Accelerating Grid Kernels for Virtual Screening on Graphics Processing Units

Irene SÁNCHEZ-LINARES, Horacio PÉREZ-SÁNCHEZ [1], and José Manuel GARCÍA
Computer Engineering and Technology Dept., Univ. of Murcia (Spain)

Abstract The discovery of new drugs can be drastically accelerated with the use of Virtual Screening (VS) methods, ongoing trend in medical research. VS methods need to screen chemical compound databases with millions of ligands but they are completely constrained by the access to computational resources. In order to solve this problem we worked on the adaptation of their main bottlenecks to GPUs using a grid approach with different interpolation methods. In our first studies for medium size proteins, our CUDA implementation runs around 30 times faster than the sequential counterpart.

Keywords. Drug Discovery, Virtual Screening, CUDA, GPUs, Grid, Interpolation

1. Introduction

The discovery of new drugs can enormously benefit from the use of Virtual Screening (VS) methods, which need to process databases with millions of ligands [5] and which are constrained by a lack of computational resoruces [6]. We propose to overcome these limitations by the exploitation of GPUs [3,11]. We described previously how the bottleneck of VS methods are related with the computation of full non-bonded interactions Kernels and how GPUs can yield speedups of up to 260 times [4]. Nevertheless, mentioned Kernels need to perform N^2 interactions calculations (N = total number of particles in the system) and even using GPUs, the required computation time grows polynomically with N so this imposes serious limitations for the simulation of big size systems. Thus we decided to look for alternatives to full Kernels and decided to use grid Kernels [9]. We have checked that just in the sequential version, speedups of 200 times versus the full non-bonded Kernel are obtained.

We will therefore unleash the potential of GPUs for the calculation of non-bonded potentials in VS using grids. Previous works have investigated this approach in a similar fashion but for long range interactions using Ewald-Mesh methods [1]. Given the molecular sizes involved in protein-ligand interactions, we will be interested only in short-range electrostatics. Related works reported a 3 times speedup using a different approach [2], a 50 times speedup focusing on the acceleration of more particular Kernels of the docking program Autodock [7], and a 7 times acceleration of the Dock6 scoring function by Yang et al. [13].

[1] Corresponding Author E-mail: horacio@ditec.um.es

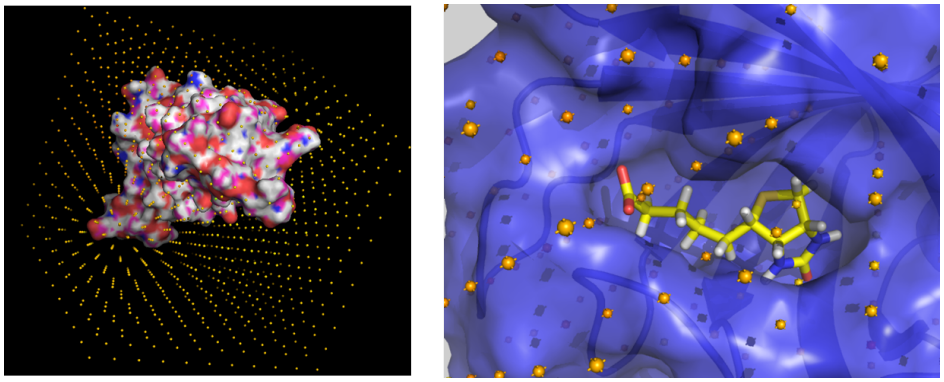

Figure 1. (A) Representation of the grid for the protein streptavidin. Length of the side of the cube (L) is 50 Å, spacing between grid points d is 5 Å, and the total number of grid points is equal to 11^3. (B) Biotin in the binding pocket of streptavidin

2. Calculation of non-bonded interactions using grids

The protein is placed inside a cube of minimal volume $Vol = L^3$ that encloses it. A three dimensional grid is created dividing the cube into $(N-1)^3$ smaller cubes of identical volume, each one of side length $d = L/N$, so that the total number of grid points is N^3. The electrostatic potential due to all protein atoms is calculated on each grid point i according to the Coulomb expression [9]. A graphical depiction of the grid for streptavidin can be seen in Figure 1(A) and in more detail for the ligand biotin on its binding pocket in Figure 1(B).

Once the protein grid is loaded into memory, the calculation of the electrostatic potential for the protein-ligand system is performed as follows; for each ligand atom i with charge q_i at point P_i we calculate which are the eight closest protein grid point neighbours. Next, an interpolation procedure is applied to estimate the value of the electrostatic potential due to all protein atoms at P_i. The same procedure is applied to all ligand atoms summing them up. Different interpolation procedures in 3D have been used [12]; linear, cubic and Gauss interpolation.

