JOURNAL OF CHEMICAL INFORMATION AND MODELING

ARTICLE

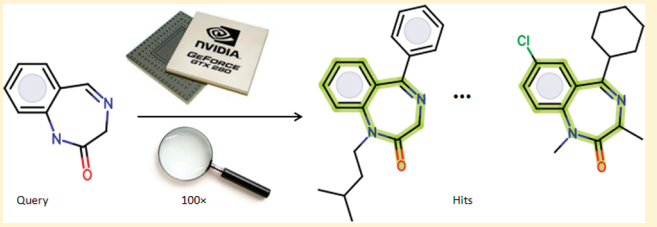
pubs.acs.org/jcim

Accelerating Chemical Database Searching Using Graphics Processing Units

Pu Liu,* Dimitris K. Agrafiotis, Dmitrii N. Rassokhin, and Eric Yang

Johnson & Johnson Pharmaceutical Research and Development, LLC, Welsh and McKean Roads, Spring House, Pennsylvania 19477, United States

ABSTRACT: The utility of chemoinformatics systems depends on the accurate computer representation and efficient manipulation of chemical compounds. In such systems, a small molecule is often digitized as a large fingerprint vector, where each element indicates the presence/absence or the number of occurrences of a particular structural feature. Since in theory the number of unique features can be exceedingly large, these fingerprint vectors are usually folded into much shorter ones using hashing and modulo operations, allowing fast "in-memory" manipulation and comparison of molecules. There is increasing evidence that lossless fingerprints can substantially improve retrieval performance in chemical database searching (substructure or similarity), which have led to the development of several lossless fingerprint compression algorithms. However, any gains in storage and retrieval afforded by compression need to be weighed against the extra computational burden required for decompression before these fingerprints can be compared. Here we demonstrate that graphics processing units (GPU) can greatly alleviate this problem, enabling the practical application of lossless fingerprints on large databases. More specifically, we show that, with the help of a ~$500 ordinary video card, the entire PubChem database of ~32 million compounds can be searched in ~0.2−2 s on average, which is 2 orders of magnitude faster than a conventional CPU. If multiple query patterns are processed in batch, the speedup is even more dramatic (less than 0.02−0.2 s/query for 1000 queries). In the present study, we use the Elias gamma compression algorithm, which results in a compression ratio as high as 0.097.



INTRODUCTION

In chemical information management systems and chemoinformatics research, molecular descriptors are the most common way to encode molecules so that they can be efficiently stored and manipulated in a computer. Since the introduction of computers in chemical research, thousands of different descriptors have been proposed, differing widely in origin, data type, dimensionality, computational cost, and applicability.[1]

One of the most popular class of descriptors are fingerprints. A fingerprint is usually a binary or integer vector where each index corresponds to a particular structural feature and the values represent the presence/absence or the number of occurrences of that feature in a given molecule. These features can either be carefully selected by domain experts, as is the case with MACCS keys[2,3] and BCI fingerprints,[4] or computed in an exhaustive manner by enumerating all possible paths and/or subgraphs of the molecular graph up to a certain size. In the former case, the number of features is relatively small (usually a few hundred), and while their choice may be appropriate for certain applications, they tend to be subjective and lack generality. Conversely, features based on exhaustive subgraph enumeration tend to be a lot more numerous but are unbiased and do not require prior knowledge about what might be relevant to the problem at hand.

Although the number of possible subgraph patterns in the chemical universe is astronomically large, the actual number of unique features each molecule contains is relatively small (usually only tens to a few hundred). In a naïve implementation where every possible feature is explicitly represented, this leads to feature vectors that are extremely long and sparse. To minimize storage space and reading time, these long fingerprint vectors are compressed into shorter bit arrays by dropping the feature count information and applying a simple modulo or hashing operation. In the compressed fingerprints, each pattern is hashed into overlapping bit segments in a much shorter bit array (usually 512 or 1024 bits). Because the number of bits available is much smaller than the number of unique features, there is unavoidably substantial collision and information loss. This reduces their discriminatory power in substructure screening and the quality of hits in similarity searching.

These limitations have stimulated the development of novel lossless fingerprint compression algorithms. In particular, Baldi and coauthors combined the statistical properties of fingerprints with integer entropy encoding.[5] The resulting lossless compression algorithms can achieve storage efficiency that is comparable to or even better than the widely used fixed-length 1024 bit fingerprints, and can substantially improve the performance of similarity searches.

As access of memory is markedly faster than disk, it is highly desirable to place the fingerprints in RAM in order to maximize performance. However, for large compound collections such as PubChem,[6] it is difficult to fit the uncompressed fingerprints in the main memory of a regular desktop computer, necessitating the use of compressed lossless fingerprints. The obvious drawback of

Received: April 11, 2011
Published: June 22, 2011

using compressed fingerprints on a large database is the need to decompress them before they can be used. For the PubChem collection, for example, each similarity or substructure search would require the decompression of 32 million fingerprints, which is extremely computationally demanding even for the most advanced central processing units (CPUs).

Graphics processing units (GPUs) offer a convenient way to attack this problem. GPUs were originally designed to accelerate 3D graphics rendering with limited functions and later evolved into flexibly programmable many-core processors with high chip-level parallelism and memory bandwidth. A modern GPU usually has several multiprocessors with hundreds of computing cores on the same chip. Rather than having a delicate balance in computation, caching, and flow control as in CPUs, GPUs specialize in high-intensity, parallel computations expecting thousands of concurrent threads, therefore providing 1 or 2 orders of magnitude increase in computational capabilities compared to conventional CPUs.

GPUs have begun to capture the attention of researchers in several computational fields and have been used in a variety of problems such as radix/merge sorting,[7] Bayesian networks and mixture models,[8] hidden Markov models,[9] support vector machines (SVM),[10−12] and similarity joins of large high-dimensional data sets.[13] In particular, GPUs have demonstrated their great potential in computational chemistry,[14,15] bioinformatics,[16] and chemoinformatics.[12,17,18] For example, the SVM-light algorithm has been parallelized on a GPU to speed up mining of high-throughput screening data by 1−2 orders of magnitude.[12] Stone et al. demonstrated that even primitive GPUs can accelerate molecular dynamics by similar factors.[15] Haque and Pande utilized GPUs for fast 3D simulation search[17] and LINGO chemical similarity calculations.[18] Most recently, it was shown that GPUs can achieve up to 34 times speedup over CPUs in protein substructure searching with a tableau-based algorithm.[16]

In this work, we demonstrate how GPUs can be utilized to accelerate integer entropy decoding of lossless fingerprints and therefore speed up substructure screening and similarity searching of large chemical databases. We show that both types of search can be carried out on the entire PubChem database containing ∼32 million compounds in less than 2 s. Our method is even more impressive when used in batch mode. With our current implementation, substructure screening of a batch of 1000 query patterns takes only ∼200 s, i.e., 0.2 s/query. Likewise, similarity searching with a cutoff of 0.8 can be completed in less than 0.02 s/query.

The remaining sections are organized as follows. We begin with a brief introduction to lossless feature count fingerprints and integer entropy coding, followed by a description of GPU hardware and CUDA C, substructure screening, similarity searching, and the implementation of our GPU-based algorithms to support these two types of searches. We then present some performance benchmarks on a diverse set of queries and conclude with a brief discussion and some suggestions for further enhancements.

