POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

High Throughput Molecular Expansion for Extreme Scale Virtual Screening Campaign

Laurea Magistrale in High Performance Computing Engineering - Ingegneria del Calcolo ad Alte Prestazioni

Author: Giuseppe Gentile

Advisor: Prof. Gianluca Palermo

Co-advisor: Davide Gadioli, Gianmarco Accordi

Academic year: 2023-2024

1. Introduction

Throughout our history, the relentless drive to discover new treatments has driven advances in medicine. Innovations in chemistry, biology, and pharmacology have gradually refined drug discovery into a systematic process, composed of different phases. Target identification phase locate a protein linked to a disease. In lead discovery phase vast libraries of chemical compounds are screened for those capable of interacting with the target to suppress effects of the disease.
Lead discovery phase relies on trial and error, with time consuming and resource intensive experiments. Early approaches involved *in vitro* testing of a limited number of compounds. However, with the rise of biocomputational methods, researchers began to simulate small numbers of molecular interactions *in silico*. Today, high-performance computing (HPC) enables the screening of billions of molecules in days.
The screening phase requires a stable conformation of molecules, that is, one of the conformations with the lowest energy among all possible coordinate sets a molecule can adopt.
Yet, as virtual screening pipelines scale up, new, *not only computational* challenges emerge. One consists in managing the storage of stable 3D molecular conformations required for docking simulations. Conventional methods precompute and store these conformers, but as compound grow into billions, storage becomes unfeasible. Alternatively, stable 3D coordinates can be generated on the fly from compact representation like SMILES [4], which preserves all information required 1to reconstruct the molecular graph. At the same time, this strategy introduce latencies, reducing HPC's speed advantage. To address this challenge, this thesis analyze, design, implement and optimize an approach to accelerate a SMILES-to-conformation expansion application. It has been designed for integration with the LiGen virtual screening pipeline. The proposed method aims to eliminate the storage bottleneck of precomputed conformers, requiring solely SMILES, while preserving the information required for docking.
The solution begins with an in-depth analysis of the original application, identifying opportunities for parallelization and optimization. Once opportunities are targeted, we investigate the usage of parallel computing to increase the throughput of minimum energy conformers.

Three implementations are developed: two utilizing CPU parallelization and another leveraging also GPU acceleration. All these versions are optimized to fully utilize high-performance computing (HPC) infrastructures equipped with HPC-grade GPUs and many CPU cores. Before delving into the proposed method, we first describe main objective differences of the state-of-the-art in molecular conformation generation against this thesis'.

In order to find stable conformations of ligands, their energy must be modeled in some way. We describe the molecular energy in function of its atom's coordinate (PES Potential Energy Surface) via MMFF94, an empirical force field defining energy in terms of distances and angles between atoms. A local minimum on the PES is a point where the gradient is zero, so that any displacements of atoms lead to an increase in energy, representing a **stable** conformer. Finding minima on multidimensional objective functions falls into the optimization field. In this thesis the focus is on a *Quasi-Newton* algorithm that has proved to be the state of the art in PES's local minima identification. The implemented algorithm used in this thesis is the L-BFGS [3], since was originally used by LiGen (software owned by Dompè). At each iteration, the direction to follow on the PES is the product between the hessian and the gradient, ensuring the reduction of the energy. The size of the step is determined by *Wolfe conditions*. The inverse Hessian approximation is updated using the previous m iteration's differences between consecutive positions and gradient, reducing the amount of storage with respect to BFGS, that in contrast stores all iterations. Software in SOTA focus on accuracy and do not target ligands. OpenMM[2] have GPUs optimizations parallelizing big molecular systems rather than set of ligands and does not take in input SMILES but molecular coordinates. LiGen-pre is a module that implement, serially, the expansion. It uses grammars to build the molecular graph from SMILES, as shown in Figure 1; molecular graph contains information about atom's connectivity. Once the connectivity graph is built, atoms are placed in 3D space. Initial coordinates are assigned using MMFF94-guided distances and angles, starting from an atom selected by breadth-first traversal. The process iteratively push or pull pairs of

