Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 2007–2016

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

GPU-accelerated Chemical Similarity Assessment for Large Scale Databases

Marco Maggioni[a,b], Marco Domenico Santambrogio[a,c,d], Jie Liang[a,b]

[a] *Department of Computer Science, University of Illinois at Chicago*
[b] *Department of Bioengineering, University of Illinois at Chicago*
[c] *Department of Computer Engineering, Politecnico di Milano*
[d] *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology*

Abstract

The assessment of chemical similarity between molecules is a basic operation in chemoinformatics, a computational area concerning with the manipulation of chemical structural information. Comparing molecules is the basis for a wide range of applications such as searching in chemical databases, training prediction models for virtual screening or aggregating clusters of similar compounds. However, currently available multimillion databases represent a challenge for conventional chemoinformatics algorithms raising the necessity for faster similarity methods. In this paper, we extensively analyze the advantages of using many-core architectures for calculating some commonly-used chemical similarity coefficients such as Tanimoto, Dice or Cosine. Our aim is to provide a wide-breath *proof-of-concept* regarding the usefulness of GPU architectures to chemoinformatics, a class of computing problems still uncovered. In our work, we present a general GPU algorithm for all-to-all chemical comparisons considering both binary fingerprints and floating point descriptors as molecule representation. Subsequently, we adopt optimization techniques to minimize global memory accesses and to further improve efficiency. We test the proposed algorithm on different experimental setups, a laptop with a low-end GPU and a desktop with a more performant GPU. In the former case, we obtain a 4-to-6-fold speed-up over a single-core implementation for fingerprints and a 4-to-7-fold speed-up for descriptors. In the latter case, we respectively obtain a 195-to-206-fold speed-up and a 100-to-328-fold speed-up.

Keywords: chemoinformatics, GPU, chemical similarity, chemical fingerprints, Tanimoto coefficient

1. Introduction

As a result of decades of research, chemoinformatics is considered a well-established field concerned with the application of computational methods to tackle chemical problems, with particular emphasis on the manipulation of chemical structural information [1]. Some of its applications are strictly related with the capability of assessing chemical similarity between compounds. For example, searching in chemical databases requires thousands of comparisons in order to return the most similar compounds to a query molecule [2]. Another example is a process known as *virtual screening* [3]. After a learning phase based on data mining algorithms, a model can predict if an arbitrary molecule

Email addresses: mmaggi3@uic.edu (Marco Maggioni), marco.santambrogio@polimi.it (Marco Domenico Santambrogio), jliang@uic.edu (Jie Liang)

1877–0509 © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.219

has the desired biological/chemical activity, reducing the number and the cost of *in-vitro* experiments. Considering Support Vector Machines [4], the model learning is based on a kernel function between molecules which roughly corresponds to chemical similarity. In addition, it is useful to apply unsupervised learning in order to aggregate molecules which shares similar activity [5]. Thus, the assessment of compound similarity has a central role in chemoinformatics.

Currently-available multimillion databases such as PubChem [6] (31 million molecules) represent a challenge for conventional chemoinformatics algorithms. The situation is even worst considering virtual libraries constructed using *combinatorial chemistry* (nearly 1 billion molecules in [7]). Scalability problems arise not only from the database size but also from the complexity of the considered algorithms. While a single database search has linear complexity $O(n)$, more sophisticated algorithms such as agglomerative hierarchical clustering may need quadratic run time $O(n^2)$ or worse. Consequently, chemoinformatics analyses can be practical only if we are able to increase the execution throughput of such a large number of compound comparisons beneath the algorithms.

Nowadays computer architectures are designed to offer an increasing level of parallelism directly available for improving the computation throughput. In accordance with this perspective, GPUs have imposed themselves as a pervasive scientific parallel architectures [8]. GPUs are pervasive since they are integral part of almost any modern computer system. GPUs are scientific since they provide high-throughput for floating point operations (up to 1.03 TFLOP/s), a desirable feature for scientific computing applications. Finally, current GPUs are parallel since they are designed according to the many-core design philosophy. Large cache memories and complex control units are replaced by a large number of simple processing cores (up to 512 in a single chip) relying in a highly parallel workload with a very regular execution pattern. Important scientific computing areas such as sequence alignment [9] and molecular dynamics simulation [10, 11] have seen an increasing popularity for GPU computing. In fact, GPUs make available to scientists and researchers around the world a supercomputing architecture at a small fraction of the cost and power consumption compared to large supercomputers or clusters. This trend has not yet been observed for chemoinformatics applications. To the best of our knowledge, very few works combine chemoinformatics problems with parallel architectures, in particular with GPUs [12, 13].

The idea of implementing similarity calculations onto a GPU is not completely new. In [12], the author consider molecules represented as canonical strings known as SMILES. Consequently, chemical similarity is based on text comparison. Despite the compelling advantage of simplicity and accuracy comparable to binary fingerprints, SMILES cannot embed structural or chemical information as for floating point descriptors. The work presented in [13] considers instead binary fingerprints and Tanimoto coefficients for learning a SVM model. However, chemoinformatics algorithms use a wide range of molecule representation and similarity coefficient so the presented analysis does not appear comprehensive. Finally, in [14], the evaluation of Tanimoto coefficient of binary fingerprints on three different architectures (a quad-core Intel architecture, a Cell architecture and a GPU architecture) have been presented. The underlying idea of comparison between different design philosophies is interesting but the proposed GPU implementation is not fairly optimized, making the entire results biased toward the quad-core and the Cell architectures.Focusing on the proposed results, it looks unusual that a single CPU thread may run three times faster than a GPU for calculating a large number of floating point Tanimoto coefficients.

