BIOINFORMATICS ORIGINAL PAPER

Vol. 26 no. 19 2010, pages 2398–2405
doi:10.1093/bioinformatics/btq444

Structural bioinformatics

Advance Access publication August 4, 2010

Ultra-fast FFT protein docking on graphics processors

David W. Ritchie* and Vishwesh Venkatraman

INRIA Nancy—Grand Est, LORIA, 615 Rue du Jardin Botanique, 54506 Vandoeuvre-lès-Nancy, France

Associate Editor: Burkhard Rost

ABSTRACT

Motivation: Modelling protein–protein interactions (PPIs) is an increasingly important aspect of structural bioinformatics. However, predicting PPIs using *in silico* docking techniques is computationally very expensive. Developing very fast protein docking tools will be useful for studying large-scale PPI networks, and could contribute to the rational design of new drugs.

Results: The *Hex* spherical polar Fourier protein docking algorithm has been implemented on Nvidia graphics processor units (GPUs). On a GTX 285 GPU, an exhaustive and densely sampled 6D docking search can be calculated in just 15 s using multiple 1D fast Fourier transforms (FFTs). This represents a 45-fold speed-up over the corresponding calculation on a single CPU, being at least two orders of magnitude times faster than a similar CPU calculation using ZDOCK 3.0.1, and estimated to be at least three orders of magnitude faster than the GPU-accelerated version of PIPER on comparable hardware. Hence, for the first time, exhaustive FFT-based protein docking calculations may now be performed in a matter of seconds on a contemporary GPU. Three-dimensional *Hex* FFT correlations are also accelerated by the GPU, but the speed-up factor of only 2.5 is much less than that obtained with 1D FFTs. Thus, the *Hex* algorithm appears to be especially well suited to exploit GPUs compared to conventional 3D FFT docking approaches.

Availability: http://hex.loria.fr/ and http://hexserver.loria.fr/

Contact: dave.ritchie@loria.fr

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on April 29, 2010; revised on July 19, 2010; accepted on July 31, 2010

1 INTRODUCTION

Protein docking is the task of calculating the 3D structure of a protein complex starting from unbound or model-built protein structures (Halperin *et al.*, 2002). As well as providing a useful technique to help study fundamental biomolecular mechanisms, using docking tools to predict protein–protein interactions (PPIs) is emerging as a promising complementary approach to rational drug design (Grosdidier *et al.*, 2009).

Although proteins are intrinsically flexible, many protein docking algorithms begin by assuming the proteins to be docked are rigid, and they employ geometric hashing (Bachar *et al.*, 1993) or fast Fourier transform (FFT) correlation techniques (Katchalski-Katzir *et al.*, 1992) to find putative initial docking poses, which are then re-scored and refined using more sophisticated but more computationally expensive techniques (Ritchie, 2008; Vajda and Kozakov, 2009). This article focuses on using graphics processor units (GPUs) to accelerate FFT-based approaches to the initial rigid body stage of a docking calculation.

The FFT approach was first used to as a rapid way to calculate shape complementarity within a 3D Cartesian grid (Katchalski-Katzir *et al.*, 1992). It was later extended to include electrostatic interactions, e.g. FTDOCK (Gabb *et al.*, 1997) and DOT (Mandell *et al.*, 2001), or both electrostatic and desolvation contributions, e.g. ZDOCK (Chen *et al.*, 2003). However, because most FFT-based approaches use 3D Cartesian grid representations of proteins, they can only compute *translational* correlations, and these must be repeated over multiple rotational samples in order to cover the 6D search space. Recently, FFT techniques have been used to calculate correlations of multi-term knowledge-based potentials (Kozakov *et al.*, 2006; Sumikoshi *et al.*, 2005). However, each cross-term in the potential requires a corresponding FFT to be calculated, and this adds to the overall computational expense. Furthermore, protein docking algorithms provide a useful way to study the nature of encounter complexes (Grünberg *et al.*, 2004), and they are beginning to be used as an *in silico* technique to help predict PPI networks (Mosca *et al.*, 2009; Yoshikawa *et al.*, 2009). Both of these approaches are computationally intensive because they involve performing many cross-docking calculations. There is, therefore, a need to develop more efficient techniques to calculate PPIs.

To address the main limitations of the Cartesian FFT approaches, we developed the spherical polar Fourier (SPF) technique, which uses *rotational* correlations (Ritchie and Kemp, 2000) to accelerate the calculation. This reduces execution times to a matter of minutes on an ordinary workstation (Ritchie, 2008). The related FRODOCK (fast rotational docking) approach has also recently demonstrated considerable performance gains compared to Cartesian grid-based FFT approaches (Garzon *et al.*, 2009). Nonetheless, further computational improvements are always desirable because greater speed may be traded for greater accuracy.

In recent years, many scientific calculations have benefited from the very high arithmetic capabilities of modern GPUs (Owens *et al.*, 2007). Initially, it required considerable skill and knowledge of graphics programming techniques to transform a scientific calculation into a form that could be executed by dedicated pixel processing hardware on a GPU. However, with the advent of programmable GPUs and software development tools such as Brook (Buck *et al.*, 2004) and the CUDA (Common Unified Device Architecture) toolkit (http://www.nvidia.com/), it is now much easier to deploy scientific software on GPUs. For example, using GPUs to calculate protein and DNA sequence alignments can give speed-ups from at least a factor of 10 (Manavski and Valle, 2008; Schatz *et al.*, 2007) to over a 100 (Suchard and Rambaut, 2009) compared to

*To whom correspondence should be addressed.

the same calculation on conventional CPUs. Similarly, GPUs can give speed-ups from around 10 to 100 for molecular dynamics simulations (Stone *et al.*, 2007; van Meel *et al.*, 2008), up to a factor of 130 for quantum chemistry calculations (Ufimtsev and Martínez, 2008), and more than a 200-fold increase for wavelet analyses of mass spectrometry data (Hussong *et al.*, 2009). GPUs have also been used to accelerate the calculation of protein accessible surface areas, for example, giving speed-up factors from around 100 to over 300, depending on the size of the protein (Dynerman *et al.*, 2009).

