# An FPGA Accelerator for Molecular Dynamics Simulation Using OpenCL

**Hasitha Muthumala Waidyasooriya [1], Masanori Hariyama [1], Kota Kasahara [2]**

[1] *Graduate School of Information Sciences, Tohoku University, Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi 980-8579, Japan*

*E-mail: {hasitha, hariyama}@tohoku.ac.jp*

[2] *College of Life Sciences, Ritsumeikan University, 1-1-1, Noji-higashi, Kusatsu, Shiga 525-8577, Japan*

*E-mail: ktkshr@fc.ritsumei.ac.jp*

**Abstract**

Molecular dynamics (MD) simulations are very important to study physical properties of the atoms and molecules. However, a huge amount of processing time is required to simulate a few nano-seconds of an actual experiment. Although the hardware acceleration using FPGAs provides promising results, huge design time and hardware design skills are required to implement an accelerator successfully. In this paper, we use a heterogeneous computing system for MD simulations, that can be used in C-based programming environment. We propose an FPGA accelerator designed using C-based OpenCL for the heterogeneous environment. We achieved over 4.6 times of speed-up compared to CPU-based processing, by using only 36% of the Stratix V FPGA resources. We also evaluate the processing times of different tasks in the heterogeneous environment.

*Keywords:* OpenCL for FPGA, molecular dynamics simulation, hardware acceleration, scientific computing.

## 1. Introduction

Molecular dynamics (MD) simulations [1] are very important in the fields of computational chemistry [2], materials science [3], bio-informatics [4], etc to study the physical properties of atoms and molecules. In MD simulations, classical physics is used to compute the movements of atoms and molecules. Currently, there are many publicly available and widely used software packages for MD simulations such as AMBER [5], Desmond [6], GROMACS [7], LAMMPS [8], CHARMM [9], etc. All of those methods are based on an iterative computation method, where the computation results of one iteration are used as the inputs in the next iteration. Each iteration consists of two major phases: force computation and motion update as shown in Fig.1. MD simulations require millions of iterations and a huge amount of processing time on general purpose CPUs to simulate few nanoseconds of the real time. Months to years of processing time is spend to find at least some useful results while simulating an actual laboratory experiment is not possible even today.

Hardware acceleration is already used to reduce the huge processing time in MD simulations. ASIC (application specific integrated circuit) implementations of MD simulation are already proposed in Refs. 10–12. However, designing such special purpose processors requires years of design, debugging and testing time and also involves a huge financial cost. Therefore, ASICs are out-of-reach for most researchers, although their performances are quite excellent. A cheap way of hardware acceleration is provided by FPGAs (feild-programmable-gate-arrays) [13,14,15]. Although the cost is extremely small compared to ASICs, the design time is still very large. FPGAs are designed using hardware description language (HDL) so that hardware design skills and experiences are required for a successful FPGA implementation. When there are algorithm changes and hardware updates, it is often required to redesign the whole FPGA architecture.

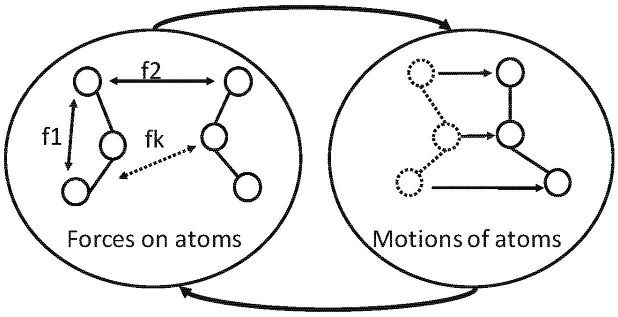


Fig. 1. Molecular dynamics simulation model. Force computation and motion update of the atoms are repeated for millions of iterations.

To overcome these problems, OpenCL for FPGA has been introduced [16]. It is a complete framework that includes firmware, software and device drivers to connect, control and transfer data to and from the FPGA. It provides a heterogeneous system consist of a host CPU and a device which is an OpenCL capable FPGA. Lightweight tasks can be processed on the host CPU while the heavyweight tasks can be offloaded to the FPGA. The host program is written in C code and the device program is written in OpenCL code [17] which is also similar to C code. FPGA implementation can be done entirely using software without requiring a single line of HDL code. Recently, some works such as Refs. 18, 19 propose FPGA accelerators using OpenCL.

