Published in final edited form as:
J Chem Inf Model. 2010 April 26; 50(4): 560–564. doi:10.1021/ci100011z.

SIML: A Fast SIMD Algorithm for Calculating LINGO Chemical Similarities on GPUs and CPUs

Imran S. Haque[†], **Vijay S. Pande**[†,‡], and **W. Patrick Walters**[¶]
Department of Computer Science, Stanford University, Stanford, CA, Department of Chemistry, Stanford University, Stanford, CA, and Vertex Pharmaceuticals Inc., Cambridge, MA

Vijay S. Pande: pande@stanford.edu

Abstract

LINGOs are a holographic measure of chemical similarity based on text comparison of SMILES strings. We present a new algorithm for calculating LINGO similarities amenable to parallelization on SIMD architectures (such as GPUs and vector units of modern CPUs). We show that it is nearly 3 times as fast as existing algorithms on a CPU, and over 80 times faster than existing methods when run on a GPU.

Introduction

The continuing exponential increase in computer power has made searches in chemical databases of thousands to millions of compounds routine. A variety of techniques exist for similarity search,[1] ranging in computational complexity from 3-D superposition methods[2–4] to simple substructure-fingerprint searches.[5,6] The Lingo method of Vidal, Thormann, and Pons[7] is a particularly simple algorithm that measures chemical similarity by computing the similarity between the SMILES representations of two given molecules. Despite its simplicity, LINGO has demonstrated accuracy comparable to path-based substructural fingerprint methods;[8] its compelling advantages are speed and the ready availability of appropriate SMILES representations for molecules.

While LINGO is one of the fastest similarity techniques in general usage, emerging problems in cheminformatics necessitate dramatically faster methods for calculating similarities. Freely available databases such as ZINC[9] (34 million molecules) and PubChem (31 million molecules) are currently being used for a variety of cheminformatic analyses. Exhaustive calculations on multimillion molecule databases such as these present a challenge to many commonly-used algorithms. Even more difficult are exhaustive databases such as GDB-13,[10] which enumerates all 970 million molecules up to 13 heavy atoms containing C, N, O, S, and Cl, according to a set of simple rules encoding chemical stability and feasibility. Virtual combinatorial libraries can also present a challenge to conventional algorithms, as even a 3 or 4 component combinatorial library can easily exceed a billion molecules.

Scalability problems arise both from database size and the algorithms considered. The largest current enumerated databases are nearly 1 billion (10^9) molecules in size;[10] future libraries are likely to be larger. While search for a single query molecule can be done in linear time, useful

Correspondence to: Vijay S. Pande, pande@stanford.edu.
[†]Stanford Computer Science
[‡]Stanford Chemistry
[¶]Vertex Pharmaceuticals Inc.

algorithms such as clustering chemical databases often run in time quadratic in the size of the database or worse.[11] Consequently, as the size of both public and corporate compound collections increases, faster similarity methods are required to handle the increasing computational load.

The GPU revolution in computing offers a way to cope with these issues. Modern GPUs (graphics processing units) are flexibly-programmable processors which offer theoretical speedups over 30-fold relative to top-end conventional CPUs. GPUs have been applied, with great effect, to related problems in computational chemistry, including molecular dynamics[12,13] and 3D similarity search.[14] Although GPUs often require algorithms to be redesigned to achieve peak speedup, such changes also often benefit modern CPUs.

In this paper we present SIML ("Single-Instruction, Multiple-LINGO"), a new algorithm (related to sparse matrix-matrix multiply) to calculate the LINGO similarity metric between molecules, especially suited for efficient execution on GPUs and the vector units of current CPUs. We describe a non-vectorized CPU version of our algorithm that is nearly 3 times as fast as existing algorithms for calculating LINGOs and a GPU implementation that is over 80 times as fast as existing methods. Our code is available at https://simtk.org/home/siml. We begin by explaining the LINGO similarity metric, continue with a description of our algorithm, present performance benchmarks, and conclude with a discussion of extensions of our work to generalizations of the considered LINGO similarities.