In the context of protein docking, Sukhwani and Herbordt (2009) have implemented the PIPER program (Kozakov *et al.*, 2006) on a C1060 GPU, to achieve a speed-up of about a factor of 18 compared to a 2 GHz CPU. As far as we know, PIPER is so far the only docking program for which GPU performance results have been published. However, because PIPER uses 3D Cartesian FFT grids, and because processing each grid takes over 0.5 s on the GPU, a 6D docking calculation still takes several GPU-hours. Although *Hex* is already much faster on one CPU than the GPU version of PIPER, we were nonetheless encouraged by these earlier successes to explore whether similar speed-ups could be achieved by implementing SPF docking correlations on a GPU.

2 METHODS

2.1 GPUs

Although the details may vary, modern GPUs consist of several programmable multi-processors (MPs) each of which comprises multiple scalar processors (SPs), or 'cores', and a modest amount of fast on-chip, or 'local', memory. This local memory is often divided into read-only 'constant' memory for storing constant parameters, and re-writable 'shared memory' which may be accessed simultaneously by multiple SPs. Additionally, GPUs often have a large amount of external 'global' memory (in the order of hundreds megabytes or several gigabytes), which is accessible by each SP. Communication between the CPU and the GPU mainly involves copying data between the CPU and global GPU memory.

The main GPU used here is an Nvidia GeForce GTX 285. This relatively high-end device has 30 MPs, 240 SPs, 1 Gb of global memory, and a clock speed of 1.48 GHz. Each SP can calculate at least one single precision floating point arithmetic operation (or 'flop') per clock cycle, and in favourable circumstances fused multiply–add instructions (three flops per cycle) can be used. Hence, this GPU is capable of at least 355 Gflops and has a theoretical maximum of 1065 Gflops. This is at least 100 times greater than that of a single core of a conventional CPU. GPUs manufactured by ATI, for example, have a somewhat different architecture, but have broadly similar computational performance. We chose to use Nvidia hardware in order to exploit the associated CUDA programming development tools and run-time libraries.

The CUDA device architecture promotes a very fine-grained 'SIMT' (Simultaneous Instructions Multiple Threads) programming model in which individual threads of execution are responsible for manipulating a small number of closely related data elements. The SIMT model is implemented using small 'kernel' functions, which have a similar syntax to the C and C++ programming languages, and which are executed in parallel by the MPs. This model is well suited for performing simple and repetitive arithmetic operations such as those found in matrix multiplications and FFTs. For such calculations, it is natural to let one SIMT thread be responsible for calculating one element of an array, for example.

Although the overall aim of CUDA is to provide an abstract way to program Nvidia GPUs, it is still necessary to understand the characteristics of these devices in order to achieve the best possible performance. For example, in contrast to conventional CPUs that often have several megabytes of fast cache memory, a considerable drawback of current GPUs is that they do not have any cache memory at all. This is significant because access to global memory is about 80 times slower than access to a register or shared memory. Hence, it is important to ensure that data traffic between the MPs and global memory is kept to a minimum, and that as much work as possible is done with data in the fast on-chip registers and shared memory.

In the CUDA SIMT model, threads are given numerical index identifiers and are grouped into 'blocks' with consecutive indices. Blocks of threads may further be grouped into 'grids' of thread blocks. Thus, threads may be indexed using 1D, 2D or 3D indexing schemes, respectively. CUDA devices schedule and execute blocks of threads by dividing them into 'warps' of 32 threads. Each warp executes one instruction at a time, so maximum efficiency is achieved when all of the threads of a warp execute the same sequence of instructions. On the other hand, access to global memory is most efficient when the data is aligned in memory on even word boundaries, and when all threads in a half-warp (i.e. either the first or second group of 16 threads in a warp) access consecutive memory elements simultaneously, for example. When this occurs, the MP can coalesce multiple memory accesses into a single transaction (the precise conditions necessary for coalescing memory accesses are described in the CUDA Programming Guide: developer.nvidia.com/object/cuda_downloads.html). By running several warps concurrently, MPs can hide the latency of global memory provided that at least one warp always has sufficient data in registers or shared memory to operate on.

To take into account the above characteristics and to optimize overall performance with a minimum of effort, we compiled from the CUDA Programming Guide and associated code examples a list of simple strategies for porting code to the GPU:

- only implement rate-limiting calculations on the GPU;
- perform non-trivial initializations on the CPU and copy the values to the GPU;
- store commonly used constants in the 'constant' memory area of the GPU;
- store complex numbers as consecutive pairs of single precision data elements;
- use the CUDA '__align__' macro to force data structures to begin on 8-byte boundaries;
- re-structure complicated data structures as regular arrays;
- round up array dimensions to multiples of 16;
- re-write a group of matrix–vector multiplications as one matrix–matrix multiplication;
- avoid using conditional statements inside loops;
- associate one array subscript with one thread index;
- access multi-dimensional arrays in natural subscript order;
- copy data between CPU and GPU memory in large chunks;
- perform matrix operations using 16 × 16 tiles of data following the 'matrixMul' example in the Nvidia developers' toolkit;
- copy data between global and shared GPU memory using 16 × 16 tiles following the 'transpose' example in the Nvidia developers' toolkit.

All of these techniques were used here, as described below.

2.2 SPF correlations

The SPF docking approach has been described previously (Ritchie and Kemp, 2000; Ritchie, 2005; Ritchie *et al.*, 2008). Nonetheless, a brief summary is given here in order to describe how it has been implemented on GPUs.