The main contribution of this work is a detailed analysis of the advantages of using many-core architectures for calculating five different commonly-used similarity coefficients (Tanimoto, Dice, Cosine, Euclidean and Hamming/Manhattan) considering both binary fingerprints and floating point descriptors as molecule representation. The practical goal is to achieve an higher execution throughput in order to cope with the aforementioned scalability issues. We propose a general GPU algorithm which outperforms the single-core CPU execution up to two orders of magnitude. Our intellectual aim is to provide a wide-breath *proof-of-concept* regarding the usefulness of GPU architectures in chemoinformatics, a class of computing problems still uncovered.

The rest of the paper is organized in the following way. Section 2 briefly presents two common techniques used for producing a binary fingerprint and a vector of real descriptors given a molecule. Moreover, it introduces several commonly-used chemical similarity coefficients. Section 3 is then dedicated to redesign the algorithm in order to be adapted to a many-core architecture. Moreover, we present further optimizations in order to minimize accesses to global memory and increase the achieved speed-up. Section 4 is dedicated to performance benchmarks considering different experimental setups, a laptop with a low-end GPU and a desktop with a more performant GPU. In the former case, we obtained a 4-to-6-fold speed-up over a single-core implementation for fingerprints and a 4-to-7-fold speed-up for descriptors. In the latter case, we respectively obtained a 195-to-206-fold speed-up and a 100-to-328-fold speed-up. Finally, Section 5 concludes with some final statements and suggestions for future work.

2. Chemical similarity

The rationale for similarity assessment lies in the *similar property principle* [1] for which structurally similar molecules tend to have similar properties and likely to exhibit the same activity. This principle permits to identify sets of potentially active compounds or drugs *in-silico* reducing the number and the cost of *in-vitro* experiments. In general, it is necessary an appropriate representation of the molecules in order to cope with the presented scalability issues while maintaining a sufficient amount of information regarding the real-world compound. Techniques based on 3D graphs and 3D superimposition methods [15] have a too high computational complexity for being applied to large molecule databases. In our work, we focus on more efficient and compact representations such as binary fingerprints and floating point descriptors. From the many-core architecture perspective, it may be argued that this choice is beneficial since GPUs can efficiently provide high-throughput calculations, especially for floating point numbers [16]. Moreover, fingerprints and descriptors preserve underlying topological and structural information necessary for having a meaningful chemical similarity assessment.

2.1. Binary fingerprints

Fingerprints represent molecules as fixed-length binary vectors where each bit corresponds to the presence of a specific dictionary fragment as depitced by Figure 1. Given an arbitrary molecule, its fingerprint can be calculated using a subgraph isomorphism algorithm [17]. This operation may be computationally expensive but it is done only once so its cost is negligible during similarity assessment. The underlying advantage of fingerprints is that topological information regarding the atomic structure can be captured. Consequently, there is a correlation between fingerprints similarity and chemical/biological similarity. On the other hand, the selection of the fragment dictionary is crucial for obtaining good results in the target chemoinformatics application. Fragments should not be too common or too rare otherwise information encoded in the fingerprint bits becomes not very discriminating or relevant for only few molecules. A dictionary can be selected by domain experts or using data mining techniques. In more detail, fingerprints based on a dictionary derived by acyclic fragment mining has been shown to be more effective for compound retrieval and classification [18].

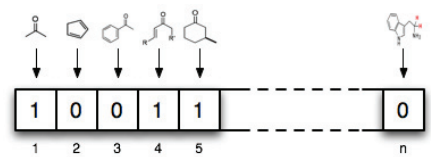

Figure 1: Each bit in the fingerprint corresponds to a fragment dictionary.

2.2. Floating point descriptors

Vectors of floating point descriptors are more suitable than fingerprints for characterizing physicochemical properties such as shape or charge. Not surprisingly, the embedding of this information produces better results for the underlying chemoinformatics application as shown in [19]. Descriptor values are usually obtained by applying algorithmic techniques to the molecular structures. The technique presented in [19] provides an highly accurate description of molecule 3D shapes by capturing the inter-atom distances as a 12-dimensional real vector. In more detail, the technique defines four precise points in the 3D space and calculates the distance from each atom, identifying four distinct distance distributions (one for each fixed point) as depicted in Figure 2. From statistical theory, a distribution is completely determined by its moments. For this reason, the technique is able to approximate the molecular 3D shape by taking the first three moments $u_k = \sum_{i=1}^{n} d_i^k / n$ of each distance distribution, producing a total of 12 real descriptors. The technique has been successively enriched with three additional descriptors in order to incorporate chirality and electrostatic considerations [20]. It may be argued that the *similar property principle* still holds. In fact, molecules with similar descriptors will have a similar 3D shape and then will likely exhibit the same activity. Last but not least, the descriptors are real-valued so any implemented operation will benefit of the floating point capabilities of GPU architectures.

