Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale,
Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

Design Considerations for Hardware Based Acceleration of Molecular Dynamics

Joseph Middleton[a], Joan Cordiner[a*]

[a] The University of Sheffield, School of Chemical, Biological and Materials Engineering, Sheffield, United Kingdom
* Corresponding Author: j.cordiner@sheffield.ac.uk

ABSTRACT

As demand for long and accurate molecular simulations increases so too does the computation demand. Beyond using new, enterprise scale processor developments - such as the ARM neoverse chips – or performing simulations leveraging Graphics Processing Unit compute, there exists a potentially faster and more power efficient option in the form of custom hardware. Using hardware description languages it is possible to transform existing algorithms into custom, high performance hardware layouts. This can lead to faster and more efficient simulations but compromises on the required development time and flexibility. In order to take the greatest advantage of the potential performance gains, the focus should be on transforming the most computationally expensive parts of the algorithms. When performing molecular dynamics simulations in a polar solvent like water, non-bonded electrostatic calculations dominate each simulation step, as the interactions between the solvent and the molecular structure are calculated. However, simply developing a non-bonded electrostatics co-processor may not be enough, as transferring data between the host program and the Field Gate Programable Arrays itself incurs a significant time delay. For any changes to be made, competitive to existing calculation solutions, the number of data transfers must be reduced. This could be achieved by simulating multiple time-steps between memory transfers, which may impact accuracy, or performing more calculations in the custom hardware.

Keywords: Modelling, Algorithms, FPGA, Molecular Dynamics, Optimisation

INTRODUCTION

Molecular dynamics is the process of simulating molecular interactions at an atomic level, this allows researchers to divine information from chemical mechanisms that cannot be easily observed or measured. Depending on the configuration, simulations can be used to determine the reaction specific parameters such as the activation and Gibbs free energy of complex or expensive experimental processes in a way that may not be obtainable in a lab. As the simulations comprise of many bound and free atoms, the workloads are a prime candidate for parallelisation as the simulation space can be divided up into smaller chunks. This allows for the simulations to be run in supercomputers, which may have thousands of nodes, rather than just a few very fast machines. The majority of the simulations are run using either Central Processing Units or using Graphics Processing Units either as co-processors to the CPU or self-contained. CPU based calculations can be offloaded across one node by running multiple threads or across many nodes by utilising libraries like OpenMP. Simulations on a GPU backend are more complex, as they require the node to communicate and synchronize data especially if the simulation is running on a single-node, multi-GPU system, or on multi-node, multi-GPU systems. This also introduces the problem of communications latency between the co-processor – in this case, the GPU – and the host which must be taken into account when running very large scale or long simulations. This communications latency can be eliminated by performing all of the calculations within the GPU, which has been developed by AMBER using the PNEND backend. Whilst coming at the cost of scalability, as this method lacks inter-node communication, calculations are performed fast enough for that to be of limited benefit.

The CPU and GPU are not the only hardware available, however, Field Programmable Gate Arrays (FPGAs) are available for commercial purchase. They allow developers to design, prototype and deploy custom hardware layouts to chips in order to perform calculations in a time defined and massively parallel way. Designing a standalone FPGA molecular dynamics solver is outside the scope of this work, and has already been achieved though it did not act as a co-processor to an existing molecular dynamics program [8, 7, 9]. The aim here is to determine, for large simulations involving the In-Vitro Transcription (IVT) process, what parts of the simulation would be best handed off to a custom hardware environment.