The SPF approach begins with a voxel-based representation of protein shape similar to that originally described by Katchalski-Katzir *et al.* (1992). However, instead of directly calculating conventional 3D Cartesian FFTs, we use the voxel samples to encode the shapes of proteins as 3D polynomial expansions of orthonormal spherical polar basis functions. For example, the

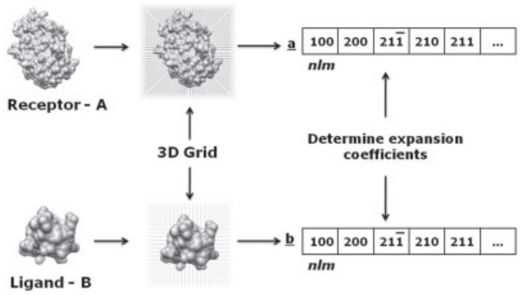
Receptor - A
3D Grid
Ligand - B
Determine expansion coefficients
a
100 | 200 | 21$\bar{1}$ | 210 | 211 | ...
nlm
b
100 | 200 | 21$\bar{1}$ | 210 | 211 | ...
nlm

Fig. 1. Schematic illustration of the calculation and storage of the SPF expansion coefficients as compact 1D coefficent vectors indexed by three subscripts, *nlm*. Overlines represent negative subscript values.

interior volume of protein A (the 'receptor') is encoded as an expansion to order N using

$$\tau_A(\underline{r})=\sum_{nlm} a^{\tau}_{nlm} R_{nl}(r) y_{lm}(\theta,\phi), \tag{1}$$

where $\underline{r}=(r,\theta,\phi)$ are 3D spherical polar coordinates, $y_{lm}(\theta,\phi)$ are normalized real spherical harmonic functions (Biedenharn and Louck, 1981), $R_{nl}(r)$ are orthonormal Gauss-Laguerre radial basis functions (Ritchie, 2005) and a^{τ}_{nlm} are the expansion coefficients. The summation ranges over all subscript values that satisfy $|m| \le l < N$. The default expansion order is $N=25$. The volumes of a surface skin region around the receptor $\sigma_A(\underline{r})$ and the corresponding volumes on protein B (the 'ligand'), $\tau_B(\underline{r})$ and $\sigma_B(\underline{r})$, are expressed in a similar way. Other properties such as electrostatic potential and charge density may also be encoded similarly. Thanks to the orthogonality of the basis functions, the expansion coefficients are easily determined by numerical integration on a 0.6 Å^3 grid (Ritchie and Kemp, 2000). This correspond to performing a forward Fourier transform in conventional Cartesian grid-based FFT approaches.

Although the SPF expansion coefficients have three subscripts, it is often convenient to store and manipulate them as compact 1D vectors, as illustrated in Figure 1. With this representation, shape complementarity may be written as a two-term overlap expression:

$$C=\int \left(\sigma_A(\underline{r})\tau_B(\underline{r})+\overline{\sigma}_B(\underline{r})\tau_A(\underline{r})\right)\mathrm{d}V, \tag{2}$$

where $\overline{\sigma}_B(\underline{r})=\sigma_B(\underline{r})-Q\tau_B(\underline{r})$, and where Q is a penalty factor that penalizes interior–interior overlaps. We use $Q=11$.

Now, it can be shown that the SPF coefficients transform among themselves under rotation according to

$$a_{nlm}(\alpha,\beta,\gamma)=\sum_{m'} R^{(l)}_{mm'}(\alpha,\beta,\gamma) a_{nlm'}, \tag{3}$$

where (α,β,γ) are Euler rotation angles and $R^{(l)}_{mm'}(\alpha,\beta,\gamma)$ are matrix elements of the real Wigner rotation matrices (Biedenharn and Louck, 1981). In other words, the effect of rotating a protein may be simulated by transforming only its expansion coefficients according to Equation (3). Similarly, it can be shown that the effect of translating a protein by a distance R along the z-axis may be simulated by transforming its expansion coefficients according to:

$$a_{nlm}(R)=\sum_{kj} T^{(|m|)}_{nl,kj}(R) a_{kjm}, \tag{4}$$

where the $T^{(|m|)}_{nl,kj}(R)$ are the matrix elements for translations of the Gauss-Laguerre basis functions (Ritchie, 2005).

Our overall strategy for calculating docking correlations, therefore, is to calculate lists of rotated and translated coefficient vectors for the receptor and ligand proteins, and to evaluate Equation (2) for all possible pairs of such vectors. However, in order to accelerate the calculation using FFT techniques, it is convenient to use both real and complex expansion coefficients. For example, Equation (1) can equally be written as

$$\tau_A(\underline{r})=\sum_{nlm} A^{\tau}_{nlm} R_{nl}(r) Y_{lm}(\theta,\phi), \tag{5}$$

where $Y_{lm}(\theta,\phi)$ are the complex spherical harmonics and A^{τ}_{nlm} are the corresponding complex expansion coefficients. It can be shown that the real and complex coefficients are related according to

$$A^{\tau}_{nlm}=\sum_{m'} a^{\tau}_{nlm'} U^{(l)}_{m'm}, \tag{6}$$

where $U^{(l)}$ is a unitary transformation matrix (Biedenharn and Louck, 1981). Hence, by taking complex linear combinations of pairs of property vectors

$$\begin{aligned} A_{nlm}&=\sum_{m'} (a^{\tau}_{nlm'}+ia^{\sigma}_{nlm'}) U^{(l)}_{m'm} \\ B_{nlm}&=\sum_{m'} (\overline{b}^{\sigma}_{nlm'}+ib^{\tau}_{nlm'}) U^{(l)}_{m'm}, \end{aligned} \tag{7}$$

the overlap Equation (2) may be calculated as

$$C=Re\Big(\sum_{nlm} A^{*}_{nlm} B_{nlm}\Big), \tag{8}$$

where $i=\sqrt{-1}$ and the asterisk denotes complex conjugation. The *in vacuo* electrostatic interaction energy may be calculated in a similar way (Ritchie and Kemp, 2000).