This paper is an extension of our previous work [20] that describe the basic FPGA accelerator for MD simulations using OpenCL. The FPGA design time has been reduced to just few hours due to software based design. Any algorithmic change could be easily implemented by just changing the software code, and the same code can be re-used in any OpenCL capable FPGA board. In this paper, we discuss the heterogeneous processing on the host and the FPGA in depth. We also evaluate the processing and the data transfer times. In this paper, we demonstrate that it is possible to achieve over 4.6 times of speed-up compared to CPU implementation for the most time consuming non-bonded force computations. This speed-up is achieved by using only 36 % of the FPGA resources. If we assume a 80% resource usage, we can achieve a similar speed-up compared to custom FPGA accelerators designed using HDL. We also highlight the problems of the FPGA-based heterogeneous systems such as data transfers between the host and the device and shows some insights to tackle those problems in future OpenCL capable FPGA-based systems.

## 2. Molecular dynamics simulation

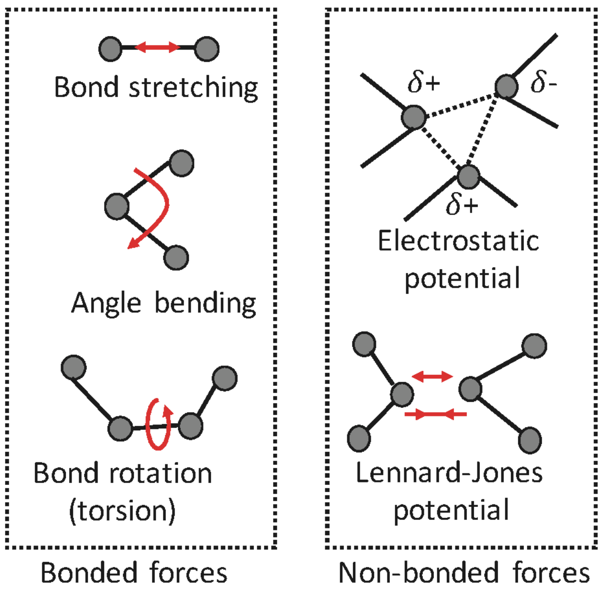


Fig. 2. Bonded and non-bonded forces consider in MD simulations.

The atoms in a system have many interaction among each other. Those can be classified in to bonded and non-bonded interactions. The bonded interactions are the acts between atoms that are linked by covalent bonds. As shown in Fig.2, stretching, angle, torsion, etc are due to bonded-forces among atoms. Bonded forces only affect a few neighboring atoms, and can be computed in $O(N)$ time for $N$ atoms. Non-bonded interactions are the acts between atoms which are not linked by covalent bonds. These are caused by the electrostatic potential, Lennard-Jones potential due to van der waals forces, etc. The forces that cause such interactions are called non-bonded forces. Those forces exist among all atoms so that the computation requires $O(N^2)$ processing time. There are several techniques available to reduce the computation cost and to accelerate non-bonded force computation.

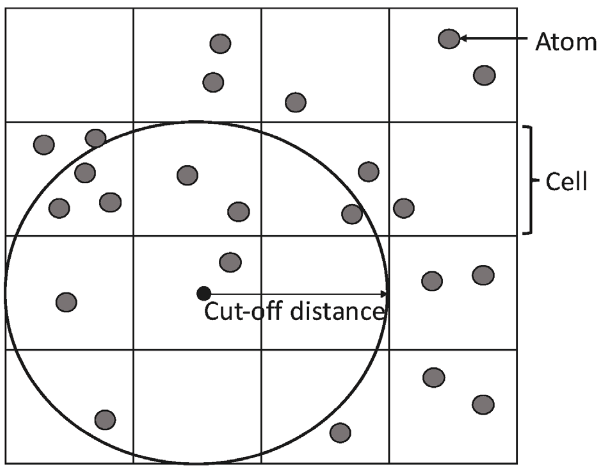


Fig. 3. Division of the simulation box in to cells.

