When using the Student t-distribution, we can freely choose the dimension of the distribution denoted by ν.

In our software implementation, we use a software pattern known as Policy, which defines a common interface for different choices of the probability distribution [21]. This allows us to determine the distribution function to use when invoking the function for forming buckets to compute VPIN values.

5 Buckets

A bucket contains a user-specified number of bars. When forming buckets, each volume bar is used as if it is a single trade with the nominal price. A VPIN value is computed from a set of buckets following each other in time. Typically, 30 - 50 bars are placed in a bucket. From the published literature, we know that the number of bars in a bucket has only minor influence on the final value of VPIN [1, 16]. In this work, we have chosen to fix the number of bars in a bucket to be 30 for all tests.

Once we fix the number of bars per bucket, then the size (or volume) of a bar and a bucket is determined by the number of buckets per day. We denote the number of bucket per day with β. In this study, we vary β from 50 to 1000.

The information required about each bucket for the computation of VPIN is the buy volume and the sell volume, nothing else. Furthermore, only the most recent few, say ω, buckets are needed for the computation of a VPIN value. The value of ω is the third free parameter we can choose to maximize the effectiveness of VPIN. However, instead of directly specifying the number of buckets to use, we specify the number of buckets as a fraction of the number of buckets in a day, σ. Using σ as the free parameter, we are using a fixed fraction of an average day's trades as the basis for computing VPIN. This appears to be a better choice than fixing the the number of buckets for computing VPIN while varying other parameters.

In our implementation, we only keep ω most recent buckets with their buy and sell volumes. We keep these values in a ring buffer that occupy a fixed piece of computer memory. This implementation strategy requires much less memory than storing all buckets in memory or in a data management system. Because the ring buffer is fixed for the entire duration of the computation, it requires only a minimal amount of work to update as the pseudocode in Listing 1 shown. These features allows VPIN values to be computed quickly.

The complete pseudocode listing of the definition of a ring buffer in C++ syntax is given in Listing 1.

Listing 1: Pseudocode of a ring buffer for storing buckets

```
class ringBuffer {
public:
    /// Constructor.  Default to 50 buckets.
    explicit ringBuffer(size_t n=50) {
        vbuy.reserve(n);
        vsell.reserve(n);
        next = 0;
    }
    void insert(double b, double s) {
        if (vbuy.size() < vbuy.capacity()) {
            vbuy.push_back(b);
            vsell.push_back(s);
            next = vbuy.size();
        } else {
            vbuy[next] = b;
            vsell[next] = s;
            ++ next;
        }
        if (next >= vbuy.capacity())
            next = 0;
    }

private:
    /// The next position to write.
    size_t next;
    /// Buy volumes in the last N buckets.
    std::vector<double> vbuy;
    /// Sell volumes in the last N buckets.
    std::vector<double> vsell;
};
```

6 VPIN

VPIN measures the average order flow imbalance in the market [15, 1]. It not only takes into account the imbalance of buyer-initiated and seller-initiated orders, but more generally the imbalance between buying and selling pressure. This difference is important, because not all buying pressure is the result of an excess of buy-initiated orders. VPINs definition has deep theoretical meaning [17, 15], however, the formula for estimating its actual value is quite straightforward. Based on a probabilistic model of information imbalance, the authors have developed a set of elegant formulas for computing VPIN. We refer the readers interested in the derivation of the formulas to references [17, 15], here we simply give the formula we use,

$$\text{VPIN} = \frac{\sum \|V_j^b - V_j^s\|}{\sum V_j}.$$

The pseudocode for computing a VPIN using the variables in the above ring buffer data structure is given in listing 2.

Listing 2: Pseudocode for computing VPIN using the ring buffer defined in listing 1

```
sum = 0.0;
for (unsigned j = 0; j < vbuy.size(); ++ j)
    sum += abs(vbuy[j] - vsell[j]) / (vbuy[j] + vsell[j]);
vpin = sum / vbuy.size();
```

In the above definition, we have neglected to specify the number of buckets to use for computing a VPIN value, i.e., the number of buckets in the ring buffer. From earlier publications on VPIN, we see that it is meaningful to associate the number of buckets with something such as fraction of an average trading day [16]. We call this time window for computing VPIN the *support window* and denote it as σ in the future discussion.

The above procedure generates VPIN values between 0 and 1. The large VPIN values indicate unusually high buy-sell imbalance. To decide what values should be considered large, it would be convenient to set a common threshold for a large class of financial instruments. However, in practice, the value of VPIN is frequently concentrated in a narrow range, and the range is different for different stocks, indices or futures contracts. In the published literature, researchers recommend normalizing the VPIN values by working with its cumulative distribution function (CDF) [14]. This strategy is equivalent to picking a constant threshold, say γ, and declare a VPIN event whenever a value is γ standard deviations larger than the average.

Assume the VPIN values follow the Normal distribution, then the CDF of VPIN can be computed by invoking the Gaussian error function from a standard math library. The relationship between the CDF and the error function is

$$\text{CDF}(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right], \tag{2}$$

where, erf is the error function, μ is the average of VPIN values, and σ is the standard deviation. We declare a VPIN event whenever the CDF of a VPIN is larger than a threshold τ. We could use the same threshold τ on a variety of trading instruments [15, 16]. Furthermore, using the value τ as the threshold on the CDF has a simple explanation. Assuming all VPIN values follow the Normal distribution, if the CDF of VPIN is greater than 0.99, then the VPIN value is larger than 99% of all VPIN values.

The VPIN values are typically much less than 1. Empirical evidences suggest that they might be better described as following the lognormal distribution. In other words, the logarithm of VPIN values might fit normal distribution better. The key challenge in computing the logarithm of VPIN values is that they might be exactly zero or very close to zero. The logarithms of these small values have very large absolute values, which can distort the values of average and standard deviation. In all of our calculations that assume lognormal distribution, we treat VPIN values less than 0.001 as 0.001, in order to avoid this difficulty with small VPIN values[5]. In the research literature, this variation of the lognormal distribution is known as following a truncated lognormal distribution [12].

In our tests, we always use the log-normal distribution. The parameter τ is the fifth free parameter we needed to determine to maximize the power of VPIN.

7 Maximum Intermediate Return

When the CDF of a VPIN value is above a given threshold τ, we say that we have encountered a VPIN event, and expect the volatility immediately following the event to be higher than usual. Our next task is to select a measure to quantify the volatility [35], which will enable us to discriminate whether a VPIN event is a true event or a false event. Specifically, we need to identify: I) the duration of a VPIN event during which the volatility is expected to be high, and II) a measure to quantify the volatility of the trades in any time window. In this work, we assume the event has the same duration for all futures contracts throughout 67-month trading history, and attempt to determine this event duration through a set of empirical measurements to be discussed later in Section 9. To measure volatility, we use the Maximum Intermediate Return (MIR) [13, 16]. The remaining of this section describes in detail the definition of MIR, the algorithm and its computational cost. A key innovation is this work is an efficient algorithm for computing MIR.

7.1 Definitions

Let's assume a VPIN event always last the same amount of time η (measured in days). Given a volatility measure M, we say a VPIN event is a true event, if its M value is greater than the average value from randomly selected time intervals of the same duration η. A false event has an M value that is less or equal to the average of the random events. It is possible to use a more sophisticated criterion, however all known definitions rely on an arbitrarily chosen threshold that determines a discrete binomial outcome (true or false). These definitions typically require the computation of the standard deviations of the volatility measures in addition to the average value required for our criteria. Generally, the additional cost of computing their average and standard deviation is relatively small compared to the time needed to compute the volatility measures. Therefore, our choice does not significantly affect the computational cost of evaluating the VPIN events. Since this work is focused on the computational costs, our choice should be nearly as good as other more sophisticated choices.

We call a false event a *false positive*. The fraction of events that are false positives is the *false positive rate*. If VPIN is perfectly effective in predicting future volatility, then we expect the false positive rate to be zero. Since the computation of the false positive rate involves a division by the number of VPIN events, we need to deal with the special case of no event. In cases where no VPIN event is detected, we take the number of events to be 0.5 and the number false positives to be 0.5 as well. Clearly, this choice is quite arbitrary. The practical reason for this choice is to prevents the problem of division by zero. Another important consideration is to prevent us from favoring parameter choices that produce no VPIN event.

[5]The actual value of the minimal VPIN value can be modified at the compile time through a macro definition.