Furthermore, if $A_{nlm}(R,\beta_A,\gamma_A)$ and $B_{nlm}(\beta_B,\gamma_B)$ represent translated and rotated complex coefficients of the receptor and ligand, respectively, the overlap score as a function of the remaining twist angle degree of freedom, α_B, may be calculated as

$$C(\alpha_B)=\sum_{m} e^{-im\alpha_B} S_m(R,\beta_A,\gamma_A,\beta_B,\gamma_B), \tag{9}$$

where the coefficients S_m are given by

$$S_m(R,\beta_A,\gamma_A,\beta_B,\gamma_B)=\sum_{nl} A^{*}_{nlm}(R,\beta_A,\gamma_A) B_{nlm}(\beta_B,\gamma_B). \tag{10}$$

Because Equation (9) has the form of a 1D Fourier series in the m index, the calculation over multiple rotational samples for α_B may be performed efficiently using a 1D FFT. We normally use an FFT length of 64, which gives angular increments of 360°/64 = 5.625°. For the remaining rotation angles, near-regular patterns of (β,γ) angles are generated from icosahedral tessellations of the sphere (Ritchie and Kemp, 2000). For example, the default icosahedral tessellation of 812 vertices gives angular samples with an average separation of $\sim$7.5°. To complete a docking search, Equation (9) is evaluated over multiple pairs (812×812) of (β,γ) molecular rotations, and the entire calculation is repeated over a range of intermolecular separations, R. We normally use 50 translational steps of 0.8 Å. This generates in the order of two billion (2×10^9) trial docking poses.

By rewriting Equations (3) and (9) to expose different Euler rotational angles as complex exponentials, it is possible to express the overall calculation as a list of 3D or even 5D FFTs. For example, a 3D rotational FFT can be calculated using

$$C(\alpha_B,\beta_B,\gamma_B)=\sum_{muv} e^{-i(m\alpha_B+2u\beta_B+v\gamma_B)} S_m(R,\beta_A,\gamma_A), \tag{11}$$

where the S_m coefficients are now given by

$$S_m(R,\beta_A,\gamma_A)=\sum_{nl} A^{*}_{nlm}(R,\beta_A,\gamma_A) B_{nlv} \Lambda^{um}_{lv}, \tag{12}$$

and where Λ^{um}_{lv} is a rotational scaling factor (Ritchie *et al.*, 2008). Equation (11) is normally evaluated using a 3D FFT grid of $48\times 24\times 48$ elements, which gives rotational steps of 7.5° for each of the three ligand rotation angles $(\alpha_B,\beta_B,\gamma_B)$. In each case, outer iterations over the remaining degree(s) of freedom must be performed to cover the 6D search space. Five-dimensional rotational FFTs may be calculated in a similar way. It might be expected that 5D FFTs would be faster than 3D and 1D FFTs, but we find that this is often not the case in practice due to the large memory requirement

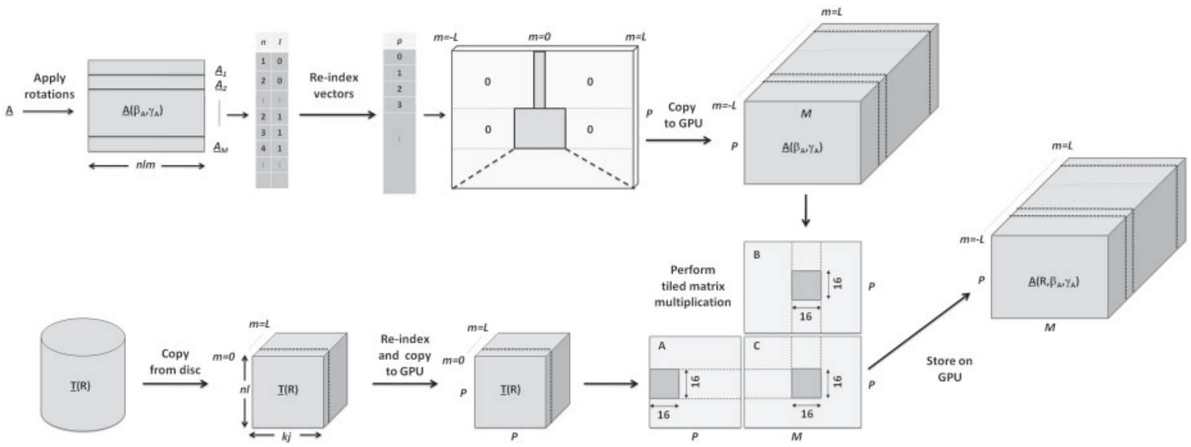

Fig. 2. Re-ordering the expansion coefficient vectors into sparse arrays suitable for tiled matrix multiplication on the GPU. Here, M rotated receptor coefficient vectors are re-indexed to give a 3D array of $2L+1$ planes of dimension $M \times P$. Similarly, the translation matrix elements are re-indexed to give $L+1$ planes of dimension $P \times P$. This allows multiple coefficient vectors to be translated efficient using tile-based matrix multiplications.

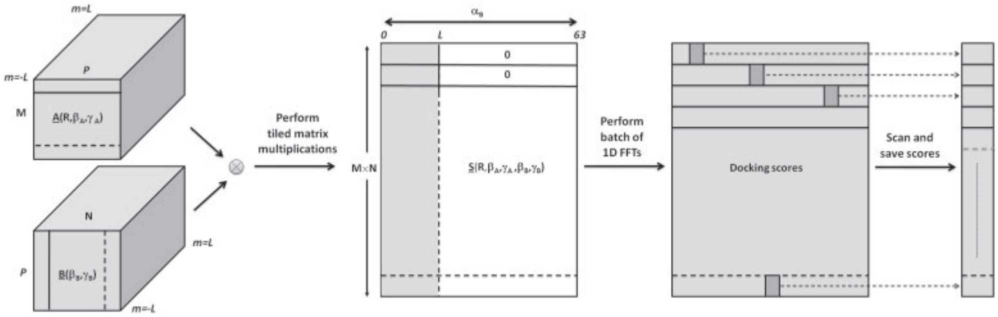

Fig. 3. Calculating multiple 1D FFTs on the GPU for multiple pairs of transformed receptor and ligand coefficient vectors. For each translation step R, M translated and rotated receptor coefficient vectors and N rotated ligand coefficient vectors are combined using Equation (10) to give $M \times N$ vectors of twist angle coefficients, $\underline{S}(R, \beta_A, \gamma_A, \beta_B, \gamma_B)$. The $\underline{S}$ coefficients are then transformed into an array of docking scores using a batch of 1D FFTs [Equation (9)], and the result is scanned to find the best score for each twist angle, α_B.

of the 5D FFT grid. Hence, only the above 1D and 3D FFT schemes are considered further here.