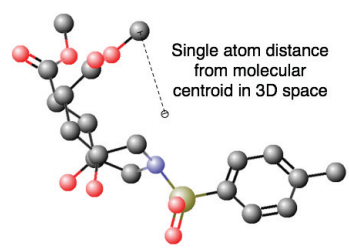

Figure 2: A numerical distribution can be created measuring the distance of each atom from a point (i.e. a centroid).

2.3. Chemical similarity coefficients

We just described how to represent a molecule as a binary fingerprint or a floating point vector. Given two molecules, their similarity can be evaluated by many different similarity coefficients that have been developed during years. The most commonly-used in chemoinformatics are reported in Table 1. All the coefficients (included euclidean and manhattan/hamming) are formulated in order to directly measure similarity. Given a molecule A, x_{iA} represents the i-component of the associated vector of floating point descriptors whereas a represents the numbers of bits sets to one in the correspondent fingerprint. Component x_{iB} and b have an analogous meaning for molecule B. Finally, c represents the number of bits that are one for both the fingerprints. It may be argued that Tanimoto, Dice and Cosine coefficients focus on common bits as evidence of similarity whereas euclidean and manhattan coefficient consider a common absence (zero bit) as a point of similarity.

Name	Real Vectors	Binary Fingerprints
Tanimoto	$S_{AB} = \frac{\sum_{i=1}^{n} x_{iA}x_{iB}}{\sum_{i=1}^{n}(x_{iA})^2 + \sum_{i=1}^{n}(x_{iB})^2 - \sum_{i=1}^{n} x_{iA}x_{iB}}$	$S_{AB} = \frac{c}{a+b-c}$
Dice	$S_{AB} = \frac{2\sum_{i=1}^{n} x_{iA}x_{iB}}{\sum_{i=1}^{n}(x_{iA})^2 + \sum_{i=1}^{n}(x_{iB})^2}$	$S_{AB} = \frac{2c}{a+b}$
Cosine	$S_{AB} = \frac{\sum_{i=1}^{n} x_{iA}x_{iB}}{\sqrt{\sum_{i=1}^{n}(x_{iA})^2 \sum_{i=1}^{n}(x_{iB})^2}}$	$S_{AB} = \frac{c}{\sqrt{ab}}$
Euclidean	$S_{AB} = \frac{1}{1 + \sqrt{\sum_{i=1}^{n}(x_{iA} - x_{iB})^2}}$	$S_{AB} = \frac{1}{1 + \sqrt{a+b-2c}}$
Manhattan/Euclidean	$S_{AB} = \frac{1}{1 + \sum_{i=1}^{n} \lvert x_{iA} - x_{iB} \rvert}$	$S_{AB} = \frac{1}{1+a+b-2c}$

Table 1: Chemoinformatics similarity coefficients adapted from [1].

3. GPU algorithm and optimizations

The assessment of chemical similarity corresponds to calculating a coefficient using floating point values (which are directly available as real descriptors) or integer values (which are derived from fingerprint bit counting). A wide range of chemoinformatics applications such as molecule clustering needs a quadratic number $O(n^2)$ of chemical similarity comparisons. Consequently, it seems reasonable to evaluate the efficiency of the GPU architecture by performing exactly n^2 comparisons (all-to-all similarity coefficient calculations). In our discussion we specifically refer to the NVIDIA's Compute Unified Device Architecture (CUDA) [8]. Due to space constraints, we introduce architectural details only when it is necessary for explaining the algorithm and the introduced optimizations. A more complete description of the considered many-core architecture and its programming model can be found in [8, 16]. For similar reasons, we focus our explanation effort on graphical representations of the proposed GPU algorithmic techniques, avoiding long sections with CUDA-like pseudocode.

3.1. Basic algorithm

The overall computation can be intuitively seen as a square matrix of independent similarity computations between column molecules and row molecules that can be executed simultaneously. This algorithmic structure represents a

case of embarrassingly parallel workload, thus is perfect for being executed on a many-core architecture. In more detail, each similarity comparison is associated with a GPU execution thread as shown by Figure 3. According to the underlying architecture, threads are logically organized in blocks. Analogously, GPU cores (also known as Stream Processors) are organized in groups of fixed size (also known as Stream Multiprocessor). At runtime, thread blocks are assigned to SMs which simultaneously execute threads on the correspondent SPs according to a warped Single Instruction Multiple Thread (SIMT) paradigm [8].

As mentioned, we cannot directly compute fingerprint coefficients as a floating point operation since we need to derive bit counting from binary vectors. Some computer architectures natively offer this operation as part of the instruction set. The GPU algorithm presented in [14] uses an approach based on repeated right-shifts for implementing the counting. We prefer an approach based on a 8-bit lookup table. Moreover, we keep this data structure in the constant memory, a special GPU memory which is equivalent to a fast read-only cache shared between all the threads. Given a fingerprint, we accumulate the total bit counting by scanning each byte and by adding the correspondent count value read from the lookup table. A simple consideration about the coefficients can speed up both CPU and GPU executions. It may be argued that given a molecule A it is necessary to compute $\sum_{i=1}^{n}(x_{iA})^2$ and a just once even if molecule A is involved in multiple chemical comparisons. Thus, it is reasonable to add a precomputing phase after which the intermediate values $\sum_{i=1}^{n}(x_{iA})^2$ and a are stored in memory for being reused later to calculate actual similarity coefficients. Since each molecule precomputing is independent, the procedure can be easily implemented on a GPU architecture as well, assigning a GPU execution thread for each computation.

