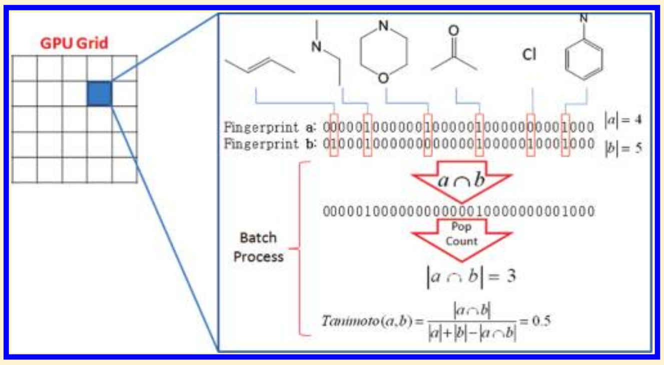
GPU Accelerated Chemical Similarity Calculation for Compound Library Comparison

Chao Ma,[†,‡,§] Lirong Wang,[‡,§] and Xiang-Qun Xie*[,‡,†,§]

[†]Department of Computational and Systems Biology, Joint Pitt/CMU Computational Biology Program, School of Medicine, [‡]Department of Pharmaceutical Sciences, School of Pharmacy, and [§]Pittsburgh Center for Chemical Methodologies & Library Development (PCMLD) and Drug Discovery Institute, University of Pittsburgh, Pittsburgh, Pennsylvania 15260, United States

ABSTRACT: Chemical similarity calculation plays an important role in compound library design, virtual screening, and "lead" optimization. In this manuscript, we present a novel GPU-accelerated algorithm for all-vs-all Tanimoto matrix calculation and nearest neighbor search. By taking advantage of multicore GPU architecture and CUDA parallel programming technology, the algorithm is up to 39 times superior to the existing commercial software that runs on CPUs. Because of the utilization of intrinsic GPU instructions, this approach is nearly 10 times faster than existing GPU-accelerated sparse vector algorithm, when Unity fingerprints are used for Tanimoto calculation. The GPU program that implements this new method takes about 20 min to complete the calculation of Tanimoto coefficients between 32 M PubChem compounds and 10K Active Probes compounds, i.e., 324G Tanimoto coefficients, on a 128-CUDA-core GPU.


GPU Grid
Fingerprint a: 00000100000001000000100000000001000
Fingerprint b: 01000100000000000000100000010001000
|a| = 4
|b| = 5
a ∩ b
00000100000000000000100000000001000
Pop Count
|a ∩ b| = 3
Batch Process
Tanimoto(a,b) = |a∩b| / (|a|+|b|−|a∩b|) = 0.5

1. INTRODUCTION

Combinatorial chemistry generates a large number of compounds and boosts the growth of various screening libraries. Thus, analysis and data mining of the vast quantity of compounds significantly contribute to the success of both virtual screening and high-throughput screening.[1] Among the analysis schemes, chemical similarity calculation is a frequently used method in computer-aided drug design.[2] The concept of chemical similarity or molecular similarity is also heavily involved in molecular diversity analysis and combinatorial library design. Many methods have been established for this purpose, covering a wide range of molecular descriptors and data mining algorithms.[3−9]

Various cheminformatics algorithms have been developed for chemical similarity measurement. The Tanimoto coefficient between molecular fingerprints is still the most popular similarity metric, because of its computational efficiency and its relevance to biological profile.[10,11] Tanimoto calculation performs structural diversity analysis through compound library comparison.[12] In addition, Database Comparison program in Tripos Sybyl[13] uses Tanimoto coefficient to characterize the degree of similarity or overlapping between two compound libraries.[14] Furthermore, Tanimoto calculation is also associated with modern machine learning algorithms, ranging from supervised kernel machine[15] to unsupervised compound clustering.[16,17] In these applications, an all-vs-all Tanimoto matrix, which contains Tanimoto coefficients between all pairs of compounds, needs to be calculated. This results in $O(N^2)$ time complexity, i.e., quadratic in the size of libraries. Despite the advance in computer hardware, exploring large chemical libraries, such as PubChem library, remains a substantial challenge.[18,19]

With the emphasis on "green high performance computing", the development of modern graphics processing units (GPU) points out potential solutions to these challenges. GPUs are specialized microprocessors for graphic rendering. Modern GPUs feature higher memory bandwidth and computing throughput in terms of floating point operations per second (FLOPS), compared to CPUs. Recently, GPUs have been applied to quantum chemistry and molecular dynamics[20,21] as well as Tanimoto calculation.[22−24] Haque et al.[22] and Liao et al.[23] reported sparse vector algorithm for all-vs-all Tanimoto matrix calculation on GPUs. The established sparse vector algorithm is proficient to process high-sparsity fingerprints.

