GPU Acceleration of a Production Molecular Docking Code[*†]

Bharat Sukhwani Martin C. Herbordt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215

Abstract: Modeling the interactions of biological molecules, or *docking*, is critical to both understanding basic life processes and to designing new drugs. Here we describe the GPU-based acceleration of a recently developed, complex, production docking code. We show how the various functions can be mapped to the GPU and present numerous optimizations. We find which parts of the problem domain are best suited to the different correlation methods. The GPU-accelerated system achieves a speedup of at least 16x for all likely problems sizes. This makes it competitive with FPGA-based systems for small molecule docking, and superior for protein-protein docking.

1 Introduction

A fundamental operation in biochemistry is the interaction of molecules through non-covalent bonding (see Figure 1). Modeling molecular *docking*, as this process is called, is critical both to evaluating the effectiveness of pharmaceuticals, and to developing an understanding of life itself. Docking applications are computationally demanding. In drug design, millions of candidate molecules may need to be evaluated for each molecule of medical importance. As each evaluation can take many CPU-hours, huge processing capability must be applied; typically, production settings rely on large clusters.

While accelerating docking using heterogeneous parallel processors has clear and obvious benefits, there has been surprisingly little work thus far. SymBioSys uses the Cell Broadband Engine in their eHITS software [13], and Servat, et al. report using the same processor to accelerate the FT-

[*]This work was supported in part by the NIH through award #R01-RR023168-01A1. Web: http://www.bu.edu/caadlab.
[†]EMail: {herbordt|bharats}@bu.edu

Dock code [16]. The present authors have previously used FPGAs to accelerate the PIPER code [17]; earlier they and collaborators demonstrated proof-of-concept FPGA-based acceleration of ZDOCK and some other systems [19]. With GPUs, the only published work so far appears to be in a dissertation by Korb [10]. There, one phase (evaluation) is accelerated, but there is no report of overall speedup.

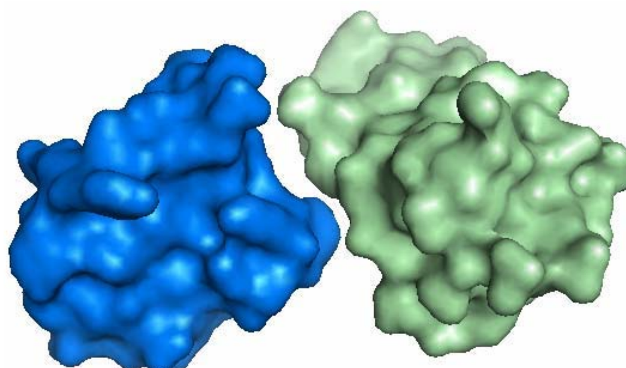

Figure 1: Docked complex of two proteins generated using Pymol [15].

The basic computational task for docking is to find the relative offset and rotation (pose) between a pair of molecules that gives the strongest interaction (see Figure 2). Hierarchical methods are often used: (i) an initial phase where candidate poses are determined (docking), and (ii) an evaluation phase where the quality of the highest scoring candidates is rigorously evaluated. This work describes the GPU-based acceleration of PIPER, a state-of-the art code that performs the first of these tasks. PIPER minimizes the number of candidates needing detailed scoring with only modest added complexity [11].

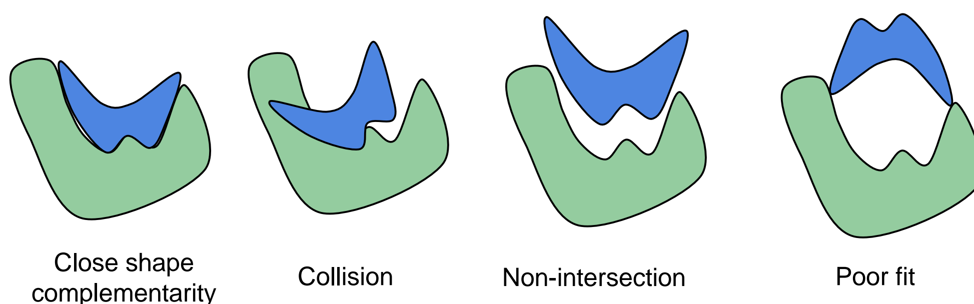

Figure 2: Shape complementarity, collisions, misses, and poor matches (after [18]).

Many docking applications including PIPER assume, at least initially, a rigid structure (see Figure 2). This still allows modeling of various force laws that govern the interaction between molecules, including geometric, electrostatic, atomic contact potential, and others. A standard technique maps the molecules' characteristics to three dimensional grids. The most energetically-favorable relative position is determined by summing the voxel-voxel interaction values for each modeled force at all positions to generate a score, and then repeating this for all possible translations and rotations.