To measure the volatility, we believe any number of well-known measures could in principle be fine [35, 4, 5, 6]. For this work, we opt for an instantaneous volatility measure called the *Maximum Intermediate Return* or *MIR* for short. Many of the well-known volatility measures are based on the realized return over the entire duration of an event. However, these measures are not appropriate for events like the "Flash Crash" of May 6, 2010, which is generally considered as a liquidity crisis, where VPIN might be effectively used [8, 15, 16, 26]. During the "Flash Crash," the return from the official beginning of the crisis (2.32pm) to the end of the session (4.15pm) was only -0.5%. However, during the same period, prices had an intermediate collapse of over -6%. The intermediate return is a much better indicator of trouble than the official return of the entire period. During the time period, the average return between any two consecutive volume buckets of size 40,000 contracts was -0.0003%, and their standard deviation was 0.8%. The standard return and volatility metrics based on these values do not capture the magnitude of the stop losses triggered during the "Flash Crash." The reason is that prices followed a characteristic V-pattern, however, those standard metrics are based on the assumption that sequential observations are statistically independent. Clearly, this independence assumption is violated in the cases of interest to us. We believe that the Maximum Intermediate Return (MIR) could correctly capture the stop losses of market participants during such extreme market events [16].

Given prices of N trades, $P_i, i = 0, 1, \ldots, N-1$, the intermediate return between trades j and k ($j < k$) is

$$R_{j,k} = P_k/P_j - 1. \tag{3}$$

Here is an example to illustrate how MIR is different from the typical uses of returns. Given a series of 51 trades where where the first 25 $R_{j-1,j} = -0.01$ and the last 25 $R_{j-1,j} = 0.01$. Most existing volatility measures use the average and the standard deviation of these sequential returns. In this case, the average return is 0 and the standard deviation is slightly greater than 0.01. In contrast, the maximum intermediate return is over 22%. We believe that this maximum intermediate return is a better metric to assess the volatility derived from failures of the liquidity provision process. We call a pair of trades, j^* and k^*, sentinels if they maximize the intermediate returns over all possible pairs of trades,

$$(j^*, k^*) = \underset{0 \le j < k < N}{\arg\max} \left| \frac{P_k}{P_j} - 1 \right|. \tag{4}$$

The *Maximum Intermediate Return* (MIR) is the intermediate return from the sentinels,

$$\text{MIR} = \frac{P_{k^*}}{P_{j^*}} - 1. \tag{5}$$

In words, MIR is the return (positive or negative) with largest magnitude between any two observations (not just sequential trades) within the sample of study. In this way, independent of whether prices follow a continuous trend, a V-pattern, or are pure noise, MIR correctly measures the maximum stop loss that a market participant may have experienced. For the reasons stated earlier, this is the metric of instantaneous volatility that matters to traders in general, and market makers in particular.

The straightforward approach to computing MIR would use a double nested loop to evaluate Equation 4. If we take the event duration to be 1 day long, then for many futures contracts, the event duration would include millions of trades. Therefore, the double loop for evaluating Equation 4 would need to compute trillions of different pairs of intermediate returns $R_{i,j}$ requiring a very significant amount of compute time. We could significantly reduce the amount of computation by considering the prices of buckets or bars instead of individual trades, because there are fewer buckets or bars. However, because the buckets and the bars are aggregates of many trades, using them will miss the actual maximum intermediate return in most cases.

To compute the exact MIR quickly, we employ two strategies. The first strategy is to have each volume bar carrying the maximum price and the minimum price of trades in the bar along with their positions in the list of trades. These values allow us to compute the true MIR in most cases; the exceptional cases will be explained after we describe our algorithm for computing MIR. The second strategy is to replace the double loop algorithm for computing MIR with a recursive procedure to be described next. After describing the algorithm, we show that the algorithm examines each price a small number of times on the average.

7.2 Algorithm for computing MIR

To simplify the description of the algorithm, we proceed as if the computation is performed using individual trades. It is straightforward to take into account of the fact we are actually using the minimum and the maximum prices of volume bars.

To determine MIR, we need to compute the largest and the smallest intermediate returns, $\overline{R}$ and $\underline{R}$. For clarity, we will call $\overline{R}$ the maximum intermediate gain and $\underline{R}$ the maximum intermediate loss. It is easy to see that MIR is either $\overline{R}$ or $\underline{R}$ depending on which one has the larger absolute value. Normally, $\overline{R}$ and $\overline{R}$ will have different signs, i.e., one positive and the other negative. We call the positive return a gain and the negative return a loss. In cases where the prices steadily rise or fall during the entire period under consideration, both $\overline{R}$ and $\underline{R}$ would have the same sign, i.e., both are positive or both are negative. It seems to counterintuitive to have a negative gain ($\overline{R} < 0$) or a positive loss ($\underline{R} > 0$). To avoid these unexpected values, we assign the one with the smaller absolute value to be 0. This choice does not affect the output of MIR, while at the same time it preserves our general notion of gain or loss.

Given N trades with prices $P_i, i = 0, 1, \ldots, N-1$. Since the prices are always positive numbers, i.e., $P_i > 0$, to find $\overline{R}$ we need to maximize the numerator and minimize the denominator in Equation 3; to find $\underline{R}$ we need to minimize the numerator and maximize the denominator in Equation 3. These observations suggest that we should locate the minimum and maximum prices within the event duration. An important fact to note about Equation 3 is that the earlier price is in the denominator and the later price in the numerator, i.e., $j < k$. Assuming P_v is the lowest price in the time duration, and $P^\wedge$ is the highest price within the time duration, if P_v appears earlier in time than $P^\wedge$, then the intermediate return $P^\wedge/P_v - 1$ must be the maximum intermediate gain $\overline{R}$, because no other combinations of prices could produce large intermediate return. While if P_v appears later in time than $P^\wedge$, then the intermediate return $P_v/P^\wedge - 1$ could be the maximum intermediate loss $\underline{R}$. There might be other intermediate returns with larger absolute value as we explain next.

Given two prices P_v and $P^\wedge$, such that $P_v < P^\wedge$, we have

$$\left| \frac{P^\wedge}{P_v} - 1 \right| > \left| \frac{P_v}{P^\wedge} - 1 \right| .$$

The implication of this relationship is that if we have found the maximum intermediate gain $\overline{R}$ from the maximum and minimum prices, then there it must be the MIR, however, if we have found the maximum intermediate loss $\underline{R}$, then it is possible to have a maximum intermediate gain with a larger absolute value even though the prices are neither the maximum price or the minimum price.

The maximum price and the minimum price within the event duration may appear multiple times. Based on the above observation, we need to locate the first appearance of the minimum price and the last appearance of the maximum price. This will maximize the chance that the minimum price appears before the maximum price, in which case we have determined the MIR. On the other hand, if the last appearance of the maximum price occurs before the first appearance of the minimum price, then we have computed the maximum intermediate loss $\underline{R}$ and need to search for the maximum intermediate gain $\overline{R}$.

Let $\overline{j}$ be the position of the last appearance of the maximum price and $\underline{j}$ be the position of the first appearance of the minimum price. Normally, $\overline{j} \neq \underline{j}$. The only possibility that the two positions might be the same is when all the values are the same. In which case, we can assign $\underline{j} = 0$ and $\overline{j} = N - 1$, and terminate the computation of MIR early.

We need to compute the maximum intermediate gain $\overline{R}$ when $\overline{j} < \underline{j}$. In this case, we need to break the N prices into three groups: (L) $j \leq \overline{j}$, (R) $j > \underline{j}$, and (M) $\overline{j} < j < \underline{j}$. The prices in the group (L) end with the maximum price. To compute the maximum intermediate gain from this group, we simply need to locate the minimum price in the group. This gives us the first candidate for the maximum intermediate gain $\overline{R_L}$. The prices in the group (R) start with the minimum price of the group, to compute the maximum intermediate gain we need to locate the maximum price of the group. This will give the second candidate for the maximum intermediate gain $\overline{R_R}$. The situation for the third candidate of the maximum intermediate gain $\overline{R_M}$ is similar to the task of computing the maximum intermediate gain over the whole set of prices in the time duration. Therefore, we can invoke the procedure for computing the maximum intermediate gain recursively. This recursive procedure terminates when there are only a small number of elements to be examined. In which case, we can use the double-nested loop procedure based on Equation 4.

When invoking the recursive procedure to compute $\overline{R_M}$, we have already computed $\underline{R}$, $\overline{R_L}$, and $\overline{R_R}$. In order for the $\overline{R_M}$ to change the overall MIR, it must be larger than $-\underline{R}$, $\overline{R_L}$, and $\overline{R_R}$. Inside the procedure for computing the maximum intermediate gain and after we have determined the maximum and minimum prices, we can provide a bound on the possible values of the maximum intermediate gain from the list of prices. Assuming P_v is the lowest price in the list of prices and $P^\wedge$ is the highest price, $\overline{(R_M)} \leq P^\wedge/P_v - 1$. If this upper bound is less than the one of $-\underline{R}$, $\overline{R_L}$, and $\overline{R_R}$, then we can terminate the procedure and return 0 because continuing the computation cannot affect the value of MIR.