2.3 Implementing SPF correlations on the GPU

From previous experience, we knew *a priori* that the rate-limiting steps in the CPU-based *Hex* docking calculations are the receptor translation [Equation (4)] and pair-wise coefficient multiplication steps [Equations (10) and (12)], all of which involve double summations over the n and l subscripts. Although calculating the initial expansion coefficients and applying the (β, γ) rotations to the receptor and ligand coefficients costs several seconds of CPU time, these are not rate-limiting steps because they can be performed before the main iteration over pairs of receptor and ligand poses. We, therefore, wrote GPU kernels only to implement Equations (4), (10) and (12), and we used the Nvidia cuFFT library for the 1D and 3D FFTs to calculate Equations (9) and (11), respectively.

Using the GPU programming strategies listed in Section 2.1, each pair of nl subscripts are first mapped to single index, p, in which successive data elements are arranged in order of the l and then n subscripts, and where the total number of elements, P, is rounded up to a multiple of 16. This scatters the elements of a compact coefficient vector into a sparse 2D array of dimension $P \times (2L+1)$ elements, where $L = N-1$. This is illustrated in the upper part of Figure 2. Re-indexing M rotated coefficient vectors in this way gives a 3D block of $P \times M \times (2L+1)$ coefficients in sparse format (Fig. 2). Similarly, the SPF translation matrix elements are re-indexed to give a sparse 3D array of $P \times P \times (L+1)$ elements. This allows all M coefficient vectors to be translated together by performing $2L+1$ matrix–matrix multiplications, as illustrated in the lower part of Figure 2. This can be done very efficiently in a GPU kernel by using one 16×16 block of threads to calculate each tile of the result matrix. Furthermore, it is straightforward to make this kernel skip completely any tile consisting entirely of zeros.

To complete the 1D FFT docking scheme, two further GPU kernels were implemented. The first of these cross-multiplies and zero-pads pairs of receptor and ligand coefficient according to Equation (10). This gives a list of $N \times M$ data vectors of length 64 which can be evaluated as a batch of 1D FFTs using the cuFFT library. The output from the FFT is an array of docking scores, or pseudo-energies, as a function of the intermolecular twist angle α_B. This array is scanned by the second kernel to identify the lowest energy for each twist angle. Finally, the resulting list of poses and energies is copied back to main CPU memory. These steps are illustrated in Figure 3. The above sequence of operations is repeated for each translational step of the 6D search.

The 3D FFT scheme generally follows the same execution path, but only a single unrotated ligand coefficient vector has to be copied to the GPU to calculate the array of 3D FFT coefficients [Equation (12)]. This array is then passed to the 3D cuFFT function, and the best poses and energies are

Table 1. Typical SPF initialization times in seconds for some example complexes using expansions to order $N = 25$ on a 2.3 GHz workstation

Receptor	#residues	Ligand	#residues	1×CPU	4×CPU
Kallikrein A	233	BPTI	58	7.5	2.0
HyHel-5 Fv	215	Lysozyme	130	8.5	2.3
TGF-β	331	FKB12	108	11.7	3.1

copied back to CPU main memory, as before. Faster implementations of the 3D FFT have been proposed for GPUs (Govindraju *et al.*, 2008; Nukada *et al.*, 2008), but we have not explored these here because significantly better docking performance is obtained using 1D FFTs, as shown below.

3 RESULTS AND DISCUSSION

3.1 Multi-threaded implementations

All calculations have been implemented using 'thread-safe' programming techniques using the Posix 'pthreads' and Windows thread libraries for Linux and Windows systems, respectively. Thus, multiple GPUs and CPU cores may be used simultaneously on most current workstations. The results presented here refer mainly to a contemporary workstation with a quad-core 2.3 GHz Xeon CPU and one GTX 285 GPU. Some overall timing results are also given for several other Nvidia devices for comparison.

3.2 Forward SPF transforms are not rate-limiting

In the SPF approach, protein shapes are sampled just once by an initial forward transform using numerical integration. Thereafter, assuming that different complexes are docked with the same search parameters, as is normally the case in *Hex*, all subsequent SPF calculations are independent of the size and nature of the proteins because they manipulate and transform SPF coefficient vectors of the same length and in the same way. Hence, calculation times for SPF docking correlations are practically constant for all complexes for a given FFT sampling scheme (1D or 3D). Table 1 shows the extent to which the initial sampling times (and, therefore, also overall execution times) vary according to the sizes of the proteins. Because this initialization step is currently not rate-limiting for docking, it has not been implemented on the GPU.

3.3 GPUs give almost identical numerical results

All FFT calculations were performed in double precision using the Intel Math Kernel Library (MKL) on the CPU and in single precision using the cuFFT library on the GPU. Hence, some small numerical differences between the CPU and GPU results are to be expected. From visual inspection of the results for docking the Kallikrein A / BPTI example, we find that the calculated GPU and CPU docking energies agree to within at least four decimal digits, or equivalently to within at least 0.05 kJ/mol for each pose (data not shown). However, as expected, some small differences between the 1D and 3D FFT schemes were observed, because the two schemes use fundamentally different rotational sampling techniques.

To quantify the effect of these differences in more detail, blind docking was performed on the 63 'rigid body' complexes of the Protein Docking Benchmark (Mintseris *et al.*, 2005). Full details of the results are presented in Supplementary Materials. Table 2 summarizes these results in terms of the mean rank and average ligand C$_\alpha$ root-mean-squared (RMS) deviation from the complex of the first 'hit' (here defined as a pose within 10 Å RMS of the complex) found within the first 1000 solutions. These values show that there is very little overall difference between the GPU and CPU calculations. This may be confirmed by closer examination of the individual docking results in Supplementary Table 1. This shows that the GPU and CPU calculations often give identical or very similar ranks to the first pose found within 10 Å RMS of the complex, although there are sometimes some fluctuations in the ranks and poses of less highly ranked predictions.