3. Code Design

In this section, we introduce the CPU and GPU designs for the calculation of the electrostatic (ES) potential using grids. We have used NVIDIA's CUDA [10] for the GPU implementations on two different machines; a) a host Intel Xeon E6850 CPU with a NVIDIA GeForce GTX 465 GPU and b) a host Intel Xeon E5620 with a NVIDIA Tesla C2050 GPU. They will be referred as Fermi and Tesla. We use GNU gcc version 4.3.4 with the -O3 flag to compile our CPU implementations, and CUDA compilation tools (release 4.0) on the GPU side.

3.1. ES energy calculation on CPU

We perform a VS experiment where a ligand database containing up to thousands of ligands is screened against a single protein molecule. The precomputed protein grid

Algorithm 1 Sequential pseudocode for the calculation of the electrostatic potential

```
1: for i = 1 to N do
2:    for j = 1 to nlig do
3:       energy[i * nlig + j] =
         interpolate(lig[i * nlig + j], ESGrid)
4:    end for
5: end for
```

is read from file and loaded onto memory. Next, the electrostatic (ES) energy of each atom is calculated using interpolation on the grid as explained before and following the pseudocode shown in algorithm 1, where N is the number of ligands, $nlig$ is the number of atoms of each ligand and the function $interpolate$ performs the calculation of the electrostatic potential for each atom.

3.2. ES energy calculation on GPU

We describe in this part the different strategies studied for the GPU implementation. All designs have in common that one thread calculates the energy of only one atom. The threads are organized in blocks of fixed size $numThreads$, being this an important optimization parameter.

3.2.1. GM; use of device memory

In this initial design the whole grid is stored in the GPU device memory. No additional optimizations are implemented.

3.2.2. ROI; truncation to the grid around the ligand

In this strategy we have implemented some optimizations with respect to the previous version (GM); In order to reduce the CPU-GPU data transfer time, we can take advantage of the fact that we only need to access the grid positions in the volume where the ligand is enclosed. So we can define a region of interest (ROI) of the grid around the ligand and send only that part to the GPU instead of the whole grid.

3.2.3. ZIP; compression of the grid

In addition, we can shorten the CPU-GPU grid transfer time with the compression of the positions of the ligand atoms. For that purpose we discretize the volume that encloses the ligands in another cubic regular grid, where each ligand atom is specified only by its grid cell index. The advantage of this approach is that if we perform a fine grain division, each atom can be conveniently represented by just 3 short integers (instead of the three doubles required for position) and reduce memory usage to a quarter.

3.2.4. SM and TM; use of shared and texture memories

We can benefit from the use of shared and texture memory to improve memory access. In the shared memory approach (SM), threads of a block cooperate to copy ROI of the grid to the shared memory in order to obtain a lower memory access penalization. It must be noticed that accessing grid data on SM is coalesced in order to leverage memory

bandwith. Regarding texture memory (TM) approach, we can store protein grid into the texture memory unit (TMI), so that just accessing different memory indexes gives us directly the interpolated [10] energy value for each atom.

3.2.5. ROI-TM-ZIP and ROI-SM-ZIP

Algorithm 2 GPU pseudocode for the calculation of the electrostatic potential

Host (CPU)

1: $CopyDataCPUtoGPU(GridROI)$
2: $clig = compress(lig)$
3: $CopyDataCPUtoGPU(clig)$
4: $nBlocks := N * nlig/numThreads$
5: $Kernel <<< nBlocks, numThreads >>> (GridROI, clig, energy)$
6: $CopyDataFromGPUtoCPU(energy)$

Kernel 1: ROI-TM-ZIP

1: **for all** nBlocks **do**
2: $dlig = decompress(clig[myAtom])$
3: $ilig = positionToROICoordinates(ROIinfo, dlig)$
4: $energy[myAtom] = accessToTextureMemory(GridROI, ilig)$
5: **end for**

Kernel 2: ROI-SM-ZIP

1: **for all** nBlocks **do**
2: $copyDataToSM(GridROI)$
3: $dlig = decompress(clig[myAtom])$
4: $ilig = positionToROICoordinates(GridROI, dlig)$
5: $energy[myAtom] = interpolate(GridROI, dlig, ilig)$
6: **end for**

Kernels shown in Algorithm 2 describe two different mixed groups of optimizations based on the previous strategies. In both Kernels, the host sends the ROI of the grid and the compressed ligand atom positions to the GPU. In Kernel 1 of Algorithm 2, each thread decompresses the coordinates of the corresponding atom and calculates its coordinates in the ROI coordinates system. Next, it performs interpolation in the TMI. In Kernel 2 of Algorithm 2, each thread also decompresses coordinates but the interpolation function is implemented in the code as described. Finally, energy values are copied back to CPU.