METHODS

A. Lossless Feature-Count Fingerprints and Integer Entropy Coding. As stated in the Introduction, in chemoinformatics applications molecules are often digitized as a vector of molecular descriptors. The majority of these vectorial representations are binary in nature; that is, they only record the presence or absence of a particular feature with one bit. The use of feature count information can make a significant difference in the rate of retrieval and quality of results, especially for topological descriptors.[19] The importance of feature counts in similarity searching, especially for graph-based descriptors, has been demonstrated in a thorough comparative study by Bender et al.[20] Counts are equally valuable in substructure screening as well. By including the number of occurrences of each feature, database molecules that contain fewer occurrences of a certain feature compared to the query pattern can be safely eliminated as potential hits, thereby greatly reducing the number of expensive atom-by-atom exact matches that need to be performed to retrieve the true hits. The count information is particularly useful for molecules with repeated units, such as alkanes, amino acids, saccharides, and other polymers.

While the theoretical number of features in the chemical universe is exceedingly large, the number of unique features for each individual molecule is relatively small. Therefore, it is often advantageous to store the fingerprints in sparse format as feature-count pairs, where the first integer encodes the index of a particular future in a look-up table/dictionary, and the second integer indicates the number of occurrences of that feature in the target molecule. The feature set used in the present study includes atom descriptors encoding atomic number, charge, aromaticity, and other atomic properties, bond descriptors encoding the bond type and bonded atom types, ring descriptors, and path and cluster descriptors that encode all possible linear and nonlinear subgraphs up to length 6, respectively (unpublished work). The specific choice of descriptors is not critical for the present work, which is concerned solely with computational efficiency. Indeed, our method can be used with any set of descriptors that meet the memory requirements described in the sequel. Our particular choice of descriptors yields on average ∼300 distinct feature-count pairs per molecule. This makes it difficult to keep them in memory on a regular desktop computer for a large database such as PubChem.

This problem can be addressed through compression. In the recent work by Baldi and coauthors,[5] two lossless fingerprint compression techniques, namely, monotone value coding and monotone length coding, were developed and compared to various fingerprint compression algorithms, i.e., modulo compression, Golomb code,[21] and Elias code.[22] In the Elias gamma method, an integer q is encoded by prepending $\lfloor \log_2 q \rfloor$ "0" bits to the binary representation of q beginning with the most significant "1" bit (e.g., the integer 5 is encoded as 00101). Therefore, the total number of bits required is $2\lfloor \log_2 q \rfloor + 1$, making this approach particularly suitable for integer arrays with lots of "1"s. In the present work, we use the following procedure to compress each sorted molecular feature-count vector $\{(K_i, C_i)\}$, where K_i is the index of the ith feature in the feature dictionary, C_i is the number of occurrences of that feature in the molecule, $i = 1, ..., N(m)$, and $N(m)$ is the number of unique features in a given molecule m.

(1) Encode $N(m)$ using Elias gamma encoding;
(2) Encode (K_1,C_1) using Elias gamma encoding;
(3) For $i = 2, ..., N(m)$ do:
  (a) Calculate $\Delta K_i = K_i - K_{i-1}$, where $\Delta K_i > 0$ since K_i's are sorted in ascending order;
  (b) Encode ΔK_i and C_i using Elias gamma encoding.

Decompression is just the reverse of the above procedure.

As suggested by the previous work of Baldi et al., the feature indices in the present work are interleaved with the feature counts.

This scheme permits early termination of the decompression process in both substructure and similarity searching and, as we demonstrate below, can greatly reduce the amount of time required for the search.

B. GPU Hardware and CUDA C. In the present work, a top-line general NVIDIA GeForce GTX 580 graphics card was used. This card had 1536 MB GDDR5 off-chip DRAM GPU memory, 64 KB constant memory, and 16 multiprocessors, each of which has 32 1.54 GHz CUDA cores. GeForce GTX 580 supports both single- and double-precision computing and is particularly powerful with 32-bit integer operations (double precision is much slower). Its peak performance can be as high as 1.58 GFLOPs. Because fingerprint manipulation involves only integer operations, the GeForce 580 card is an ideal cost-effective device for our purpose.

Early general purpose GPUs (GPGPUs) supported a limited instruction set and programmatic access to the GPU often required intimate knowledge of the underlying GPU hardware. With the development of high-level languages, such as OpenCL,[23] CUDA C,[24] and DirectCompute, the low-level programming of the GPU is hidden behind an abstraction layer that allows the user to focus on the algorithm logic rather than the idiosyncrasies of the video hardware. In particular, CUDA (compute unified device architecture) is a general purpose parallel computing architecture, first introduced in GeForce 8800 GTX by NVIDA. The corresponding software environment known as CUDA C includes a C-like language that provides programmers with a convenient way to harness the enormous parallel computing capability of the GPU. CUDA C has a mature and well-optimized compiler and development environment, and a large user community. In addition, CUDA was shown to perform better than OpenCL with respect to data transfer and computational efficiency, and it was for these reasons that we chose it for the present work.[25]

In the CUDA programming model, the parallel codes (denoted as *kernels*) are executed in GPU coprocessors (*devices*), while the rest of the program is run on a CPU (*host*). The host and the devices maintain separate memory spaces. The host (CPU) program can access the computer memory (*host memory*) directly and the GPU memory (*device memory*) through CUDA API calls, while the kernel can only access device memory. A kernel function, when invoked, can be executed in parallel by different CUDA threads, which are organized into a grid of thread blocks. Each block can be treated as a virtual multiprocessor where each thread has a fixed number of registers and the threads within the same block communicate through barrier synchronization and high-efficiency, low-latency, per-block shared memory. In GeForce GTX 580, each block can have up to 1024 threads sharing 48 KB shared memory and 32 K 32-bit registers.

C. Substructure Screening. Substructure search is one of the most common and widely supported types of search in chemical databases. The goal is to retrieve all the molecules in the database that contain a given query pattern. Most chemical database management systems implement substructure search in two distinct steps: screening and verification. In the screening phase, the fingerprints of the query pattern are generated and compared to the fingerprints of the molecules in the database (which are computed once in a preprocessing step when the database is indexed and updated dynamically as more molecules are added to the database or the structures of existing molecules are changed). Any molecules that pass the screening phase are then compared to the query through exact atom-by-atom matching to verify whether they are true hits. Since atom-by-atom matching is much more expensive that comparing fingerprints, the fingerprint features need to be as selective as possible so as to minimize the number of molecules that need to be verified.

For a molecule to be a screening hit, it must contain all of the features that are present in the query pattern. Furthermore, if count information is also included, the feature counts of the molecules must be no less than the corresponding counts of the query. Mathematically, if $\{K_i(m), C_i(m)\}$ and $\{K_i(q), C_i(q)\}$ represent the set of features and their corresponding counts for a database molecule m and a query q, respectively, a molecule is a potential hit if and only if

$$\forall\ K_i(q) \in \{K_i(q)\} \rightarrow\ \exists\ K_j(m) \in \{K_j(m)\},$$
$$K_j(m) = K_i(q) \text{ and } C_j(m) \geq C_i(q) \qquad (1)$$

Therefore, it is not always necessary to decompress the entire compressed feature-count fingerprint, as the decompression can be stopped whenever a query feature does not appear in the database molecule or the corresponding count is larger. For a discriminative fingerprint, the majority of nonmatching molecules can be screened out, leaving a much smaller number of candidates to be verified by exact atom-by-atom matching.