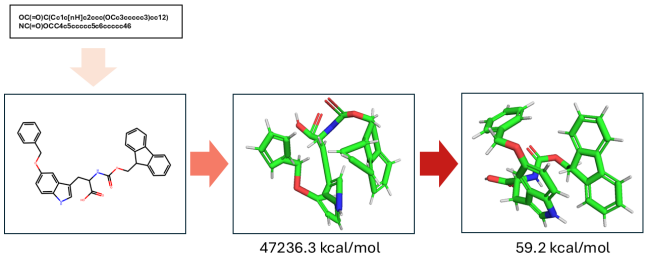

Figure 1: Ligen stages, from SMILES to minimum energy conformation.

atoms until every bound is satisfied. Using the generated coordinates, reach stable conformers by L-BFGS optimization.

After having illustrated SOTA methods and compared them to our aim, in Section 2 we explain proposed CPU parallelization, detailing the two versions and their rationale with any computational strategies employed and motivations of the chosen optimizations. Section 3 shows how the application is accelerated by offloading parts of the CPU implementation to GPU. Section 4 presents the results and performance analysis of our implementation. Finally, Section 5 concludes the thesis, offering also suggestions for future work.

2. CPU acceleration

Profiling the serial LiGen application showed that most of the time was spent on calculating energy. For this reason, a latency optimization was implemented with OpenMP, evaluating each different energy contributions in parallel. However, this strategy follows a single-molecule oriented parallelization, where threads are dynamically created and destroyed for each molecule, not exploiting the high volume of data actually used end-to-end. A change of perspective to the end-to-end computation allowed to notice that the SMILES to low-energy conformation is **embarrassingly parallel**, both from a **molecule-wise** point of view (each molecule can be processed end-to-end independently from others) and from a **stage-wise** point of view (while energy minimization of a molecule is carried out, coordinate generation of another one can be performed). We refer to a *stage* as either: the reading of the SMILES, generation of initial coordinates, generation of batch of molecules (with initial guess coordinates) or energy minimization. Both parallel perspectives are implemented us-

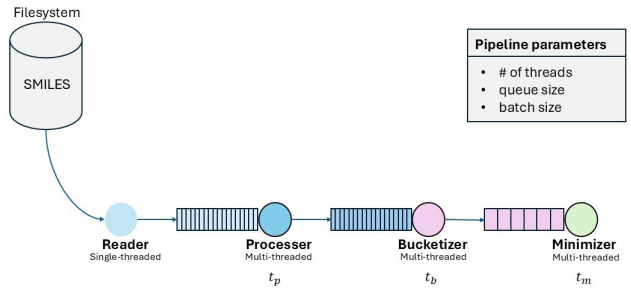

Figure 2: Ligen stages parallelized, organizing computations in pipeline, implementing parallelism both intra and inter stage.

ing *libdpipe*: a library that allow to implement pipeline stages in parallel using *stl threads* and thread safe finite queues with FIFO policy to handle content of each stages. In Figure 2, stages processes elements from its input queue, perform its computation and pushes the results into the output queue, which is handled by the next stage in the pipeline. The *bucketizer* creates batches of ligands to be minimized together. Some ligands may take longer to minimize than others. By grouping them into small batches, the workload is more evenly distributed across minimizers, reducing variability of minimization times with respect to single ligands minimization. Pipeline approach not only allows for the parallelization of each computational stage, but also provides a straight forward mechanism for bottleneck detection. When a stage's queue fills up we fall in two cases: if no other stages follow or other queues are not full, that stage is the slowest of the pipeline. Otherwise, next stages are creating a back-pressure effect that congest previous stages.
Analysis of queue's size proved the minimizer to be the slowest stage, falling in the first case explained previously, and bucketizer and processer falling in the second scenario.