4. Results

4.1. Grid spacing and interpolation accuracy

Figure 2 shows how grid spacing d influences accuracy in the different interpolation procedures. We wondered about the smallest possible value of d that yields good accuracy and that uses the less possible number of points for the grid, and therefore memory. We found that a value of $d = 0.5$ Å gives good accuracy for the three interpolation methods. Smaller values of d do not improve accuracy significantly while they require more memory (it depends on $(1/d)^3$). From the other side, higher values of d, like 1.5 Å yield unacceptable results for all studied interpolation methods in the rugged parts of the

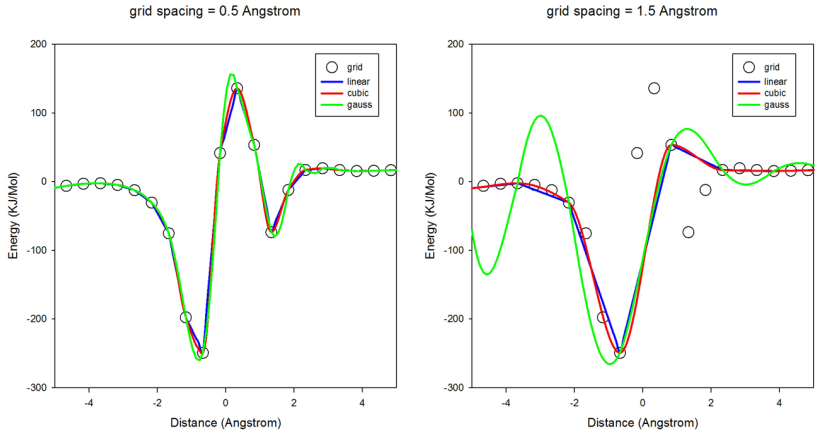

Figure 2. Interpolation results obtained in a part of the grid streptavidin-biotin for grid spacing values of 0.5 (left picture) and 1.5 (right picture) Å, and using different interpolation procedures. For clarity of the comparison we show the values for the grid points pertaining to a grid with a spacing of 0.5 Å.

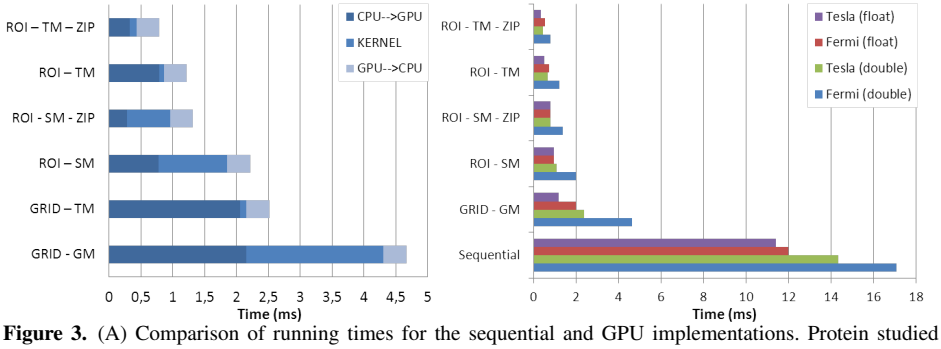

Figure 3. (A) Comparison of running times for the sequential and GPU implementations. Protein studied is streptavidin and the screening was performed using a ligand database containing 2000 ligands, each one containing around 32 atoms. (B) Total running times for the two GPUs used in our study and in float and double precision.

curve, which happens often due to the typical charge distribution in proteins. Therefore we accepted a value of $d = 0.5$ Å as optimal. Regarding the interpolation procedure we discarded the Gauss interpolation given its worst results in the rugged parts of the curve, if we compare it with cubic and linear interpolations, which yield similar accuracy. We finally decided to use only the latter given its lower computational cost. We also discarded interpolation methods of higher order since in the ROI strategy (grid is reduced around the ligand) they would not be able to access grid points out of ROI, yielding wrong results.