Methods

Overview of LINGO

The LINGO algorithm[7] models a molecule as a collection of substrings of a canonical SMILES representation. Specifically, a SMILES string (after some simple transformations, such as resetting all ring closure digits to zero) is fragmented into all its contiguous substrings of length q, producing the set of "q-Lingos" for that molecule. The similarity between a pair of molecules A and B is then defined by the following equation:

$$T_{A,B} = \frac{1}{\ell} \sum_{i=1}^{\ell} \left(1 - \frac{|N_{A,i} - N_{B,i}|}{N_{A,i} + N_{B,i}} \right) \quad (1)$$

In this equation, ℓ represents the number of Lingos present in either A or B, and $N_{x,i}$ represents the number of Lingos of type i present in molecule x. In this paper, we will use the lowercase "Lingo" to refer to a substring of a SMILES string and the all-capitals "LINGO" to refer either to the method, or the Tanimoto similarity resulting from the method.

Grant et al. have presented an efficient algorithm to calculate this Tanimoto value given a pair of SMILES strings.[8] Their algorithm, given two SMILES strings of lengths m and n, first constructs in $\Theta(m)$ time a finite state machine representation of one SMILES string and all its valid Lingos. It then processes the second string through this FSM in $\Theta(n)$ time, yielding a total runtime linear in the sum of the length of the two strings.

The use of the LINGO algorithm requires choosing a particular value of q, the SMILES substring length. Values of q that are too small retain little structural information about molecules; values that are too large induce too many distinct Lingos, making it increasingly unlikely that a pair of molecules will have Lingos in common. Both Vidal et al.[7] and Grant et al.[8] demonstrated that setting $q = 4$ (i.e., considering SMILES substrings of length 4) had the best performance in a variety of cheminformatics applications.

Our Algorithm

The LINGO Tanimoto equation can be written in a simpler form using set notation. Specifically, we treat a molecule as a multiset of q-Lingos; a multiset is a generalization of a set that allows each element of a set to have multiplicity greater than 1. The union of two multisets contains all elements present in either set, with multiplicity equal to the maximum multiplicity of that element in either set. Similarly, the intersection of two multisets has only those elements present in both sets, with multiplicity equal to the minimum multiplicity in either set. In this model, the LINGO Tanimoto between a pair of molecules takes the following simple form:

$$T_{A,B} = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

Computer systems typically represent a character with 8 bits; thus, there is a one-to-one correspondence between q-Lingos and 8*q*-bit integers. The optimality of $q = 4$ is fortuitous, as 4-Lingos correspond to 32-bit integers, which map directly to the typical word length of modern CPUs and GPUs (but see the conclusion for discussion of efficient implementations where $q \neq 4$). This mapping means that current hardware can perform a comparison between two 4-Lingos in one operation, rather than the 4 that would be needed for individual characters. We thus represent each molecule as a multiset of 32-bit integers: a sorted vector of 4-Lingos (represented as integers), and a parallel vector containing the multiplicity of each Lingo.

This multiset intersection/union can also be interpreted as a modified vector Tanimoto operating on sparse vectors *A* and *B*. In this interpretation, *A* and *B* are extremely-high dimensional vectors, in which each coordinate specifies the number of Lingos present in the molecule of a given type; the dimensionality is determined by the number of possible Lingos. If each Lingo consists of four 8-bit characters, then the implicit dimensionality is 2^{32}; however, since most of these Lingos will not be present, the vector is highly sparse. Consider the vector Tanimoto equation:

$$T_{AB} = \frac{\langle A, B \rangle}{\langle A, A \rangle + \langle B, B \rangle - \langle A, B \rangle} \tag{3}$$