A listing of the pseudocode for implementing this recursive procedure is given in Listing 3.

Listing 3: Pseudocode for computing MIR recursively

```
// Given an array P, find the first occurrence of smallest value pmin (↩
    store its position to imin) and the last occurrence of the largest ↩
    value pmax (store its position to jmax)
find_min_max(P, i, j, pmin, imin, pmax, jmax)
    imin = i; jmax = i; pmin = P[i]; pmax = P[i];
    for (++i; I < j; ++ i) {
        if (P[i] < pmin) pmin = P[i], imin = i;
        if (p[i] >= pmax) pmax = P[i], i1 = i;
    }

// Given an array P, find the minimum value between i and j
find_min(P, i, j)
    pmin = P[i]
    for (++i; i < j; ++i)
        if (P[i] < pmin) pmin = P[i]
    return pmin

// Given an array P, find the maximum value between i and j
find_max(P, i, j)
    pmax = P[i]
    for (++i; i < j; ++i)
        if (P[i] > pmax) pmax = P[i]
    return pmax
```

```

// Given an array P, find the maximum intermediate gain for values ↩
    between i and j.
max_intermediate_gain(P, i, j, thr)
    find_min_max(P, i, j, pmin, imin, pmax, jmax);
    res =     pmax/pmin - 1.0;
    // done
    if (imin <= i1) return res;
    // no need to continue
    if (res <= thr) return 0.0;
    // group L
    pmax0 = find_min(P, i, jmax);
    rl = pmax / pmax0 - 1;
    // group R
    if (thr < RL) thr = rl;
    pmin1 = find_max(P, imin+1, j)
    rr = pmin1 / pmin - 1;
    if (thr < RR) thr = rr;
    // group M
    rm = max_intermediate_gain(P, jmax+1, imin, thr);
    if (thr < RM) thr = rm;
    return thr;

// Given a list of prices, compute MIR
// Assume the number of elements in P to be np
compute_MIR(P)
    find_min_max(P, 0, np, pmin, imin, pmax, jmax);
    thr =     pmax/pmin - 1.0;
    // done
    if (imin <= i1) return thr;
    loss = pmin/pmax - 1.0;
    thr = - loss;
    // group L
    pmax0 = find_min(P, 0, jmax);
    rl = pmax / pmax0 - 1;
    if (thr < RL) thr = RL;
    // group R
    pmin1 = find_max(P, imin+1, np);
    rr = pmin1 / pmin - 1;
    if (thr < RR) thr = RR;
    // group M
    rm = max_intermediate_gain(P, jmax+1, imin, thr);
    if (thr < RM) thr = rm;
    if (thr > -loss)
        return thr;
    else
        return loss;
```

Next, we briefly discuss the use of bars instead of individual trades for computing MIRs.

In our implementation of Listing 3, we actually use bars with minimum and maximum prices instead of individual trades. It is straightforward to modify the functions `find_min`, `find_max`, and `find_min_max` to use the information in the bars to find the trades that have the minimum and the maximum trade prices. When the two extreme prices do not appear in the same bar, then the above algorithm can proceed as described. When the two extreme prices do appear in the same bar, we need to distinguish two cases. When the minimum price comes before the maximum price, we have reached the maximum intermediate gain and would terminate the procedure. However, when the minimum price comes after the maximum price, we have information to compute the maximum intermediate loss, but no way to determine whether or not we have missed an intermediate gain with a larger absolute value than the maximum intermediate loss. It is theoretically possible for us to miss the correct MIR using the bars even with the minimum and the maximum prices. In our tests, we have found this not to be a concern. Shall this not be the case in any other use cases, the user would have to compute MIR from individual trades. In which case, it would be even more important to use the algorithm outlined in Listing 3 because the number of trades in an event duration would generally be much larger than the number of bars.

Since the MIR actually has both positive and negative values, when we say the volatility is high, we generally mean that the MIR value has a large absolute value. In our implementation, we separate out the positive MIR values from the negative ones, and compute the average in these two groups separately. If an event produces a positive MIR, its value has to be larger than the average of all positive MIR values in order to be a true event. If an even produces a negative MIR, its value has to be less than the average of all negative MIR values in order to be a true event. Any events with an MIR value between the two average values are considered as false positives.

7.3 Computational cost

It is easy to see that the double nested loop implementation of Equation 4 will read N prices $N(N+1)/2$ times. In computational complexity theory, this is known as an order N^2 algorithm. Next we outline a set of arguments to show that the pseudocode given in Listing 3 reads N prices order N times. In other word, we have designed a linear algorithm. When N is large, a linear algorithm can be arbitrarily faster than an order N^2 algorithm. Therefore, it is worthwhile to establish this fact clearly.

Assume the prices given to the above procedure are randomly distributed and there is only one minimum price and one maximum price, then the position of the minimum price and the maximum price are statistically independent. In terms of computational cost, we see that the function `compute_MIR` and `max_intermediate_gain` each has four parts: (1) a single pass through all incoming elements to compute the minimum and the maximum prices, (2) a loop through the group L, (3) a loop through the group R, and (4) a recursive call to another function on group M. Given N input prices, loop 1 reads all N elements of the array and the total cost can be counted as N. If the minimum price appears before the maximum prices, the computation terminates. The probability of this is $1/2$ because of the independence assumption. Based on the same assumption, the average numbers of elements in the three groups are the same, $N/3$. The cost of looping through group L and group R is $2N/3$. The cost of recursion is basically repeating the current procedure but on $N/3$ prices rather than N of them. Thus the total cost is

$$\begin{aligned} T(N) &= N + \frac{1}{2}\left(\frac{2N}{3} + T(N/3)\right) = \frac{4N}{3} + \frac{1}{2}T(N/3) \\ &= \frac{4N}{3} + \frac{1}{2}\frac{4}{3}\frac{N}{3} + \frac{1}{2}\frac{1}{2}T(N/9) = \frac{4N}{3}\left(1 + \frac{1}{6} + \frac{1}{36} + \ldots\right) \\ &= \frac{4N}{3}\frac{6}{5} = 8N/5 \qquad (6) \end{aligned}$$

Had the function for computing the maximum intermediate gain simply gone through the group M once, the expected cost of computing MIR would have been $N + N/2$. The recursive calls to compute the maximum intermediate gain on group M increases the total cost. However, because the recursive invocation only happens half the time and only on an input array of $1/3$ the size, we see that the total cost of computing MIR is only slightly larger than $N + N/2$. On average, we read each input array element 1.6 times.

The above analysis is for the average case, it is possible that the recursive procedure needs to be invoked on a large fraction of the input values, for example when the input values are in the descending order. In this case, $\overline{j} = 0$, $\underline{j} = N - 1$, and the group M has $N - 2$ elements. This recursion may continue for $N/2$ times, which will take N^2 time altogether. However, in this case, our algorithm will terminate at some point, because the upper bound on the maximum intermediate gain will become less than the absolute value of the maximum intermediate loss $\underline{R}$, therefore, avoid using N^2 execution time.

8 Parallelization

The process of computing VPIN and MIR for each futures contract can proceed independently. For example, computing VPIN values on ES does not require any information from any other futures contracts. This allows us to run a number of parallel tasks, one for each contract instruments. This parallelization strategy is often referred to as the "task parallelism" in High-Performance Computing literature [27, 23]. Given that we have all the trading records, we could further parallelize the computation procedures by dividing the trading records from different time periods as different tasks. However, it is more straightforward to handling all computations involving on contract instrument in a single parallel task than splitting the computation onto multiple processors and the time required for each task is relatively short as we will see in the next section. we anticipate that in a real-time setting each parallel task is likely to be handling one contract instrument. Therefore, it is more useful to measure its computational cost.

Our parallel implementation uses POSIX threads [30, 23], which is more widely supported than alternatives such as MPI [22] and Hadoop [41]. The parallel job spawns a user-specified number of parallel processes. The parallel processes share a common task list, which holds the list of contract instruments [31]. When a process finishes its assigned task, it takes the next available task from this shared list. The parallel job completes when the last process finishes its assigned tasks.

Since different contract instruments have wildly different number of trades, ES has 478 million trades while MW has only 33 thousand trades, each parallel task may require very different amount of compute time. To minimize the time spent by the parallel job, we need to ensure the tasks assigned to each parallel process complete in the same amount of time. In our case, the computation time for each task is a linear function of the number of trades for the contract instrument, we choose to assign the instrument with the largest number of trades first. This strategy also work well for other computational tasks where the cost is a monotonic function of the input size.