4.2. Analysis and performance of the sequential code

In Figure 3(B) we can see the timing results obtained for the sequential code in a ligand database screening with 2000 ligands. The trilinear interpolation needs to access eight adjacent cells of a ligand atom positions. It implies two memory accesses to four different rows of the grid. Furthermore, we cannot exploit the use of the cache due to the fact

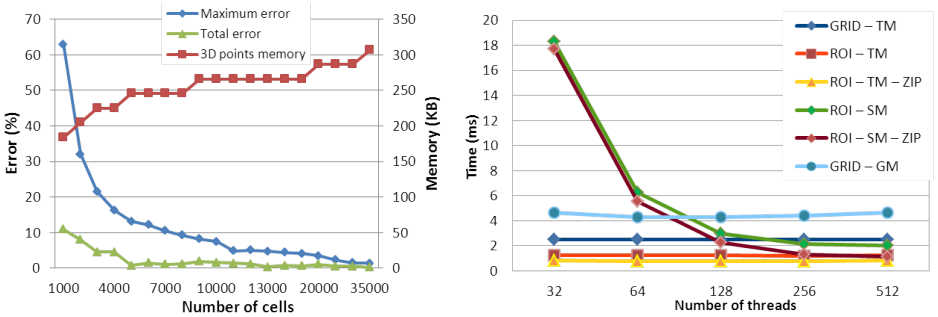

Figure 4. (A) Values of the maximum and total error per atom obtained when using the compressed grid for representing the ligand database for several values of the number of cells and memory consumption in KB (B) Influence of the number of threads per block on the running time for the different implementation strategies studied.

that the atoms are spread in random positions in the 3D space. Therefore most of the RAM accesses represent a bottleneck. Nevertheless, we have used this grid Kernel as starting point and investigated how to adapt it to the GPU architecture, since it is widely used in most biomolecular simulation methods. An additional reason is the 150 to 200 speedups in the sequential version versus the full kernel [4] for several grid densities and size ranges of rigid proteins.

4.3. Compression of the ligand database atomic positions

In Figure 4 we can observe how the number of cells influences the error for the calculation of the electrostatic potential. As one would expect, increasing the number of cells reduces on average the maximum error for the calculation of the potential per atom, and the same for the total error. A maximum error of 0.25% is obtained when we use 35000 cells to compress the whole ligand database. At the same time, memory consumption increases only linearly given the efficiency of the compression method used and only around 300 KB are needed to store the whole ligand database.

4.4. Threads per Block

We have also investigated the influence of the number of threads per block as can be seen in Figure 4(B). Since the designs ROI-SM and ROI-SM-ZIP use the shared memory, they are more affected by the value of the block size than the others. If the number of threads per block is smaller than the ROI size, threads need to perform too many iterations to copy the whole ROI into the shared memory while the high bandwidth memory is unused. For a number of threads per block higher than the ROI size, the memory access bandwidth is improved because there are many simultaneous memory accesses. The global and texture memory are cached (only in the Fermi architecture) and the data copy is performed automatically in background independently of the number of threads. As a consecuence we choose a number of threads equal to 512 as optimal for the shared memory designs.

4.5. Texture Memory

In the TM strategy we have first checked whether we obtain the same interpolation results than in the sequential version and this is confirmed in Figure 5. We can also see how the use of this memory unit decreases considerably the time needed for the calculation of the interpolation. It is clearly shown in Figure 3(A) in cases GRID-GM to GRID-TM and ROI-SM or ROI-SM-ZIP to ROI-TM. Therefore it is a good idea to use always the TMI when linear interpolation is required. Finally, if we look at Figure 3(A) it is clear that ROI-TM-ZIP and ROI-SM-ZIP offer the best performance since they combine all the best advantages from the previous strategies.

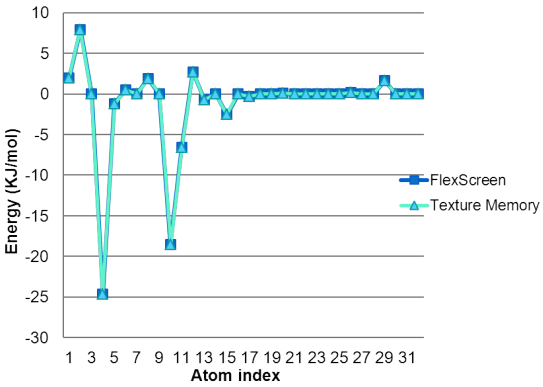

Figure 5.: Value of the electrostatic potential calculated for each atom of biotin in the binding pocket of streptavidin comparing the grid approach used in FlexScreen and using the Texture Memory of the GPU.