The resulting computational complexity is large. With typical grid sizes of $N = 128$ in each dimension and the total number of angles $10,000$, 10^{10} relative positions are evaluated for a single molecule pair. Typically, the outer loop consists of the rotations while the translations are handled with a 3D correlation. Since the latter require $O(N^6)$ operations, this type of exhaustive search was long thought to be computationally infeasible [12]. The introduction of the FFT to docking [9] reduced the complexity of each 3D correlation to $O(N^3 \log N)$ for steric (shape only) models; further work expanded the method to electrostatic [3] and solvation contributions [1].

Docking computations are generally used to model one of two types of interactions: between proteins (protein-protein docking) or between a protein or other large molecule and a small molecule (small molecule docking). In the latter case the large molecule is referred to as the substrate or receptor and the small molecule as the ligand. Protein-protein docking is important for basic science, while small molecule docking is the method primarily of interest in drug discovery. In both cases, one molecule has a grid size of up to 128^3; in protein-protein the second molecule is similar, but in small-molecule the ligand is typically an order of magnitude smaller (per dimension). This difference leads to there being a divergence in optimizations, with docking codes sometimes specializing in one domain or the other. We have found that this divergence emerges in accelerated docking as well.

In our previous work [18, 19] we showed that, for FPGA-based coprocessors, the original direct correlation–rather than an FFT–is sometimes the preferred method for computing rigid molecule docking. This is largely due to the efficiency with which FPGAs perform convolutions with the modest

precision (2-7 bits) of the original voxel data. Note that this precision goes to 48 or 106 bits (single or double precision complex floating point) for the FFT. In [17], we extended these methods to facilitate integration into PIPER, adding support for multiple and complex energy functions. The result was, for small-molecule docking, a multi-hundred-fold speed-up of PIPER's correlation computation and a 20-fold speed up of the entire application.

In that work we also found the limits of the correlation-based approach for current generation FPGAs. Since the FFT has the advantage over direct convolution in asymptotic complexity, the question is at what molecule sizes this advantage begins to dominate over other factors, such as precision. We found that the split occurs almost directly on the small/large molecule boundary: for ligands less than 25^3 direct correlation yields significant acceleration; for larger ligands the FFT, even on the host, is superior.

Our basic result here is a set of GPU-based solutions that work well for both protein-protein and small-molecule docking domains, with performance speed-ups of at least 16x being achieved across the entire range. We also find a divergence of methods for large and small molecule applications, although the excellent FFTs available on the GPU push the cross-over point down to a smaller ligand size.

The significance is as follows. We believe this to be the first published study of a production docking code accelerated with a GPU, and because of its cost effectiveness, we anticipate wide distribution. Further significance is the finding with respect to the cross-over point between 3D correlations and FFTs on GPUs. This could be of interest in the many other applications where these operations are fundamental. And finally, the comparison with the FPGA points to the best ways to build cost-effective rigid-molecule docking systems using the current generation of accelerator technology. Other contributions are the implementations of numerous other parts of the application, including results filtering and system integration. We have also conducted numerous experiments to optimize coordination of multiple passes for different force components, and data distributions among blocks and blocks among SMPs.

The rest of this paper is organized as follows. We next give a brief overview of PIPER. There follows the design, implementations, and details of how they were determined. We conclude with results and discussion.

2 The PIPER Docking Program

2.1 Overview

A primary consideration in docking is preventing the loss of near-native solutions (false negatives); as a result, rigid molecule codes tend to retain a large number (thousands) of docked conformations for further analysis even though only a few hundred will turn out to be true hits. "Improving these methods remains the key to the success of the entire procedure that starts with rigid body docking [11]." PIPER addresses this issue by augmenting commonly used scoring functions (shape, electrostatics) with a desolvation function computed from pairwise potentials. A fundamental innovation in PIPER is the finding that eigenvalue-eigenvector decomposition of the pairwise interaction matrix can substantially reduce this added complexity.

PIPER's energy-like scoring function is computed for every rotation of the ligand (smaller molecule) with respect to the receptor (larger molecule). It is defined on a grid and is expressed as the sum of P correlation functions (for each energy term) for all possible translations α, β, γ of the ligand relative to the receptor

$$E(\alpha,\beta,\gamma) = \sum_{p}\sum_{i,j,k} R_p(i,j,k)L_p(i+\alpha, j+\beta, k+\gamma) \quad (1)$$

where $R_p(i,j,k)$ and $L_p(i+\alpha, j+\beta, k+\gamma)$ are the components of the correlation function defined on the receptor and the ligand, respectively. For every rotation, PIPER computes the ligand energy function L_p on the grid and performs repeated FFT correlations to compute the scores for the different energy functions. For each pose, these energy functions are combined to obtain the overall energy for each pose. Finally, a filtering step returns some number of poses based on score and distribution.