Molecular Dynamics

There are two varieties of Molecular modeling, molecular dynamics and quantum dynamics. Both have their distinct advantages and disadvantages. Quantum Dynamics is the most computationally demanding, with it realistically being limited to small molecules, but provides the most information and most accurate capture of real phenomena. Molecular dynamics on the other hand can be used to simulate large systems with reasonable accuracy in timescales of nano and microseconds. It achieves this by abstracting the difficult calculations performed by QM systems, into lookup tables which can be later adjusted to better replicate experimental results for the research area the force field has been tuned to. Prominent force-fields are both CHARMM-39 and AMBER, with both being tuned for bio-molecules and their associated structures [12]. The sometimes-extensive parameterisation into these force fields represents a major drawback to molecular simulations, as optimised force-fields struggle to accurately model molecular behaviour outside of the target domain [1]. Approaches exist that both attempt to mitigate this and to increase the scope of molecular simulations by allowing for non-ground state simulations, with the methods allowing for good accuracy whilst maintaining reasonable calculation speeds [2]. Many other abstractions are made to MD simulations to make them more performant, one of the most prominent is the treatment of solvents in large simulation domains. Due to the nature of a solvent, its presence within the simulation is typically the dominant force, and without further abstractions or clever modelling methods - such as periodic boundaries, the simulation of the solvent can take up a large portion of the compute time taken to simulate each timestep. In simulation domains that use Particle Mesh Ewald (PME), the number of atoms scales in line with Equation 1. Where N is a real number of elements within the simulation and where the equation models the growth in compute complexity for N.

$$Cost(N) = \mathrm{N}\log(\mathrm{N}) \tag{1}$$

Because the solvent particles are not numerous, it is necessary to simplify the structure and define the number of electrostatic interactions, with the most common water solvent model being TIP3P, which has a fixed charge and 3 polar interaction sites [4]. Even with this abstraction, when simulating large biomolecules, establishing periodic boundaries may only have a limited effect on the number of solvent molecules. An alternative is to replace the individual solvent molecules with a continuum dielectric model which is sampled to generate the electrostatic forces, These implicit solvent models can offer significant performance benefits, up to 100 times faster for specific simulations with the AMBER implementation of Generalised Born (GB) [10]. Despite the improvements in simulation speed, the scaling of strict GB computation complexity is worse than that for explicit methods, Equation 2. Where *n* is the real number of elements within the simulation solvent.

$$Cost(n) = n^2 \tag{2}$$

Despite the worse computational complexity implicit simulation methods can sample the solvent space faster. Though this faster simulation speed becomes more prominent when compared against explicit models with large solvent volumes. Using GB models also increases the design considerations of the simulation, as just as not all force-fields will produce accurate results, not all GB models are able to produce the same results in different simulation environments [3]. In addition, in order to facilitate the fast conformal sampling desired the collision frequency of the solvent is reduced, meaning that using implicit models necessitates the loss of viscosity modelling [10].

Simulation Hardware

The majority of modern simulations are no longer performed on the CPU, both because of an increase in complexity and because there are alternative calculation methods in the form of consumer and datacenter GPUs which offer significantly better parallel performance. Instead, CPUs are used to orchestrate the management of the processes, and in multi-node simulations, to handle internode communication. There are two primary methods of GPU integration, the standalone method discussed already, AMBER PNEND manages all of the simulation steps on the GPU. Alternatively, the CPU will correlate and propagate the simulation timesteps, whilst the GPU performs all of the calculations on the data. This method is beneficial as it allows for the coordination of more resources, with simulations being able to be performed over multi-nodeGPU systems, though has a significant drawback in that the data is not available instantly. The primary method for connecting CPU and GPU systems is the PCIe bus, which has a non-trivial multi mi-

crosecond latency [13], especially between complex devices. This connection lag becomes more problematic as GPUs become more powerful, essentially meaning that devices are sitting idle for longer, increasing the amount of time required to run the simulation. New hardware designs such as Heterogeneous System Architecture can be used to mitigate this latency, which allows both devices to have a unified memory pool, eliminating the need to transfer the data. Other GPU bound processes also experience this problem, and other Host-Device communication protocols exist, which have lower latencies, such as NVLink for use in SXM socket GPUs.