Table 2. Mean rank, RMS deviation, and number of hits obtained for exhaustive unbound–unbound docking of the 63 'rigid body' complexes of the Docking Benchmark (version 2)

3D CPU	3D GPU	1D CPU	1D GPU
Rank (RMS) Hits	Rank (RMS) Hits	Rank (RMS) Hits	Rank (RMS) Hits
147 (7.8) 6.4	147 (7.9) 6.2	166 (7.9) 5.8	165 (7.9) 5.6

This table summarizes the results presented in Supplementary Table 1. For each FFT docking scheme, the overall results are listed as the mean rank, the average ligand C$_\alpha$ RMS deviation with respect to the crystal structure of the complex, and the average number of 'hits' found within the top 1000 docking poses. Any pose for which the ligand RMS is within 10 Å of the complex is considered to be a 'hit'. Means of ranks were calculated using the mean log rank formula of Ritchie *et al.* (2008). All docking runs used default search parameters with a steric scan using $N = 18$ followed by shape plus electrostatic re-scoring using $N = 25$.

Table 2 also shows that the 3D FFT scheme gives marginally better results than the 1D scheme. We believe this is because the 3D FFT scheme tends to over-sample rotation space near the poles, and hence has a slightly better chance of sampling a near-native poses than the more regular icosahedral sampling pattern used in the 1D FFT scheme. Clearly, using different rotational and translational sampling densities for either FFT scheme would cause comparable fluctuations in the results. Overall, these tables show that the GPU and CPU calculations give almost identical numerical results, and that the effect of any arithmetic differences is very small compared to using different orientational sampling patterns.

3.4 Over 100-fold GPU speed-up for 1D correlations

Figure 4 shows the overall docking correlation rates at different polynomial expansion orders for both the 1D and 3D FFT calculation schemes described above. This shows that a GPU can calculate the 1D FFT docking scheme significantly faster than the same calculation on the CPU, and indeed also significantly faster than the 3D FFT scheme on both the GPU and CPU. As might be expected, 3D FFT calculation rates are less sensitive to the polynomial order than the 1D FFTs because they require less explicit matrix arithmetic and they can benefit more from the $O(N \log N)$ nature of the FFT. However, for the relatively low expansion orders used here, 3D FFTs on the GPU are not substantially faster than using a single CPU core. On the other hand, because *Hex* performs an initial scan of the search space using $N = 18$, and because only a small fraction of the remaining poses are re-scored using $N = 25$, the overall benefit of using a GPU to calculate the 1D docking scheme is dramatic. As shown in Figure 4, the 1D FFT scheme can score 236 million poses per second using $N = 18$ on the GPU, and 104 million poses per

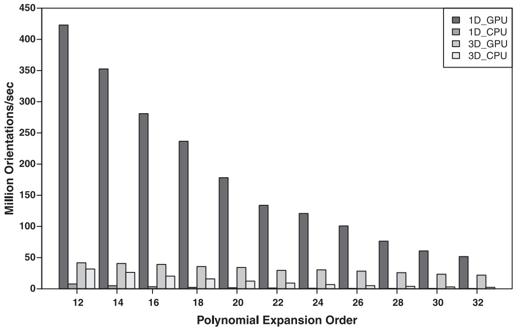

Fig. 4. Docking correlation rates (millions of poses per second) using the 1D and 3D correlation schemes at various polynomial expansion orders. The GPU rates include all data transfers between the GPU and CPU, the time required to re-index and translate the coefficients, and the time spent in the cuFFT library. All rates exclude the costs of calculating the initial forward transform and reading the translation matrices from disc.

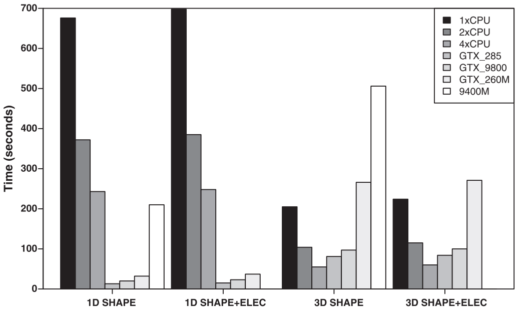

Fig. 5. Total execution times for exhaustive *Hex* docking runs with default search parameters on a variety of GPU devices compared to using from one to four CPU cores simultaneously. The GPU devices have the following specifications: GTX 285: 240 cores, 1.48 GHz, 1 GB memory (workstation); GTX 9800: 128 cores, 1.7 GHz, 512 MB memory (workstation); GTS 260M: 112 cores, 1.37 GHz, 1 GB memory (laptop); 9400M: 16 cores, 1.4 GHz, 256 MB memory (laptop). The 9400M device has insufficient memory for shape plus electrostatic correlations.

second using $N = 25$. These rates correspond to speed-up factors of 100 and 130, respectively, compared to a single CPU core.

Although the above speed-ups are impressive, overall execution times are reduced by smaller factors than these because it is not feasible to implement all of the steps of a docking calculation on the GPU. Furthermore, because current CPUs typically have up to four identical cores, it is perhaps fairer to compare one GPU with four CPU cores. Although other protein docking programs such as DOT (Mandell *et al.*, 2001), PIPER (Kozakov *et al.*, 2006) and ZDOCK 3.0.1 (Mintseris *et al.*, 2007) can distribute the workload over multiple nodes in a compute cluster using, for example, the MPI message passing library (http://www.open-mpi.org/), to our knowledge *Hex* is currently the only protein docking program that uses multi-threading techniques to exploit multi-core processors. Figure 5 compares the overall execution times of a typical *Hex* docking run on a variety of GPU devices with using from one to four CPU cores simultaneously. In all cases, adding electrostatics to the scoring function costs little because it is only calculated for the best 25 000 poses in the final re-scoring step.

Table 3. Total execution times in seconds for ZDOCK, PIPER and *Hex*

	ZDOCK 1×CPU	PIPER[a] 1×CPU	PIPER[a] 1×GPU	*Hex* 1×CPU	*Hex* 4×CPU	*Hex*[b] 1×GPU
3D	7172	468 625	26 372	224	60	84
(3D)[c]	(1195)	(42 602)	(2398)	224	60	84
1D	–	–	–	676	243	15

In this table, the ZDOCK and *Hex* values are measured execution times for unconstrained exhaustive docking of the Kallikrein A / BPTI complex, and are tabulated according to the number of CPUs and GPUs used. Dense rotational sampling was used in ZDOCK with a Cartesian grid size of $(92\text{Å})^3$. A comparable rotational sampling density was used in *Hex*, as described in the main text.