The procedure used to determine whether a database molecule m with compressed fingerprint $\boldsymbol{F}(m)$ is a potential hit from a query q that contains $N(q)$ feature-count pairs is as follows:

(1) Set isMatch = true;
(2) Extract the number of feature-count pairs $N(m)$ from the compressed fingerprint $\boldsymbol{F}(m)$ using Elias gamma decoding;
(3) For ($i = 0$, $j = 0$; $i < N(m)$; i++) do:
  (a) Extract ith feature-count pair $(K_i(m),C_i(m))$ from $\boldsymbol{F}(m)$ using Elias gamma decoding;
  (b) For (; $j < N(q)$; j++) do:
    (i) Read jth feature-count pair $(K_j(q),C_j(q))$ from the query pattern q;
    (ii) If $K_j(q) = K_i(m)$, do:
      (1) If $C_j(q) > C_i(m)$, set isMatch = false and exit loop b;
      (2) j++;
      (3) Exit loop b;
    (iii) If $K_j(q) < K_i(m)$, set isMatch = false and exit loop b;
    (iv) Exit loop b;
(4) If isMatch = true and $j < N(q)$ or $K_j(q) > K_i(m)$, set isMatch = false;
(5) If isMatch = true, mark molecule m as a screening hit.

For the above algorithm to work, the feature-count pairs in the molecule and the query are assumed to be sorted in ascending order of K (feature index). We remind the reader that K is the index of a feature in a look-up table, and not the original feature code itself.

D. Similarity Searching. Similarity searching is another common chemoinformatics task. It attempts to retrieve the molecules from a database that are similar to a query molecule within a predetermined similarity cutoff. Nearest neighbor searching is a very common problem in computer science, and a number of elaborate branch-and-bound algorithms have been devised to eliminate the need for a full database scan.[26] The Tanimoto coefficient is the most commonly used similarity measure in chemoinformatics because it is conceptually and computationally straightforward and has been shown to return meaningful hits. Very importantly, it satisfies the triangle inequality,[27] which is a necessary condition for a distance function to be a true metric.

Currently, almost all of the similarity calculations are based on binary fingerprints with only a few exceptions,[5,19,20,28,29] although the generalization to feature-count fingerprints has been proposed multiple times.[30,31]

In the present work, the similarity between two molecules m and q is calculated according to the generalized min−max measure:[30,31]

$$S(m,q) = \frac{\sum_i \min\{C_i(m), C_i(q)\}}{\sum_i \max\{C_i(m), C_i(q)\}} \quad (2)$$

where i runs over the union of distinct features in molecules m and q. It is worth mentioning that the feature-count version of the min−max measure also satisfies the triangle inequality because each feature-count pair $\{K_i(m), C_i(m)\}$ in molecule m can be decomposed into $C_i(m)$ binary features $K_{i,1}(m)$, ..., $K_{i,C_i}(m)$, where $K_{i,j}(m)$ denotes the jth appearance of the ith feature in molecule m. Therefore it can be transformed into the binary version of the Tanimoto coefficient, where the triangle inequality has been proven.[27] As with substructure screening, in similarity search not all the fingerprints need to be decompressed. Swamidass and Baldi developed a method with sublinear retrieval time without loss of accuracy.[5] Their method substantially reduces the search space and the authors have shown that only the molecules satisfying the following condition need to be further considered:

$$TotC(q)t \le TotC(m) \le TotC(q)/t \quad (3)$$

where t is the similarity cutoff and Tot$C(m)$ and Tot$C(q)$ are the sums of feature counts for the database molecule m and query q, respectively.

Our similarity search algorithm for a query molecule q and a database molecule m is shown below. To facilitate the filtering in eq 3, the sums of feature counts for the database and query molecules are precomputed and stored in memory. As with substructure screening, $N(m)$ and $N(q)$ denote the number of feature-count pairs in molecules m and q, respectively.

(1) Skip the molecule if eq 3 is not satisfied;
(2) Set sharedCounts = 0;
(3) Extract the number of feature-count pairs $N(m)$ from the fingerprint $\boldsymbol{F}(m)$ using Elias gamma decoding;
(4) For ($i = 0$, $j = 0$; $i < N(m)$; i++) do:
  (a) Extract ith feature-count pair $(K_i(m),C_i(m))$ from the compressed fingerprint $\boldsymbol{F}(m)$ for molecule m using Elias gamma decoding;
  (b) For (; $j < N(q)$; j++) do:
    (i) Read the jth feature-count pair $(K_j(q),C_j(q))$ for the query pattern q;
    (ii) If $K_j(q) = K_i(m)$, do:
      (1) Set sharedCounts = sharedCounts + $\min(C_i(m),C_j(q))$;
      (2) j++;
      (3) Exit loop b;
    (iii) Else if $(K_j(q) > K_i(m))$, exit loop b;
(5) Set similarity(m,q) = sharedCounts/(Tot$C(q)$ + Tot$C(m)$ − sharedCounts);
(6) If similarity$(m,q) \ge$ cutoff, mark molecule m as a hit.

E. GPU Implementation. In both substructure and similarity searching, the entire database must be scanned against the query pattern. There is no complex recursion or flow control involved in the calculation, and the computations on each database molecule are independent of each other. In the present implementation, each molecule is naturally assigned to a separate thread. Since no communication between threads is necessary, Elias gamma encoding and decoding can be straightforwardly translated from CPU to GPU. More importantly, this eliminates any expensive communication between thread blocks. Because compressed feature-count fingerprints take only a few tens to a few hundreds of bytes[5] (∼250 bytes in the current study), a modern GPU with 1 or 2 GB memory can easily hold a few million molecules. Therefore, millions of parallel tasks can be generated, ensuring that the full GPU power is harnessed by eliminating transaction latency between memories. This makes substructure/similarity searching ideally suited to GPU technology.

As stated earlier, the GPU kernel cannot access CPU memory. To perform parallel kernel computations on a given set of data, that data must be transferred from the host to the device memory. This could be time-consuming and even become the bottleneck for large data sets. For instance, searching the PubChem database would require ∼1.5 s to transfer ∼7.5 GB of compressed fingerprint data.

Figure 1 illustrates the schematic flow of our GPU implementation. Because the device memory on the GPU is much smaller than that on the CPU, the fingerprints for a large chemical collection such as PubChem, at least at present, need to be split into smaller chunks. Each of these chunks is sent to the GPU global memory for computation and is then processed by b thread blocks of t threads each. The CUDA occupancy calculator[24] is employed to provide insight into the choice of block size and register allocation. A 512-thread ($t = 512$) block with 16 registers per thread is empirically determined to provide good overall performance, and $b = \lceil n/t \rceil$, where n is the number of molecules in the given chunk. The feature-count vector of each query pattern is copied into the low-latency constant memory and shared by all of the threads in the GPU. With the compressed fingerprints and feature-count pairs for a query pattern, the substructure screening and similarity searching algorithms described in the last two subsections are implemented and run on GPU. The corresponding results are transferred back to CPU memory.