3. GPU acceleration

Energy minimization resulted to be the bottleneck almost with any threads configuration, at the same time target HPC infrastructures still have GPUs available and we decided to seize this optimization opportunity. The adopted strategy consists in handling *intra-bucket* parallelization with GPUs and *inter-bucket* as showed in Section 2 with CPU acceleration. In this way ligands in bucket are minimized in parallel on the GPU, and data preparation of the bucket for GPU phase is executed in parallel for each t_m, and transfers to device are automatically handled with streams to overlap copies of bucket to device. In contrast to SOTA approaches that use many blocks for a molecule, we map a *block* for a ligand of the batch, and a *warp* per block[1], so to use warp primitives for any vector operation and exploit the fastest pipeline on GPU (registers). Batch size is proportional to the occupancy of the kernel. Any vector operation is parallelized with warp-reduce primitives, exploiting high bandwidth of registers: each thread calculate its contribute and then reduce.
Evaluating an energy contribution's interaction means calculate the energy (a scalar) and gradient $(\partial x, \partial y, \partial z)$ for atoms' coordinates involved in an interaction. To compute those two quantities, coordinates of the atoms of each interaction must be accessed, and the pattern of access influences performance. Being energy a scalar, is easily evaluated through a thread-reduce. For gradient calculation, instead, every interaction partial derivative must be written to the gradient vector in the atom indexes of the two atoms. Moreover, coordinates changes at every step of the algorithm, so they must be accessed every time the energy evaluation is performed. The order of the accesses to coordinate and gradient vector affect the caching system; poor coalesced accesses reduce performance. Coordinates and gradient order of accesses is established by the *interaction array*, that we want to organize in order to maximize coalesced accesses. Since non-bonded interactions scales to $O(\frac{N^2}{2})$, the interaction matrix of indexes can be seen as a non-full upper triangular matrix (for symmetry of interactions). Non-bonded interactions matrix in row major order, storing entries only for atom pairs separated by at least three bonds (*1-4* scaling) has variable row sizes, leading to unpredictable memory access of coordinates. Additionally, two threads may update the same gradient index if their interactions share an atom, which is frequent in row major order.
This was proven by profiling with *nsight compute*: target atom accesses exhibited a 12% inefficiency in global memory transactions due to non coalesced pattern, same percentage also for gradient accesses. Atomic operations on the gradient vector, contributed less to stalls. In

the proposed optimization, dummy interactions are created for pairs closer than 3 bonds, setting their parameters to 0, nullifying energy contribute; this allow to consider all atoms, hence a full triangular matrix. Two patterns are used to reduce memory access; the first for the atoms subset with index up to the smallest multiple of 32 of N, defined $m = 32 * \lfloor n/32 \rfloor$, and the other for leftovers atoms. In the first, every thread prefetch in registers a set of unique source and target atom's coordinate, meaning no thread has the same atom's coordinates of another one. This prefetching is performed in $m/32$ warp iteration, and is coalesced on coordinate vector because access are to (consecutive) indexes I (for source), $I+1$ (for target), where I is the $m \times m$ identity. Once all coordinates are in registers, target atom's coordinate are shared using warp primitives to next thread, that in next iteration will compute the interaction with the received target atom and its source (that is constant for every thread). This patten allow also to accumulate gradient's value for source and target atoms to registers before writing to memory, without atomic operations. Accumulated gradient value for that target atom is shared together with the atom's coordinates. With Figure 3 we represent a matrix with $m = 32$. Each warp is identified by the main color, than each thread inside the warp is identified by a its warp color, but with different nuance. The prefetching is the main diagonal. Indexes on diagonal are consecutive, and allow to recycle coordinates from previous iterations from the thread $t + 1$.

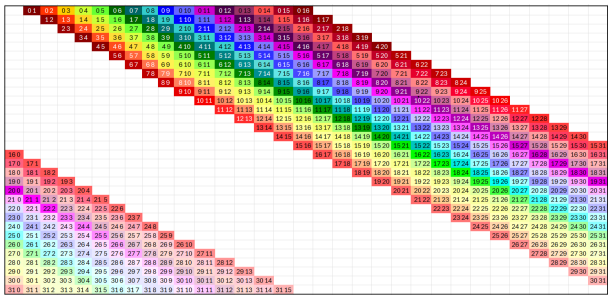

Figure 3: Non bonded interaction matrix pattern for $m = 32$.

Interactions with atoms index greater than m are instead decomposed into a rectangular matrix with atom indexes $[0, .., m] \times [m, .., n]$ and the upper triangular matrix with $[m, .., n] \times [m, .., n]$. The upper triangular matrix is handled as the naive implementation, since consists in few interactions. For each column of the rectangular matrix instead, $\frac{m}{32}$ warp iterations are performed, with threads always handling unique row indexes. In this pattern each thread has same target atom, so reduction is performed on a leader thread that writes the gradient to the target atom's index.