2.2 PIPER Program Flow and Performance Profile

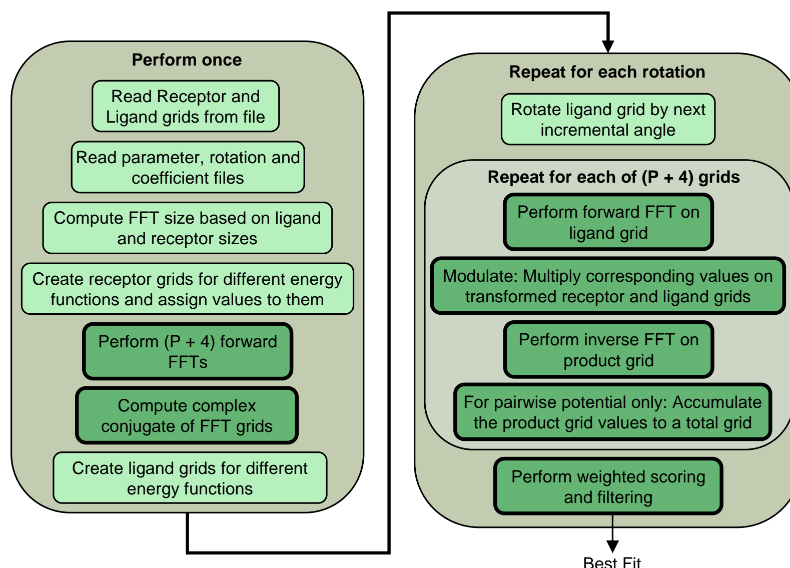

Figure 3: Program flow of Piper. Blocks in dark green with bold border indicate steps accelerated on the GPU.

Figure 3 shows the sequence of steps followed by the PIPER program to perform docking of a ligand to a receptor. The ligand and receptor atoms are read from input files, along with certain parameters and coefficients. These are used in scoring, filtering top scores, rotation, and charge assignment. Next, PIPER determines the size of the padded FFT grid (based on the sizes of the ligand and receptor) and generates the receptor and ligand grids for the various energy functions. Then the receptor grids for the energy functions are assigned values and their forward FFTs and complex conjugates are computed. The number of forward FFTs to be performed equals P + 4. The “4” are the following: attractive van der Waals, repulsive van der Waals, Born component of electrostatic energy, and Coulomb component of electrostatic. The “P” are the top P desolvation terms. In our accelerated version, we perform the forward FFTs and complex conjugates on the GPU.

For each rotation, PIPER multiplies the ligand with the next rotation vector and assigns new values to ligand grids for different energy functions. We leave these two steps to be performed on the host. After grid assignment, forward FFTs of each ligand grid are performed and the transformed

Table 1: PIPER run times for one rotation. Steps performed once are negligible over thousands of rations.

Phase	Run time (seconds)	% total
Ligand rotation	0.00	0%
Charge assignment	0.23	2.3%
FFT of ligand grids	4.51	45.4%
Modulation of grid-pairs	0.22	2.2%
IFFT of ligand grids	4.51	45.4%
Accumulation of desolvation terms	0.24	2.4%
Scoring and filtering	0.23	2.3%
Total	9.94	100%

grid is multiplied with the corresponding transformed receptor grid. The multiplied grid is then inverse transformed. For the case of the desolvation terms, the inverse transformed grids are accumulated to obtain the total score for desolvation energy. A weighted sum of the scores for the various energy functions is then computed and the top scores reported. In our GPU accelerated version, we perform all the per-rotation steps (except rotation and grid assignment) in GPU. This accelerates the bulk of the work performed by PIPER (see Table 1).

3 Mapping Piper to a GPU

3.1 Overview

From Table 1 we see that the essential tasks to be mapped to the GPU are the FFTs. We also observe, however, that while these comprise 91% of the runtime, Amdahl's law tells us that this limits speedup to a factor of 11. Therefore as many of the other (comparatively minor) tasks as possible must also be addressed. Of these, we find that modulation and accumulation are straightforward, but that filtering brings up some interesting issues (described in Section 3.4). Unfortunately, charge assignment is highly complex and is still done on the host.

With respect to the FFT task (actually the use of the FFT to perform the correlation task): As described in the introduction, both the FFT and direct correlation are useful in the overall solution;

they are described in the next two subsections. We find that one key issue is, as expected, the grid size. We also find, however, that the complexity of the energy function (i.e., the number of desolvation terms) has a significant effect on the choice of implementation.

3.2 Direct Correlation

To perform correlation on a GPU, the ligand and receptor grids need to be transferred to the device memory. Since every multiprocessor needs access to both these grids, they either need to be stored in device's global memory, accessible by all the multiprocessors, or duplicated in the local shared memory of each of the multiprocessor. Since receptor grids are large and the shared memory per multiprocessor is relatively small, it is not possible to copy these grids to the shared memories; rather, we store the receptor grids in the global memory. Since the ligand grids are much smaller, we tried to store them variously in the device's shared memory or constant cache, both of which provide much faster access compared with global memory. We found that access time from constant memory and shared memory is identical, unless there is a cache miss on the constant memory, in which case, the data is accessed from global memory.