Our program is implemented in C++ [37], which is better suited for High-Performance Computing than more commonly used scripting languages such as Python [39]. However, Python and similar languages are also used to process massive amounts of data, and many of them are able to make use of POSIX pthreads for parallel computations as well.

9 Experimental evaluation

In this section, we present the test results using the 94 most active futures contracts. Our computation procedure will first produce volume bars, group the bars into buckets, and then compute the VPIN values. In this process, we have a number of different parameters to choose. The six parameters we test are listed in

Table 1: Parameters to control the computation of VPIN and MIR. The six parameter choices form 16,000 different combinations. When any of them is not specified, the following default values are used unless otherwise specified: π = closing, $\beta = 200$, $\sigma = 1$, $\eta = 1$, normal distribution for BVC, $\tau = 0.99$.

	Description	**Choices to consider**
π	Nominal price of a bar	Closing, average, weighted average, median, weighted median
β	Buckets per day	50, 100, 200, 500, 1000
σ	Support window	0.25, 0.5, 1, 2 (days)
η	Event duration	0.1, 0.25, 0.5, 1 (days)
ν	Parameter for BVC	Normal distribution or Student t-distribution with ν = 0.1, 0.25, 1, 10
τ	Threshold for CDF of VPIN	0.9, 0.93, 0.95, 0.96, 0.97, 0.98, 0.99, 0.999

Table 1. In this section, we first describe the test machine we used, describe a way to verify that the VPIN events are indeed different from random selection of time points, and move to have more quantitative way to measure the difference between MIRs described above. Table 1 shows 6 parameters with four or five different choices each, with a total of 16,000 different combinations. We will use these parameter choices to demonstrate the effectiveness of our program and to identify the parameter combinations that produce the lowest false positive rates.

9.1 Test setup

We conducted our tests using an IBM DataPlex machine located at the National Energy Research Supercomputing Center (NERSC)[6]. The HDF5 files are stored on the file system generally known as the global scratch space, which is a GPFS system exported to all NERSC machines. This file system is heavily used and therefore we observed significant variations in the time needed to read the same data file. For example, the file ES.h5 is about 5.7GB in size. The minimum observed time for reading this data file is 45.9s, the median time is 48.8s and the average time is 50.4s. We see that the average time is almost 4.5s longer than the minimum. The average value is larger than the median, which indicates there are large values that are significantly larger than other. We performed these reading operations always using a single thread. The median reading speed is about 117MB/s and the fastest reading rate is about 125MB/s. These reading rates are typically of what can be achieved using a single thread. Generally, by using more computer nodes to read the same file, we could achieve higher aggregate reading speed.

9.2 Time breakdown

To understand where the compute time is spent in the C++ program, we measure the time used by the key functions. In Figure 3, we show four such functions: reading the data from the HDF5 file into memory, constructing the volume bars, computing the VPIN values (including forming the buckets), and computing the MIR values. Overall, we see that the time required to read the data into memory dominates the total execution time. The next largest amount of time is spent on constructing the volume bars. In this case, we are using the closing price as the nominal price of the bar, therefore, the time to form the bars is relatively

[6]Information about NERSC available online at `http://nersc.gov/`

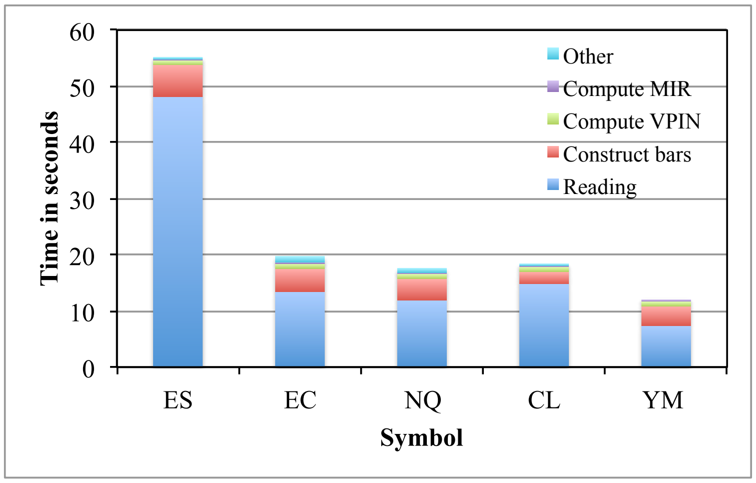


Figure 3: A breakdown of time (seconds) spent in major steps of the C++ program. The volume bars are constructed with their closing prices as their nominal prices. The time to compute to MIR includes the time to compute the MIR values of 10,000 random events.

small. Had we used the median price or the weighted median price, the time to form the volume bars would be even longer according to Figure 2.

The time needed to compute VPINs and MIRs are about the same in all five cases. This is because those steps are operating on about the same number of objects. The input for computing VPINs is the volume bars. Because the number of volume bars is about the same in each trading day, and the number of trading days is the same for these five futures, they produce similar number of buckets and about the same number of VPINs. The numbers of VPIN events are different, but they are relatively few compared to the number of random events used as references. Our program always uses 10,000 random events as references. The time for computing MIR values includes the time to compute MIRs for both VPIN events and reference events. Since each event contains the same number of volume bars, the time to compute MIRs is roughly the same. The time for computing MIR is relative small, we might feel the effort spent in producing a recursive algorithm was not worth it. However, in these cases, the number of bars in an event is 6,000. The simple double loop algorithm for computing MIR would have taken thousands of times longer, which would have made the time to compute MIR the dominant operation. Therefore the effort to reduce the time for computing MIRs was necessary.

The test job that runs over all 16,000 combinations listed in Table 1 as well as over all 94 futures contracts took 19 hours 26 minutes and 41 seconds using a 32-core computer node and about 74GB of memory. In this case, we read the input data into memory once and repeat the different combinations of parameters without reading the data again. This reduces the I/O times and make it much more efficient to try different combinations. The average time to compute VPIN, detect VPIN events, and compute the false positive rates is 1.49 seconds on a single CPU core,

$$\frac{32 \times (19 hours + 26 minutes + 41 seconds)}{16000 \times 94} = 1.49 seconds.$$

In other words, one CPU core can process 67-month data in 1.5 seconds. This speed is able to keep up with the real-time data stream of trading data.

A python version of this program that stores bars and buckets in a SQLite database took more than 12

Table 2: Days of VPIN events on ES. A total of 36 different VPIN events are detected with $\tau = 0.99$. On a couple of days, multiple events are detected.

Year	Month	Days	Notable event
2007	08	16	Coutrywide Financial liquidity crunch
2008	01	22, 23	FOMC warns weak outlook
2008	09	29	Dow drops 778 points
2008	10	06, 07, 08, 10, 13, 14, 16, 17, 20, 22, 23, 24, 27, 28, 29, 31	Stock Crash 2008
2008	11	13, 14, 17, 20	Global financial crisis
2010	05	06	Flash Crash
2011	08	09, 10, 11	Downgrading of US's credit rating

Table 3: The number of VPIN events E and the probability p determined by the Kolmogorov-Smirnov test that the sequence of the VPIN events follows the same distribution as random events.

	ES		EC		NQ		CL		YM	
Nominal prices	E	p	E	p	E	p	E	p	E	p
Closing	26	$2e{-}6$	32	$5e{-}7$	24	$2e{-}5$	36	$5e{-}4$	32	$1e{-}6$
Average	27	$5e{-}9$	25	$4e{-}4$	22	$4e{-}5$	36	$4e{-}5$	28	$2e{-}10$
Weighted average	27	$5e{-}6$	27	$4e{-}6$	22	$7e{-}5$	36	$7e{-}6$	27	$6e{-}10$
Median	26	$2e{-}6$	36	$6e{-}7$	24	$2e{-}6$	35	$2e{-}5$	28	$6e{-}6$
Weighted median	25	$2e{-}7$	32	$6e{-}6$	22	$2e{-}4$	36	$5e{-}6$	30	$6e{-}10$

hours to complete one parameter combination on ES, while the C++ version took about 60 seconds on the same machine. The C++ program is about 720 times faster than the python program.

9.3 Events detected

Table 2 shows the days of VPIN events on ES. To show that the detected events correspond to notable real-world events, we have listed some important market events that occurred on the same day. Since these events are known to cause high volatility in the markets, we see that VPIN events are correlated to these high volatility events.

9.4 Kolmogorov-Smirnov test

Now we turn our attention to examine the effectiveness of VPIN. We first demonstrate that VPIN events produce a different sequence of MIRs compared to random events. To do this, we apply the widely used Kolmogorov-Smirnov test on the two sequences, and report the probability that they follow the same distribution in Table 3. The first thing we observe from Table 3 is that all the probability values are less than 10^{-4}, which indicates that the two sequences of MIR values are extremely unlikely to be drawn from the same probability distribution. These probability values are representative of all futures weve tested. In other words, the events identified by VPIN values are extremely different a sequence of randomly selected events.