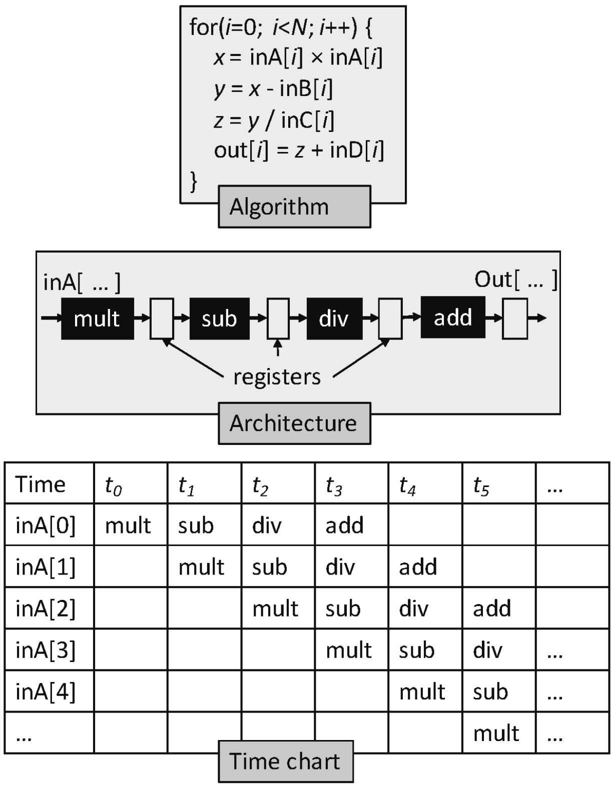
MD simulation is done for a system that is usually represented by a box of atoms. To reduce the computation complexity, the box is divided in to multiple cells. Fig.3 shows a 2-D representation of the cell division. A cut-off distance is set between two atoms and the neighboring cell-pairs within the cut-off distance are extracted to a cell-pair list. Non-bonded force computation is done for the atoms of the cell-pairs in the list. As a result, we do not have to consider all atom-pair combinations for the force computation. Since the atoms move in the box, the cell-pair list is updated in each iteration. A periodic boundary condition is used when an atom leaves the box. We assume that the same box is replicated at the boundaries so that an atom leaves from the box reappears from the opposite direction. Using this method, we can simulate a large system by using only a small number of atoms.

Even with these techniques, MD simulation takes a huge amount of processing time. Non-bonded force computation occupies most of the total processing time. Therefore, we accelerate the Non-bonded force computation using FPGA. The FPGA acceleration is based on the MD simulation software "myPresto/omegagene" [21,22].

## 3. MD simulation accelerator architecture using OpenCL

### *3.1. Parallel processing using loop-pipelining*

OpenCL for FPGA uses pipelines on FPGA to implement parallel computations. Fig.4 shows a typical pipeline that contains four operations, multiplcation, subtraction, division and addition. Multiple functional units are used to implement different operations. Registers are placed in between two functional units to temporally hold the intermediate results. A pipeline stage is assigned for the operations processed between two adjacent registers. In each clock cycle, data proceeds from one stage to another. The data set in Fig.4 contains $N$ data values. In each clock cycle, data are read in serial manner. For example, in clock cycle $t_0$, data $inA[0]$ is read and the multiplication operation is done. In clock cycle $t_1$, subtraction corresponds to $inA[0]$ and multiplication corresponds to data $inA[1]$ are done simultaneously as shown in the time chart. Similarly, in clock cycles $t_3, t_4, ...$, all four operations are done simultaneously for different data. At this time, we call the pipeline is fully filled. In this computing method, parallel processing is done in multiple pipeline stages, even the data are read in serial. The clock period of the pipeline is decided by the slowest stage, which is called the bottleneck stage. If we divide the slowest stage in to multiple pipeline stages that have smaller processing times, we can increase the clock frequency.



```
for(i=0; i<N; i++) {
    x = inA[i] × inA[i]
    y = x - inB[i]
    z = y / inC[i]
    out[i] = z + inD[i]
}
```

| Time | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | ... |
|---|---|---|---|---|---|---|---|
| inA[0] | mult | sub | div | add | | | |
| inA[1] | | mult | sub | div | add | | |
| inA[2] | | | mult | sub | div | add | |
| inA[3] | | | | mult | sub | div | ... |
| inA[4] | | | | | mult | sub | ... |
| ... | | | | | | mult | ... |

Fig. 4. Parallel processing using pipelines. Multiple operations are done on different data simultaneously.

The Altera offline compiler (AOC) reads an OpenCL code and implements pipelines for the computations inside loops. This is called "loop-pipelining". A pipeline stage is designed for each operation in the OpenCL code. In fact, even a single operation such as 32-bit multiplication is divided into several pipeline stages to reduce the processing time of a stage. As a result, we can achieve high frequencies that are over 200MHz. Since FPGAs have a large number of registers, pipelines with thousands of stages can be designed easily.

When there are no data dependencies between nested loops and if AOC identifies such situations, AOC automatically generates a fully pipelined architecture. Otherwise, a separate pipeline is implemented for the inner-loop. Therefore, an iteration of the outer-loop proceeds only after all the iterations of the inner-loop are finished. The computation of the outer-loop stalls until the inner-loop finishes its execution. To avoid this, we can fully unroll the inner-loops. However, fully unrolled loops process all its iterations in parallel, so that a lot of hardware resources are required to implement such processing.