Both the shared and the constant memory, however, are small and thus limit the size of the largest ligand that can fit in its entirety. An important consideration is the complexity of energy function used. With 4 pairwise potential terms, we can fit a ligand grid of size up to 7^3 in shared memory and 8^3 in constant memory. For larger ligand grids, we store the ligand in global memory, degrading the performance.

Work assigned to different thread-blocks can be distributed in many ways. We tried two schemes: In both, we launch the kernel with a 2D array of thread blocks, each with a 3D array of threads. In the first scheme, each thread-block is responsible for computing a part of the 2D result plane for all the 2D planes in the 3D result grid (see Figure 4(a)). In the second distribution, we assign different 2D planes to different thread-blocks. The threads on each of those thread blocks compute a larger part of the 2D plane, but only for the planes assigned to the current thread block (see Figure 4(b)).

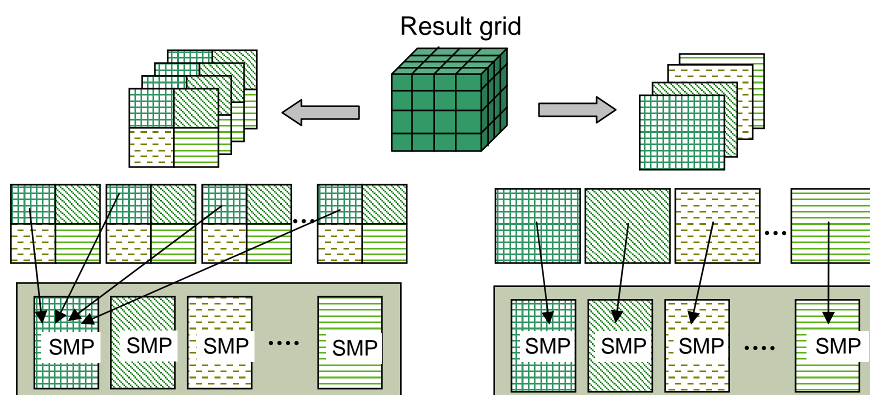

Figure 4: Distribution of work on the GPU for direct correlation: (a) Each thread block works on part of every 2D plane, (b) Different 2D planes are assigned in their entirety to different thread blocks.

Both distributions result in similar runtimes, though one or the other can have better performance for various non-cubic grids.

Based on our experiments, we noticed the following peculiarities of direct correlation on the GPU with respect to the PIPER energy functions:

– Only a limited number of pairwise potential terms can be supported without swapping or storing the ligand on global memory; more terms would result in larger memory requirements for storing the ligand grids. We could support ligand grid sizes of up to 8^3 when using 4 pairwise potential terms (8 correlations in total).

– Unlike FFT-based correlation, where the time to perform correlation depends on the size of the padded FFT grid (which is equal to the sum of the ligand and receptor sizes), the direct correlation time depends mainly on the size of the ligand. Thus, for the same FFT size, direct correlation can perform significantly better for smaller ligands whereas FFT correlation runtimes remain unchanged.

– For smaller ligand grids, we performed a further optimization: we store voxels for multiple rotations of the ligand in constant memory. This allows the correlation inner loop to compute multiple scores in each iteration. This yields two benefits - the loop overhead is amortized over multiple rotations. More importantly, each receptor voxel, fetched from the global memory (which is not cached) gets used

multiple times, thus reducing the overall fetch time by the number of rotations that can be computed at once. Since access to global memory has higher latency, reducing accesses to global memory results in significant performance improvement. For 4^3 ligand grids, we can perform 8 rotations in one iteration, achieving a speedup of 2.7x over direct correlation performed one rotation at a time.

– Direct correlation results in a performance improvement over FFT correlation only for ligand grids which can fit in the shared or constant memory. Accessing ligand from global memory results in significant performance loss.

– Direct correlation provides filtering of the top scores since different multiprocessors, computing different regions of the result map, can find and report their local best score. An extra step of finding the best of these local best scores, as well as flagging the cells for exclusion still needs to be performed by a master thread (see Section 3.4).

3.3 FFT

Performing correlation using FFTs involves computing a forward FFT of the two grids, modulation (multiplication) of the transformed grids, and an inverse FFT of the product grid. In our GPU-accelerated PIPER code, we perform these steps, along with accumulation and scoring, on the GPU. Performing operations on the GPU requires transferring the data to the device memory, which can sometime more than offset the benefit achieved from parallel execution. It is thus important to take transfer time into consideration.

For computing the forward and inverse FFTs, we use the NVIDIA CUFFT library. The forward FFT of the receptor grid is performed only once, since the receptor grid is held fixed. For each energy function, the receptor grid is copied to the device and the CUFFT library call is made. The complex conjugate of the transformed grid is then performed on the GPU (by dividing the grid between different thread groups and different threads). Note that the transformed grid is already present in the device memory, and that computing the complex conjugate on the device does not involve an extra data transfer.