If addition is kept as normal and multiplication of two numbers is replaced by taking the minimum of the two it is trivial to prove that this modified vector Tanimoto calculates the exact same quantity as the multiset definition given above (indeed, an equivalent expression, motivated differently, is given by Grant et al.[8]). This interpretation (which uses the same representation in memory as the multiset interpretation) has the advantage that the set magnitudes $\langle A,A \rangle$ and $\langle B,B \rangle$ need not be calculated for every Tanimoto calculation; only the intersection size $\langle A,B \rangle$ must be calculated each time. Indeed, the set magnitudes are determined by the SMILES length ℓ and Lingo size q: $\langle x,x \rangle = \ell_x - q + 1$. This sparse vector algorithm therefore saves the computation involved in calculating the set union in the multiset algorithm.

Given a pair of molecules represented in the paired-array manner listed above, the LINGO Tanimoto between them can be calculated in linear time using an algorithm similar to merging sorted lists (Algorithm 1).

GPU implementation

The finite-state-machine algorithm of Grant et al.[8] is poorly adapted to execution on a GPU or other SIMD processor because it requires either a large amount of branching or a moderately-sized lookup table to implement FSM state transitions. While it is possible to write branch-

dense code for a GPU, such code typically performs poorly because different lanes in the vector unit must execute different code paths. The latter option, a large random-access lookup table, falls prey to memory constraints. The GPU has only a very small amount of low-latency "shared" memory with true random access on-chip (16KB on recent NVIDIA GPUs). If the lookup table is larger than the available shared memory, then it must be placed in off-chip "global" memory; however, good performance on GPUs requires that global memory be accessed with spatial locality among threads. This behavior is unlikely for a state-transition lookup table.

In contrast, our algorithm is well-suited for GPU implementation. In calculating a similarity matrix, we assign each row of the matrix to a "thread block", or virtual core, on the GPU. Each thread block executes many threads, which calculate elements of the row in parallel. Each thread block stores the multiset representation of its query molecule (the molecule against which all column elements are compared) in shared memory to minimize global memory accesses. Furthermore, we store database molecules (ones along the columns of the similarity matrix) in column-major layout (such that the Lingos/counts for each molecule lie in one column). This layout maximizes the spatial locality of memory accesses by consecutive threads.

In architectures that require loads by adjacent lanes of a vector unit to be adjacent in memory for high performance (e.g., Intel/AMD SSE, GPUs without texture caching), it is possible to make a small modification to our algorithm to guarantee this characteristic. We let *j*, the database molecule multiset index, be shared among all threads. We ensure that each thread has advanced its *i* pointer (query multiset index) as far as possible before *j* is incremented, so all threads always read from the same row in the transposed database multiset matrix. In architectures with relaxed locality requirements (e.g., GPUs with 2-D texture caching support), this row-synchronization can be removed, allowing higher performance.

Results

Benchmarking Methodology

Our benchmark problem is the calculation of an all-vs-all LINGO Tanimoto matrix for a set of compounds, which is a common first step for many algorithms in cheminformatics, such as clustering. It is also relevant to doing multiple similarity searches, as the calculation of each row of the matrix is equivalent to performing a database search against a different query molecule.

For this particular problem it would be possible to halve the calculation by only evaluating the upper or lower triangle of the Tanimoto matrix, since using the same molecule set for both query and database compounds induces a symmetric matrix. However, in the general case different molecule sets may be used for the query and database sets (along the rows and columns of the resulting matrix), resulting in an asymmetric matrix. We therefore benchmarked the computation of the full Tanimoto matrix, rather than only the upper or lower half, for greater generality. We benchmark the following Tanimoto calculation methods:

- OE: OpenEye implementation (OELingoSim) of the LINGO similarity method
- CPU: CPU implementation of our sparse vector algorithm using precalculated magnitudes and normal (row-major) database layout
- GPU: GPU implementation of our sparse vector algorithm using precalculated magnitudes, transposed (column-major) query layout, and 2-D texture fetching