In this article, we present a new algorithm to calculate Tanimoto coefficients between pairs of compounds and to perform compound library comparison on GPUs. Different from Haque's and Liao's work, our algorithm achieves better performance when processing low-sparsity molecular fingerprints. Furthermore, compound library comparison can be executed on GPUs after Tanimoto coefficients are solved. In the present studies, we start with algorithm design and then compare its performance with existing GPU and CPU algorithms using three different chemical libraries and three CPU or GPU systems. The results show that the program that implements this novel approach runs up to 39 times faster than the Sybyl Database Comparison program and nearly 10 times faster than the existing GPU-based sparse vector algorithm. Furthermore, the program completes the calculation of 324G Tanimoto coefficients in 20 min, for comparing 10K

Received: December 14, 2010
Published: June 21, 2011

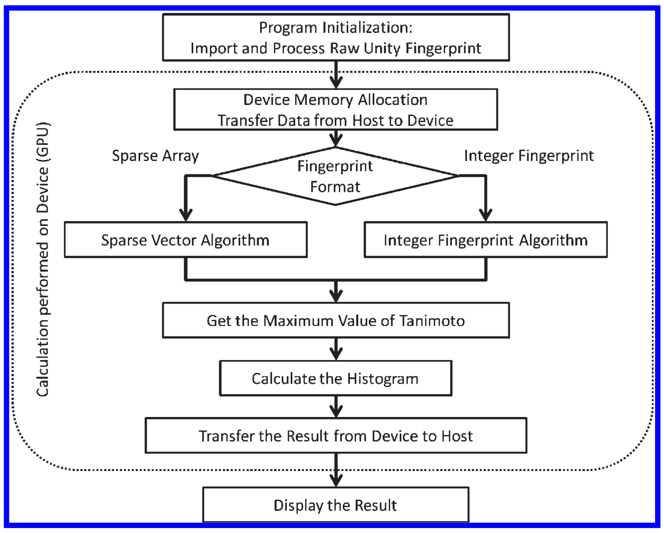

Program Initialization: Import and Process Raw Unity Fingerprint
Calculation performed on Device (GPU)
Device Memory Allocation Transfer Data from Host to Device
Sparse Array
Fingerprint Format
Integer Fingerprint
Sparse Vector Algorithm
Integer Fingerprint Algorithm
Get the Maximum Value of Tanimoto
Calculate the Histogram
Transfer the Result from Device to Host
Display the Result

Figure 1. The general workflow of compound library comparison on the computer hardware where a GPU is located.

Active Probes compounds with PubChem library. The algorithm is implemented with graphical user interface (GUI) for ease of use. The binary, source code, and user instruction are available at http://www.cbligand.org/gpu. The program can be customized to adapt any binary fingerprints and carry out kNN (k-nearest neighbor) search.

2. METHODS AND CALCULATIONS

2.1. Overview of Compound Library Comparison. The concepts and details regarding the GPU-accelerated compound library comparison are described in this section. In cheminformatics, Tanimoto coefficient is one of the most popular chemical similarity indices, and is usually calculated based on binary molecular fingerprints. The Tanimoto coefficient between a pair of compounds that have molecular fingerprints **a** and **b** can be formulated as

$$\mathrm{Tanimoto}(\mathrm{a},\mathrm{b}) = \frac{N_{\mathrm{ab}}}{(N_{\mathrm{a}} + N_{\mathrm{b}}) - N_{\mathrm{ab}}}$$

where N_{ab} is the number of common "1" bits that occur in both fingerprint **a** and fingerprint **b**; N_{a} is the number of "1" bits in fingerprint **a**; and N_{b} is the number of "1" bits in fingerprint **b**. In our calculation, the Tanimoto coefficient is computed with Unity[13] fingerprint as illustration, whereas other fingerprints can be applied as well. Unity fingerprint is a 992-bit binary vector, which encodes the presence or absence of a set of predefined structural patterns.

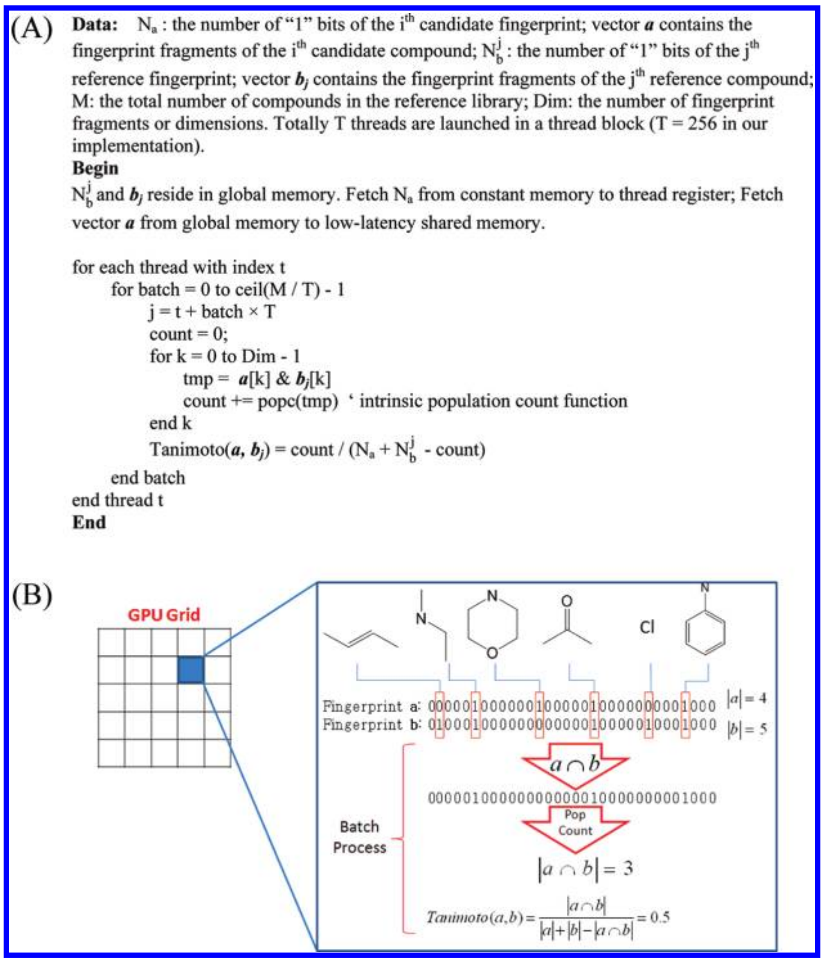
The compound library to be compared to is named as reference library, while the other compound library is named as candidate library. The goal is to characterize how well the candidate library is represented in the reference library. Among established programs, Tripos Sybyl[14] examines the Tanimoto similarity between each candidate compound and its nearest neighboring compound (most similar compound) in a reference library. Let $\boldsymbol{X}$ denote an $m \times n$ Tanimoto matrix, in which $X_{i,j}$ is the Tanimoto coefficient between the i^{th} compound in candidate library and the j^{th} compound in reference library. Finally, the distribution of $\boldsymbol{y}$ indicates the degree of overlapping between the two libraries, where $\boldsymbol{y}$ is an m-element vector and $\boldsymbol{y}_i = \max_{1\le j\le n} X_{i,j}$ for $1 \le i \le m$.