## *3.2. FPGA architecture using OpenCL*

```
foreach cell-pair in the cell-pair list do {
   CELL1 = cell-pair -> cell1
   CELL2 = cell-pair -> cell2
   foreach atom in CELL1 do {
      check condition 1
      check condition 2
      ...
      foreach atom in CELL2 do {
        check condition 1
        check condition 2
        ...
        calculate force
      }
   }
}
```

Fig. 5. Algorithm of the non-bonded force computation method using cell-pair list.

As explained in section 2, non-bonded force computation is the hardest and the most time consuming part of the MD simulation. Therefore, we accelerate this computation using FPGAs. The software programs [22] use cell-pair list to reduce the computation amount. However, this process involves many conditional checking and nested loops. The outline of the cell-pair list based computation algorithm is shown in Fig.5. It consists of three loops. The outer-loop proceeds for each cell-pair in the list. In the inner-loops, two atoms from each cell in the cell-pair are selected to compute non-bonded forces. Many conditional branches are used to reduce the computation amount [21]. The loop boundaries of the inner loops are depend on the number of atoms in a cell. Since different cells contains different number of atoms, the loop boundaries vary with cells. Moreover, Since the atoms moves in each it-

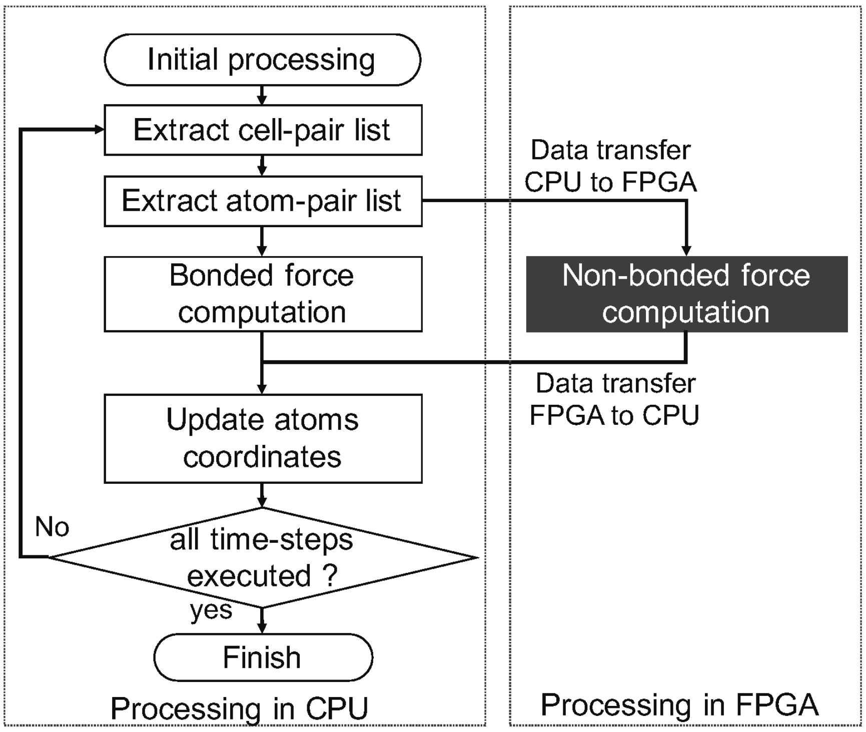


Fig. 6. Flow-chart of the CPU-FPGA heterogeneous processing.

eration, the atoms in a cell also changes. Therefore, the loops boundaries are not fixed and those are data dependent.

Due to such data dependencies, AOC cannot generate a fully pipelined architecture for this algorithm. Moreover, we cannot manually unroll the inner-loops since the loop boundaries are not fixed. Therefore, this algorithm is not suitable for OpenCL implementation due to the stalls in the outer-loops as explained in section 3.1.

To solve this problem, we separate the force computation from the atom-pair selection. We first extract the complete list of atom-pairs based on the cell-pair list. Then we perform the force computation for each atom-pair in the list. The atom-pair-list extraction is just a searching procedure that does not contain heavy computations. On the other hand, force computation contains many multiplications and divisions. Therefore, we use the host computer for atom-pair list extraction and transfer the list to the FPGA for force computation. Once the list is extracted, only a single loop is sufficient for the force computation of all the atom-pairs in the list. As a result, AOC can implement loop-pipeling on FPGA to accelerate the computation.

Fig.6 shows the flow-chart of the proposed CPU-FPGA heterogeneous processing. Bonded-force computation is done on CPU while non-bonded force computation is done on FPGA. After computing all the forces, the atom coordinates are updated using the Newton's equations. Then a new atom-pair-list is extracted. In this method, we get two overheads, atom-pair-list data transfer to FPGA and force data transfer from FPGA. We will further discuss this problem in the evaluation and suggest some solutions.