For a large database, the GPU memory will be flushed by each data chunk because the GPU cannot accommodate all of the compressed fingerprints. Thus, for every new query, these data chunks need to be wastefully retransferred to GPU memory again and again. This suggests that the current approach would be even more suitable for batch processing of multiple queries because the data transfer would need to occur only once. Of course, this is not an issue if the entire database can fit into GPU memory (e.g., for a collection of a few million compounds or for shorter feature-count fingerprints) or if additional GPUs are installed on the same server.

It is worth emphasizing that the number of registers per thread is very limited. When a large lookup table is used in the algorithm, one may copy it to the shared memory, which is another scarce resource. If still larger, the table will have to reside in off-chip device memory with high latency. This would have a significant adverse impact on overall performance. Therefore, whenever possible, it is beneficial to reduce the consumption of registers and shared memory even at the expense of more computations.

All the host code was executed on an Intel Xeon CPU E5530 with 12 GB of RAM, and all the kernel code on a NVIDIA GeForce GTX 580 GPU. The core algorithms (the encoding and decoding of feature-count fingerprints and the substructure and similarity search) were implemented in CUDA C[24] and wrapped into a distributable dynamic-link library, which was consumed by C# code extending the functionality of Microsoft SQL Server 2008 (similar in concept to Oracle's cartridge technology). All of

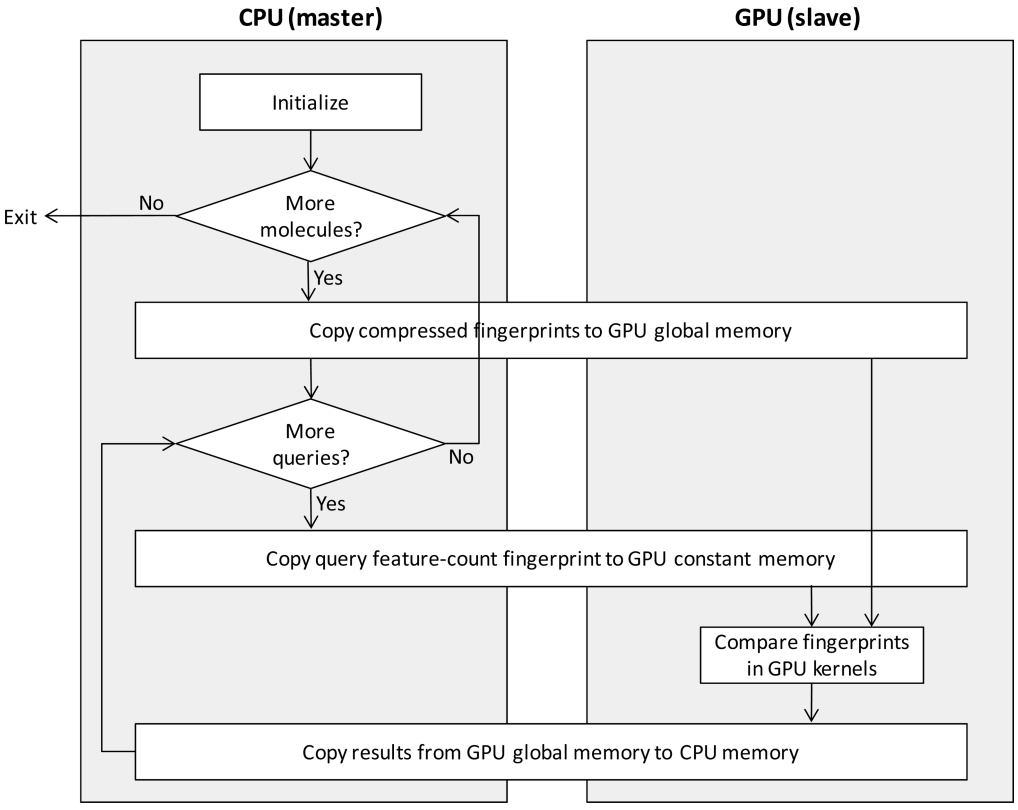


Figure 1. Flowchart illustrating the GPU implementation of substructure screening and similarity searching.

the code was compiled with Microsoft Visual Studio in x64 mode using the maximized speed option. The GPU core algorithms were also implemented on CPU for head-to-head comparison.

RESULTS

A. Database Statistics and Method Validation. To validate our algorithm and assess its performance on large chemical collections, the entire PubChem compound database[6] was downloaded on Jan. 31st, 2011 from the PubChem ftp site (ftp://ftp.ncbi.nih.gov/pubchem/Compound/). The zipped SD files downloaded were parsed by an in-house C# program and the resulting information, including various identifiers, SMILES, charge, mass, formula, and other properties in the SD files were loaded onto a SQL server 2008 database. Overall, this PubChem snapshot contained about 31.5 million molecules. The structures extracted from the SD files were used to calculate the feature-count fingerprints using our in-house Mt toolkit, which is part of the DirectedDiversity[32] and ABCD software suites.[33]

Following the practice of Baldi and co-workers,[5] rather than storing the raw 64-bit hashed features, we only considered the features appearing in the database and sorted the feature-count pairs in descending order of their frequency of occurrence in the database. The 64-bit raw hashed feature codes and the indices in the sorted list were stored in the database as a look-up table, allowing us to translate the raw hash codes into compact indices in the sorted list. The indices and their corresponding counts in each molecule were then compressed using the algorithm described above and stored in the database for future use in substructure and similarity searching.

Since real-world databases evolve over time, there could be hundreds or even thousands of new molecules added to the database daily, thus introducing new features and counts and changing the relative order of features in the sorted list. Updating the lookup table every time there is a change in the database would be time-consuming, and the compression rate and the overall performance could deviate from the reported values. To emulate the real-world scenario, instead of using the entire PubChem database, we used only the first 2 million molecules to construct the lookup table of N_{max} unique features. It is worth emphasizing that the choice of N_{max} is arbitrary as long as it is larger than the number of unique features in the molecule set used to construct the lookup table. The indices of existing features for the remaining molecules in the database were retrieved from this lookup table, and any new features absent from this table were assigned an index determined by the formula (N_{max}+HashValue%1021).

Figure 2 shows the histogram of the number of unique features per molecule (a) and the frequencies of the top 1000 features across the entire PubChem database (b). From Figure 2a, it is evident that the majority of molecules have 100−500 unique features, with an average around 300. Figure 2b shows that, although only the first 2 million molecules were used in the construction of the lookup table, the resulting table captures all of the frequent features and their relative occurrences in the entire PubChem database.