Figure 1 shows the general diagram for compound library comparison on GPUs. First, Unity fingerprints are indexed and transferred to allocated graphical memory. Next, depending on the format of the indexed fingerprints, "Sparse Vector" kernel or "Integer Fingerprint" kernel is launched on a GPU to calculate the Tanimoto matrix, $\boldsymbol{X}$. After the Tanimoto matrix calculation, a parallel reduction kernel is executed to identify the maximum of each row of $\boldsymbol{X}$, i.e. $\boldsymbol{y}_i$. Finally, the distribution of $\boldsymbol{y}_i$ is examined through computing and displaying its histogram.

2.1.1. Integer Fingerprint Algorithm on GPUs. The Tanimoto equation consists of three elements: N_{a}, N_{b}, and N_{ab}. In library comparison, the number of "1" bits of each compound, N_{a} or N_{b}, is used repeatedly. Thus, it is wise to precalculate N_{a} and N_{b} in the fingerprint indexing procedure. In this algorithm, binary molecular fingerprints, such as Unity fingerprint, are saved into 32-bit integers. This approach reduces time and space complexity for low-sparsity fingerprints. First, one integer is capable of representing 32 fingerprint bits, so only 124 bytes are required to save the whole Unity fingerprint of any compound (Unity fingerprint has 992 bits). Furthermore, the throughput of calculating N_{ab} can be guaranteed by efficient intrinsic "&" operator and population count (pop-count) instructions on GPUs. The "&" operator finds the common "1" bits between two 32-bit fingerprint fragments ($\mathrm{A}\cap\mathrm{B}$), while the pop-count instruction returns the number of the common "1" bits. Therefore, the total number of common "1" bits between two fingerprints, i.e. N_{ab}, can be obtained through applying "&" and pop-count instructions on every 32-bit fingerprint fragment.

In GPU parallel programming, the i^{th} thread block (or virtual GPU core) is responsible for the i^{th} row of the Tanimoto matrix, $\boldsymbol{X}_{i,\bullet}$, and every thread in the block calculates one or more elements of $\boldsymbol{X}_{i,\bullet}$. Each thread block therefore compares a candidate compound to the whole reference library, and every thread in the block calculates the Tanimoto coefficients between the candidate compound and some reference compounds. For coalesced memory access, the reference and candidate library fingerprints are organized in column and row major 2D arrays, respectively. Figure 2A summarizes algorithm pseudocode for a given thread block, and Figure 2B graphically illustrates the calculation procedure.

2.1.2. Sparse Vector Algorithm on GPUs. The calculation strategy of sparse vector algorithm is similar to the one of integer fingerprint algorithm previously mentioned: the i^{th} thread block (or one virtual core) is responsible for the i^{th} row of the Tanimoto matrix, and every thread in the block calculates one or more elements of that row. However, the sparse vector algorithm adapts an alternative approach to calculate Tanimoto coefficients between pairs of fingerprints, as reported by Haque et al. and Liao et al.[22,23] Figure 3 illustrates the representation of fingerprints in sparse vector format. Figure 3A displays the fingerprints of the i^{th} reference and j^{th} candidate compounds in original binary format. Instead of using the binary format, the fingerprints can be indexed into sparse vector format as well. As shown in Figure 3B, each element indicates the index, of which the bit is "1" in the binary format, and the indices are sorted in an increasing order. In the Tanimoto matrix calculation, the fingerprints from reference library and candidate library are organized in a column-major (Figure 3C) and row-major (Figure 3D) layout in order to achieve coalesced memory access. Sparse vectors of a compound library are sorted according to vector length with the intention of minimizing the possibility of divergent execution branches. For a specific pair of reference and candidate compounds,



(A) **Data:** N_a : the number of "1" bits of the ith candidate fingerprint; vector ***a*** contains the fingerprint fragments of the ith candidate compound; N_b^j : the number of "1" bits of the jth reference fingerprint; vector $\boldsymbol{b_j}$ contains the fingerprint fragments of the jth reference compound; M: the total number of compounds in the reference library; Dim: the number of fingerprint fragments or dimensions. Totally T threads are launched in a thread block (T = 256 in our implementation).