The tests shown in Table 3 use the CDF threshold of 0.99, which means that if VPIN values follow

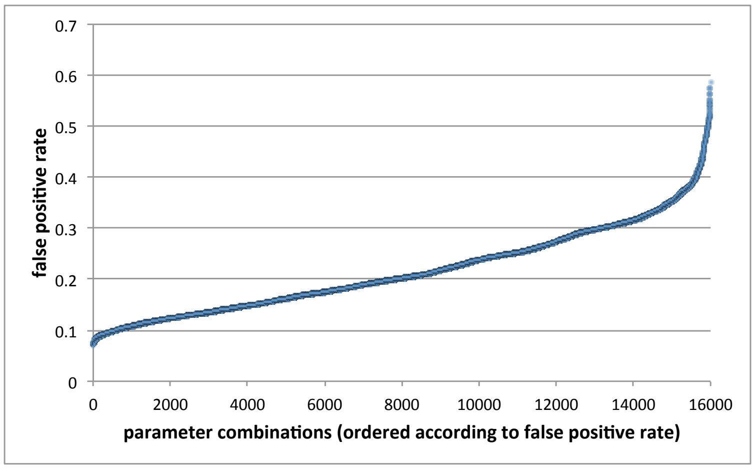


Figure 4: The avearge false positve rates of 94 futures contracts under different VPIN parameter choices. The 16,000 parameter choices are ordered according to their false positive rates.

Table 4: the 10 parameter combinations that produced the smallest average false positive rates, α.

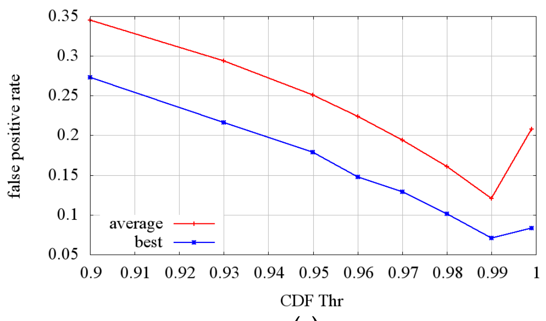
π	β	σ	μ	ν	τ	α
Median	200	1	0.1	1	0.99	0.071
Weighted median	1000	0.5	0.1	1	0.99	0.071
Weighted median	200	0.5	0.1	0.25	0.99	0.072
Weighted median	200	0.5	0.1	1	0.99	0.073
Median	200	1	0.1	10	0.99	0.073
Median	500	0.5	0.1	0.1	0.99	0.074
Median	200	1	0.1	Normal	0.99	0.074
Weighted median	200	1	0.1	1	0.99	0.074
Weighted median	200	1	0.25	1	0.99	0.074
Weighted average	200	1	0.1	1	0.99	0.075

the lognormal distribution, we expect 1% of the VPIN values would be larger than the threshold. There are about 300,000 VPIN values computed in each case, 1% of which would be 3,000. Clearly, the actual number of events is far less than 3,000.

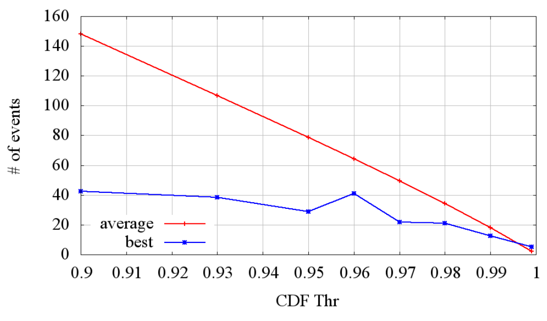
9.5 False-positive rates

To determine whether an event is a true event or a false positive, we use the average MIR from a set of randomly selected time windows as the reference for comparison. In our test program, we always choose 10,000 random events to compute the averages of the positive MIR values and negative MIR values.

The choices of parameters given in Table 1 define 16,000 different combinations. We apply all these 16,000 combinations to all 94 different futures contracts. For each of the 16,000 combinations, we compute the average false positive rate of the 94 test cases, one for each futures contract. Figure 4 shows the average false positve rates of 16,000 different parameter combinations. There are 604 combinations with $\alpha < 0.1$, and 70 combinations with $\alpha > 0.5$. The median value of the false positive rates is about 0.2 (i.e., 20%). Clearly, it is worthwhile to carefully choose the parameters for VPIN.



(a)



(b)

Figure 5: The false positive rates (left) and the number of VPIN events (right) detected with different choices of CDF threshold values.

Table 5: The best parameter combinations under different CDF thresholds.

π	β	σ	μ	ν	τ	α
Closing	1000	2	0.1	1	0.9	0.2734
Weighted median	200	2	0.1	0.25	0.93	0.2162
Median	200	2	0.1	10	0.95	0.1786
Weighted median	200	1	0.1	1	0.96	0.1474
Median	100	2	0.1	10	0.97	0.1292
Weighted median	500	1	0.25	0.25	0.98	0.1013
Median	200	1	0.1	1	0.99	0.0709
Average	100	0.25	0.1	0.25	0.999	0.0832

Table 4 shows the ten combinations that produced the lowest average false positive rates over all 94 futures contracts. From Table 4 we observe that each of the ten combinations produces an average false positive rate of around 7%. Given the diversity of the data, these false positive rates are remarkably low. The 7% false postive rate is about 1/3rd of the median value ($\sim$ 20%) of all false positve rates in our test. Next, we examine the parameter combinations listed in Table 4 in more detail.

To achieve the lowest false positive rates, we could use the values from the first row in Table 4. However, it is not immediately obvious that this choice would be good for other type of data or the best choice would remain to be the best had we tested more parameter combinations. Here we attempt to discover the patterns in the parameters that achieve good performance in Table 4.

Among the ten best parameter combinations, we see that the threshold τ on CDF is always 0.99, the event duration η is 0.1 days in nine out of the ten cases, the number of buckets per day β is 200 in eight cases, the support for computing VPIN σ is 1 day in six cases, the bulk volume classification parameter ν is 1 in six cases, and the Weighted Median is the preferred pricing method for volume bars in five cases. Based on this observation, we say that the value of τ has strongest influence on the false positive rate. A different value of τ would strongly affect the false positive rate. In contrast, the pricing method π seems to be the least influential; it could take a different value in five of the ten cases while still achieving very good false positive rates. Next, we study these two parameters in more detail. It would be interesting to study the other four parameters more carefully as well, but we will leave that for a future study.

Table 6: Statistics on prices for volume bars and the resulting VPIN values over all trades of ES.

Pricing Method		Closing	Average	Weighted Average	Median	Weighted Median
Price changes	Mean	$-5e-6$	$-5e-6$	$-5e-6$	$-5e-6$	$-5e-6$
	Standard Deviation	0.231	0.184	0.181	0.209	0.205
VPIN	Mean	0.072	0.090	0.092	0.077	0.078
	Standard Deviation	0.020	0.023	0.024	0.022	0.23

9.6 CDF Thresholds

Figure 5 shows the false positive rates and the number of VPIN events detected under different CDF thresholds (τ). Table 5 shows the parameters combinations that give rise to the lowest α under different choices of τ. The red line in Figure 5 marked as average is the average over all 2,000 combinations of the other five parameters and 94 different futures contracts, and the blue line marked as best is produced from the combination of the other five parameters that produces the smallest average false positive rate over all 94 futures contracts. If the average is close to the best, then the choices of the other five parameters do not have a large influence on the false positive rates. In our tests, the differences between the average rates and the best rates range from about 7% when τ=0.9 to about 5% when τ=0.99. The exception is that when τ=0.999, the difference is 12.5%. Given that the false positive rate is 27% when τ=0.9, a difference of 7% is relatively small. This agrees with our previous observation that τ has a much stronger influence on the false positive rates than the other parameters.

To understand how the CDF threshold affects the false positive rates, we also plotted the number of VPIN events detected in Figure 5. Generally, fewer events are detected with higher CDF thresholds. Had the VPIN values followed the log-normal distribution as assumed, the number of events should be proportional to $1-\tau$ and should descrease linearly as τ increases. We see the "average" number of events is pretty close to be decreasing linearly as τ increases. However, with the best parameter combinations, the number of events detected descreases much slower as τ increases. This suggests that the "average" parameter choices are more reflecting the "random" fluctuation in the VPIN values, which the "best" parameter choices are much better in picking up the buy-sell imbalance reflected in the test data.