4. Results

The CPU-based latency optimization proved to be sub-optimal, yielding no more than a 2× speedup on the end-to-end ligands expansions of an EXSCALATE dataset, regardless of the number of allocated threads. We attribute this low advantage to two main factors: (1) the overhead from dynamically creating and destroying threads for each molecule and (2) the relatively short minimization time for individual ligands, which limits the benefits of parallel execution. The higher the volume of ligands, the more overhead in creating and destroying threads increases, so, we concluded this approach does not scale well with the number of threads and number volume of data. The final implementation aim to increase the throughput of minimized conformers through an hybrid parallelization approach targeted at different levels of granularity. Analyzing heterogeneous levels of parallelization required a careful analysis of the pipeline conducted with a subset of the EXSCALATE dataset. First, queues of the CPU only parallelization are analyzed, together with the thread allocation to assess its scalability. Pipeline's throughput is dictated by the slowest stage of the pipeline, since is the one that frees its queue more slowly than others, that ideally would go faster if the bottleneck stage was not present. Sizes of queues are measured over time using one thread for each stage. In first seconds of Figure 4 the minimizer can't stay at the pace of the guesser that produces initial guess at an high frequency: this is represented by the spike of the green line, so, the minimizer's queue aren't freed at the same pace of other stages. It then follows the spike of the orange line, that is a back-pressure effect caused by the minimizer; meaning that the bucketizer would forward buckets at frequencies that the minimizer can't sustain, so, it start to accumulate buckets in its queue too.

Figure 4: Pipeline's stages queue size measured every second.

The strategy to face the bottleneck has been to iteratively increase the number of threads for the current bottleneck stage. First, threads of the bottleneck are increased, queues are analyzed to understand if there is a new stage as bottleneck. This behavior can happen due to the increase of threads. Anytime a bottleneck is faced, this procedure is repeated. The same procedure is tested also with the GPU minimizer with an NVIDIA A100. An heatmap showing the procedure conducted is represented Figure 5; on each cell the speedup with respect to the serial version is shown.

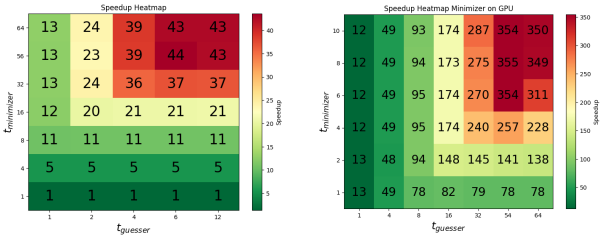

Figure 5: Fine tuning procedure on CPU version of the pipeline (left) and GPU (right)

In the first row of the CPU pipeline, even if we increase the number of guessers, the throughput is dictated by the minimizer, so, speedup is constant in that row. Increasing the number of minimizer's threads ends up reaching a plateau of speedup: in the first column there is advantage up to the $t_{minimizer} = 16$, but then we have to shift toward configurations with higher number of guesser (toward right). In the optimal configuration, all physical threads of the LIGATE machine are allocated: 56 ($minimizer$) + 6 $guesser$ + 1 ($reader$) + 1 ($bucketizer$) = 64. Creating software threads introduce context switches slightly increasing time of execution: this happen in all cell's with sum of threads higher than 64 (top-right). On the GPU version, threads that with the CPU version were allocated to the minimizer now get allocated to the guesser, thanks to the offloading on GPU of the minimization algorithm that increase the throughput of minimized ligands. Comparing the two results, the trend obtained is a rotation of configuration, visually captured by the fact that colors in CPU parallelism scaled in the direction of rows for first configurations, now they scale in the direction of column (guesser). Top-right corner indicates the optimum configuration zone, highlighting the need of shifting to at least 54 threads to overcome the bottleneck of the guesser.