Begin

N_b^j and $\boldsymbol{b_j}$ reside in global memory. Fetch N_a from constant memory to thread register; Fetch vector ***a*** from global memory to low-latency shared memory.

```
for each thread with index t
    for batch = 0 to ceil(M / T) - 1
        j = t + batch × T
        count = 0;
        for k = 0 to Dim - 1
            tmp = a[k] & b_j[k]
            count += popc(tmp)  ‘ intrinsic population count function
        end k
        Tanimoto(a, b_j) = count / (N_a + N_b^j - count)
    end batch
end thread t
End
```

(B)

GPU Grid
N
O
Cl
Fingerprint a: 000001000000100000010000000001000 |a| = 4
Fingerprint b: 0100010000000000000100000100001000 |b| = 5
a ∩ b
000001000000000000001000000000001000
Pop Count
|a ∩ b| = 3
Batch Process
Tanimoto(a,b) = |a∩b| / (|a|+|b|−|a∩b|) = 0.5

Figure 2. (A) The pseudocode for a GPU thread block to calculate the Tanimoto coefficients between a candidate compound and a reference library using integer fingerprint format. (B) Graphical illustration of Tanimoto calculation on a CUDA core. The GPU algorithm is designed to count the number of common elements between two 32-bit fingerprint fragments in one single step.

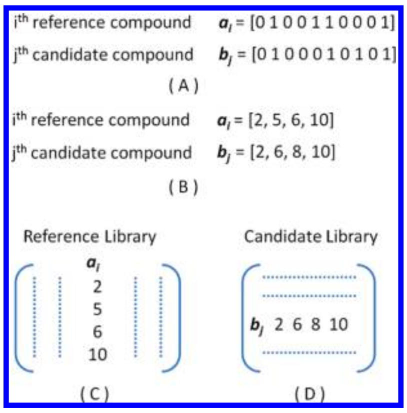
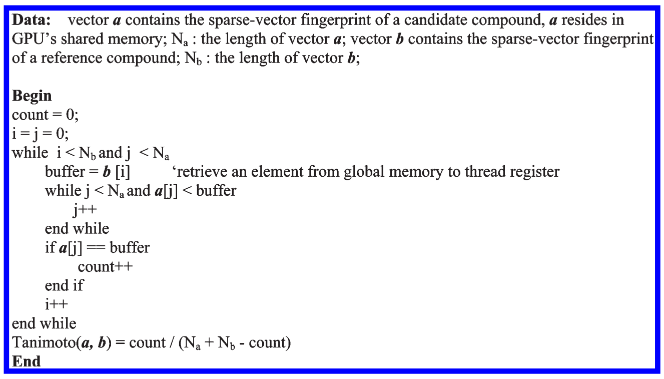

ith reference compound $\boldsymbol{a_i}$ = [0 1 0 0 1 1 0 0 0 1]
jth candidate compound $\boldsymbol{b_j}$ = [0 1 0 0 0 1 0 1 0 1]
(A)
ith reference compound $\boldsymbol{a_i}$ = [2, 5, 6, 10]
jth candidate compound $\boldsymbol{b_j}$ = [2, 6, 8, 10]
(B)
Reference Library
Candidate Library
$\boldsymbol{a_i}$
2
5
6
10
$\boldsymbol{b_j}$ 2 6 8 10
(C)
(D)

Figure 3. Illustration of data structure of sparse vectors: (A) original binary fingerprint format; (B) sparse vector format; (C) column major alignment for reference library; and (D) row major alignment for candidate library.



Data: vector ***a*** contains the sparse-vector fingerprint of a candidate compound, ***a*** resides in GPU's shared memory; N_a : the length of vector ***a***; vector ***b*** contains the sparse-vector fingerprint of a reference compound; N_b : the length of vector ***b***;

```
Begin
count = 0;
i = j = 0;
while  i < N_b and j < N_a
    buffer = b [i]        ‘retrieve an element from global memory to thread register
    while j < N_a and a[j] < buffer
        j++
    end while
    if a[j] == buffer
        count++
    end if
    i++
end while
Tanimoto(a, b) = count / (N_a + N_b - count)
End
```

Figure 4. The pseudocode for calculating the Tanimoto coefficient between a specific pair of compounds that are represented in sparse vector fingerprints.

Figure 4 outlines the algorithm to calculate the Tanimoto coefficient between them.

2.1.3. Find Maximum Tanimoto and Create Histogram on GPUs. A parallel divide-and-conquer algorithm is used to find the maximum values of Tanimoto coefficients. Every thread block is in charge of searching the maximum value from one row of the Tanimoto matrix. In a block, threads independently find the local maximum values from disjoint sets of Tanimoto coefficients. Then, parallel reduction step determines the global maximum (y_i) of each row.

Despite its simple idea, histogram creation is a nontrivial job on GPUs, because GPUs have a limited amount of memory

Table 1. Number of Compounds and Coverage Statistics for Three Testing Compound Libraries[a]

library ID	no. of molecules	lowest coverage (%)	average coverage (%)	highest coverage (%)
A	9902	5.2	19.8	43.2
B	24755	5.3	18.7	44.4
C	56079	2.4	19.1	41.8

[a] Coverage ratio is defined as the percentage of "1" bits in the 992-bit Unity fingerprint. Library A was generated from TimTec 10K-Active-Probes library (http://www.timtec.net/actiprobe-10k.html); Library B was generated from TimTec 25K-Active-Probes library (http://www.timtec.net/actiprobe-25k.html); Library C was generated from Maybridge Screening Collection.