Fig.7 shows the processing done in the FPGA accelerator. The FPGA accelerator access the atom-pair list in the global memory and read atom-pairs one-by-one in serial manner. For each atom-pair, their distance is calculated and compared with the cut-off distance. If it is larger than the cut-off distance, further computations are stopped and a new atom-pair is read. Otherwise, potential and force computations are done. The outputs are the force data.

Fig.8 shows the proposed CPU-FPGA heterogeneous processing system for molecular dynamic simulations. The FPGA board is connected to the CPU through a PCI express bus. Initially, the atom-pair list and the atom coordinates are transferred from the host computer to the global memory (DRAM) of the FPGA board. After the computations are done on the FPGA board, force data are read by the host computer. This data transfer is done through the PCIe port of the host computer motherboard. The FPGA accelerator read the input data from the global memory and performs the computation. The outputs are written back to the global memory. The data read, computation and write-back is fully pipelined, so that force data are written to the global memory in every clock cycle after the pipeline is filled.

Fig.9 shows the accelerator architecture expected to be generated by AOC. It contains several computation modules for force and distance computations. The distance between the two atoms in the atom-pair is calculated using $x, y, z$ coordinates of the atoms. This computation requires several subtraction, multiplication and addition operations and one squire-root operation. The Lennard-Jones (L-J) potential computation requires divisions, additions and many multiplications [23]. Similarly, electrostatic potential computation requires divisions, additions, subtractions and multiplications [21]. After the potentials are computed, non-bonded force computation is done and the output force data are written to the global memory. The total computation requires, many additions, subtractions, divisions, multiplications and square-root operations. AOC generates pipelines stages for all operations. Therefore, even we process atom-pairs in serial manner, the pipelined architecture allows many parallel operations for different atom-pairs at the same time. As a result, we can achieve a considerably large processing speed.

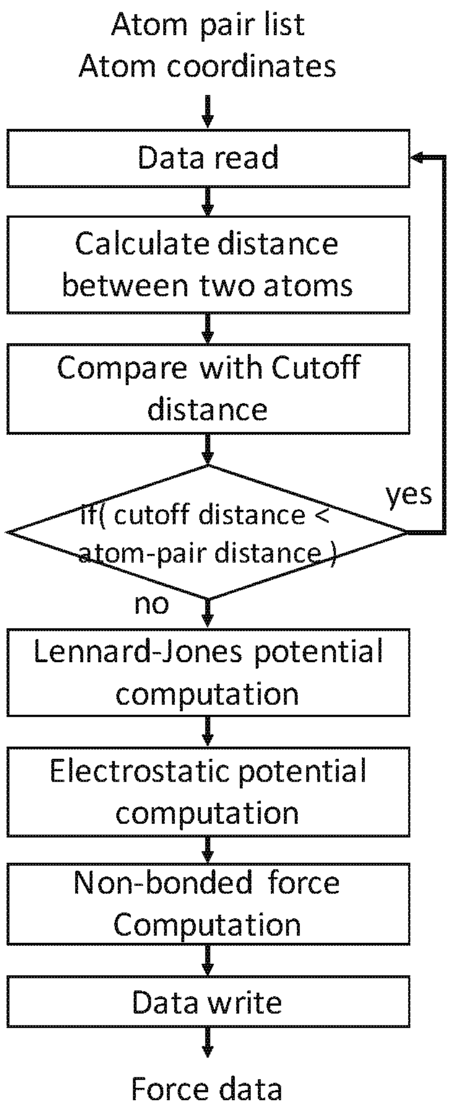

Atom pair list
Atom coordinates
Data read
Calculate distance between two atoms
Compare with Cutoff distance
if( cutoff distance < atom-pair distance )
yes
no
Lennard-Jones potential computation
Electrostatic potential computation
Non-bonded force Computation
Data write
Force data

Fig. 7. Flow-chart of the processing in FPGA accelerator.

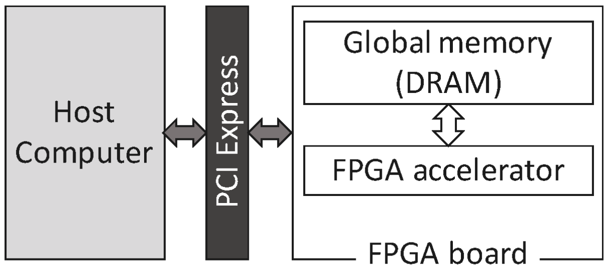


Fig. 8. Diagram of the CPU-FPGA heterogeneous processing system for molecular dynamic simulations.