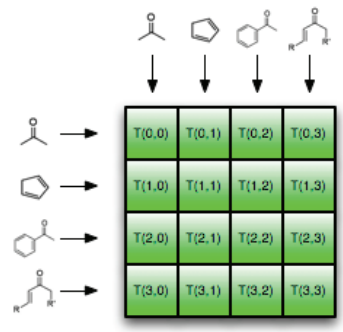

Figure 3: Chemical similarity computation decomposed into GPU threads.

3.2. Tiling technique

The presented GPU algorithm can benefit the large number computational units available on a many-core architecture. However, in most of the available GPU architectures threads are forced to directly access a large global memory without taking advantage of any transparent caching mechanism for exploiting data locality. As seen from the unimpressive GPU results obtained in [14], this drawback may represent a severe bottleneck which slows down the computation up to a disadvantageous speed up compared to single-core CPU execution. On the other hand, GPU architectures make available for each SM a fast cache memory known as shared memory which can be used for resolving the highlighted problem. We can also argue that the explicit caching technique presented in this section is beneficial for GPUs based on the recent Fermi architecture [8] (the first GPU with a transparent cache memory).

A general optimization technique known as *tiling* is available in literature [16, 21]. The underlying idea takes advantage of the CUDA programming model for which threads in the same block can utilize shared memory for caching data and synchronizing themselves. Thus, it is possible to decompose the original computation $n \times n$ matrix into $b \times b$ tiles (or blocks). Within a tile, threads need to access b row molecules and b column molecules for performing chemical similarity comparisons. Instead of having two accesses to global memory for each threads, molecules are collaboratively preloaded in shared memory as shown by Figure 4. The subsequent memory accesses will be made locally reducing the number of global memory accesses by a b factor. In more detail, we have $2b^2$ accesses without tiling. Considering a total of $2b$ accesses for preloading the molecules, the reduction factor can be easily calculated as $2b^2/2b = b$. In other words, larger tiles permit a better cache sharing leading to a reduction in the number of accesses to global memory. Practically speaking, we should maintain blocks as large as

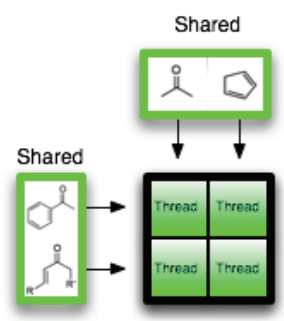

Figure 4: A 2x2 thread tile preloads row and column molecules in shared memory.

possible keeping in mind that there exist architectural constraints regarding the shared memory available for a SM and the maximum number of threads that a SM scheduler can deal with.

3.3. Sliding-tile technique

The GPU algorithm can be further improved by considering sliding tiles. Until now we have considered a biunivocal relation between GPU threads and similarity comparisons. However, it may be argued that a thread can calculate more than a similarity coefficients (i.e. a thread can calculate an entire row of coefficients). This consideration leads to sliding tiles which iteratively move along a row for calculating all the coefficients in the matrix. In other words, each thread will compute coefficient with offset $[0, b, 2b, 3b, ..., n - b]$ from its initial position (in the original approach we had instead n/b distinct threads corresponding to different blocks). Sliding tiles can reduce the number of accesses to global memory by reusing row molecules in shared memory. As depicted by Figure 5, rows are loaded once whereas column molecules are updated according to the tile position.

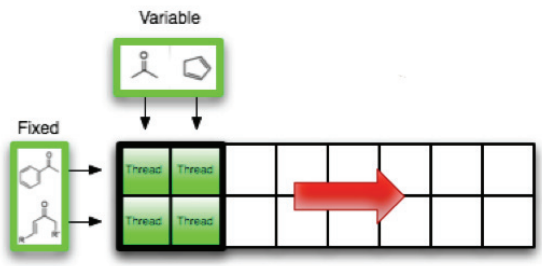

Figure 5: Row molecules are fixed whereas column molecules depend by the tile position.

The presented technique almost halves the number of accesses to global memory. Considering (n/b) blocks on a rows, the *tiling* technique requires $2b \cdot (n/b) = 2n$ accesses. Considering sliding tiles, we need b accesses for loading row molecules and $b \cdot (n/b) = n$ accesses for loading column molecules. The reduction factor is then $2n/(n + b) \approx 2$ assuming $b \ll n$. It may be also noted that sliding tiles require the same amount of shared memory needed by the *tiling* technique since column molecules are always loaded in the same memory location.