Table 2. Compound Library Comparison Performance Using Integer Fingerprint Algorithm on Machine 2

reference and candidate libraries	time (s)	throughput (kTanimotos/s)	effective bandwidth (GB/s)
A vs A	2.16	45477	5.75
A vs B	5.30	46284	5.86
A vs C	11.97	46398	5.88
B vs A	5.25	46690	5.91
B vs B	13.30	46086	5.84
B vs C	29.61	46885	5.94
C vs A	11.83	46974	5.95
C vs B	29.58	46934	5.94
C vs C	68.09	46184	5.85

Table 3. Compound Library Comparison Performance Using Integer Fingerprint Algorithm on Machine 3

reference and candidate libraries	time (s)	throughput (kTanimotos/s)	effective bandwidth (GB/s)
A vs A	0.47	209507	26.43
A vs B	0.95	257483	32.68
A vs C	2.12	261684	33.18
B vs A	0.94	261884	33.03
B vs B	2.29	267136	33.90
B vs C	4.85	286175	36.26
C vs A	2.00	278064	35.17
C vs B	4.80	288914	36.63
C vs C	11.37	276519	35.03

Table 4. Compound Library Comparison Performance Using Sparse Vector Algorithm on Machine 2

reference and candidate libraries	time (s)	throughput (kTanimotos/s)
A vs A	22.11	4434
A vs B	52.92	4631
A vs C	122.75	4523
B vs A	53.33	4596
B vs B	126.63	4839
B vs C	295.00	4705
C vs A	122.14	4546
C vs B	291.38	4764
C vs C	674.17	4664

allowing low-latency random access. In this case, GPU threads maintain conflict-free counters to create subhistograms in parallel. Finally, a 100-bin histogram is generated by merging all the subhistograms. This histogram displays the distribution of y_i and depicts the degree of similarity between two compound libraries.

2.2. Computation Protocols. The computing performance of the established GPU-accelerated algorithms was evaluated through comparing three compound libraries against each other. For the three libraries, 992-bit Unity fingerprints were generated by Tripos Sybyl software and formatted into the data structures required by sparse vector or integer fingerprint algorithms. Details regarding the testing compound libraries can be found in Table 1, including the TimTec libraries (Library A and B) and Maybridge Screening Collection (Library C). Sometimes, a data entry in SDF files may contain two or more separate structures, e.g., compound and organic solvent, and such entries have been removed from these libraries. The coverage statistics in Table 1 reflects the sparsity of Unity fingerprints, when applied on the sample libraries.

The GPU integer fingerprint and sparse vector algorithms were implemented in our CudaCLA program. The executable file, together with source code and documents, is available at http://www.cbligand.org/gpu. The program was compiled by Visual Studio C++ 2005 and CUDA 3.0 toolkit and implemented with graphical user interface. For comparison studies, we selected three computer systems, including the following:

- Machine 1: CPU: Intel Xeon 5160 at 3.0 GHz
- Machine 2: CPU: AMD Athlon 3800+ at 2.0 GHz; GPU NVIDIA Quadro FX 580 (32 CUDA cores at 1.12 GHz); NVIDIA drivers: 197.13
- Machine 3: CPU: Intel Core i7 860 at 2.8 GHz; GPU NVIDIA Geforce GTS 250 (128 CUDA cores at 1.78 GHz); NVIDIA drivers: 197.13

The performance of our GPU program was compared to the commercial Sybyl Database Comparison[14] program. Sybyl Database Comparison program, as a CPU program, was tested on the three machines, while the performance of integer fingerprint and sparse vector algorithms were tested on GPUs of machine 2 and 3. It is worth pointing out that the Sybyl Database Comparison program is also based on Unity fingerprints.

3. RESULTS AND DISCUSSION

The GPU-accelerated compound library comparison is developed for the first time using integer fingerprint algorithm, and its performance is summarized in Tables 2 and 3. For comparisons, the performance of sparse vector algorithm is listed in Tables 4 and 5. In the tables, the calculation time includes the time spent on graphical memory allocation, data transfer, Tanimoto calculation using either algorithm, searching for the maximum values, and histogram creation (the procedures enclosed by dotted line in Figure 1). Tanimoto matrix calculation generally accounts for more than 95% of the GPU time. Therefore, kTaninotos/s (kilo Tanimoto coefficients per second) is simply used as performance metric. The results from testing the Sybyl Database Comparison program on the three machines can be found in Table 6.

3.1. GPU-Based Sparse Vector Algorithm Compared with Published Results. Our implementation of sparse vector algorithm achieved an average throughput of 28634 kTanimotos/s on GTS 250, the graphics device of machine 3 (Table 5). Haque et al.[22] attained 60900 to 64230 kLINGOS/s on the same GPU, GTS 250. Despite certain difference between Tanimoto and LINGO, the same underlying algorithms find the overlapping structural patterns. In addition, the sparse vector algorithm has a linear time complexity in the length of sparse vectors. The average length of sparse vectors in Haque's work ranged from 29.31 to 31.65 out of thousands of possible patterns. On the other hand, the average of sparse vectors in our sample libraries ranged from 185 to 196 out of a total of 992. Given the lower sparsity of the Unity fingerprints and the longer length of the sparse vectors in this study, the performance of our implementation of sparse vector algorithm was comparable to Haque's results. Similarly, Liao et al.[23] reported that all-vs-all Tanimoto matrix calculation for PB83 library took 17.1 s on device GTX 280 (240 CUDA cores). In their study, the PB83 library contained 27674 molecules, and the maximum length of sparse vectors was 459. In our case, the library comparison of Library B to itself (Table 5) took 20.45 s on GTS 250 (128 CUDA cores). Library B contained 24755 molecules and the maximum length of sparse vectors was 440, which was similar to PB83. Note that GTX 280 had higher memory bandwidth and more cores than GTS 250. Although these experiments were carried out with different machines, fingerprints, and data sets, the side-by-side analysis showed that the performance of our implemented sparse vector algorithm was at the same magnitude of published results.