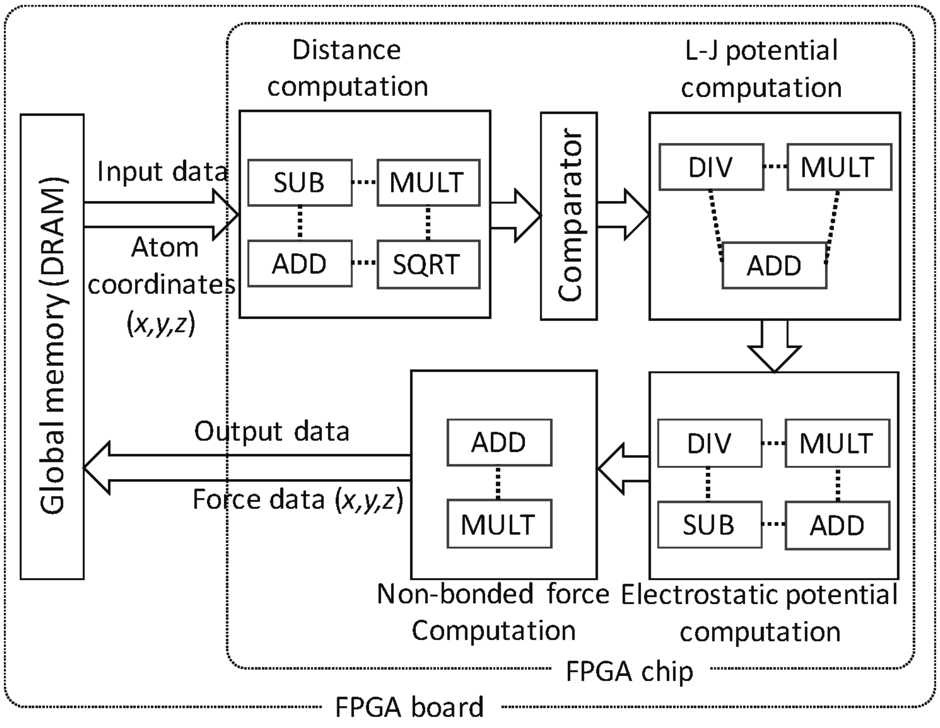


Fig. 9. FPGA accelerator architecture.

## 4. Evaluation

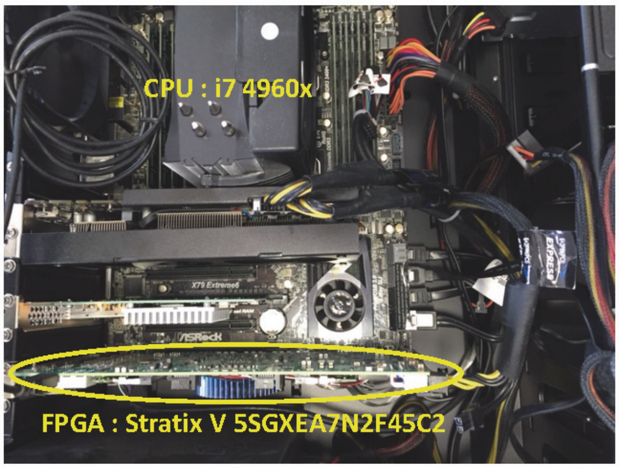


Fig. 10. The implemented heterogeneous system with a CPU and an FPGA.

For the evaluation, we used DE5 board that contains Stratix V 5SGXEA7N2F45C2 FPGA. Fig.10 shows a picture of this heterogeneous system. The operating system is CentOS 6.7. The FPGA is configured using Quartus II 15.0 with OpenCL SDK. The molecular dynamics simulation contains 22,795 atoms. The width, height and the depth of the box are $61.24 \times 10^{-10}$ m each.

Table 1. FPGA resource usage.

| Resource | Usage | Percentage used (%) |
|---|---|---|
| Logic (ALMs) | 83,653 | 35.64 |
| Registers | 91,682 | 9.76 |
| Memory (Mbits) | 3.35 | 6.70 |
| DSPs | 49 | 19.14 |

Table 1 shows the FPGA resource utilization details. The most used resource is ALMs (adaptive logic modules) and 36% of the total ALMs are used for the implementation. From those resources, around 19% of the resources are used for the I/O implementations such as PCIe and memory controllers, etc. As a result, we can increase the parallel processing by 4 times using 80% of the FPGA resources. The measured clock frequency of the accelerator is 202 MHz.