As might be expected from CPUs, they are designed for general purpose compute tasks, GPUs whilst being specifically tuned for graphics operations, at least originally, are capable of doing a great many things. Though when performing computational work, the specific area of interest is typically the floating point mathematical operations they can carry out in parallel. This means that only a section of each chip will be operating, whilst the entire chip is powered on and operating. This presents a problem, as the throughput of the chip is limited by the number of physical compute cores and the operating speed of the chip. Increases in the chip speed lead to higher power consumption, including powered on components, such as memory banks, that are not being used. Additionally, in order to focus on Artificial Intelligence workloads like training and inference, manufacturers are reducing the amount of single and double precision floating point computing to dedicate more chip space to half and float8 workloads, though there are ways to recover some of this loss, and it is more prevalent in consumer grade chips [5].

In cases where standalone or co-processor GPU workloads are not supported, CPUs need to be used, at a significant performance penalty. This can be mitigated by using high core count chips, such as the Neoverse series of ARM chips, which are capable of scaling to 128 individual cores and performing 16 Fused Add Multply operations per cycle [6].

Instead of taking the general purpose approach to hardware, it is possible to use FPGAs, which sit in between the general purpose hardware solutions and completely custom Application Specific Integrated Circuits (ASICs), a category which both CPUs and GPUs fall into. As discussed earlier they allow for the design and implementation of custom hardware designs, which can be entire algorithms and allow for complete flexibility in their implementation; at least within the bounds of the FPGA manufacturer. The designs are typically limited in speed to the MHz range, which when compared to modern CPU and GPUs appears to be a significant performance reduction. FGPA designs are specific to the desired task, minimising the amount of general purpose overhead the chips must perform, and the large number of onboard components allows for them to maintain many data processing pipelines, which encourages parallel operation. Having the physical program laid out in the hardware allows for data to be processed at every step along it, rather than just at the instruction pointer like in a CPU. As well as MD solvers for FGPAs as mentioned earlier, solvers have also been created for ASICs, in the form of ANTON and MDGrape [14, 15]. With the FPGA hardware solutions holding their own against contemporary GPU hardware and the ASIC solutions capable of performing multi microsecond simulations per day. However, ASIC development costs are extraordinarily high, being in the region of Millions of USD, which places them out of the scope of many developers. In addition to this cost, because of the fixed nature of the designs, hardware must be extensively tested to ensure there are no bugs, with this verification typically being performed in FPGA based hardware. Both custom hardware solutions have significant efficiency benefits over complete GPUs however, which can consume up to 500W. This can be a critical factor, especially if simulations are being performed over a significant length of time, and helps to reduce the overall cost of the simulations, as whilst both custom solutions have a higher setup cost, the running costs can be significantly less.

SIMULATION

Molecular Dynamics is a very capable tool for investing bio-molecules, as they are typically very complex and consequently reactions are not well understood. This is why simulation techniques must be implemented to investigate what cannot be observed directly. As such predicting protein physical properties and interactions with other molecules are important problems to the development of better healthcare. Whilst also being difficult to observe, some interesting but complex processes are also very expensive to investigate. One of these being mRNA, which even in research grade environments can easily consume USD$10,000 to USD$100,000 in materials over a set of experiments. mRNA investigation has gained significant traction after the pandemic, due largely to the fact that it uses the bodies own protein synthesis mechanisms to deliver the designed biological payloads, whether these be vaccines or therapeutic payloads. The process of mRNA manufacturing is an area of intense development, and in many cases determining specific reaction kinetics, such as the Activation Energy, Gibbs Free Energy and reaction coordinate diagrams would allow for the production of better higher order models. Which in turn would allow for experiments to be planned to both corroborate results and plot out the manufacturing space.