Table 5. Compound Library Comparison Performance Using Sparse Vector Algorithm on Machine 3

reference and candidate libraries	time (s)	throughput (kTanimotos/s)
A vs A	3.68	26636
A vs B	8.71	28158
A vs C	20.08	27658
B vs A	8.63	28413
B vs B	20.45	29963
B vs C	47.35	29320
C vs A	19.53	28429
C vs B	46.54	29831
C vs C	107.33	29301

3.2. Integer Fingerprint Algorithm versus Sparse Vector Algorithm.

Sparse vector algorithm is designed to handle high-sparsity molecular fingerprints. However, many of the popular molecular fingerprints are binary and have low sparsity, such as Unity, FP2, MACCS, and PubChem fingerprints. The integer fingerprint algorithm is capable of attaining higher computation efficiency for these types of fingerprints, because it can process many fingerprint "bits" in a single iteration. On the contrary, the sparse vector algorithm scans through every present structural pattern to search for the overlapping ones. Moreover, the "while" and "if-else" statements in the algorithm easily result in divergent execution paths on GPU, which reduces the degree of parallelism. The advantage of integer fingerprint algorithm on low-sparsity fingerprint is demonstrated by the experiment. As shown in Tables 4 and 5, the throughput of sparse vector algorithm varies from 4434 kTanimotos/s to 4839 kTanimotos/s on the GPU of machine 2 and from 26636 kTanimotos/s to 29963 kTanimotos/s on the GPU of machine 3. Integer fingerprint algorithm delivers much higher throughput ranging from 45477 to 46974 on the GPU of machine 2 (Table 2) and from 209507 kTanimotos/s to 288914 kTanimotos/s on the GPU of machine 3. Among the nine cases, integer fingerprint algorithm is 9.5 to 10.3 times faster than sparse vector algorithm on machine 2 and 7.9 to 9.8 times faster than sparse vector algorithm on machine 3.

The performance of integer fingerprint algorithm is further assessed by effective memory bandwidth (the rate at which data is read from, and stored to graphical memory). The effective bandwidth of library comparison using integer fingerprint algorithm can be determined according to the following equation

$$\frac{n \times (m+1) \times 124 + n \times (m+1) \times 4 + n \times m \times 2 \times 4 + n \times 2 \times 4}{\text{time in seconds}} \times 2^{-30}\,\text{GByte/s}$$

where n is the size of a candidate library, and m is the size of a reference library. Each virtual core reads a candidate fingerprint into cache and compares it to the whole reference library. As there are n candidate compounds and a Unity fingerprint occupies 124 bytes, the Tanimoto matrix calculation reads $n \times (m + 1) \times 124$ bytes from graphical memory. The number of "1"

Table 6. Performance of Sybyl Database Comparison Program That Carries out Compound Library Comparison with Unity Fingerprint

	machine 1		machine 2		machine 3	
reference and candidate libraries	time (s)	[a]effective throughput (kTanimotos/s)	time (s)	[a]effective throughput (kTanimotos/s)	time (s)	[a]effective throughput (kTanimotos/s)
A vs A	8.38	11689	34.38	2852	10.58	9267
A vs B	23.60	10385	90.19	2718	29.38	8343
A vs C	78.77	7049	306.57	1811	98.02	5663
B vs A	21.96	11161	86.14	2845	27.78	8823
B vs B	52.98	11569	208.63	2937	66.17	9260
B vs C	190.91	7268	748.42	1854	235.45	5895
C vs A	73.34	7568	284.55	1951	91.59	6061
C vs B	176.48	7846	689.49	2013	225.55	6152
C vs C	393.43	7988	1535.34	2047	511.54	6143

[a] For easy comparison, the effective throughput is approximated assuming that the program finishes the whole Tanimoto matrix calculation, which is not necessarily required in practice. The throughput is in the unit of kTanimotos/s.

bits of reference fingerprints and candidate fingerprints are accessed by each core, which brings $n \times (m + 1) \times 4$ byte data. The Tanimoto matrix has $n \times$ m float-type (32-bit) elements. The matrix is accessed twice (matrix generation and searching for maximum Tanimoto coefficients). GPU therefore reads and writes $n \times m \times 2 \times 4$ bytes. Finally, the array containing the maximum of each row of the Tanimoto matrix is accessed, which leads to input and output of $n \times 2 \times 4$ bytes. In fact, the effective memory bandwidth is somehow underestimated by this equation, because the total calculation time includes some other procedures, such as memory allocation and host-to-device data transfer.

As summarized in Tables 2 and 3, the implemented integer fingerprint achieved average effective bandwidth of 5.88GB/s on machine 2 and 33.59GB/s on machine 3. The program from NVIDIA SDK[25] was used to test peak memory bandwidth, revealing that the GPU of machine 2 had 12.88GB/s device-to-device copy bandwidth, and the GPU of machine 3 had 57.36GB/s device-to-device copy bandwidth. The program therefore made good utilization of hardware resources.

Due to the use of intrinsic instructions ("&" operator and population count), integer fingerprint algorithm yielded much higher throughput than sparse vector algorithm regarding Tanimoto calculation. The analysis on effective memory bandwidth further depicted absolute computation performance. As a result, integer fingerprint algorithm is considerably superior to sparse vector algorithm when Tanimoto coefficient is measured on low-sparsity fingerprints, e.g., Unity fingerprint.