From our test results, we see that τ=0.99 or 0.98 might be reasonable choices. Either of these choices produces less than 10% false positive events and detects around 20 events per futures contract. The false positive rates are pretty low and the number of events is small enough that a human expert might be able examine each event to verify the validity of the signals.

9.7 Pricing the bars

Previous publications on VPIN generally use the closing price as the price of a bar [1, 10, 16]. Because the prices of trades in a bar are typically quite close to each other, it is somewhat surprising that computing prices differently can affect the VPIN events detected. Figure 6 shows the prices computed using different methods in a brief time window where the price of ES changes quickly. We see that the prices computed are quiet close to each other, which agree with our expectation.

To see how the small differences between the prices from different pricing methods lead to different VPIN values, we recall that the VPIN values critically depend on sequential price changes and their standard

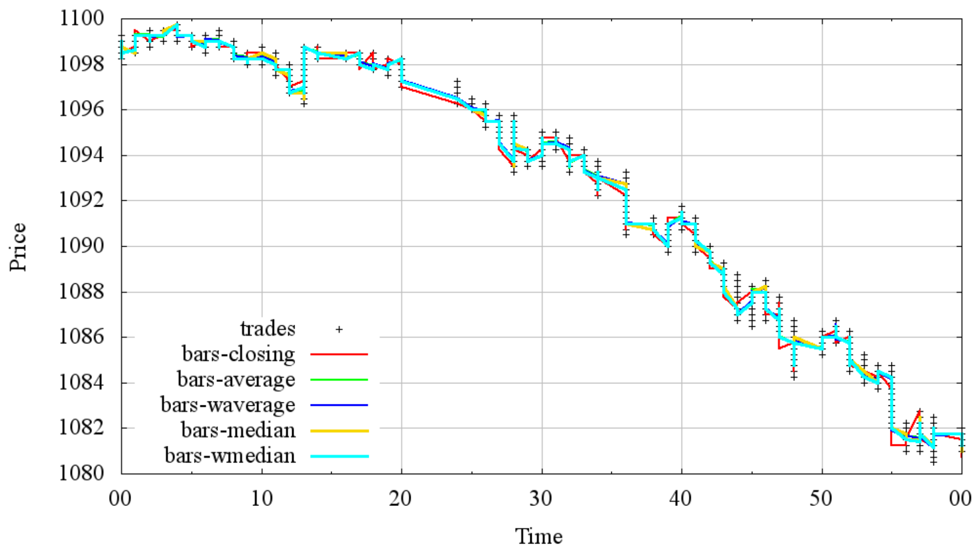
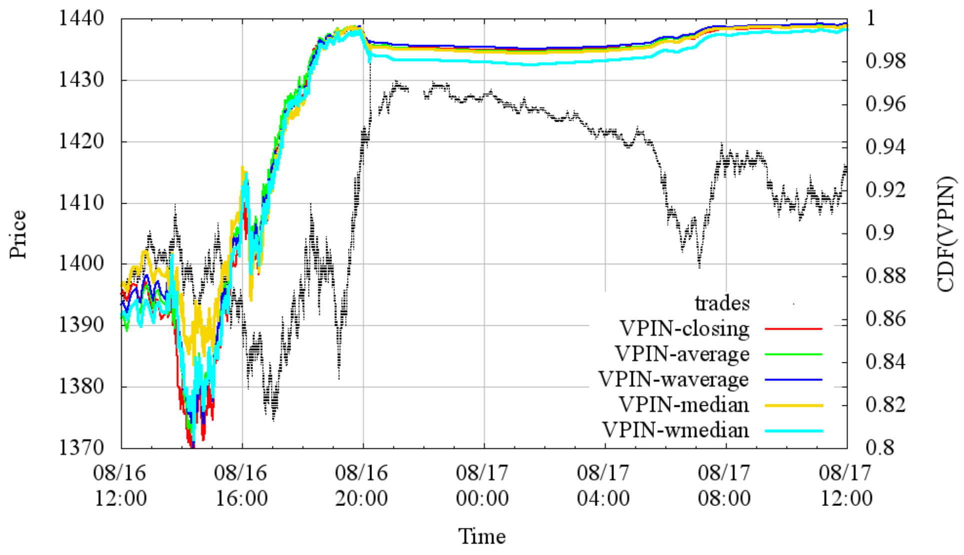


Figure 6: The nominal prices of volume bars of ES in a one-minute interval with a sharp price drop. All five pricing strategies produce similar values even during this period of dramatic price changes.

deviation. These sequential price change statistics affect the Bulk Volume Classification (BVC) and the projected imbalance between buy and sell that determine the value of VPIN. Table 6 lists the basic statistics about the sequential price changes and the resulting VPIN values. We can see that the average price changes are very small compared to the actual prices of ES. From Figure 6 we see the price of ES is around 1000, while the average price changes are about 10^{-6}, such differences are too small to be observed in Figure 6. The standard deviations of the price changes are about 0.2, which are five orders of magnitude larger than mean sequential price changes. In addition, the differences among the standard deviations are also larger than the difference among the mean sequential price changes. Since the standard deviations of the price changes have a more directly influence on the computation of BVC (cf Eq. 1), and therefore VPIN. We see more noticeable differences among VPIN computed with different pricing methods as shown in Table 6. The standard deviations of VPIN values are around 0.02 and the difference among the average VPIN values are nearly one standard deviation apart from each other. Clearly, the VPIN values should be considered different.

Next, we take a detailed look at two specific VPIN events to see if there are general patterns in the VPIN values. Figure 7 shows the VPIN values for two different days where VPIN values reach above 0.99, August 16, 2007 and September 29, 2008. In these cases, we see the CDF (and therefore VPIN) values are generally quite close to each other, especially, when they are close to 1 and when their are rising quickly. This suggests that the different pricing methods should have nearly the same power in detecting VPIN events. In other words, the VPIN values computed using different pricing methods should reach the threshold for VPIN events at nearly the same time.

To see the difference among the onset time of VPIN events detected with different price methods, we show the time of day of all VPIN events detected with the threshold of 0.99 in Figure 8. In this figure, the vertical axis is the time of the day while the horizontal axis is the day of an event. Each vertical column of points indicates a set of VPIN events on the same day. The most noticeable feature in the figure is that there are always multiple different points clustered together for each day, which indicates that different pricing



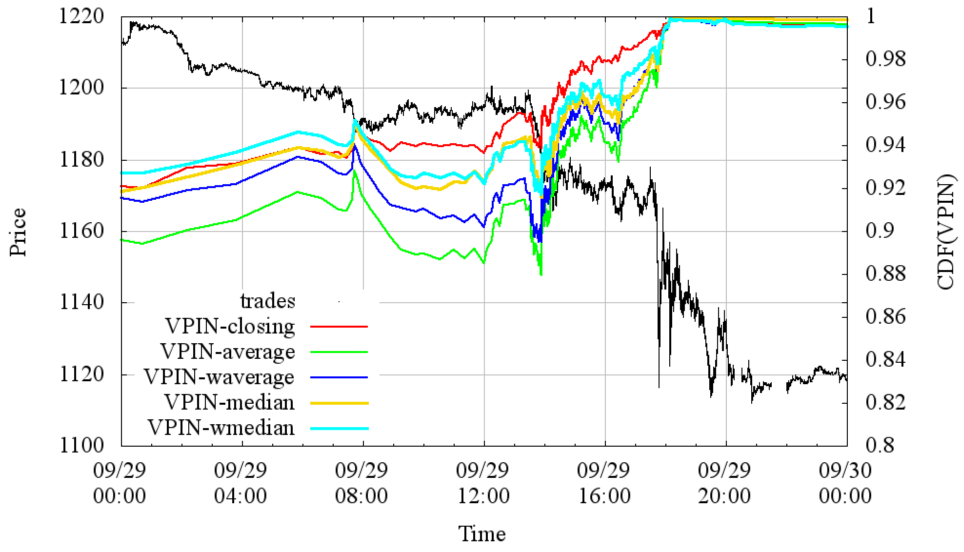


Figure 7: The VPIN values computed from different pricing strategies for two time periods around 8/16/2007 and 9/29/2008.

methods lead to the detection of the same event. Therefore, the two events shown in Figure 7 are common cases. In most cases, the onset time of VPIN events are less than 2 hours apart. In fact, in many cases, the symbols representing the different pricing methods are right on top of each other indicating the onset time is within minutes of each other. This suggests that VPIN is detecting events based on underlying information imbalance not the details of the pricing methods.

Clearly, the details of the pricing methods actually matter because not all pricing methods lead to the same set of events. Furthermore, a number of the VPIN events have wildly different onset time. Additional study of these pricing strategies is necessary to understand these differences.