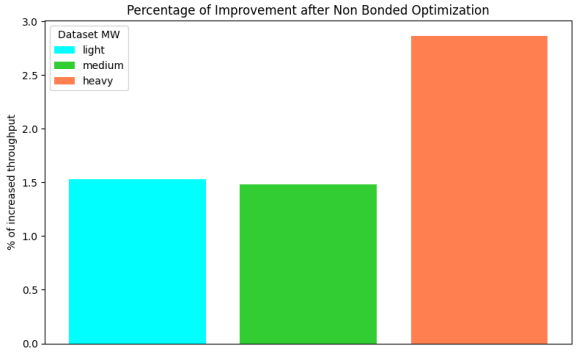

Figure 6: Percentage of increased throughput after optimizing non bonded interactions.

Then, the percentage of increased throughput of the pipelines with and without non-bonded optimization with EXSCALATE datasets with different molecular weights is provided in Figure 6. Larger ligands benefit more from non bonded optimization, having to a greater increase in throughput.

Profiling the non-optimized version with Nsight Compute, target atom accesses exhibited a 12% inefficiency in global memory transactions, as well as gradient vector. In the optimized version, target atoms are accessed from global memory only in pre-fetching phase; the percentage of inefficiency was reduced down to 0.16%. Finally, Table 1 report the end-to-end throughput (from SMILES to conformation, including I/O) of the two versions of the pipeline on the target machine.

Table 1: Throughput and speedup of the fine-tuned pipelines version on LIGATE, with EXSCALATE datasets.

Version	Light	Medium	Heavy
CPU $[kmol/s]$	2.85	1.02	0.49
GPU $[kmol/s]$	9.80	5.06	3.59
Speedup	**3.4**	**4.9**	**7.3**

Kernel minimization time for buckets of very light molecules was observed to be low, thus the overhead of launching the kernel is likely the limiting factor in terms of speedup. The latency of the kernel launches is hidden the more the kernel times increase (longer minimization for bigger ligands). The provided speedup are still restricted by the available cores of the machine. The fine tuning procedure on this machine drove us to a thread configuration where the guesser is dictating the throughput of the pipeline for some part of the computation. If we had more cores available we'd be able to make the minimizer always the one who establish the throughput, probably ending up with a bigger speedup factor.
Moreover, being the ported application a part of a bigger software, that is currently using GPU for the docking phase, by minimizing buckets of ligands on GPU we have the advantage that ligands are already ready to use in device memory.

5. Conclusions

Computational methods in drug discovery have significantly reduced costs and the time required for research. VS campaigns now enable in-silico evaluation of potential drug candidates before laboratory experiments are conducted. As these campaigns enlarge the search for effective compounds, novice challenges for storage methods are faced. Compact formats like SMILES helped to reduce storage needs compared to molecular coordinates. Yet, docking algorithms require molecules in low-energy conformations. The proposed method in this thesis accelerates the expansion from SMILES to low-energy conformations by building upon the SOTA LiGen-pre application. Analysis of LiGen-pre revealed that the energy minimization is the primary computational bottleneck. To overcome this, the workflow have been parallelized with CPU, and minimization is offloaded to GPU, ensuring scalability across hardware infrastructures. This approach increases throughput without any loss in the quality of the generated ligands. This thesis may be expanded to protein energy minimization, or in general bigger bio-molecular systems. Triangular interactions pattern could be exploited also for interactions parameter retrieval if numerical stability for dummy interactions is assessed. This would reduce global memory accesses, exploiting more registers bandwidth and potentially increase performance even further. Also, the code may be adapted to portable frameworks like SYCL or HIP.

References

[1] Gianmarco Accordi, Davide Gadioli, Emanuele Vitali, Luigi Crisci, Biagio Cosenza, Andrea Beccari, and Gianluca Palermo. Out of kernel tuning and optimizations for portable large-scale docking experiments on gpus. *The Journal of Supercomputing*, 80(3):11798–11815, 2024.

[2] Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, and et. al. Harrigan. Openmm 7: Rapid development of high-performance algorithms for molecular dynamics. *PLOS Computational Biology*, 13(7):e1005659, 2017.

[3] Dong C. Liu and Jorge Nocedal. On the limited memory broyden–fletcher–goldfarb–shanno method. *Mathematical Programming*, 45(1):503–528, 1989.

[4] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Modeling*, 28(1):31–36, 1988.