3.4. Wide sliding-tile technique

We mentioned that choosing a large b positively affects the memory performance despite its applicability is constrained by two architectural constraints. However, it is possible to partially overcome the problem related with the maximum number of threads contained in a tile. The shortcut still relies on the fact that a thread can calculate more than a single similarity coefficient. Consequently, a single block can covers a tile k-times wider (row-wise direction). In more detail, each thread will be responsible of calculating coefficients located at row-offset $[0, b, 2b, ..., (k - 1)b]$ depicting a wider tile. The memory access optimization will be obtained by increasing the constant memory dedicated to row molecules of a factor b and by keeping the iterative sliding mechanism for covering the entire calculation matrix. In the example shown by Figure 6, a single thread block covers a double tile since each thread computes two coefficients at different row offsets $[0, b]$. Again, we can quantify the reduction regarding the number of accesses to global memory. Considering k as width factor of the new tile, the previous sliding tile technique requires $k(n + b)$ accesses. Considering instead wider tiles, we need $k \cdot b$ accesses for loading row molecules and still n accesses for loading column molecules. The reduction factor is then $k(n + b)/(n + kb) \approx k$ assuming $b \ll n$ as well as $kb \ll n$. It may be argued that the presented technique increases the amount of shared memory required for a block while keeps unchanged the number of threads within a block. Consequently, the tile size will be dependent by only the former architectural constraint. The optimization principles introduced in [21] also focus on loop unrolling in order to reduce the impact related with loop overhead (i.e. condition testing or index variable incrementing). However, this optimization may reduce the number of active thread blocks when too many additional thread registers are required. Moreover, loop unrolling is not flexible to diverse descriptor/fingerprint length since it is an hard-coded technique. For these reasons, we prefer to not consider loop unrolling in our optimized GPU implementation.

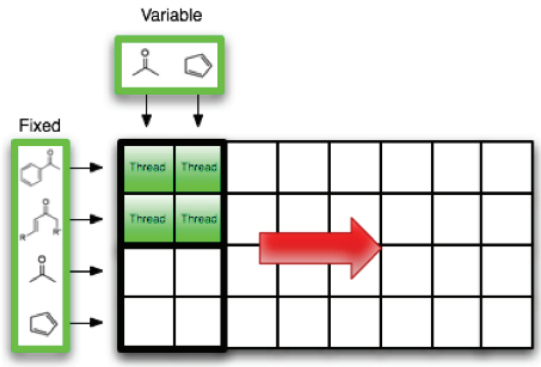

Figure 6: A wider sliding tile corresponding to the same thread block.

4. Results

We develop a CUDA implementation of our general GPU algorithm for each chemical similarity coefficient presented in Table 1. The main goal is to assess the advantages related with many-core architectures and the adopted optimization principles. The baseline performance is measured in terms of execution time necessary for a single-core CPU to compute an all-to-all chemical similarity matrix. The GPU speed-up is then related to this time measurement. We should recall that chemical similarity computation is a common and relatively simple operation used within various chemoinformatics algorithms. Therefore, it does not exist a stand-alone optimized code to compare with. In any case, we compile our experimental implementation always activating the available optimization flags (i.e. -O4 in gcc compiler) in order to assess performance against a reasonably optimized code. Within this section, we also provide a multithreaded execution of the algorithm on a dual-core CPU in order to present a fair comparison between the different computing architectures.

4.1. CUDA parameters and molecule dataset

According to the CUDA programming model [22], the number of active thread blocks is limited by the maximum number of thread in a SM. Consequently, we fixed $b = 16$ in order to have 256 threads for block and a good amount of memory sharing within the block. Depending by the computing capability actually implemented in the GPU device, the number of active block ranges from 3 to 6. We also fixed $k = 4$ in order to provide sufficiently large tiles without exceeding the available shared memory for SM.

We test our GPU algorithm on a set of randomly generated molecule fingerprints and a set of randomly generated molecule descriptors. As mentioned, our work is focused on measuring the performance speed-up so we conveniently generate random data. It may be argued that this simplification does not affect the obtained speed-up since algorithm runtime only depends by the number of molecules. The fingerprints are 256-bits long in order to propose a result comparison with Tanimoto calculations performed in [14]. The descriptor vectors are composed by 15 single-precision floating point values accordingly with the method described by [20]. We considered exponentially larger datasets from 1600 molecules to 51200 molecules. The corresponding chemical similarity matrix has quadratic size and may not always fit in the GPU memory. For this reason, when the matrix is too large we decompose it in 128MBytes chunks which are computed one after another by the GPU.

4.2. Results on an integrated laptop GPU

For our experimental result, we considered two different platforms corresponding to common computer systems easily available to scientists and researchers around the world. This choice is in agreement with the thought that GPU architectures represent the supercomputing for the masses [16]. The first computer system is a laptop equipped with an Intel Core 2Duo@2.53Ghz , with an integrated NVIDIA GeForce 9400M@450MHz (16 CUDA cores and 256MBytes of DDR3 memory shared with CPU) and with MacOSX as operating system. We compiled our CUDA kernel using NVIDIA nvcc compiler based on GNU gcc compiler. Table 2 shows the results regarding chemical

similarity coefficients calculated for both binary fingerprints (upper table) and floating point descriptors (lower table). Each row corresponds to an increasingly large dataset. For each possible coefficient, we report the performance speed-ups from the baseline single-core CPU execution, respectively for the basic GPU algorithm and for the wide sliding-tile GPU algorithm. Looking at the results, it is easy to note that we obtain a 4-to-6-fold speed-up. This result can be consider satisfactory considering the underlying computer system where a fairly powerful CPU is coupled with a low-end integrated GPU. We can also note how the introduced algorithmic optimizations assure an additional 2-to-3-fold speed-up compared with the basic GPU algorithm. The best average performance speed-up is obtained with Tanimoto coefficient since it has the simplest formulation. Recalling again Table 1, it may be argued that Tanimoto coefficient contains an unique floating point division whereas the other coefficients also contain multiplications by two or roots.