T7RNA transcriptase is a relatively large protein, and

the DNA molecular structures and mRNA chains are long, which prevents the use of tight periodic boundaries within the simulation. Not all of the reactions that involve these species are appropriate, the interaction of covalent bonds cannot be modeled using MD, and instead a mixed QM/MD approach must be used. Simulations that break ionic bonds however, are not restricted. This leads to the use of large solvent volumes which must be solved for each timestep. If a correlation between the simulation temperature and the Gibbs free energy was desired then an explicit solver would need to be used. Though in the case of the reaction coordinate diagram, a faster implicit solver could be used. Knowing that it is likely that the simulation run time will largely be dominated by electrostatic forces, the electrostatic interaction algorithm could make an impactful first target for transition to a FPGA based co-processor.

Approaching electrostatic interactions.

In order to take advantage of the hardware co-processor being a part of the system, both the general purpose electrostatic algorithm, which supports both the explicit solvent model and inter-atomic electrostatic interactions, and the implicit solvent model should be implemented. As discussed previously, the hardware designs take up physical space on the FGPA, this is in antithesis to how CPUs work, which operate on information at the instruction pointer and is not dissimilar to how GPUs are designed, though the embedded pipelines tend to be more general. In order to take the greatest advantage of the physical nature of the design, most FGPA designs operate like pipes, taking data and using the design to transform it into the desired output, which allows for more data to be processed at a time. The downside of the physical layout is that there are finite resources available onboard, and increasingly complex pipelines reduce the number that can be fit in parallel onboard, which in turn reduces the calculation throughput.

Designing

Since the idea is to use this as a co-processor to assist in calculations being performed on a host, it makes sense to convert parts of an existing simulation program to ensure maximum compatibility. This also allows for comparisons to be made between the co-processor and original code, to help eliminate any calculation drift or bugs that were introduced in the transfer. It is possible to convert the existing code almost directly into a Hardware Description Language (HDL) through a process known as High Level Synthesis (HLS). Though this approach is almost entirely automated, it can struggle to develop performant designs both due to the existing code semantics being designed for a general purpose architecture. The most performant option would be to port the algorithms by hand to HDL, which would allow for them to be tightly coupled to the physical resources onboard the FPGAs as it allows for complete control over the physical hardware implementation. This process is very labour intensive and requires all of the support infrastructure to be instantiated manually, such as the memory controllers. Though most importantly, especially so for long, mathematically intensive algorithms, it would require that all of the floating point units be designed and linked up by hand.

Mitigation

Laying out the structure of the floating point operations manually in the hardware represents a significant amount of work, not only does it involve instantiating the hardware, but also determining how the data will flow through it. In order to do this in a scalable way, a semi-supervised automation approach is suggested as it would be easier to manage than a fully automated approach that would inherit the same problems as HLS. In this semi-supervised approach, the algorithm would be broken down into a graph representation, where nodes represent individual calculation steps, and the connections between them the flow of the data.

Once the pipeline is in this format, the physical layout can be organised by a series of optimisation algorithms, which determine where the most low-latency operations would be required, and where calculations are able to take longer to resolve. Once the fast and slow data paths have been determined, the configuration of the nodes can be transformed into instantiated floating point units, before being placed onto the FPGA fabric. This approach allows for designs to be optimised by configurable constraints, which supports flexible designs without having to rewrite large sections of the hardware design.

CONCLUSION

Both FPGA and ASIC solutions are very competitive against modern hardware when running in stand-alone configurations, and will likely compare well in comparison to co-processor designs as they are all equally limited by the latency of the PCIe bus. Even outside of a speed comparison, FPGA solutions are more energy efficient than the alternatives making them a compelling choice. Though compared to stand-alone solutions an electrostatic co-processor is limited, it could potentially be beneficial for large solvent volumes within transcription simulations, and further developing a semi-supervised process for designing floating point algorithms would be helpful when expanding the design to become a self-sufficient solution.

ACKNOWLEDGEMENTS

I would like to acknowledge Crina-Maria Pricop for helping me write this paper, her patience was enormously

helpful. I would like to thank Joan Cordiner for being an excellent supervisor.