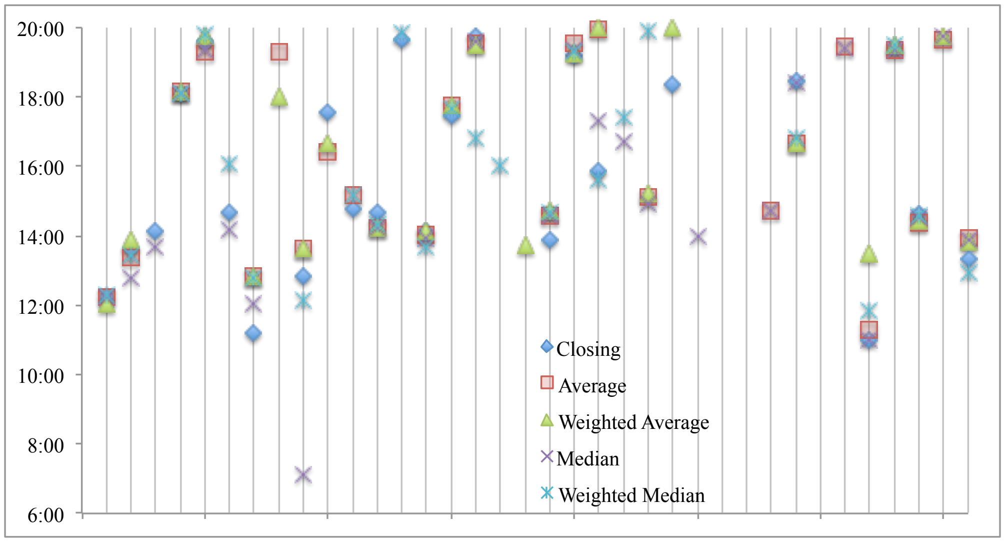


Figure 8: Time of day of VPIN events detected with different pricing methods for volume bars on ES. The horizontal axis is the day of a VPIN event and the light vertical lines mark each day listed in Table 2.

10 Summary

In this study we set out to apply the HPC experience gathered over decades of Big Data analysis by the National Laboratories on a set of data analysis tasks on high-frequency market data. While some financial practitioners, like NANEX, use these techniques routinely, however there are still relative few research publications on handling massive amounts of data. We have demonstrated an interesting use case for HPC techniques in this paper.

We implemented a popular liquidity metric known as VPIN on a large investment universe of nearly 100 most liquid futures contracts over all asset classes. To our knowledge, this is the first microstructure study to analyze such a vast investment universe. In doing so, we demonstrated that the technology required to monitor real-time liquidity conditions over all markets already exists, and can be deployed with a modest investment.

With our approach, we were able to quickly examine 16,000 different parameter combinations for evaluating the effectiveness of VPIN on all liquid futures traded in 67 months, on all asset classes. Our results confirm that VPIN is a strong predictor of liquidity-induced volatility, with false positive rates as low as 7%. Here recommended parameter choices to achieve this performance: pricing the volume bar with the median prices of the trades, buckets per day = 200, bars per bucket = 30, support window for computing VPIN = 1 day, event duration = 0.1 day, bulk volume classification with Student t-distribution with $\nu = 0.1$, and threshold for CDF of VPIN = 0.99. This set of parameters produce an observed false positive rate that is one third of that median value achieved among the 16,000 test cases.

This HPC task took about 20 hours on a single computer node at the National Energy Research Supercomputing Center (NERSC), many thousands times faster than commonly used alternatives. This speed up is achieved by using (1) a more efficient data organization, (2) more efficient data structures and algorithms for computing VPIN and related quantities, (3) better utilization of computer resources through parallelization. We believe that these techniques are useful to many data analysis challenges posed by todays high-frequency markets.

11 Acknowledgements

The authors would like to thank Dr. Marcos Lopez de Prado for valuable discussions on computing VPIN and for insightful information about the financial markets. Without his help, we would not be able to start nor to complete this work. We are grateful to TickData for providing the large data set used in this study. We would also like to thank Professors David Easley and Maureen O'Hara for providing valuable comments on an early draft of this paper.

This work is supported in part by the generous donations to the Center of Innovative Financial Technologies (CIFT) at Lawrence Berkeley National Laboratory, and by the Director, Office of Laboratory Policy and Infrastructure Management of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This work used resources of The National Energy Research Scientific Computing Center (NERSC).

A List of futures contracts

Table 1 lists the contracts in out study; we list their descriptions, rolling rules, trading venues, classes as well as the total number of trades (in thousands). The rolling rules are as follows:

1. An integer value in the "Roll" column indicates the number of days before expiration when we have applied the roll.

2. When the "Roll" column registers a * followed by an integer value, it means that the roll was applied that number of days before expiration, unless the volume shifted to the next serial contract before that day.

3. A zero value in the "Roll" column indicates that the roll date was purely determined by the transfer of volume from one contract to the next.

Table 7: List of futures contracts and their total volume of trades from January 2007 to July 2012 (in thousands).

Symbol	Description	Roll	Exchange	Class	Volume (000)
AD	Australian Dollar	11	CME	Currency	63,595
AX	Australian 10 yr bond	0	ASX	Interest rates	3,889
AY	Australian 3 yr bond	0	ASX	Interest rates	3,159
BL	Euro-bobl 5 yr	*28	Eurex	Interest rates	27,228
BN	Euro-bund 10 yr	*28	Eurex	Interest rates	53,292
BO	Soybean oil	*20	CBOT	Grain	17,796
BP	British Pound	10	CME	Currency	73,695
BR	Brazilian Bovespa futures	0	BM&F Bovespa	Equity	5,582
BZ	Euro-schatz 2yr	*28	Eurex	Interest rates	16,136
CB	Canadian 10 yr	*27	MSE	Interest rates	8,276
CC	Cocoa ICE	*15	ICE	Softs	4,733
CD	Canadian Dollar	10	CME	Currency	52,101
CF	CAC 40 index futures	0	NYSE Liffe	Equity	56,521
CL	Light Crude NYMEX	*19	NYMEX	Energy	165,208
CN	Corn	*20	CBOT	Grain	41,954
CO	Brent Crude ICE	0	ICE	Energy	79,182
CT	Cotton #2	*20	ICE	Softs	4,494
DA	Dax futures	17	Eurex	Equity	97,337
DJ	DJIA Futures	12	CBOT	Equity	596
DX	Dollar index ICE	1	ICE	Currency	11,352
EC	Euro FX	10	CME	Currency	188,837
ED	Eurodollar	*15	CME	Interest rates	11,864
EN	Nikkei 225 Futures	0	SGX	Equity	26,729
ES	S&P 500 E-mini	12	CME	Equity	478,029
FC	Feeder cattle	*29	CME	Meat	1,181
FT	FTSE 100 index	21	NYSE Liffe	Equity	54,259

FV	T-Note 5 yr	*28	CBOT	Interest rates	59,830
GC	Gold Comex	*27	COMEX	Metal	62,875
GL	Long gilt	5	NYSE Liffe	Interest rates	16,353
GO	Gasoil	0	ICE	Energy	15,858
HG	Copper High Grade	*27	COMEX	Metal	12,393
HI	Hang-Seng index futures	28	Hong Kong Stock Exchange	Equity	55,812
HO	Heating Oil #2	*27	NYMEX	Energy	22,740
IB	IBEX 35	18	MEFF Renta Variable	Equity	16,791
II	FTSE MIB	18	Borsa Italiana	Equity	16,775
JA	Platinum	0	Tokyo Commodity Exchange	Metal	51
JB	Japanese 10 yr bond	12	Tokyo Stock Exchange	Interest rates	5,401
JE	Japanese Yen E-mini	10	CME	Currency	221
JG	Gold	0	Tokyo Stock Exchange	Metal	136
JO	Orange Juice	1	ICE	Softs	690
JY	Japanese Yen	10	CME	Currency	76,006
KC	Coffee ”C”	*20	ICE	Softs	5,910
KE	Korean 3 yr	20	Korea Exchange	Interest rates	3,707
KM	KOSPI 200	0	Korea Exchange	Equity	46,121
LB	Lumber	*24	CME	Softs	330
LC	Live cattle	*29	CME	Meat	6,945
LH	Lean Hogs	*24	CME	Mean	5,521
MD	S&P 400 MidCap	12	CME	Equity	42
ME	Mexican Peso	10	CME	Currency	7,782
MG	MSCI EAFE Mini	12	NYSE Liffe	Equity	2,022
MI	S&P 400 MidCap E-mini	12	CME	Equity	28,266
MS	Soybeans E-mini	*20	CBOT	Grain	58
MW	Wheat E-mini	*20	CBOT	Grain	33
ND	Nasdaq 100	12	CME	Equity	788
NE	Nikkei 225 Futures	0	Osaka Securities Exchange	Equity	8,519
NG	Natural Gas	0	NYMEX	Energy	50,847
NK	Nikkei 225 Futures	0	CME	Equity	6,048
NQ	Nasdaq 100	12	CME	Equity	173,211
NZ	New Zealand Dollar	11	CME	Equity	3,809
OA	Oats	*20	CBOT	Grain	410
PA	Palladium	*26	NYMEX	Metal	1,160
PB	Pork Bellies	*29	CME	Meat	69
PL	Platinum	*26	NYMEX	Metal	2,086
PT	S&P Canada 60	9	MSE	Equity	11,374
QG	Natural Gas E-mini	0	NYMEX	Energy	2,167
QM	Crude Oil E-mini	*19	NYMEX	Energy	11,436
RL	Russell 2000	12	ICE	Equity	91
RM	Russell 1000 Mini	12	ICE	Equity	418