The transformed receptor grid for each energy function is left in the device memory. This allows us to perform modulation on the GPU without recopying the grids. This simple optimization results in significant performance improvements since it avoids the need to transfer large amount of data from host to device for each rotation. This approach, however, is possible only if there is enough device memory to simultaneously store multiple grids of N^3 complex entries each. We find that our Tesla C1060 card easily fits all 22 grids (4 + P; P = 18 desolvation grids) for a large grid size (N = 128).

For each rotation, the host rotates the ligand and computes the new grid values for the various energy functions. The ligand grids are then copied to the GPU memory and the forward FFT is performed. The transformed grid is then multiplied with the corresponding transformed receptor grid. This multiplication is performed on the GPU, with different threads multiplying different parts of the grids. Since the transformed grids for both the receptor and the ligand are present on the device, performing multiplication on the device does not incur any data transfer. Finally, the product grid is inverse transformed to obtain the correlation scores. The above steps are for each of the different energy grids. In the case of the desolvation grids, the inverse FFT is followed by an extra step to accumulate the scores of the different desolvation energy terms to obtain a total desolvation energy score. This accumulation is also performed on the GPU, yielding similar performance benefits as performing modulation on the device.

In contrast with the direct correlation, FFT-based correlation performs individual correlations in serial order. This enables scalability to any number of correlations, thus allowing any number of pairwise potential terms in PIPER desolvation energy function. Addition of any new energy function is also straightforward. Further, unlike direct correlation on the GPU, where the size of the ligand grid is limited by the size of the constant memory, FFT correlation can support any ligand grid size, so long as the total FFT grid fits in global memory.

3.4 Scoring and Filtering

For each rotation, after the correlation scores for the various energy functions have been computed, two steps remain: computing the total score for each translation (goodness of fit) and selecting the translations with the best scores (filtering). Scoring simply involves computing the weighted average of the scores for the various energy functions. The PIPER filtering algorithm selects the top scoring conformations from different regions of the result map. One complication is that PIPER computes scores using multiple sets of weights: the scores returned are the best over all of the sets. The number of scores returned per rotation is a parameter with a default of one.

To perform scoring on the GPU, the scoring coefficients must first be made available to the GPU multiprocessors. Since the GPU constant memory is cached and provides faster access, we use it to store the coefficients. Copying the coefficients from host to constant memory is performed only once and the transfer time is very small compared to the total runtime.

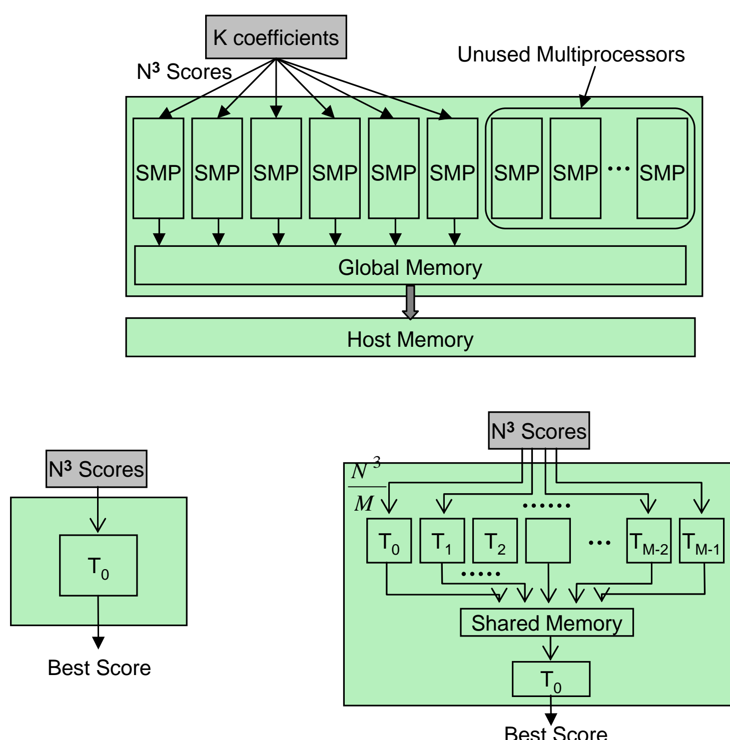

Figure 5: Scoring and filtering on the GPU: (a) Different coefficients are distributed among different multiprocessors, with each containing (b) one thread, or (c) multiple threads.

The work on the GPU is divided by distributing the K coefficient sets onto K different multiprocessors, each computing its respective best scores independent of the rest (see Figure 5(a)). In our initial approach, we assigned just a single thread to each thread block (running on each of the K multiprocessors). Thus, each thread computes N^3 weighted scores and stores the top score in the device global memory (see Figure 5(b)). Contrary to our expectations, this version resulted in significant slowdown compared to the serial code. This was a clear example of where the multiprocessors are highly underutilized and the communication cost outweighs the computation benefit.

In our second scheme, we assigned different coefficient sets to different thread blocks with each thread block now containing M threads (M = 500 for the results presented). The correlation grids are divided equally among all the threads, with each thread computing N^3/M weighted scores and finding the best scores within its subset (Figure 5(c)). For each thread block, we allocate an array in the device shared memory. The best score found by each thread and the corresponding score index are stored in this shared array. Once all the threads finish processing their subset of results, a master thread (thread 0) processes the shared array to find the best of these best scores and stores it in the array of top scores in the device global memory.