All CPU performance testing was performed at Vertex on machine 1. GPU performance testing was performed on two different machines at Stanford to assess code scaling:

- Machine 1: CPU: AMD Phenom II X4 920 (4 cores @ 2.8GHz)
- Machine 2: CPU: Intel Core 2 Quad Q6600 (4 cores @ 2.4GHz); GPU: NVIDIA GeForce GTS 250 (128 SP @ 1.84 GHz)
- Machine 3: CPU: Intel Xeon E5420 (4 cores @ 2.5GHz); GPU: NVIDIA Tesla T10 (240 SP @ 1.44 GHz)

The performance of our CPU code and the OpenEye LINGO implementation were measured using a test code written in C++. We used OpenMP (a multi-platform standard API for shared memory parallel programming) to parallelize both CPU methods to obtain multi-core results. Our GPU implementation is written in Python, using the PyCUDA library to access the GPU. Tanimoto calculations for both GPU and CPU codes were implemented as requesting 240 rows of a Tanimoto matrix at a time to enable parallelization over rows. All timing results were averaged over 6 runs.

Machines 1 and 2 used gcc 4.3.2; machine 3 used gcc 4.1.2. CPU code was compiled with `gcc -O2 -fopenmp`. CUDA version 2.3 and NVIDIA driver version 190.29 were used on both machines 2 and 3.

All SMILES strings used for testing were obtained from the ZINC database's subset of molecules from Maybridge.[9] Relevant statistics for the chosen benchmark sets are shown in Table 1.

Performance

As a baseline performance benchmark, we evaluated the performance of the OpenEye algorithm and our CPU multiset/sparse-vector algorithm on the construction of a similarity matrix for a 4,096 molecule subset of the Maybridge database. The performance results on Machine 1 are shown in Table 2. To evaluate GPU performance, we measured the time to construct a similarity matrix on sets of 4,096, 8,192, and 32,768 molecules from Maybridge, on machines 2 and 3. Measured times include the time to transfer sparse-vector-representation molecules to the GPU as well as the time to transfer calculated Tanimotos back to the host. GPU results are presented in Table 3.

Analysis

Our multiset/sparse-vector methods for calculating LINGO Tanimotos show significant speedup relative to the existing best-in-class techniques for calculating LINGOs. The CPU implementation shows a 2.75× speedup relative to the OpenEye method. This algorithm incurs additional initialization overhead for each query string at the start of each row, whereas in our method the preprocessing (constructing sorted-array multiset representations of the SMILES strings) is performed at startup and not repeated. However, this startup cost is amortized over the large calculation, amounting to less than 1% of the execution time even for the parallel implementation of the CPU-based algorithm (last row, Table 2), and does not significantly affect the achieved speedups. It is notable as well that our CPU implementation is not vectorized to take advantage of the SSE capabilities on the tested Phenom II CPU. As explained above, our algorithm is amenable to vectorization, and it may be possible to obtain even larger speedups on the CPU.

Even larger speedups are achieved by moving the algorithm onto a GPU. On the 4K-size problem, machine 2's midrange GPU shows nearly 20× speedup relative to our CPU sparse-vector algorithm, and nearly 55× relative to the OE algorithm. The throughput for this midrange GPU falls off somewhat as the matrix size increases; this is likely due to the increasing average size (29.3 Lingos at 4K to 31.6 Lingos at 32K) increasing the work per Tanimoto calculation.

In contrast, the throughput on the high-end GPU (machine 3) continues to rise as the matrix size increases, indicating that even higher throughput may be possible on larger problems. At the largest size we tested, the high-end Tesla T10 GPU shows 30× speedup relative to our CPU method and nearly 83× speedup relative to the OE technique.

Because they take advantage of the inherent parallelism in large-scale similarity calculations by parallelizing over row computations, our methods also scale very well to larger hardware. Our CPU algorithm reaches 96.3% of linear speedup when parallelized over 4 cores with OpenMP. Our GPU code also shows excellent scaling; at the largest problem size, we observe 97.9% of the expected throughput scaling, based on the increased number of shaders and decreased clock speed on the Tesla T10 relative to the GeForce GTS 250.