Elias gamma encoding can achieve a very high compression rate for vectors dominated by small integers. For instance, only a single "on" bit is needed to encode the integer "1". Figure 3a shows the histogram of feature counts across the entire PubChem and reveals that the majority of feature counts are 1′s and

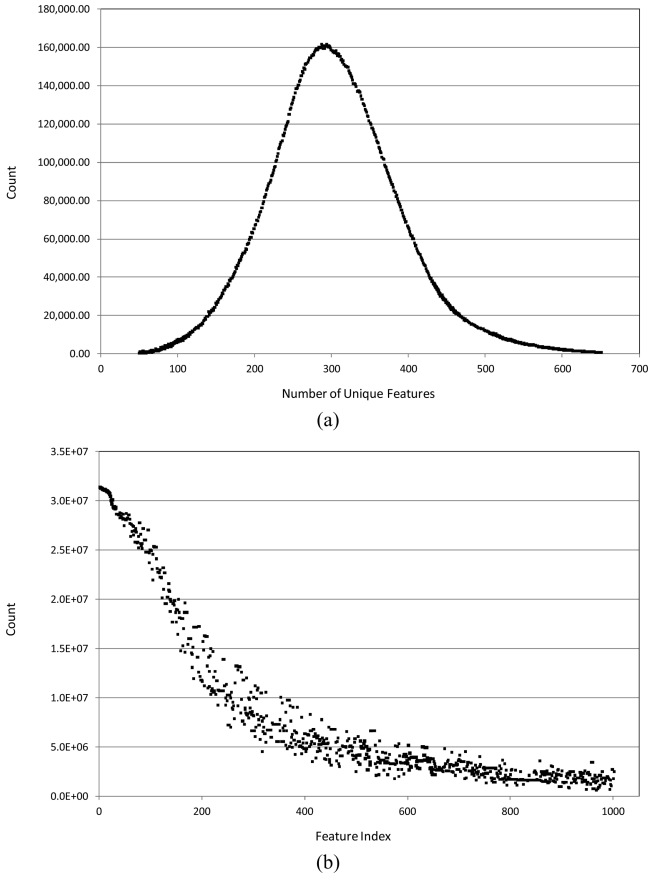


Figure 2. Feature statistics on a snapshot of the PubChem database containing ~32 million molecules. (a) Distribution of unique features in a single database molecule. (b) Distribution of counts of the 1000 most frequent features (sorted; where, for example, 3.5E+07 represents $3.5 \times 10_7$).

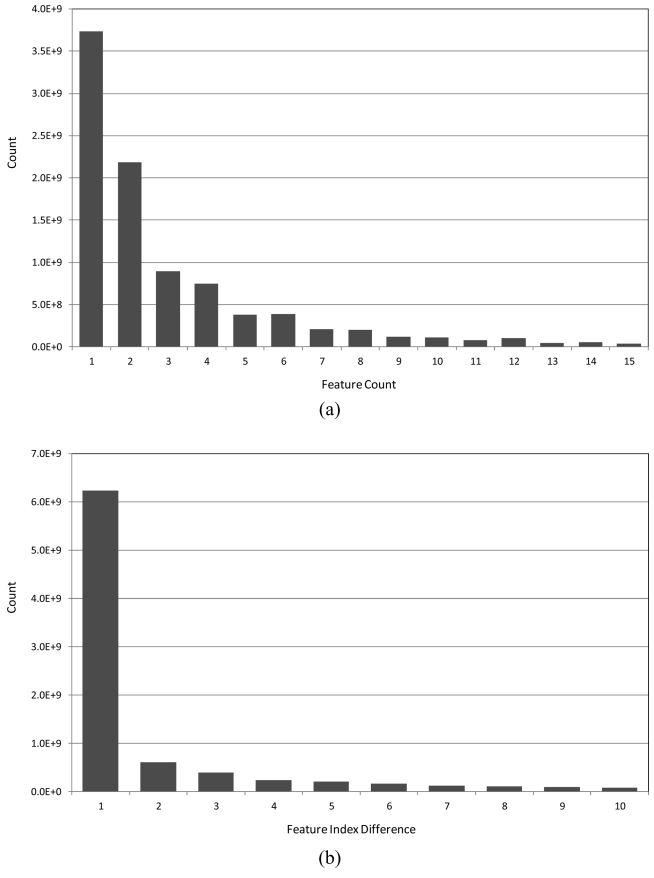


Figure 3. (a) Histogram of feature counts across the entire PubChem database (where, for example, 4.0E+9 represents 4.0×10^9). (b) Histogram of the differences between consecutive feature indices. These distributions demonstrate that the majority of the feature counts and index differences are 1′s and other small integers, validating our choice of Elias gamma algorithm for fingerprint compression.

over 97% are less than or equal to 15. Another interesting observation is that the feature indices for a given molecule often increase semisequentially. Figure 3b shows the histogram of feature index differences of the neighboring features over the entire PubChem collection. Over 65% of index differences are 1's and 90% are less than or equal to 15, which only require 1−7 bits to encode under the Elias gamma method. Calculations showed that step 3a (Methods, section A) in the current encoding scheme can improve the compression rate by 200% and only 7.5 GB of memory is required for the entire PubChem collection, leading to a compression rate of 9.7%.

B. Substructure Screening. Two sets of query molecules were used to calibrate the performance of our GPU code. The first is a diverse group of patterns extracted from the recent publication by Golovin and Henrick,[34] and the second includes 1000 molecules selected at random from PubChem.

A GPU can have hundreds of computing cores, can support thousands of threads, and thus has an intrinsic advantage over CPUs on "single program multiple data" (SPMD) tasks such as database scanning. Table 1 summarizes the timing results for screening the ~32 million molecules in the PubChem database against each of the 14 substructure queries used by Golovin and Henrick. The hardware specifications for the CPU and GPU have been described in detail in Methods, section E. The table includes two pairs of columns. The first pair, labeled GPU1 and CPU1, lists the timings of the "naïve" search where the fingerprint of each database molecule is fully decompressed before it is compared to the fingerprint of the query. The second pair, labeled GPU2 and CPU2, lists the timings of the "intelligent" search detailed in Methods, section C, where decompression is terminated as soon as the program determines that the molecule could not be a possible match on the basis of the already decompressed feature-counts. All timings are reported in milliseconds.

To obtain reliable statistics, each entry in Table 1 represents the average over multiple runs. Because CPU calculations are substantially slower than GPU, we used 15 independent runs for CPU1 and CPU2 and 200 runs for GPU1 and GPU2. As seen in Table 1, the full decompression of 32 million fingerprints is computationally very demanding, requiring ~420 s on a single CPU core. By taking advantage of the massively parallel GPU architecture, the computational time can be reduced 100-fold to ~4 s. Thus, by investing ~$500 in a GeForce GTX 580 video card, one can accelerate the decompression and substructure screening calculation by 2 orders of magnitude.