If the number of top scores to be reported per rotation (per coefficient) is greater than 1, then thread 0 performs an additional task of flagging the cells neighboring the current best score and so exclude them from consideration in the next iteration. This is done to avoid reporting multiple top scores from the same region. In the serial version, Piper maintains an array of integers, with one entry for each of the N^3 result cells. In our GPU version, maintaining such a large array on the GPU shared memory is not feasible. We first tried to maintain a small array on shared memory, containing the indices of the cells to be excluded in the next iteration. This requires each thread to traverse the entire array to check if the current cell is present and must be excluded. This leads to significant slowdown, with GPU filtering performing 2-3x slower than filtering on host. In our second approach, we tried storing the N^3 array on device global memory, with each thread reading only one array entry to determine exclusion. This improved the performance significantly, with GPU filtering now

performing better than CPU filtering. Note that the size of the array in global memory increases to KN^3 since all the coefficients are being processed in parallel. The performance hit due to accessing global memory is not significantly large and can be attributed to the memory coalescing support in current generation GPUs.

The performance improvements obtained by performing filtering on GPU is two-fold. First, and the more obvious, performance improvement comes from processing multiple scores in parallel on the GPU. This is achieved both at the thread-block level (different coefficients being processed on different threads) and the thread-level (different parts of the grids being processed by different threads). The performance reported in this paper is for 6 coefficient sets (which is the default in PIPER). Since each coefficient is assigned to one multiprocessor, we use only 6 of the 30 multiprocessors available on the Tesla C1060 GPU. Somewhat surprisingly, further distributing the work does not significantly improve performance. There are two reasons: first, in the above approach the filtering time is already a very small fraction of total runtime. Second, since the CUDA architecture does not support fine-grained synchronization and requires each thread-block to be able to execute independently and in any order, distributing filtering work across multiple thread blocks is not very straightforward. In future, we plan to implement this, though it is not a high priority.

The second performance improvement comes from the side effect of performing filtering on the device. Since grid modulation is performed on the GPU, performing filtering on host would require transferring 5 modulated grids (each with N^3 elements) to the host for every rotation; instead only the few top scores need to be transferred.

4 Results

The results presented here were generated using an NVIDIA Tesla C1060 GPU card and an Altera Stratix-III EP3SL340 FPGA. All systems, including the serial reference, ran on a 2008-era 2GHz quad-core Intel Xeon processor. Only a single core was used. Since the same PIPER tasks that lend themselves to acceleration are also parallelizable, speedup results should be scaled accordingly. For

the GPU FFT we use the CUFFT library from NVIDIA. For data type, the serial and GPU versions both use single precision floating point, while the FPGA uses the original fixed point. Docking output from the various implementations does not noticeably differ.

The major results are as follows:

- Overall speedup of the GPU-accelerated over the original serial implementation as a function of ligand size,
- Performance comparison of the FFT-based and direct correlations as a function of ligand size,
- Speedups for the various GPU-accelerated PIPER tasks, and
- A comparison with the FPGA-accelerated implementation.

Table 2: CPU times, GPU times, and speedups for various steps.

Phase	CPU time (ms)	GPU Time (ms)	Speedup
Forward FFT (once per rotation, per energy grid	205	9.3	22
Complex conjugate (once per rotation, per energy grid)	10	0.01	1000
Modulation (once per rotation, per energy grid)	10	0.01	1000
Inverse FFT (once per rotation, per energy grid	205	11.8	17
Accumulation of desolvation terms (once per rotation)	240	0.09	2667
Scoring and filtering (once per rotation)	230	39.5	6
Total runtime per rotation	9980	609	16.4

Table 2 compares the serial with the GPU-accelerated runtimes for the various tasks. Correlation on the GPU is performed using the FFT/IFFT pair. The grid size is 128^3; the number of pairwise potential terms is 18 (the default). Clearly, the complex conjugate, modulation, and accumulation tasks afford very high speedups due to their inherent parallelism. Speedup for the scoring-and-filtering task is modest due to the utilization issues already described, but does not significantly affect the total performance. The overall speedup achieved per rotation, including the time for the unaccelerated rotation and grid assignment tasks, is over 16x. Perhaps the most likely near-term source of further speedup is through continued improvements in GPU FFTs (see, e.g., [5, 14]).