	Tanimoto		Dice		Cosine		Euclidean		Hamming	
# Molecule Fingerprints	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
1600	2.30x	5.75x	2.26x	4.25x	2.29x	4.31x	2.29x	4.31x	2.19x	4.31x
3200	2.30x	6.01x	2.25x	4.43x	2.27x	4.48x	2.28x	4.49x	2.17x	4.48x
6400	2.40x	5.89x	2.29x	4.46x	2.26x	4.41x	2.28x	4.47x	2.15x	4.33x
12800	2.79x	6.09x	2.28x	4.51x	2.28x	4.54x	2.28x	4.51x	2.21x	4.44x
25600	3.01x	6.34x	2.39x	4.74x	2.40x	4.78x	2.41x	4.79x	2.98x	6.32x
51200	2.94x	5.87x	2.93x	5.84x	2.98x	5.98x	2.97x	5.91x	2.95x	5.90x
Average	2.62x	5.99x	2.40x	4.71x	2.41x	4.75x	2.42x	4.75x	2.44x	4.96x
	Tanimoto		Dice		Cosine		Euclidean		Manhattan	
# Molecule Descriptors	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
1600	2.91x	4.87x	2.81x	4.68x	3.09x	4.97x	2.77x	4.43x	2.13x	3.69x
3200	2.90x	5.10x	2.87x	5.09x	3.17x	5.47x	2.63x	4.61x	2.11x	3.87x
6400	2.86x	4.83x	2.83x	5.89x	3.07x	6.26x	2.99x	4.20x	2.44x	3.59x
12800	3.08x	5.61x	3.06x	7.10x	3.23x	7.43x	3.19x	4.73x	2.62x	4.01x
25600	3.81x	7.35x	3.78x	9.27x	3.84x	9.34x	4.33x	6.50x	3.79x	5.83x
51200	4.01x	9.92x	4.00x	9.89x	3.96x	7.68x	4.48x	6.79x	3.90x	6.03x
Average	3.26x	6.28x	3.22x	6.99x	3.39x	6.86x	3.40x	5.21x	2.83x	4.50x

Table 2: GPU speed-ups over single-core CPU laptop execution.

The lower part of Table 2 refers instead to coefficients calculated on molecule descriptors. Looking at the results, we have again a satisfactory 4-to-7-fold speed-up influenced by a 2-fold speed-up associated with the optimizations only. Similarly to what done for fingerprint bit counting, we can precompute $\sum_{i=1}^{n}(x_{iA})^2$ and $\sum_{i=1}^{n}(x_{iB})^2$ for Tanimoto, Dice and Cosine coefficients. This optimization is visible in the obtained results (Euclidean and Manhattan coefficients have lower speed-ups). We can also observe a correlation between speed-up and molecule dataset size. In general, a GPU algorithm can reach an high FLOP/s rate when it is not memory bounded (in other words, it has many floating point operations and few memory accesses). This is not the case of fingerprint coefficients where few floating point operations are performed. Consequently, the speed-up seems to be bounded. We should also point out that the GPU scheduler takes advantage from an high workload, hiding thread memory latency with execution of other threads. This may reasonably explain the trend of better speed-ups with larger molecule workloads.

4.3. Results on a modern desktop GPU

The second computer system that we considered in our experiments is a desktop equipped with an AMD Athlon 64 X2@2GHz, with a recent NVIDIA GeForce GTX460@780Mhz (336 CUDA core and 1GByte of dedicated GDDR5 memory) and with Windows XP as operating system. The NVIDIA nvcc compiler is now based on Visual Studio compiler. Considering the hardware configuration unbalanced toward GPU computing, we expect to obtain a speed-up around two orders of magnitude. We can estimate an approximate 21-fold improvement over the laptop system since the number of CUDA core is increased from 16 to 336. The above part of Table 3 shows the results regarding chemical similarity coefficients calculated on binary fingerprints. As we can see, we obtain a 195-to-206-fold speed-up compared to the baseline single-threaded CPU time. Not surprisingly, this result is even better of the theoretical 126-fold improvement (obtainable as the average laptop GPU speed-up 6 multiplied by 21). Moreover, the introduced algorithmic optimizations approximatively account for a 2-fold improvement. Analogously to the laptop platform, the best speed-up is obtained with the simpler coefficient (Tanimoto). It might be argued that calculations are not memory bounded anymore since the speed-up is correlated with the molecule dataset size. In fact, the faster GDDR5 memory offers enough bandwidth in order to not represent a bottleneck.