While our algorithm is well suited to very-large-scale similarity calculations involving the calculation of multiple rows of a Tanimoto matrix, it is not a good choice for small calculations such as the scan of a single molecule against a SMILES database. Our algorithm has a setup cost linear in the total number of molecules considered (query+database), whereas the Grant et al. FSM-based algorithm only incurs cost for each query molecule. In the 1×N (one query versus database of size N) case, the FSM algorithm thus has O(1) initialization cost, whereas our sparse vector algorithm has O(N) initialization cost. For large (multiple-row) calculations this setup cost is amortized because the same sparse-vector representations of database molecules are reused at each row.

Conclusions

The growing size of chemical databases requires the development of faster methods of chemical similarity comparison in order to implement database search, clustering, and other chem-informatic algorithms. We have described a new algorithm for calculating the LINGO similarity measure which is nearly 3× faster than the current best-in-class method. This algorithm is well-suited to implementation on SIMD architectures like GPUs, and demonstrates over 80-fold speedup versus OpenEye's implementation of LINGO similarities when run on a high-end GPU. Furthermore, this algorithm scales well to large hardware, enabling high performance on new generations of CPU and GPU hardware. Implementations of our algorithm are available at https://simtk.org/home/siml.

Our current implementations are focused on the LINGO algorithm with $q = 4$ (that is, considering SMILES substrings of length 4) because they have been shown to be optimal for typical similarity searches.[7,8] However, in some cases, a different value of q may be desirable (for example, extremely small molecules, in which there are few distinct Lingos, or cases in which more specificity, and a larger Lingo size, is required). Our algorithm is trivially extensible to $q < 4$; such Lingos correspond one-to-one with integers smaller than 32 bits, so the remaining available bits can be replaced by zero and the algorithm remains as previously described. It is also possible to extend the method to larger q. One possibility is to extend the integer size — by using 64-bit integers, up to $q = 8$ can be handled using the same method. Recent CPUs and upcoming GPUs natively support 64-bit integers, potentially imposing little overhead (except doubled memory traf-fic) relative to the use of 32-bit integers. Another option is to compress the Lingos. SMILES does not have a full 256-character alphabet, and therefore does not require a full 8 bits per character. By preprocessing strings the width of each character in a Lingo can be reduced. For example, if only 64 characters are used in a set of molecules, each character will only require 6 bits, and up to $q = 5$ can be supported still using 32-bit integers. Our algorithm is insensitive to these issues, as it operates on integers of arbitrary width. While certain integer sizes may be faster on given hardware, methods such as Lingo compression can be handled in preprocessing, and do not affect the core multiset algorithm.

By providing nearly 2 orders of magnitude in speedup, our algorithm enables dramatically larger calculations than previously possible. The use of GPU acceleration in particular allows searches that previously required use of a cluster of hundreds of machines to be performed on a user's desktop; alternatively, lengthy searches can be performed in near-realtime. We anticipate that these search capabilities will enable a new class of large-scale cheminformatics applications for data fusion on very large databases.

Acknowledgments

The authors would like to thank Kim Branson for assistance with running CPU benchmarking and Anthony Nicholls and Roger Sayle for helpful comments on the manuscript. ISH and VSP acknowledge support from Simbios (NIH Roadmap GM072970).