[a] The PIPER times are estimated for 54 000 rotational sample steps (as in ZDOCK dense sampling) using the per-rotation times quoted in Table 1 of Sukhwani and Herbordt (2009), i.e. 2.0 GHz CPU: 9.98 s/rotation and C1060 GPU: 0.556 s/rotation, for a receptor grid size of 128^3 and a ligand grid of 32^3. The PIPER CPU time given here has been scaled to that of a 2.3 GHz processor, and the GPU time (C1060: 240 cores, 1.3 GHz) has been scaled to that of a GTX 285 (240 cores, 1.48 GHz).

[b] Here, only the GPU is used in the docking search, although the initialization step uses four CPU cores (see Table 1).

[c] Times given in brackets for ZDOCK and PIPER have been corrected from the times measured for 12-term (ZDOCK) and estimated for 22-term (PIPER) runs to a hypothetical two-term potential like the two-term pseudo-energy used in *Hex*.

Figure 5 also shows that using two CPU cores nearly doubles the overall speed, but using four CPU cores gives only about a 3-fold speed-up. On the other hand, for the 1D FFT scheme, using one GPU core is still at least 10 times faster than using four CPU cores together, which is clearly a significant improvement. Furthermore, on our GPU-based server (Macindoe *et al.*, 2010), we find that using two GPUs together is twice as fast as one GPU. However, because the 1D FFT scheme is so fast, much of this gain is masked by file transfer and network overheads on the web server.

3.5 Speed comparison with ZDOCK and PIPER

To allow a more direct comparison between the performance of the SPF representation and conventional Cartesian FFT grid-based docking approaches, Table 3 compares overall docking times for *Hex* with the execution time for ZDOCK 3.0.1 measured on the same CPU along with CPU and GPU execution times for PIPER, which have been estimated from the timings given by Sukhwani and Herbordt (2009) using the same 'dense' rotational sampling as ZDOCK (54 000 ligand rotational steps of ∼6°). The ZDOCK dense sampling mode is similar to the default *Hex* sampling scheme ($812 \times 64 = 51\,968$ ligand rotations). However, it should be noted that the ZDOCK and PIPER grid sizes of $(92 \times 1.2\text{Å})^3$ and $(128 \times 1.0\text{Å})^3$, respectively, are both larger than the default translational step size used in *Hex* (0.8 Å). Furthermore, ZDOCK 3.0.1 and PIPER both employ multi-term potentials derived from 12 residue types in ZDOCK (Mintseris *et al.*, 2007) and using up to 22 cross-terms in PIPER (Kozakov *et al.*, 2006), whereas *Hex* uses a much simpler two-term shape complementarity model with an optional two-term *in vacuo* electrostatic contribution.

Bearing these similarities and differences in mind, Table 3 shows that using the *Hex* 1D FFT scheme on a high-end GPU is about 475 times faster (7172/15) than ZDOCK 3.0.1, about 31 200 times faster (468 625/15) than PIPER on a single 2.3 GHz CPU core, and about 1750 times faster (26 372/15) than PIPER on a comparable GPU. Table 3 also shows the overall *Hex* execution times for different numbers of CPU cores. This shows that using one GPU to calculate

1D FFT scheme gives the best overall performance, being over 45 times faster (676/15) than the corresponding calculation on a single CPU core, whereas the GPU accelerates the 3D FFT scheme by only a very modest factor of about 2.5 (224/84).

Given that one complex FFT calculation can correlate two shape terms in *Hex* or two potentials in ZDOCK or PIPER, the underlying difference in speed between the 1D SPF (GPU) and 3D Cartesian (CPU) approaches may be estimated to be about a factor of 80 for ZDOCK (i.e. 7172/15/6) and 2840 for PIPER (i.e. 468 625/15/11) per pair-wise property correlation. This clearly indicates that PIPER contains some very expensive steps compared to ZDOCK. Comparing the *Hex* 3D SPF CPU time with the 3D FFT Cartesian calculation in ZDOCK in a similar way gives a SPF/Cartesian speed-up factor of about 14 (i.e. 7172/84/6). Hence, the relatively low GPU/CPU factor of 2.5 for the *Hex* 3D SPF correlations indicates that the *Hex* 3D CPU implementation is in fact very well optimized. Overall, Table 3 shows that the fastest GPU scheme (1D SPF) is about 15 times faster (224/15) than the fastest CPU scheme (3D SPF) which is itself about 32 times faster (7172/224) than the 3D Cartesian-based FFT in ZDOCK.

3.6 1D SPF correlations are well suited for GPUs

Taking into account that Sukhwani and Herbordt (2009) compared PIPER using cuFFT on the GPU with FFTW (http://www.fftw.org) on the CPU (which is known to be slower for FFTs than MKL), our 3D SPF results also show that GPUs do not give substantial speed-ups for 3D FFT calculations. This is because 3D FFT algorithms require multiple passes through the data volume, and on the GPU this exposes the latency of repeatedly accessing global memory without the benefit of fast cache memory. On the other hand, because multiple 1D FFTs can be processed in a single pass over global GPU memory, it follows that the SPF 1D FFT scheme is especially well suited to exploit current GPU architectures.

4 CONCLUSION

The *Hex* 1D and 3D FFT docking schemes have been implemented on CUDA GPUs. Although the CPU version of *Hex* is already much faster than conventional Cartesian grid-based FFT docking algorithms, GPU-based correlations using the 1D FFT scheme are accelerated by at least a factor of 100 compared to a single CPU core, and a very satisfactory 45-fold overall speed-up is achieved for the 1D FFT scheme. This corresponds to an 15-fold speed-up compared to the 3D FFT scheme on the CPU. However, only a very modest GPU/CPU speed-up factor of about 2.5 is obtained for the 3D FFT scheme. This shows that the *Hex* 1D FFT docking scheme is especially well suited to exploit current GPU architectures. On a contemporary high-end GPU, the 1D FFT scheme allows an exhaustive protein docking calculation to be completed in just 15 s, which is at least two orders of magnitude faster than leading conventional Cartesian-based docking algorithms such as ZDOCK and PIPER. Thus, for the first time, exhaustive FFT-based protein docking may now be carried out in interactive time-scales using a modern GPU. This algorithmic improvement will facilitate the use of docking techniques to help study PPIs and PPI networks.