In GPU2 and CPU2, the decompression is terminated early whenever a new feature or a larger number of occurrences of an existing feature is identified in the query pattern. This strategy can substantially speed up the screening process for CPU, reducing the computation from 420 to 9−110 s. For GPU, the speedup of early termination is only 2-fold. This is due to the fact

Table 1. Timings for Substructure Screening of Golovin and Henrick's Query Patterns, Averaged over Multiple Independent Runs (ms)[a]

SMARTS	no. of features	GPU 1	CPU 1	GPU2	CPU2
S1C=NC=C1	73	4749.8	421 741.0	2227.5	110 839.7
CP(O)(O)=O	97	4376.7	420 123.5	2418.1	100 332.2
C34CCC1C(CCC2CC(=O)CCC12)C3CCC4	112	4700.4	422 037.4	1555.2	7 160.5
NC1=NN=C(S)S1	115	4054.9	422 084.7	2001.4	55 459.1
OC2=CC(=O)C1=C(C=CC=C1)O2	143	3995.4	421 208.6	1915.6	41 901.1
CNc1ncnc2[n](C)cnc12	144	3993.3	423 925.5	1733.7	30 027.6
Nc1ncnc2[nH]cnc12	147	4014.7	422 714.1	1792.6	37 584.9
CCCCCP(O)(O)=O	163	4566.5	422 766.2	2138.4	67 838.9
CCCCCCCCCCCP(O)(O)=O	169	4530.9	422 333.7	1790.3	23 841.2
Nc1ncnc2[n](cnc12)C3CCCC3	172	3994.8	422 890.5	1705.5	27 675.0
CC12CCC3C(CCC4=CC(O)CCC34C)C1CCC2	194	4021.8	421 581.3	1555.3	5 989.7
N2CCC13CCCCC1C2CC4=C3C=CC=C4	214	4742.5	421 245.7	1583.8	11 254.9
C1C2SCCN2C1	227	4329.1	424 169.7	2108.1	73 902.9
ONC1CC(C(O)C1O)[n]2cnc3c(NC4CC4)ncnc23	359	4187.8	419 477.0	1558.6	9 297.9

[a] The results were sorted in ascending order of the number of unique features in query patterns.

that, in the GPU implementation, the data transfer between the main and GPU memory becomes the rate limiting step; indeed, it takes ~1.5 s just to transfer the fingerprints of 32 million compounds from the main to GPU memory. Although early termination of fingerprint decoding can greatly reduce decompression time, the time required for data transfer is unaffected. Another interesting observation is that the timings for CPU2 fluctuate greatly while the rest of the results only undergo moderate undulations. In the CPU2 calculations, the overall time depends greatly on the composition of the feature-count pairs. If one feature from a given query pattern is new to the lookup table, this feature will be assigned a large index according to the formula (N_{max}+HashValue%1021). If the counts of the other features are relatively small, this will force the decompression of the feature-count pairs for all of the known features in the table for every fingerprint in the database. Therefore there is almost no savings for such a novel query pattern. Fortunately, this is not the case for most queries of interest. The two most time-consuming queries in this table are S1C=NC=C1 and CP(O)(O)=O. All of the features involved in these queries are rather common. However, because of their simple chemical structures, the counts for each feature are very small, thus greatly limiting their discriminatory ability. In contrast, molecules CC12CCC3C(CCC4=CC(O)CCC34C)C1CCC2, ONC1CC(C(O)C1O)[n]2cnc3c(NC4CC4)ncn-c23, and C34CCC1C(CCC2CC(=O)CCC12)C3CCC4 are much more complex, and the feature counts are very large. This explains the exceptional efficiency of CPU2 for these three patterns. For CPU1 and GPU1, the decompression of the whole fingerprint is mandatory, while, for GPU2, it is the data transfer that takes most of the time. Therefore, we expect much less fluctuation and more consistent database performance for these three implementations.

Substructure screening performance was also evaluated on a second set of queries, comprising 1000 randomly chosen PubChem structures. The timing results were very similar to those of the first set, with an average of 428 373.8, 4259.5, 19 172.1, and 1740.6 ms for CPU1, GPU1, CPU2, and GPU2, respectively.

As discussed above, the impressive computational speed of the GPU is somewhat offset by the relatively slow data transfer between CPU and GPU for large blocks of data. This suggests that the current GPU approach would be even more useful for batch processing of multiple queries. Indeed, when the 1000 query molecules in the second set were processed in batch, the average screening time over 100 runs for GPU1 and GPU2 was 2723.0 and 227.6 s, respectively, which corresponds to 0.2276 s/query in the latter case. Such performance is nothing short of impressive, particularly if one takes into account the extremely low cost of the hardware.

C. Similarity Searching. The two sets of queries that were used in substructure screening were also employed to assess performance enhancement in similarity searching. The results are summarized in Table 2. Similar to the substructure screening experiments, CPU1 and GPU1 represent the naïve search where the fingerprints of the database molecules are fully decompressed before they are compared to the fingerprint of the query, whereas CPU2 and GPU2 refer to the intelligent search that takes advantage of eq 3 to filter out a large portion of nonmatching molecules before the time-consuming decompression.

To obtain reliable results, 15 and 100 independent runs were performed for CPU and GPU, respectively, and the average timings are reported in Table 2 for a similarity cutoff of 0.8. Overall, the timing results for similarity searching are very similar to those for substructure screening. For CPU1 and GPU1, because the most time-consuming step is the decompression of 32 million fingerprints, there is no large fluctuation for any of the quethery patterns. Since the algorithm can take advantage of the massive parallelism of the GPU, similarity searching is greatly accelerated and the time is reduced from 450 to 5 s going from CPU to GPU. A 5 s time frame is certainly bearable when considering that ~32 million compounds need to be scanned. For GPU2, although for certain queries most of the database compounds can be filtered out by using eq 3 and the calculation can be split among hundreds of cores, the time in fingerprint data transfer is unavoidable. Therefore, the relative variation of the timings on GPU2 is much smaller when compared with that of CPU2.

As with the substructure search, for CPU2 the timing results depend greatly on the query pattern. It takes more than 225, 219, and 184 s to retrieve similar compounds for complex molecules such as N2CCC13CCCCC1C2CC4=C3C=CC=C4, ONC1CC(C(O)-

Table 2. Timings for Similarity Searching of Golovin and Henrick's Query Patterns Using a Similarity Threshold of 0.8, Averaged over Multiple Independent Runs (ms)[a]

SMILES	no. of features	GPU 1	CPU 1	GPU2	CPU2
S1C=NC=C1	73	4337.5	441 230.9	1636.5	189.1
CP(O)(O)=O	97	4682.6	431 254.0	1888.9	409.3
C34CCC1C(CCC2CC(=O)CCC12)C3CCC4	112	4547.9	445 300.5	3483.6	184 270.5
NC1=NN=C(S)S1	115	4653.1	448 241.3	1609.7	547.1
OC2=CC(=O)C1=C(C=CC=C1)O2	143	4511.2	450 122.1	1714.2	9 087.5
CNc1ncnc2[n](C)cnc12	144	4660.5	456 805.7	1783.8	11 536.3
Nc1ncnc2[nH]cnc12	147	4683.1	450 563.7	1729.8	6 952.9
CCCCCP(O)(O)=O	163	5355.3	443 393.5	1958.4	4 991.9
CCCCCCCCCCCP(O)(O)=O	169	5637.5	445 315.5	2307.3	47 900.5
Nc1ncnc2[n](cnc12)C3CCCC3	172	4746.6	462 273.3	2350.4	66 055.5
CC12CCC3C(CCC4=CC(O)CCC34C)C1CCC2	194	4911.1	460 358.1	2844.5	146 745.5
N2CCC13CCCCC1C2CC4=C3C=CC=C4	214	5473.9	462 275.7	3725.7	226 682.7
C1C2SCCN2C1	227	5700.3	470 450.4	1738.8	9 489.5
ONC1CC(C(O)C1O)[n]2cnc3c(NC4CC4)ncnc23	359	5530.0	484 013.3	3748.3	219 994.0

[a] The results were sorted in ascending order of the number of unique features in query patterns.