In Figure 6 we focus on the performance of the correlation task. We plot the two GPU-accelerated methods, FFT-based and direct correlation, and the FPGA-accelerated direct correlation. The ligand

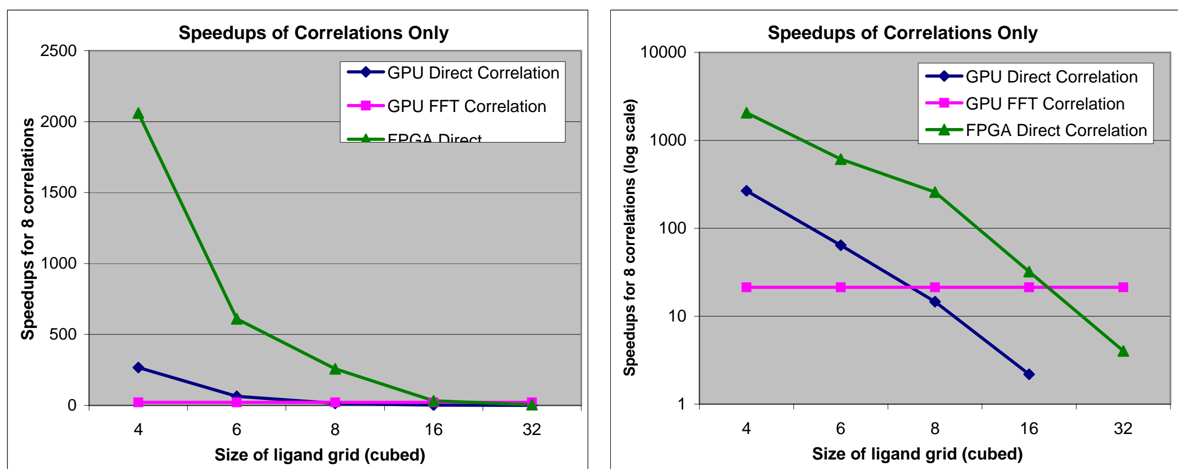

Figure 6: Shown is the speedup of the correlation task with respect to ligand size on GPU- and FPGA-accelerated systems.

grid size is varied from 4^3 to 32^3, but the total correlation size is fixed at 128^3.

As expected, the FFT maintains constant performance while the correlation-based methods drop sharply with ligand size. For GPUs alone, direct correlation is better for small ligands, while the FFT maintains good performance throughout. The crossover point is at a ligand size of about 8^3. For the FPGA, the crossover point with the GPU FFT is around 18^3.

For the GPU direct correlation, the drop in performance is due to two factors: the larger inner loop and the need to access the ligand grid from global memory. For the FPGA direct correlation, the drop in performance is mainly due to the inability to fit the entire ligand grid on chip and thus the need to swap different parts of the ligand grid in and out of the FPGA pipeline.

In Figure 7 we examine the total end-to-end speedups of both the GPU- and FPGA-accelerated codes with respect to the range of likely ligand sizes. For the GPU we use direct correlation for the smallest ligand and the FFT for the rest. In this graph we see two phenomena. The first is that the performance improvement on the small ligands is brought down to earth by the overhead, but is still substantial. The second is that the FPGA's advantage in small-molecule is diminished somewhat by greater overhead. It remains substantial, however, in the 8^3 - 12^3 range. For ligands much larger than

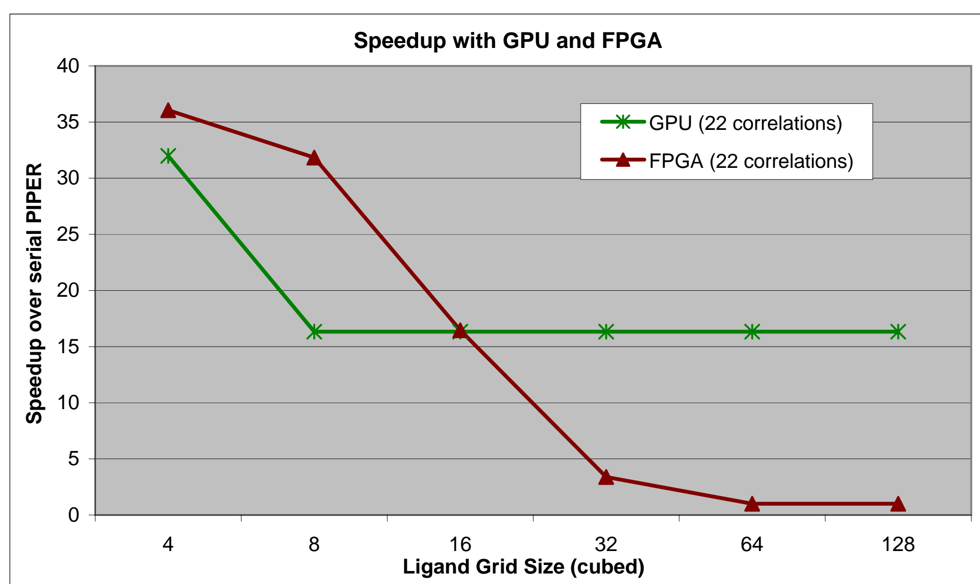

Figure 7: Shown is the overall end-to-end speedup of PIPER with respect to ligand size on GPU- and FPGA-accelerated systems.

16^3 the GPU dominates.