3.3. Performance of GPU- and CPU-Based Programs for Compound Library Comparison. Table 6 shows that Sybyl Database Comparison, as a traditional CPU program, runs fastest on machine 1, with an average throughput of 9169 kTanomotos/s. Nevertheless, the program yields an average throughput of 2336 kTanimotos/s on machine 2 and 7290 kTanimotos/s on machine 3. Thus, the performance of Database Comparison program on machine 1 is compared to the GPU program. The GPU implementation of compound library comparison using integer fingerprint algorithm shows an average 5.28 times speedup on the GPU of machine 2 (Table 2) in comparison with Sybyl Database Comparison program on machine 1 (Table 6). Further speedup can be seen on machine 3. Table 3 and Table 6 show that the GPU implementation is up to 39.47 times as fast as the CPU program and 30.44 times speedup on average. FX 580 (the graphic device in machine 2) is considered as a low-end or entry-level device, as it only has 32 CUDA cores. Nevertheless, the GPU program still runs much faster than the commercial CPU program. This is mainly due to the multicore parallel computing architecture of modern GPUs. In the GPU program, multiple threads are launched concurrently for Tanimoto calculation, searching for the maxima, histogram creation, and so on.

Additionally, the GPU program using sparse vector algorithm shows an average 3.26 times speedup relative to Sybyl Database Comparison program, when run on machine 3 (see Table 5 and Table 6). Nevertheless, the implementation using sparse vector algorithm is not necessarily as fast as the CPU program, when tested on machine 2. The integer fingerprint algorithm significantly outperforms the Database Comparison program even on low-end device, whereas the sparse vector algorithm only shows moderate speedup on machine 3. These results illustrate the fact that extra attention is required for GPU algorithm design and implementation in order to achieve optimal performance.

3.4. Validation on PubChem Database. The integer fingerprint algorithm was further tested on a large compound database

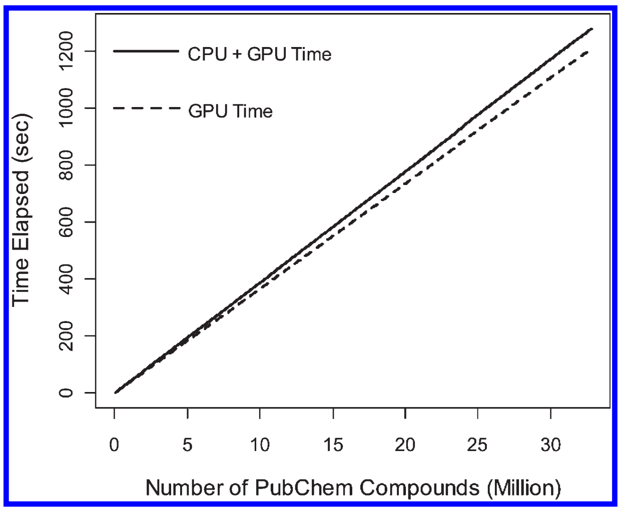


Figure 5. The plot of elapsed time versus processed PubChem compounds. Dashed line shows the elapsed GPU time as a function of the number of compounds. The GPU time includes time allocated for host-device data transfer, Tanimoto matrix calculation, histogram creation, and so on (Figure 1). The solid line demonstrates the total calculation time. This is equal to GPU time plus CPU time. CPU time is for hard drive I/O, task scheduling, etc.

with the intention to examine its scalability and consistency. In this experiment, Library A (9902 compounds) was compared with 32.79 M PubChem structures (by April 2011). The computation was performed on machine 3, and 324.69G Tanimoto coefficients were generated. Figure 5 plots the amount of aggregated time as a function of the number of PubChem structures. The whole process took 1279 s, and GPU time accounted for 94% of it, i.e., 1207 s. Thus, the average throughput was 269041 kTanimotos/s, which resembled the results summarized in Table 3. Figure 5 also reveals that the elapsed time is strictly linear to the quantity of processed structures, suggesting that the program generates stable throughput regardless of diverse structures.

4. CONCLUSION

The ever-growing chemical libraries demand the development of efficient algorithms and programs for chemical similarity calculation that plays a fundamental role in cheminformatics. As a similarity index, Tanimoto coefficient is widely involved in data mining of small molecules, such as compound clustering, diversity analysis, and k-nearest-neighbor search (kNN). However, the quadratic time complexity in the number of compounds could be a problem for these techniques, particularly when large compound data sets are presented. Parallel Tanimoto calculation on modern GPUs points out a potential solution to these challenges. In this article, we report a GPU-accelerated integer fingerprint algorithm and its application for calculating Tanimoto coefficients. The test calculation shows that the integer fingerprint algorithm runs up to 10 times faster than the published sparse vector algorithm on GPUs and up to 39 times faster than the CPU-based commercial program. The GPU-accelerated integer fingerprint algorithm produces high throughput for low-sparsity binary fingerprint. For example, more than 200 million Tanimoto coefficients can be calculated every second on a decent device, such as GTS 250. With this speed, comparing 9902 Active Probe compounds with PubChem library, that has totally 324G Tanimoto coefficients to calculate, can be completed in about 20 min. In this study, compound library comparison can be interpreted as 1-nearest-neighbor search. This approach could be easily upgraded to k-nearest-neighbor search by a minor modification (The source code is accessible at http://www.cbligand.org/gpu.). Additionally, the all-vs-all Tanimoto matrix calculation in the GPU program can also be adapted by other algorithms. Currently, we are conducting the application of the GPU-based algorithm to modeling quantitative structure−activity relationship for large data sets.

AUTHOR INFORMATION

Corresponding Author

*Phone: 412-383-5276; Fax: 412-383-7436. E-mail: xix15@pitt.edu.

ACKNOWLEDGMENT

This project is supported by grants from NIGMS P50-GM067082, R01DA025612 and Pitt CRDFA fund 37969. We would like to thank Dr. Herbert Barry III for proofreading of the manuscript.