# Molecule Fingerprints	Tanimoto		Dice		Cosine		Euclidean		Hamming	
	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
1600	108.45x	174.53x	102.01x	163.68x	111.82x	179.68x	109.13x	174.89x	96.61x	171.42x
3200	110.41x	180.11x	103.30x	168.44x	112.57x	183.98x	110.01x	177.99x	97.07x	176.61x
6400	121.59x	199.87x	117.43x	188.69x	116.87x	186.61x	117.32x	191.05x	112.67x	196.85x
12800	132.77x	219.63x	126.40x	208.94x	121.16x	189.24x	124.62x	204.11x	128.27x	217.08x
25600	138.15x	229.18x	132.21x	219.26x	141.32x	234.30x	139.22x	230.22x	136.06x	226.43x
51200	139.17x	236.54x	136.47x	225.00x	147.12x	243.40x	145.21x	239.32x	145.92x	240.18x
Average	125.19x	206.68x	119,64x	195,67x	125.14x	202.87x	124.25x	202.93x	119,43x	204.76x
# Molecule Descriptors	Tanimoto		Dice		Cosine		Euclidean		Manhattan	
	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
1600	106.11x	109.69x	104.45x	108.85x	111.92x	115.80x	71.10x	87.00x	240.64x	296.13x
3200	107.16x	111.32x	107.42x	111.48x	107.82x	119.02x	71.10x	87.60x	243.90x	306.76x
6400	111.44x	115.94x	112.36x	117.12x	115.68x	124.05x	86.73x	96.37x	282.43x	320.98x
12800	115.71x	120.56x	117.30x	122.76x	123.54x	129.08x	102.36x	105.13x	320.95x	335.20x
25600	117.43x	122.39x	117.58x	124.23x	125.05x	130.96x	102.93x	105.80x	327.32x	341.44x
51200	145.59x	152.29x	148.21x	155.68x	151.82x	159.08x	116.91x	120.12x	354.58x	368.86x
Average	117.54x	121.53x	117.89x	123.35x	122.64x	129.67x	91.86x	100.34x	294.97x	328.23x

Table 3: GPU speed-ups over single-core CPU desktop execution.

The lower part of Table 3 completes the experimental results with the coefficient calculated on molecule descriptors. The achieved performance is improved by a 100-to-328-fold speed-up. As we can see, the introduced algorithmic optimizations have a small role in this improvement. It is also easy to note how Manhattan coefficient has the best speed-up by far. This result is probably related with the simplicity of the Manhattan coefficient and with the optimizations natively implemented by the NVIDIA Fermi GPU architecture [8]. Tanimoto, Dice and Cosine coefficient calculations are still optimized by precomputing $\sum_{i=1}^{n}(x_{iA})^2$ and $\sum_{i=1}^{n}(x_{iB})^2$ leading to approximatively a 25% performance advantage respect to Euclidean coefficient. Moreover, the computation is not memory bounded since the speed-up is still correlated with the number of molecule descriptors considered.

4.4. Comparison with dual-core CPU architecture

We implemented a multi-core version of our code using Pthreads libraries [23]. Since the proposed workload is a case of embarrassingly parallelism, it is straightforward to distribute the computation among the threads. In more detail, we use four threads in order to keep busy the dual-core CPU architectures available. The obtained results are reported in Table 4. As expected, the dual-core implementation linearly increases the performance by a factor of about 2x. When there is possibility of precomputing (i.e. Tanimoto coefficient), this increase is even bigger. However, we can observe that GPUs still have a considerable advantage, especially taking account of performance relative to cost and power consumption. We can also argue that a similar performance scalability can be observed using multiple GPUs.

Tanimoto on 51200 Fingerprints				
GPU model	Precomp. (speed-up)	Dual Core (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
9400M	1.75x	4.15x	2.94x	5.87x
GTX460	1.44x	3.02x	139.17x	236.54x
Manhattan on 51200 Descriptors				
GPU model	Precomp. (speed-up)	Dual Core (speed-up)	Base GPU (speed-up)	Opt. GPU (speed-up)
9400M	-	1.76x	3.90x	6.03x
GTX460	-	1.82	354.58x	368.86x

Table 4: Dual-core CPU speed-ups versus GPU speed-ups.

4.5. Comparison with previous work

Last but not least, we try to make a comparison between our work and the all-to-all fingerprint Tanimoto calculation proposed by [14]. Details about our Tanimoto coefficient calculation are reported in Table 5. We have adopted the 256-bits long fingerprint format as well as the lookup table approach for counting bits proposed in [14]. Despite these similarities, we obtain discrepant results even in the baseline single-core CPU time. In more detail, in [14] it has been claimed that 2048 fingerprints are processed by a single thread in 49.76 seconds whereas our single-core implementation can process a slightly larger amount in just a fraction of the time (3200 fingerprints in 8.73 seconds). In this scenario, the only meaningful

Tanimoto			
# Molecule Fingerprints	Base. CPU (ms)	Optimized GPU	
		(ms)	(speed-up)
1600	2173	12	174.53x
3200	8730	48	180.11x
6400	40745	203	199.87x
12800	168137	765	219.63x
25600	701303	3060	229.18x
51200	2907592	12292	236.54x

Table 5: Tanimoto coefficient performance for comparison with the work proposed by [14].

comparison can be assessed from relative GPU speed-ups. Assuming 8192 fingerprints, the single-core and GPU implementations proposed by [14] have runtimes of $832.21s$ and $2385.14s$. This result corresponds to a slow down factor of $832.21s/2385.14s = 0.34x$. For same task, we have clearly shown how our optimized GPU algorithm achieves an average performance speed-up of 206.68x, three orders of magnitude better than [14].