5 Discussion

In this work we describe a GPU-accelerated production docking code, PIPER, which achieves an end-to-end speed-up of at least 16x for all commonly used problem sizes. We find that for small ligands direct correlation is superior to the FFT; the crossover point lies at about 8^3. The FPGA remains superior to the GPU for low precision correlations, and so has better performance for ligand sizes less than about 16^3. The performance difference for the correlation alone (for these small ligands) is substantial; when the rest of the computation is included, however, the difference in speed-up is more modest (see Figure 7). For all ligands larger than 16^3, the GPU version continues to give excellent performance, while the FPGA stops being cost-effective at around 25^3.

Putting these results into the users' perspective, the GPU-accelerated version is clearly cost-effective—with respect to an unaccelerated workstation—for both protein-protein and small molecule domains. If the user is interested only in small molecule docking, then the FPGA-accelerated version

could be preferred. The performance difference, however, might not be great enough to justify the higher cost and the reduced flexibility of that technology. If, however, some of the overhead (in particular in charge assignment) can be reduced, then the FPGA may find a clearer niche. It is also possible that an efficient 3D FFT could be developed for FPGAs; this would shrink the performance gap for large molecule pairs.

The FFT is one of the most widely used tools in embedded and high performance computing and developing efficient versions for the GPU has received much attention (see, e.g., [5, 4, 14]). We gratefully take advantage of this fine work. The application of GPU correlation to various signal and image processing applications has also been widely studied (see, e.g., [2, 7, 8]). We believe this to be the first published study of GPU-acceleration of a complete docking code.

Acknowledgments. We thank members of the Structural Bioinformatics group at Boston University for their help in understanding the PIPER code.

References

[1] Chen, R., and Weng, Z. A novel shape complementarity scoring function for protein-protein docking. *Proteins: Structure, Function, and Genetics 51* (2003), 397–408.

[2] Fialka, O., and Cadik, M. FFT and convolution performance in image filtering on GPU. In *Proceedings of the Conference on Information Visualization* (2006).

[3] Gabb, H., Jackson, R., and Sternberg, M. Modelling protein docking using shape complementarity, electrostatics, and biochemical information. *Journal of Molecular Biology 272* (1997), 106–120.

[4] Govindaraju, N., Lloyd, B., Dotsenko, Y., Smith, B., and Manferdelli, J. High performance discrete Fourier transforms on graphics processors. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2003).

[5] Govindaraju, N., Lloyd, B., Dotsenko, Y., Smith, B., and Manferdelli, J. High performance discrete

Fourier transforms on graphics processors. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC08)* (2008).

[6] Gu, Y., and Herbordt, M. FPGA-based multigrid computations for molecular dynamics simulations. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines* (2007), pp. 117–126.

[7] Harris, G., Haines, K., and Smith, L. GPU accelerated radio astronomy signal convolution. *Radio Astronomy 22*, 1-2 (2008), 129–141.

[8] Huang, J., Ponce, S., Park, S., Cao, Y., and Quek, F. GPU-accelerated computation for robust motion tracking using CUDA framework. In *Proceedings of the IET International Conference on Visual Information Engineering* (2008).

[9] Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A., Aflalo, C., and Vakser, I. Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques. *Proc. Nat. Acad. Sci. 89* (1992), 2195–2199.

[10] Korb, O. *Efficient Ant Colony Optimization Algorithms for Structure- and Ligand-Based Drug Design*. PhD thesis, University of Konstanz, 2008.

[11] Kozakov, D., Brenke, R., Comeau, S., and Vajda, S. PIPER: an FFT-based protein docking program with pairwise potentials. *Proteins: Structure, Function, and Genetics 65* (2006), 392–406.

[12] Kuntz, I., Blaney, J., Oatley, S., Langridge, R., and Ferrin, T. A geometric approach to macromolecule-ligand interactions. *Journal of Molecular Biology 161* (1982), 269–288.

[13] May, M. Playstation cell speeds docking programs. *Bio-IT World July 14* (2008).

[14] Nukada, A., Ogata, Y., Endo, T., and Matsuoka, S. Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC08)* (2008).

[15] Pymol. *http://pymol.sourceforge.net*, 2008.

[16] Servat, H., Gonzalez-Alvarez, C., Aguilar, X., Cabrera-Benitez, D., and Jimenez-Gonzalez, D. Drug design issues on the Cell BE. In *Proceedings of 3rd International Conference on High Performance and Embedded Architectures and Compilers* (2008), pp. 176–190.

[17] Sukhwani, B., and Herbordt, M. Acceleration of a production rigid molecule docking code. In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications* (2008), pp. 341–346.

[18] VanCourt, T., Gu, Y., and Herbordt, M. FPGA acceleration of rigid molecule interactions. In *Proceedings of the IEEE Conference on Field Programmable Logic and Applications* (2004).

[19] VanCourt, T., and Herbordt, M. Rigid molecule docking: FPGA reconfiguration for alternative force laws. *Journal on Applied Signal Processing v2006* (2006), 1–10.