REFERENCES

(1) Harper, G.; Pickett, S. D. Methods for mining HTS data. *Drug Discovery Today* **2006**, *11* (15−16), 694–9.

(2) Nikolova, N.; Jaworska, J. Approaches to Measure Chemical Similarity − a Review. *QSAR Comb. Sci.* **2003**, *22* (9−10), 1006–1026.

(3) Nilakantan, R.; Bauman, N.; Haraki, K. S. Database diversity assessment: New ideas, concepts, and tools. *J. Comput.-Aided Mol. Des.* **1997**, *11* (5), 447–452.

(4) Shemetulskis, N. E.; Weininger, D.; Blankley, C. J.; Yang, J. J.; Humblet, C. Stigmata: An Algorithm To Determine Structural Commonalities in Diverse Datasets. *J. Chem. Inf. Comput. Sci.* **1996**, *36* (4), 862–871.

(5) Boyd, S. M.; Beverley, M.; Norskov, L.; Hubbard, R. E. Characterising the geometric diversity of functional groups in chemical databases. *J. Comput.-Aided Mol. Des.* **1995**, *9* (5), 417–424.

(6) Martin, E. J.; Blaney, J. M.; Siani, M. A.; Spellmeyer, D. C.; Wong, A. K.; Moos, W. H. Measuring Diversity: Experimental Design of Combinatorial Libraries for Drug Discovery. *J. Med. Chem.* **1995**, *38* (9), 1431–1436.

(7) Cummins, D. J.; Andrews, C. W.; Bentley, J. A.; Cory, M. Molecular Diversity in Chemical Databases: Comparison of Medicinal Chemistry Knowledge Bases and Databases of Commercially Available Compounds. *J. Chem. Inf. Comput. Sci.* **1996**, *36* (4), 750–763.

(8) Singh, N.; Guha, R.; Giulianotti, M. A.; Pinilla, C.; Houghten, R. A.; Medina-Franco, J. L. Chemoinformatic Analysis of Combinatorial Libraries, Drugs, Natural Products, and Molecular Libraries Small Molecule Repository. *J. Chem. Inf. Comput. Sci.* **2009**, *49* (4), 1010–1024.

(9) Zhang, L.; Xiao, Q.; Ma, C.; Xie, X.-Q.; Floreancig, P. E. Construction of a Bicyclic β-Benzyloxy and β-Hydroxy Amide Library through a Multicomponent Cyclization Reaction. *J. Comb. Chem.* **2009**, *11* (4), 640–644.

(10) Bajorath, J. Integration of virtual and high-throughput screening. *Nat. Rev. Drug Discovery* **2002**, *1* (11), 882–94.

(11) Martin, Y. C.; Kofron, J. L.; Traphagen, L. M. Do structurally similar molecules have similar biological activity?. *J. Med. Chem.* **2002**, *45* (19), 4350–8.

(12) Xie, X.-Q.; Chen, J.-Z. Data Mining a Small Molecule Drug Screening Representative Subset from NIH PubChem. *J. Chem. Inf. Model.* **2008**, *48* (3), 465–475.

(13) *SYBYL 8.0*; Tripos International: 1699 South Hanley Rd., St. Louis, Missouri, 63144, USA, 2008.

(14) *Tripos Bookshelf 8.0*; Tripos International: 1699 South Hanley Rd., St. Louis, Missouri, 63144, USA, 2007.

(15) Wassermann, A. M.; Geppert, H.; Bajorath, J. Searching for Target-Selective Compounds Using Different Combinations of Multiclass Support Vector Machine Ranking Methods, Kernel Functions, and Fingerprint Descriptors. *J. Chem. Inf. Model.* **2009**, *49* (3), 582–592.

(16) 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.

(17) Butina, D. Unsupervised Data Base Clustering Based on Daylight's Fingerprint and Tanimoto Similarity: A Fast and Automated Way To Cluster Small and Large Data Sets. *J. Chem. Inf. Comput. Sci.* **1999**, *39* (4), 747–750.

(18) Wang, Y.; Xiao, J.; Suzek, T. O.; Zhang, J.; Wang, J.; Bryant, S. H. PubChem: a public information system for analyzing bioactivities of small molecules. *Nucleic Acids Res.* **2009**, *37* (Web Server issue), W623–33.

(19) Xie, X.-Q. Exploiting PubChem for virtual screening. *Expert Opin. Drug Discovery* **2010**, *5* (12), 1205–1220.

(20) Götz, A. W.; Wölfle, T.; Walker, R. C., Quantum Chemistry on Graphics Processing Units. In *Annu. Rep. Comput. Chem.*, Ralph, A. W., Ed.; Elsevier: 2010; Vol. 6, pp 21−35.

(21) Xu, D.; Williamson, M. J.; Walker, R. C., Advancements in Molecular Dynamics Simulations of Biomolecules on Graphical Processing Units. In *Annu. Rep. Comput. Chem.*, Ralph, A. W., Ed.; Elsevier: 2010; Vol. 6, pp 2−19.

(22) 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* (4), 560–564.

(23) 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* (12), 2718–25.

(24) Sachdeva, V.; Freimuth, D.; Mueller, C., Evaluating the Jaccard-Tanimoto Index on Multi-core Architectures. In *Computational Science − ICCS 2009*; Allen, G.; Nabrzyski, J.; Seidel, E.; van Albada, G.; Dongarra, J.; Sloot, P., Eds.; Springer: Berlin/Heidelberg: 2009; Vol. 5544, pp 944−953.

(25) NVIDIA CUDA ZONE. http://www.nvidia.com/object/cuda_home_new.html (accessed Dec 10, 2010).