4.6. Floating point accuracy influence on different GPUs

We have also performed this study in several NVIDIA GPUs, both in simple and double precision, in order to check how the architectural design affects performance and precision. In Figure 3(B) we can observe that on average, Tesla GPU runs faster than Fermi GPU. For both cases the running times are smaller working on single than in double floating point precision, as one would expect. In the results obtained in the different GPU strategies presented, Tesla also outperforms Fermi due to the higher number of cores (448 versus 352). This is more accurate in the cases like GRID-GM where interpolation computations take a high percentage of the total running time.

5. Conclusions and outlook

In this work we have efficiently shown how the CUDA language can be used to efficiently exploit the GPU architecture and as far as we know, this is the first GPU implementation of a grid Kernel in a VS methodology. For typical and representative VS calculations of a database with 2000 ligands we have obtained speedups of up to 20x (Fermi architecture) and 30x (Tesla architecture) versus the sequential grid Kernel counterpart and we have determined which are the optimal values of the most relevant parameters like the number of threads per block, grid spacing and number of cells of the compressed ligand database atomic positions. We have also shown that the most optimal interpolation procedure tested is the linear one. Between the strengths of our implementation, we are satisfied with our in-depth study of different and alternative GPU implementation strategies, and between the weaknesses of our work we think we could have studied more different interpolation procedures.

On the application side, this GPU grid Kernel can be used in many different simulation methodologies that imply interactions between many particles; rigid-rigid systems (like the studied cases here; docking of biotion in streptavidin) or mixed rigid-flexible systems (our next step). This Kernel can also be used with sligth modifcations in another fields like astrophysics.

At the moment we are targeting our efforts to the implementation of this GPU Kernel in the VS program FlexScreen [8], where the authors have already contributed to its development, adding new functionalities related with an improved scoring function that can efficiently handle not only electrostatics but other new physical terms in the description of the protein-ligand interactions.

The source code of the program is available upon request.

Acknowledgements

This research was supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grants 00001/CS/2007 and 15290/PI/2010, and also by the Spanish MEC and European Commission FEDER under grants CSD2006-00046 and TIN2009-14475-C04 and a postdoctoral contract from the University of Murcia (30th December 2010 resolution).

References

[1] David S. Cerutti, Robert E. Duke, Thomas A Darden, and Terry P. Lybrand. Staggered Mesh Ewald: An Extension of the Smooth Particle-Mesh Ewald Method Adding Great Versatility. *Journal Of Chemical Theory And Computation*, 5(9):2322–2338, 2009.

[2] Zhi-wei Feng, Xu-hong Tian, and Shan Chang. A Parallel Molecular Docking Approach Based on Graphic Processing Unit. In *Bioinformatics and Biomedical Engineering (iCBBE), 2010 4th International Conference on*, pages 1–4, 2010.

[3] G. Giupponi, M. J. Harvey, and G De Fabritiis. The impact of accelerator processors for high-throughput molecular modeling and simulation. *Drug Discovery Today*, 13:1052–1058, 2008.

[4] Ginés Guerrero, Horacio Pérez-Sánchez, Wolfgang Wenzel, José Cecilia, and José García. Effective parallelization of non-bonded interactions kernel for virtual screening on gpus. In *5th International Conference on Practical Applications of Computational Biology; Bioinformatics (PACBB 2011)*, volume 93, pages 63–69. Springer Berlin / Heidelberg, 2011.

[5] John J Irwin and Brian K Shoichet. ZINC–a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005.

[6] WL Jorgensen. The many roles of computation in drug discovery. *Science*, 303(5665):1813–1818, 2004.

[7] S Kannan and R Ganji. Porting Autodock to CUDA. *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, 2010.

[8] Daria B Kokh and Wolfgang Wenzel. Flexible side chain models improve enrichment rates in in silico screening. *Journal of Medicinal Chemistry*, 51(19):5919–5931, oct 2008.

[9] EC Meng, BK Shoichet, and ID Kuntz. Automated Docking with Grid-Based Energy Evaluation. *Journal of Computational Chemistry*, 13(4):505–524, 1992.

[10] NVIDIA. *NVIDIA CUDA Programming Guide 4.0*. 2010.

[11] Horacio Pérez-Sánchez and Wolfgang Wenzel. Optimization methods for virtual screening on novel computational architectures. *Current Computer-Aided Drug Design*, 7(1):44–52, 2011.

[12] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2nd edition, 1992.

[13] Hailong Yang, Qiongqiong Zhou, Bo Li, Yongjian Wang, Zhongzhi Luan, Depei Qian, and Hanlu Li. GPU Acceleration of Dock6's Amber Scoring Computation. *Advances in Computational Biology*, 680:497–511, 2010.