5. Conclusions

In this paper, we have shown how chemical similarity assessment in large scale database can take advantage of GPU computing. We provide details about some methods for efficiently representing molecules and about similarity coefficients commonly used in chemoinformatics. We presented a general GPU algorithm for all-to-all chemical comparisons between molecule fingerprints and between molecule descriptors, introducing the necessary optimizations in order to minimize accesses to global memory. Previous works found in literature [12, 13] combine chemoinformatics with GPU computing but none of them contains a comprehensive covering of the chemical similarity topic. We tested the proposed algorithm on different experimental setups, obtaining up to a 206-fold speed-up for fingerprints and up to a 328-fold speed-up for descriptors. We also compared our Tanimoto coefficient calculation with [14], outperforming the previous result by three order of magnitute. Finally, our aim was to provide a wide-breath *proof-of-concept* regarding the usefulness of GPU architectures in chemoinformatics, a class of computing problems still uncovered. Consequently, we plan to investigate about other chemoinformatics techniques that may take advantage of GPU computing. **Acknowledgments**: this work was supported by National Institute of Health grants GM086145 and GM079804.

[1] A. R. Leach, V. J. Gillet, An Introduction to Cheminformatics, Springer, 2005.
[2] P. Willet, J. M. Barnard, G. M. Downs, Chemical similarity searching, Chemical Information and Modeling 38 (6) (1998) 983–996.
[3] D. Wilton, P. Willet, K. Lawson, G. Mullier, Comparison of ranking methods for virtual screening in lead-discovery programs, Chemical Information and Modeling 2003 (43) (2003) 2.
[4] R. N. Jorissen, M. K. Gilson, Virtual screening of molecular databases using a support vector machine, Chemical Information and Modeling 45 (3) (2005) 549–561.
[5] D. Butina, Unsupervised data base clustering based on daylight's fingerprint and tanimoto similarity: A fast and automated way to cluster small and large data sets, Chemical Information and Modeling 39 (4) (1999) 747–750.
[6] Pubchem - http://pubchem.ncbi.nlm.nih.gov/.
[7] L. C. Blum, J.-L. Reymond, 970 million druglike small molecules for virtual screening in the chemical universe database gdb-13, American Chemical Society 131 (25) (2009) 8732–8733.
[8] Nvidia, Nvidia's next generation cuda compute architecture: Fermi, Whitepaper.
[9] S. A. Manavsky, G. Valle, Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment, BCM Bioinformatics 9 (2) (2008) 1–9.
[10] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabucco, K. Schulten, Accelerating molecular modeling applications with graphics processors, Computational Chemistry 28 (16) (2007) 2618–2640.
[11] D. Xu, M. J. Williamson, R. C. Walker, Advancements in molecular dynamics simulations of biomolecules on graphical processing units, Annual Reports in Computation Chemistry 6 (2010) 2–19.
[12] Q. Liao, J. Wang, Y. Webster, I. A. Watson, Gpu accelerated support vector machines for mining high-throughput screening data, Chemical Information and Modeling 49 (12) (2009) 2718–2725.
[13] I. S. Haque, V. S. Pande, Siml: A fast simd algorithm for calculating lingo chemical similarities on gpus and cpus, Chemical Information and Modeling 50 (4) (2010) 560–564.
[14] V. Sachdeva, D. M. Freimuth, C. Mueller, Evaluating the jaccard-tanimoto index on multi-core architectures, International Conference on Computational Science (2009) 944–953.
[15] J. A. Grant, M. A. Gallardo, B. Pickup, A fast method of molecular shape comparison: A simple application of a gaussian description of molecular shape, Computational Chemistry 17 (14) (1996) 1653–1666.
[16] D. B. Kirk, W. W. Hwu, Programming Massively Parallel Processors, Morgan Kaufmann, 2010.
[17] J. R. Ullmann, An algorithm for subgraph isomorphism, ACM 23 (1) (1976) 31–42.
[18] N. Wale, G. Karypis, Acyclic subgraph based descriptor spaces for chemical compound retrieval and classification, IEEE International Conference of Data Mining 23 (1) (2006) 679–689.
[19] P. J. Ballester, W. G. Richards, Ultrafast shape recognition to search compound databases for similar molecular shapes, Computational Chemistry 28 (10) (2007) 1711–1723.
[20] M. S. Armstrong, G. M. Morris, P. W. Finn, R. Sharma, L. Moretti, R. I. Cooper, W. G. Richards, Electroshape: Fast molecular similarity calculations incorporating shape, chirality and electrostatics, Computer-aided Molecular Design 24 (9) (2010) 789–801.
[21] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, W. W. Hwu, Optimization principles and application performance evaluation of a multithreaded gpu using cuda, Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (2008) 74–82.
[22] Nvidia, Cuda c programming guide 3.2 (november 2010) - http://developer.nvidia.com/object/cuda_3_2_downloads.html.
[23] Posix threads programming, https://computing.llnl.gov/tutorials/pthreads/.