References

1. Nikolova N, Jaworska J. Approaches to Measure Chemical Similarity - a Review. QSAR & Combin Sci 2003;22:1006–1026.
2. Grant JA, Gallardo MA, Pickup BT. A fast method of molecular shape comparison: A simple application of a Gaussian description of molecular shape. J Comput Chem 1996;17:1653–1666.
3. Miller MD, Sheridan RP, Kearsley SK. SQ: A Program for Rapidly Producing Phar-macophorically Relevent Molecular Superpositions. J Med Chem 1999;42:1505–1514. [PubMed: 10229621]
4. Rush TS, Grant JA, Mosyak L, Nicholls A. A shape-based 3-D scaffold hopping method and its application to a bacterial protein-protein interaction. J Med Chem 2005;48:1489–1495. [PubMed: 15743191]
5. Carhart RE, Smith DH, Venkataraghavan R. Atom pairs as molecular features in structure-activity studies: definition and applications. J Chem Inf Comput Sci 1985;25:64–73.
6. Durant JL, Leland BA, Henry DR, Nourse JG. Reoptimization of MDL Keys for Use in Drug Discovery. J Chem Inf Comput Sci 2002;42:1273–1280. [PubMed: 12444722]
7. Vidal D, Thormann M, Pons M. LINGO, an Efficient Holographic Text Based Method To Calculate Biophysical Properties and Intermolecular Similarities. J Chem Inf Model 2005;45:386–393. [PubMed: 15807504]
8. Grant JA, Haigh JA, Pickup BT, Nicholls A, Sayle RA. Lingos, Finite State Machines, and Fast Similarity Searching. J Chem Inf Model 2006;46:1912–1918. [PubMed: 16995721]
9. Irwin JJ, Shoichet BK. ZINC — A Free Database of Commercially Available Compounds for Virtual Screening. J Chem Inf Model 2005;45:177–182. [PubMed: 15667143]
10. Blum LC, Reymond JL. 970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13. J Am Chem Soc 2009;131:8732–8733. [PubMed: 19505099]
11. 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;39:747–750.
12. Friedrichs MS, Eastman P, Vaidyanathan V, Houston M, Legrand S, Beberg AL, Ensign DL, Bruns CM, Pande VS. Accelerating molecular dynamic simulation on graphics processing units. J Comput Chem 2009;30:864–872. [PubMed: 19191337]
13. Stone JE, Phillips JC, Freddolino PL, Hardy DJ, Trabuco LG, Schulten K. Accelerating molecular modeling applications with graphics processors. J Comput Chem 2007;28:2618–2640. [PubMed: 17894371]
14. Haque IS, Pande VS. PAPER - Accelerating parallel evaluations of ROCS. J Comput Chem 2009;31:117–132. [PubMed: 19421991]

```
Indices into lists a/a_c and b/b_c
i = j = 0;
Accumulator for magnitude of intersection between a and b
isct = 0;
while i < ℓ_a and j < ℓ_b do
    if a[i] == b[j] then
        isct += min(a_c[i],b_c[j]) ;
        i += 1;
        j += 1;
    else if a[i] < b[j] then
        i += 1;
    else
        j += 1;
    end
end
return (isct / (m_a + m_b − isct))
```

Algorithm 1.
Sparse-vector algorithm to calculate LINGO Tanimoto between two molecules in sorted multiset representation

Table 1

Characteristics of SMILES sets from Maybridge used for benchmarking

# molecules	Average SMILES length (characters)	Average # distinct Lingos
4,096	35.17	29.31
8,192	35.41	29.52
32,768	38.23	31.65

Table 2

Similarity matrix construction performance on CPU, 4,096 molecules, Machine 1 (parallel = 4 cores).

Algorithm	Time (ms)	Throughput (kLINGOS/sec)
OE	15060	1113
CPU	5460	3070
OE parallel	3880	4320
CPU parallel	1420	11830
Read SMILES + construct multisets	10.5	—

Table 3

Similarity matrix construction performance on GPU. Times include transfer of molecules to GPU and transfer of Tanimotos back to host. Note that total work performed is quadratic in the number of molecules.

Machine	Number of molecules	Time (ms)	Throughput (kLINGOS/sec)
2	4096	275	60900
2	8192	1026	65410
2	32768	16717	64230
3	4096	215	77900
3	8192	778	86300
3	32768	11637	92270