RR	Rough Rice	*20	CBOT	Grain	334
SB	Sugar #11	*25	ICE	Softs	16,761
SF	Swiss Franc	10	CME	Currency	34,252
SM	Soybean meal	*20	CBOT	Grain	13,890
SP	S&P500	12	CME	Equity	6,142
ST	Sterling 3 months	*15	NYSE Liffe	Interest rates	1,765
SV	Silver	*27	COMEX	Metal	24,375
SW	Swiss Market Index	13	Eurex	Equity	18,880
SY	Soybeans	*20	CBOT	Grain	35,247
TP	TOPIX	0	Tokyo Stock Exchange	Equity	8,416
TS	10-year interest rate swap futures	*28	CBOT	Interest rates	41
TU	T-Note 2 yr	*28	CBOT	Interest rates	24,912
TW	MSCI Taiwan	27	SGX	Equity	24,533
TY	T-Note 10 yr	*28	CBOT	Interest rates	95,793
UB	Ultra T-Bond	*28	CBOT	Interest rates	9,341
UR	Euribor 3 months	*15	NYSE Liffe	Interest rates	3,747
US	T-Bond 30 yr	*28	CBOT	Interest rates	57,588
VH	STOXX Europe 50	13	Eurex	Equity	196
WC	Wheat	*20	CBOT	Grain	21,159
WT	WTI Crude	0	ICE	Energy	18,164
XB	RBOB Gasoline	*27	NYMEX	Energy	17,575
XG	Gold mini-sized	*27	NYSE Liffe	Metal	3,223
XP	ASX SPI 200	13	ASX Group	Equity	16,716
XX	EURO STOXX 50	13	Eurex	Equity	80,299
YM	Dow Jones E-mini	12	CBOT	Equity	110,122
YS	Silver mini-sized	21	NYSE Liffe	Metal	1,434

References

[1] David Abad and José Yagüe. From PIN to VPIN: An introduction to order flow toxicity. *The Spanish Review of Financial Economics*, 10(2):74 – 83, 2012.

[2] Anat R Admati and Paul Pfleiderer. A theory of intraday patterns: Volume and price variability. *Review of Financial Studies*, 1(1):3–40, 1988.

[3] Yacine Aït-Sahalia, Per A Mykland, and Lan Zhang. How often to sample a continuous-time process in the presence of market microstructure noise. *Review of Financial Studies*, 18(2):351–416, 2005.

[4] Yakov Amihud, Haim Mendelson, and Maurizio Murgia. Stock market microstructure and return volatility: Evidence from italy. *Journal of Banking & Finance*, 14(2):423–440, 1990.

[5] Torben G Andersen and Tim Bollerslev. Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International Economic Review*, pages 885–905, 1998.

[6] Torben G Andersen, Tim Bollerslev, Francis X Diebold, and Paul Labys. The distribution of realized exchange rate volatility. *Journal of the American statistical association*, 96(453):42–55, 2001.

[7] E Wes Bethel and Hank Childs. *High Performance Visualization: Enabling Extreme-scale Scientific Insight*. Chapman & Hall, 2012.

[8] E. Wes Bethel, David Leinweber, Oliver Rübel, and Kesheng Wu. Federal market information technology in the post-flash crash era: Roles for supercomputing. *The Journal of Trading*, 7(2):9–25, 2012. http://dx.doi.org/10.3905/jot.2012.7.2.009.

[9] Lawrence Blume, David Easley, and Maureen O'hara. Market statistics and technical analysis: The role of volume. *The Journal of Finance*, 49(1):153–181, 1994.

[10] B Chakrabarty, R Pascual, and A Shkilko. Trade classification algorithms: A horse race between the bulk-based and the tick-based rules. http://ssrn.com/abstract=2182819, December 2012.

[11] Robert W Colby and Thomas A Meyers. *The Encyclopedia of Technical Market Indicators*. USA: Dow Jones-Irwin, 1988.

[12] Edwin L Crow and Kunio Shimizu. *Lognormal distributions: Theory and applications*, volume 88. CRC PressI Llc, 1988.

[13] Marcos Mailoc López de Prado. *Advances in High Frequency Strategies*. PhD thesis, Cornell University, 2012.

[14] D Easley, M Lopez de Prado, and M O'Hara. The exchange of flow toxicity. *The Journal of Trading*, 6(2):8–13, 2011.

[15] D Easley, M Lopez de Prado, and M O'Hara. The microstructure of the 'flash crash': Flow toxicity, liquidity crashes and the probability of informed trading. *The Journal of Portfolio Management*, 37:118–128, 2011.

[16] D Easley, M Lopez de Prado, and M O'Hara. Flow toxicity and liquidity in a high frequency world. *Review of Financial Studies*, 25(5):1457–1493, 2012.

[17] D Easley, N Kiefer, M OHara, and J Paperman. Liquidity, information, and infrequently traded stocks. *Journal of Finance*, 51:1405–1436, 1996.

[18] David Easley, Marcos López de Prado, and Maureen O'Hara. The volume clock: Insights into the high frequency paradigm. Available at SSRN `http://ssrn.com/abstract=2034858`, 2012.

[19] David Easley, Marcos M. Lopez de Prado, and Maureen O'Hara. The microstructure of the 'flash crash': Flow toxicity, liquidity crashes and the probability of informed trading. *The Journal of Portfolio Management*, 37(2):118–128, November 2009.

[20] David Easley and MAUREEN O'HARA. Time and the process of security price adjustment. *The Journal of finance*, 47(2):577–605, 1992.

[21] E Gamma, R Helm, R Johnson, and J Vlissides. *Design Patterns*. Addison-Wesley, Reading, Massachusetts, 1995.

[22] William Gropp, Ewing L Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, volume 1. MIT press, 1999.

[23] Georg Hager and Gerhard Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.

[24] HDF Group. HDF5 user guide, 2011. `http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html`.

[25] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft, October 2009.

[26] AA Kirilenko, APS Kyle, M Samadi, and T Tuzun. The flash crash: The impact of high frequency trading on an electronic market. Available at SSRN: `http://ssrn.com/abstract=1686004`, 2010.

[27] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing*. Benjamin/Cummings, Redwood City, 1994.

[28] Charles Lee and Balkrishna Radhakrishna. Inferring investor behavior: Evidence from torq data. *Journal of Financial Markets*, 3(2):83–111, 2000.

[29] Charles Lee and Mark J. Ready. Inferring trade direction from intraday data. *Journal of Finance*, 46:733–746, 1991.

[30] Bil Lewis, Daniel J Berg, et al. *Multithreaded programming with Pthreads*. Sun Microsystems Press, 1998.

[31] Timothy G Mattson, Beverly A Sanders, and Berna L Massingill. *Patterns for parallel programming*. Addison-Wesley Professional, 2004.

[32] Albert J Menkveld and Bart Yueshen. Anatomy of the flash crash. Available at SSRN `http://ssrn.com/abstract=2243520`, 2013.

[33] M. O'Hara. *Market microstructure theory*. Blackwell, 2007.

[34] D. L. Shell. A high-speed sorting procedure. *Commun. ACM*, 2(7):30–32, 1959.

[35] Robert J Shiller. *Market volatility*. The MIT Press, 1992.

[36] Arie Shoshani and Doron Rotem. *Scientific Data Management: Challenges, Technology, and Deployment*, volume 3. Chapman and Hall/CRC, 2010.

[37] Bjarne Stroustrup. *C++ Programming Language, 3/e*. Pearson Education India, 1994.

[38] Erik Theissen. A test of the accuracy of the lee/ready trade classification algorithm. *Journal of International Financial Markets, Institutions and Money*, 11(2):147–165, 2001.

[39] Guido Van Rossum et al. Python programming language. `http://python.org/`, 1994.

[40] Xavier Vives. *Information and learning in markets: the impact of market microstructure*. Princeton University Press, 2010.

[41] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.