Table 2 shows the processing times of one iteration on CPU and FPGA. Note that, in this evaluation, we implemented the force computation method shown in Fig.5 on FPGA using OpenCL, without applying any algorithmic changes. That means, both the CPU and the FPGA execute almost the same code. According to the CPU implementation results, the non-bonded force computation takes 79% of the total CPU processing time. This shows, we can reduce most of the processing time by accelerating the force computation. However, using the same code on FPGA does not give any acceleration as demonstrated by the processing time results on FPGA. The processing time is more than 129 times larger compared to that in CPU.

Table 2. Comparison of the processing time using the straight forward method by using the same software code used in CPU implementation.

| | **CPU** | **FPGA** |
|---|---|---|
| Non-bonded force computation | 0.68 s | 88.03 s |
| Total computation | 0.86 s | 88.24 s |

Table 3 shows the processing time comparison when the proposed atom-pair list based method is implemented on FPGA. We achieved a speed-up of 4.6 times compared to CPU implementation. The work in [15] reports 11.1 times speed-up using a single FPGA, while the work in [24] reports over 13 times speed-up using multiple FPGAs. However, direct comparison is difficult since both CPUs and FPGAs of the previous works are quite different from what we have used. In our implementation, we used only 36% of the FPGA resources. If we used upto 80% of the FPGA resources, we can theoretically increase the speed-up to 18.4 % if memory bandwidth permits. This evaluation shows that a considerable speed-up can be achieved using OpenCL implementation.

Table 4 shows the processing times of different tasks in the heterogeneous computing system. The non-bonded force computation takes the same time as the bonded force computation since it is done in the FPGA. Moreover, both computations can be done in parallel in CPU and FPGA. Therefore, we can reduce some of the processing time by overlapping the computation in FPGA with that in CPU. However, the data transfer between the CPU and FPGA is still a problem. The atom-pair list and the coordinate data are transferred from the CPU to the FPGA. Theoretically, there are $O(N^2)$ of atom-pairs. However, atom numbers are integers and only two bytes are required to represent those. Moreover, the list can be compressed by removing redundant data. For example, three atom-pairs [atom1-atom2], [atom1-atom3] and [atom1-atom4] are represented as [atom1:atom2,atom3,atom4] by removing the redundant atom1. The number of atom coordinates in $x,y,z$ directions are only three times of the number of atoms $O(N)$. Therefore, data transfer from CPU to FPGA is relatively fast. However, the data transfer from the FPGA to CPU accounts for almost half of the total processing time. This data transfer contains the force data in $x,y,z$ directions. Since force is calculated for all atom pairs, the number of data are $O(N^2)$. Moreover, four byte floating-point data are used to represent each force value, so that data compression is difficult. As a result, this data transfer takes most of the processing time. Overall, data transfers take more than 62% of the total processing time.

Table 3. Comparison of the processing time using proposed atom-pair list based implementation.

| | **CPU** | **FPGA** | **Speed-up** |
|---|---|---|---|
| Measured results | 0.68 s | 0.14 s | 4.6 |
| Estimation based on 80% resource usage | 0.68 s | 0.037 s | 18.4 (maximum) |

This problem can be solved by using SoC (system-on-chip) based OpenCL capable FPGAs, which will be released in near future. Such a system contains a multicore CPU and an FPGA on the same chip. Since both the host and the device are on the same chip, PCI express based data transfers are no longer required. We can use on-board data transfers which are much faster. If the shared memory is utilized instead of the global memory to store the intermediate results, we can completely eliminate the data transfers.

Table 4. Total processing time per one iteration of the heterogeneous system.

| Task | Time (s) |
|---|---|
| Non-bonded force computation in FPGA | 0.14 |
| Bonded force computation in CPU | 0.14 |
| Data transfer: CPU to FPGA | 0.07 |
| Data transfer: FPGA to CPU | 0.25 |
| Total processing time (simultaneous processing on CPU and FPGA) | 0.51 |

## 5. Conclusion

We propose an FPGA Accelerator for MD simulations using OpenCL. We use an atom-pair list based algorithm that requires a single loop to represent the force computation in OpenCL. This allows AOC (Altera offline compiler) to automatically generate an efficient pipelined architecture. We achieved over 4.6 times speed-up compared to CPU implementation by using only 36% of the FPGA resources. Maximum of 18.4 times speed-up is possible by assuming an 80% resource utilization. Such a performance is similar to an HDL-designed custom accelerator.

Since the proposed architecture is completely designed by software, the same program code can be reused by recompiling it for any OpenCL capable FPGA board. We can also implement any future algorithm change by just updating the software and recompiling it by using just few hours of design time. However, the data transfers between CPU and FPGA is still a problem. This problem can be solved by future SoC based FPGA boards that contain a multicore CPU and an FPGA on he same chip. Therefore, PCI express based data transfers can be replaced by much faster on-board data transfers. We may also able to use shared memory to completely eliminate data transfers.