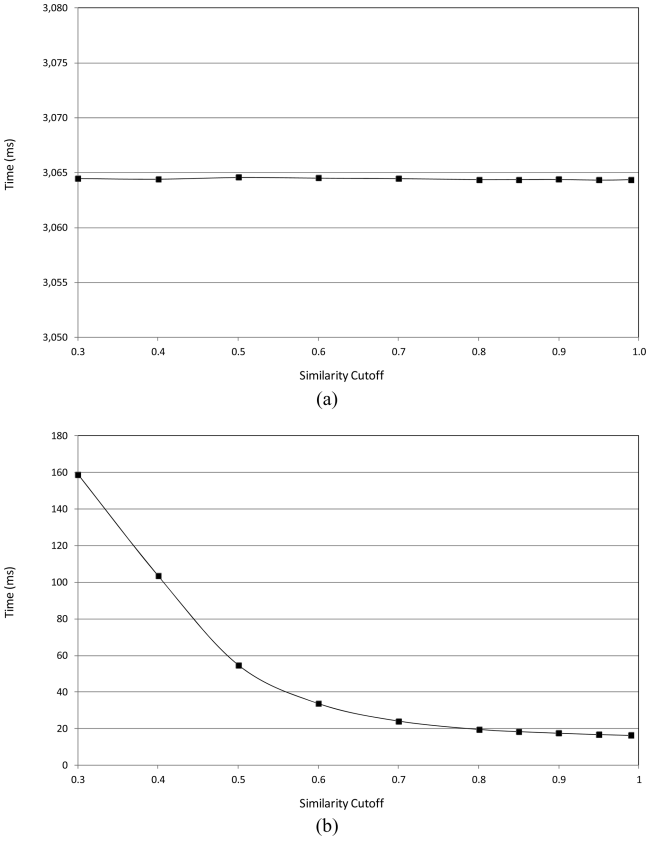


Figure 4. Average Timings of Similarity Searches for the "Naïve" and "Intelligent" Algorithms as a Function of Similarity Cutoff, Collected over Multiple Independent Runs. (a) "Naïve" algorithm involving full decompression. (b) "Intelligent" algorithm utilizing eq 3 for filtering out molecules that cannot possibly exceed the similarity cutoff.

C1O)[n]2cnc3c(NC4CC4)ncnc23, and C34CCC1C(CCC2CC-(=O)CCC12)C3CCC4, respectively, while the similarity searches of simple patterns such as S1C=NC=C1, CP(O)(O)=O, and NC1=NN=C(S)S1 require a mere 0.18, 0.40, and 0.55 s. This is understandable if one scrutinizes eq 3 and the distribution of the total feature counts. The histogram of total feature counts across the entire PubChem database (results not shown) is a slightly negatively skewed normal-like distribution. It has a single peak around 984, and only fewer than 5% of the molecules have less than 521 or more than 1861 feature counts. As shown in the work by Swamidass and Baldi,[30] the percent of fingerprints that need to be decoded is $N(\mathrm{Tot}(C(q)/t-\mu)/\sigma) - N((\mathrm{TotC}-(q)^*t-\mu)/\sigma)$ when the distribution is assumed to be normal with mean μ and variance σ. If $\mathrm{Tot}C(q)$ of the pattern q lies around the peak region, the discriminatory ability of eq 3 is rather small, but when $\mathrm{Tot}C(q)$ falls into the tail regions, only a tiny fraction of fingerprints needs to be decompressed.

It is obvious that the screening efficiency of eq 3 also depends on the similarity cutoff t. The smaller the cutoff, the more molecules need to be compared against the query. Thus, the similarity calculation becomes increasingly time-consuming as the cutoff decreases. To demonstrate this point and the suitability of GPUs for batch queries, the first 100 molecules in the second query set (1000 PubChem compounds) were sent to the GPU in one batch, and similarity searches were performed over a series of cutoffs, i.e., 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.95, and 0.99. (The sampling of cutoff values is more dense in the high-value region, which reflects the fact the scientists are usually more interested in compounds that are very similar to the query pattern.) Again, 10 independent runs were carried out for GPU1 and 100 runs for GPU2, since the later is computationally more affordable. The average timings per molecule are summarized in Figure 4. As explained above, because all the fingerprints were fully decompressed and compared against the query pattern, it is not surprising that the timing is insensitive to the cutoff, as manifested in Figure 4a. Figure 4b clearly demonstrates that the time required to complete the search is a decreasing function of the similarity cutoff. Again, it is worth emphasizing that the GPU algorithm is particularly suitable for batch searches, requiring less than 0.02 s/query for a cutoff greater than 0.8. For a small (and probably meaningless) cutoff of 0.3, the similarity search over the entire PubChem database takes on average about 0.16 s/query.

DISCUSSION

The lossless feature-count fingerprint is an attractive option to describe the relationships between molecules because of its excellent retrieval performance. The predominance of small counts makes Elias gamma encoding an ideal choice as a compression algorithm. On the basis of our observations, the feature indices after sorting are usually semisequential. Rather than encoding the indices themselves, we encode the differences between neighboring indices, thus achieving 200% improvement in compression performance. The overall compression rate is exceptionally high, ~0.097.

The compression algorithm makes it possible to fit the feature-count fingerprints of a very large database such as PubChem in the fast-accessing RAM of a regular desktop server. Although this can greatly boost the performance of many chemoinformatics database tasks, these compressed fingerprints need to be decompressed before they can be utilized. The decompression and subsequent comparison of such fingerprints presents a significant computational challenge even for most state-of-the-art CPUs. For instance, substructure screening of the entire ~32 million molecule PubChem database could take more than 100 s even with the help of sophisticated pruning algorithms, and similarity searching with a cutoff of 0.8 could take more than 200 s for certain patterns. With the help of a single video card, such queries can be completed in ~2 s. If multiple queries are submitted to the GPU at once for batch processing, the average query time for substructure screening and similarity searching using a cutoff of 0.8 or higher is reduced to 0.2 and 0.02 s, respectively. Moreover, since CUDA C is just a subset of standard C, the present algorithm can be straightforwardly implemented, compiled into a library, and called from other languages and/or programs. In our specific implementation, the methods were packaged into a dynamically linked library (DLL) and invoked through managed C# code as stored procedures or user-defined functions on Microsoft SQL Server.

Finally, it is worth pointing out that the current algorithms can be further optimized by overlapping the data transfer and kernel calculations, by overlapping the CPU and GPU computations, or by installing additional GPUs on the same server.

AUTHOR INFORMATION

Corresponding Author

*Tel.: (215) 628-6803; e-mail: puliu45@gmail.com.

REFERENCES

(1) Todeschini, R.; Consonni, V. *Handbook of Molecular Descriptors*; Wiley-VCH: Weinheim, Germany and New York, 2002.

(2) Durant, J. L.; Leland, B. A.; Henry, D. R.; Nourse, J. G. Reoptimization of MDL Keys for Use in Drug Discovery. *J. Chem. Inf. Comput. Sci.* **2002**, *46*, 1273–1280.

(3) McGregor, M. J.; Pallai, P. V. Clustering of Large Databases of Compounds: Using the MDL "Keys" as Structural Descriptors. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 443–448.

(4) Barnard, J. M.; Downs, G. M. Chemical Fragment Generation and Clustering Software. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 141–142.