ACKNOWLEDGEMENTS

We thank the Nvidia Professor Partner Programme for the gift of a graphics card.

Funding: Agence Nationale de la Recherche, grant number ANR-08-CEXC-017-01.

Conflict of Interest: none declared.

REFERENCES

Bachar,O. *et al.* (1993) A computer vision based technique for 3D sequence-independent structural comparison of proteins. *Protein Eng.*, **6**, 279–288.

Biedenharn,L.C. and Louck,J.C. (1981) *Angular Momentum in Quantum Physics*. Addison-Wesley, Reading, MA.

Buck,I. *et al.* (2004) Brook for GPUs: stream computing for graphics hardware. *ACM Trans. Graph.*, **23**, 777–786.

Chen,R. *et al.* (2003) ZDOCK: an initial-stage protein-docking algorithm. *Proteins Struct. Funct. Bioinform.*, **52**, 80–87.

Dynerman,D. *et al.* (2009) CUSA and CUDE: GPU-accelerated methods for estimating solvent accessible surface area and desolvation. *J. Comput. Biol.*, **16**, 523–537.

Gabb,H.A. *et al.* (1997) Modelling protein docking using shape complementarity, electrostatics and biochemical information. *J. Mol. Biol.*, **272**, 106–120.

Garzon,J.I. *et al.* (2009) FRODOCK: a new approach for fast rotational protein-protein docking. *Bioinformatics*, **25**, 2544–2551.

Govindraju,N.K. *et al.* (2008) High performance discrete Fourier transforms on graphics processors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Article no. 2, IEEE Press, Piscataway, NJ, pp. 1–12.

Grosdidier,S. *et al.* (2009) Computer applications for prediction of protein-protein interactions and reational drug design. *Adv. App. Bioinf. Chem.*, **2**, 101–123.

Grünberg,R. *et al.* (2004) Complementarity of structure ensembles in protein-protein docking. *Structure*, **12**, 2125–2136.

Halperin,I. *et al.* (2002) Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins Struct. Funct. Genet.*, **47**, 409–443.

Hussong,R. *et al.* (2009) Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics*, **25**, 1937–1943.

Katchalski-Katzir,E. *et al.* (1992) Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. *Proc. Natl Acad. Sci.*, **89**, 2195–2199.

Kozakov,D. *et al.* (2006) PIPER: an FFT-based protein docking program with pairwise potentials. *Proteins Struct. Funct. Bioinform.*, **65**, 392–406.

Macindoe,G. *et al.* (2010) HexServer: an FFT-based protein docking server powered by graphics processors. *Nucleic Acids Res.*, **38**, W445–W449.

Manavski,S.A. and Valle,G. (2008) CUDA-compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, **9**, S10.

Mandell,J.G. *et al.* (2001) Protein docking using continuum electrostatics and geometric fit. *Protein Eng.*, **14**, 105–113.

Mintseris,J. *et al.* (2005) Protein-protein docking benchmark 2.0: an update. *Proteins Struct. Funct. Bioinform.*, **60**, 214–216.

Mintseris,J. *et al.* (2007) Integrating statistical pair potentials into protein complex prediction. *Proteins Struct. Funct. Bioinform.*, **69**, 511–520.

Mosca,R. *et al.* (2009) Pushing structural information into the yeast interactome by high-throughput protein docking experiments. *PLoS Comput. Biol.*, **5**, e1000490.

Nukada,A. *et al.* (2008) Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Article no. 5, IEEE Press, Piscataway, NJ, pp. 1–11.

Owens,J.D. *et al.* (2007) A survey of general-purpose computation on graphics hardware. *Comput. Graph. Forum*, **26**, 80–113.

Ritchie,D.W. and Kemp,G.J.L. (2000) Protein docking using spherical polar Fourier correlations. *Proteins Struct. Funct. Genet.*, **39**, 178–194.

Ritchie,D.W. *et al.* (2008) Accelerating and focusing protein-protein docking correlations using multi-dimensional rotational FFT generating functions. *Bioinformatics*, **24**, 1865–1873.

Ritchie,D.W. (2005) High-order analytic translation matrix elements for real-space six-dimensional polar Fourier correlations. *J. Appl. Cryst.*, **38**, 808–818.

Ritchie,D.W. (2008) Recent progress and future directions in protein-protein docking. *Curr. Protein Pept. Sci.*, **9**, 1–15.

Schatz,M.C. *et al.* (2007) High-throughput sequence alignment using graphics processors. *BMC Bioinformatics*, **8**, 474.

Stone,J.E. *et al.* (2007) Accelerating molecular modeling applications with graphics processors. *J. Comput. Chem.*, **28**, 2618–2640.

Suchard,M.A. and Rambaut,A. (2009) Many-core algorithms for statistical phylogenetics. *Bioinformatics*, **25**, 1370–1376.

Sukhwani,B. and Herbordt,M.C. (2009) GPU acceleration of a production molecular docking code. In *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. ACM, New York, NY, pp. 19–27.

Sumikoshi,K. *et al.* (2005) A fast protein-protein docking algorithm using series expansions in terms of spherical basis functions. *Genome Inform.*, **16**, 161–173.

Ufimtsev,I.S. and Martínez,T.J. (2008) Quantum chemistry on graphical processor units. 1. Strategies for two-electron integral evaluation. *J. Chem. Theory Comput.*, **4**, 222–231.

Vajda,S. and Kozakov,D. (2009) Convergence and combination of methods in protein-protein docking. *Curr. Opin. Struct. Biol.*, **19**, 164–170.

van Meel,J.A. *et al.* (2008) Harvesting graphics power for MD simulations. *Mol. Simul.*, **34**, 259–266.

Yoshikawa,T. *et al.* (2009) Improving the accuracy of an affinity prediction method by using statistics on shape complementarity between proteins. *J. Chem. Inf. Model.*, **49**, 693–703.