REFERENCES

1. V. Zapletal et al., “Choice of Force Field for Proteins Containing Structured and Intrinsically Disordered Regions,” Biophysical Journal, vol. 118, no. 7, pp. 1621–1633, Feb. 2020, doi: https://doi.org/10.1016/j.bpj.2020.02.019.
2. A. Warshel and M. Levitt, “Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme,” Journal of Molecular Biology, vol. 103, no. 2, pp. 227–249, May 1976, doi: https://doi.org/10.1016/0022-2836(76)90311-9.
3. A. V. Onufriev and D. A. Case, “Generalized Born Implicit Solvent Models for Biomolecules,” Annual Review of Biophysics, vol. 48, no. 1, pp. 275–296, May 2019, doi: https://doi.org/10.1146/annurev-biophys-052118-115325.
4. C. Vega and E. de Miguel, “Surface tension of the most popular models of water by using the test-area simulation method,” The Journal of Chemical Physics, vol. 126, no. 15, p. 154707, Apr. 2007, doi: https://doi.org/10.1063/1.2715577.
5. Hiroyuki Ootomo and R. Yokota, “Recovering single precision accuracy from Tensor Cores while surpassing the FP32 theoretical peak performance,” International Journal of High Performance Computing Applications, vol. 36, no. 4, pp. 475–491, Jun. 2022, doi: https://doi.org/10.1177/10943420221090256.
6. ARM Holdings, “Neoverse V1 introduction,” Jun. 2021. Accessed: Jan. 04, 2025. [Online]. Available: https://teratec.eu/library/pdf/forum/2021/A05-03.pdf
7. Brahim Betkaoui, D. B. Thomas, and W. Luk, “Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing,” Dec. 2010, doi: https://doi.org/10.1109/fpt.2010.5681761.
8. C. Yang et al., “Fully integrated FPGA molecular dynamics simulations,” arXiv (Cornell University), Nov. 2019, doi: https://doi.org/10.1145/3295500.3356179.
9. P. Hamm, “Toward an FPGA-based dedicated computer for molecular dynamics simulations,” The Journal of Chemical Physics, vol. 162, no. 5, Feb. 2025, doi: https://doi.org/10.1063/5.0248834.
10. R. Anandakrishnan, A. Drozdetski, Ross C. Walker, and Alexey V. Onufriev, “Speed of Conformational Change: Comparing Explicit and Implicit Solvent Molecular Dynamics Simulations,” Biophysical Journal, vol. 108, no. 5, pp. 1153–1164, Mar. 2015, doi: https://doi.org/10.1016/j.bpj.2014.12.047.
11. Wilfred and A. E. Mark, “Validation of molecular dynamics simulation,” vol. 108, no. 15, pp. 6109–6116, Apr. 1998, doi: https://doi.org/10.1063/1.476021.
12. K. Vanommeslaeghe et al., “CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields,” Journal of Computational Chemistry, vol. 31, no. 4, pp. 671–690, Mar. 2010, doi: https://doi.org/10.1002/jcc.21367.
13. N. Zilberman *et al.*, “Where Has My Time Gone?” Accessed: Mar. 14, 2025. [Online]. Available: https://api.repository.cam.ac.uk/server/api/core/bitstreams/b0e91ca7-9b3b-4f31-b1d8-05c6668c8a66/content
14. D. E. Shaw *et al.*, “Anton, a special-purpose machine for molecular dynamics simulation,” *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 1–12, Jun. 2007, doi: https://doi.org/10.1145/1273440.1250664.
15. Itta Ohmura, G. Morimoto, Y. Ohno, A. Hasegawa, and Makoto Taiji, “MDGRAPE-4: a special-purpose computer system for molecular dynamics simulations,” vol. 372, no. 2021, pp. 20130387–20130387, Aug. 2014, doi: https://doi.org/10.1098/rsta.2013.0387.