The heterogeneous computing system we used can contain multiple FPGAs and we can connect many such systems to build a computing cluster. Such would be our future works to increase the processing speed further.

## Acknowledgments

This work is supported by MEXT KAKENHI Grant Number 15K15958.

## References

1. D. C. Rapaport, *The art of molecular dynamics simulation*, (Cambridge university press, 2004).
2. F. Jensen, *Introduction to computational chemistry*, (John Wiley & Sons, 2013).
3. C. Z. Wang, and K. M. Ho. Material simulations with tight-binding molecular dynamics, *Journal of phase equilibria*, **18** (6) (1997) pp. 516–529.
4. V. Daggett, Protein folding-simulation, *Chemical reviews* **106** (5) (2006) pp. 1898–1916.
5. D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, The amber biomolecular simulation programs, *Journal of computational chemistry*, **26** (16) (2005) pp. 1668–1688.
6. K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, et al., Scalable algorithms for molecular dynamics simulations on commodity clusters, in *Proc. ACM/IEEE SC Conference*, (2006), pp. 43–43.
7. S. Pronk, S.Pall, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M.R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, et al., Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit, *Bioinformatics* **29** (7) (2013) pp. 845–854.
8. S. Plimpton, P. Crozier, and A. Thompson, LAMMPS-large-scale atomic/molecular massively parallel simulator, *Sandia National Laboratories* **18** (2007).
9. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, CHARMM: a program for macromolecular energy, minimization, and dynamics calculations, *Journal of computational chemistry*, **4** (2), pp. 187–217.
10. T. Narumi, Y. Ohno, N. Okimoto, A. Suenaga, R. Yanai, and M. Taiji, A high-speed special-purpose computer for molecular dynamics simulations: MDGRAPE-3, in *NIC Workshop*, (2006), **34** pp. 29–36.
11. D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, et al., Anton, a special-purpose machine for molecular dynamics simulation, *Communications of the ACM*, **51**, (7), (2008) pp. 91–97.
12. D. E. Shaw, J. Grossman, J.A. Bank, B. Batson, J.A. Butts, J.C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton, et al., Anton 2: raising the bar for performance and programmability in a special-

purpose molecular dynamics supercomputer, in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, (IEEE, 2014), pp. 41–53.

13. E. Cho, A. G. Bourgeois, and F. Tan, An FPGA design to achieve fast and accurate results for molecular dynamics simulations, in *Parallel and Distributed Processing and Applications*, (Springer, 2007), pp. 256–267.
14. M.Chiu and, M. C. Herbordt, Molecular dynamics simulationson high-performance reconfigurable computing systems, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, **3** (4) (2010), pp. 23:1–23:37.
15. M. A. Khan, *Scalable molecular dynamics simulation using FPGAs and multicore processors*. (PhD thesis, Boston University College of Engineering, 2013).
16. Altera SDK for OpenCL. *https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html*, (2016).
17. The open standard for parallel programming of heterogeneous systems. *https://www.khronos.org/opencl/*, (2015).
18. S. Tatsumi, M. Hariyama, M. Miura, K. Ito, and T. Aoki, OpenCL-based design of an FPGA accelerator for phase-based correspondence matching, in *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, (USA, Las-Vegas, 2015), pp. 90–95.
19. N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, Throughput-optimized OpenCL-based FPGA accel- erator for large-scale convolutional neural networks, in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'16*, (2016), pp. 16–25.
20. Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Kota Kasahara, Architecture of an FPGA Accelerator for Molecular Dynamics Simulation Using OpenCL, in *Proc. 15th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2016)*, (Japan, Okayama, 2016), pp. 115–119.
21. T. Mashimo, Y. Fukunishi, N. Kamiya, Y. Takano, I. Fukuda, and H. Nakamura, Molecular dynamics simulations accelerated by GPU for biological macromolecules with a non-Ewald scheme for electrostatic interactions, *Journal of chemical theory and computation*, **9** (12) (2013), pp. 5599–5609.
22. mypresto, *http://presto.protein.osaka-u.ac.jp/myPresto4/index.php?lang=en*, (2015).
23. Alan Hinchliffe, *Molecular modelling for beginners*, John Wiley & Sons, 2005.
24. S.Kasap and K.Benkrid, Parallel processor design and implementation for molecular dynamics simulations on a FPGA-based supercomputer, *Journal of Computers*, **7** (6) (2012), pp. 1312–1328.