(5) Baldi, P.; Benz, R. W.; Hirschberg, D. S.; Swamidass, S. J. Lossless Compression of Chemical Fingerprints Using Integer Entropy Codes Improves Storage and Retrieval. *J. Chem. Inf. Model.* **2007**, *47*, 2098–2109.

(6) Bolton, E.; Wang, Y.; Thiessen, P. A.; Bryant, S. H. PubChem: Integrated Platform of Small Molecules and Biological Activities. In *Annual Reports in Computational Chemistry*; American Chemical Society: Washington, DC, USA, 2008; Vol. 4, pp 217−241.

(7) Satish, N.; Harris, M.; Garland, M.Designing Efficient Sorting Algorithms for Manycore GPUs. *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*; IEEE Computer Society: Washington, DC, USA, 2009; pp 1−10.

(8) Suchard, M. A.; Wang, Q.; Chan, C.; Frelinger, J.; Cron, A.; West, M. Understanding GPU Programming for Statistical Computation: Studies in Massively Parallel Massive Mixtures. *J. Comput. Graph. Stat.* **2010**, *19*, 419–438.

(9) Liu, C. cuHMM: A CUDA Implementation of Hidden Markov Model Training and Classication. http://code.google.com/p/chmm (accessed Feb 12, 2011).

(10) Do, T.-N.; Nguyen, V.-H. A Novel Speed-up, S. V. M. Algorithm for Massive Classification Tasks. *Proceedings of RIVF'2008*; IEEE Press:: Ho Chi Minh, Vietnam, 2008; pp 215−220.

(11) Catanzaro, B.; Sundaram, N.; Keutzer, K. Fast Support Vector Machine Training and Classification on Graphics Processors. In *ICML'08: Proceedings of 25th International Conference on Machine Learning*; Cohen, W. W., McCallum, A., Roweis, S., Eds. ACM Press: New York, NY, and Helsinki, Finland, 2008; pp 104−111.

(12) Liao, Q.; Wang, J.; Webster, Y.; Watson, I. A. GPU Accelerated Support Vector Machines for Mining High-Throughput Screening Data. *J. Chem. Inf. Model.* **2009**, *49*, 2718–2725.

(13) Lieberman, M. D.; Sankaranarayanan, J.; Samet, H. A Fast Similarity Join Algorithm Using Graphics Processing Units. *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*; IEEE Computer Society: Washington, DC, USA, 2008; pp 1111−1120.

(14) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; Legrand, S.; Beberg, A. L.; Ensign, D. L.; Bruns, C. M.; Pande, V. S. Accelerating Molecular Dynamic Simulation on Graphics Processing Units. *J. Comput. Chem.* **2009**, *30*, 864–872.

(15) Stone, J. E.; Phillips, J. C.; Freddolino, P. L.; Hardy, D. J.; Trabuco, L. G.; Schulten, K. Accelerating Molecular Modeling Applications with Graphics Processors. *J. Comput. Chem.* **2007**, *28*, 2618–2640.

(16) Stivala, A. D.; Stuckey, P. J.; Wirth, A. I. Fast and Accurate Protein Substructure Searching with Simulated Annealing and GPUs. *BMC Bioinf.* **2010**, *11*, 446.

(17) Haque, I. S.; Pande, V. S. PAPER-Accelerating Parallel Evaluations of ROCS. *J. Comput. Chem.* **2009**, *31*, 117–132.

(18) Haque, I. S.; Pande, V. S.; Walters, W. P. SIML: A Fast SIMD Algorithm for Calculating LINGO Chemical Similarities on GPUs and CPUs. *J. Chem. Inf. Model.* **2010**, *50*, 560–564.

(19) Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *J. Chem. Inf. Model.* **2010**, *50*, 742–754.

(20) Bender, A.; Jenkins, J. L.; Scheiber, J.; Sukuru, S. C. K.; Glick, M.; Davies, J. W. How Similar Are Similarity Searching Methods? A Principal Component Analysis of Molecular Descriptor Space. *J. Chem. Inf. Model.* **2009**, *49*, 108–119.

(21) Golomb, S. W. Run-length Encodings. *IEEE Trans. Inf. Theory* **1965**, *12*, 399–401.

(22) Elias, P. Universal Codeword Sets and Representations of the Integers. *IEEE Trans. Inf. Theory* **1975**, *21*, 194–203.

(23) Tsuchiyama, R.; Nakamura, T.; Lizuka, T.; Asahara, A. *The OpenCL Programming Book*; Fixstars: Oakland, CA, 2010.

(24) *NVIDIA CUDA C Programming Guide 3.2*; NVidia: Santa Clara, CA, 2010.

(25) Karimi, K.; Dickson, N. G.; Hamze, F. A Performance Comparison of CUDA and OpenCL. arXiv.org, 2010; Vol. abs/1005.2581.

(26) Xu, H.; Agrafiotis, D. K. Nearest Neighbor Search in General Metric Spaces Using a Tree Data Structure with a Simple Heuristic. *J. Chem. Inf. Model.* **2003**, *43*, 1933–1941.

(27) Lipkus, A. H. A. Proof of the Triangle Inequality for the Tanimoto Distance. *J. Math. Chem.* **1999**, *26*, 263–265.

(28) Hert, J.; Willett, P.; Wilton, D. J.; Acklin, P.; Azzaoui, K.; Jacoby, E.; Schuffenhauer, A. Comparison of Topological Descriptors for Similarity-Based Virtual Screening Using Multiple Bioactive Reference Structures. *Org. Biomol. Chem* **2004**, *2*, 3256–3266.

(29) Liu, R.; Zhou, D. Using Molecular Fingerprint as Descriptors in the QSPR Study of Lipophilicity. *J. Chem. Inf. Model.* **2008**, *48*, 542–549.

(30) Swamidass, S. J.; Baldi, P. Bounds and Algorithms for Fast Exact Searches of Chemical Fingerprints in Linear and Sublinear Time. *J. Chem. Inf. Model.* **2007**, *47*, 302–317.

(31) Swamidass, S. J.; Chen, J.; Bruand, J.; Phung, P.; Ralaivola, L.; Baldi, P. Kernels for Small Molecules and the Prediction of Mutagenicity, Toxicity, and Anti-Cancer Activity. *Bioinformatics* **2005**, *21*, i359–368.

(32) Agrafiotis, D. K.; Lobanov, V. S.; Salemme, F. R. Combinatorial Informatics in the Post-genomics Era. *Nat. Rev. Drug Discovery* **2002**, *1*, 337–346.

(33) Agrafiotis, D. K.; Alex, S.; Dai, H.; Derkinderen, A.; Farnum, M.; Gates, P.; Izrailev, S.; Jaeger, E. P.; Konstant, P.; Leung, A.; Lobanov, V. S.; Marichal, P.; Martin, D.; Rassokhin, D. N.; Shemanarev, M.; Skalkin, A.; Stong, J.; Tabruyn, T.; Vermeiren, M.; Wan, J.; Xu, X. Y.; Yao, X. Advanced Biological and Chemical Discovery (ABCD): Centralizing Discovery Knowledge in an Inherently Decentralized World. *J. Chem. Inf. Model.* **2007**, *47*, 1999–2014.

(34) Golovin, A.; Henrick, K. Chemical Substructure Search in SQL. *J. Chem. Inf. Model.* **2009**, *49*, 22–27.