Information Engineering Express
International Institute of Applied Informatics
2017, Vol.3, No.2, P.11 - 23

# OpenCL-Based Implementation of an FPGA Accelerator for Molecular Dynamics Simulation

Hasitha Muthumala Waidyasooriya,
Masanori Hariyama [*], Kota Kasahara [†]

## 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 propose an OpenCL-based heterogeneous computing system with an FPGA accelerator. OpenCL is a c-like programming environment to design FPGA accelerators. We achieved over 4.9 times speed-up compared to CPU-based processing, by using only 16% of the Arria 10 FPGA resources. The speed-up is limited by the memory access bandwidth. It is possible to achieve 24 times speed-up by using an FPGA board with over 50 GBps bandwidth.

*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.

[*] Tohoku University, Miyagi, Japan
[†] Ritsumeikan University, Shiga, Japan

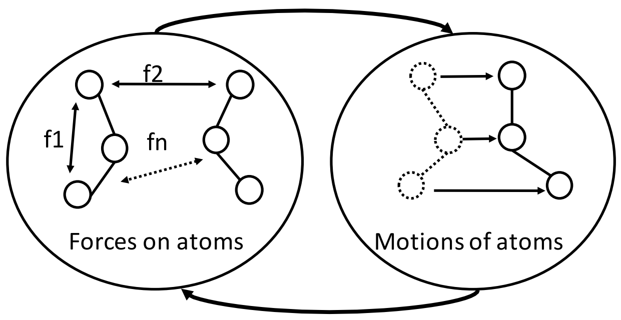

Figure 1: Molecular dynamics simulation model. Force computation and motion update of the atoms are repeated for millions of iterations.

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, 11, 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.

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 [18, 19] propose FPGA accelerators using OpenCL.

This paper is an extension of our previous works in [20, 21] that describe the basic FPGA accelerator for MD simulations using OpenCL. In this paper, we discuss the OpenCL-based implementation in detail. We also discuss how to optimize the memory access and compare the performance of different implementations. A detailed evaluation is done by using a recently introduced Arria 10 FPGA. According to the evaluation, we achieved 4.9 times speed-up compared to CPU implementation, for the most time consuming non-bonded force computation task. If we assume a 80% resource usage, we can achieve better performance compared to previous works that implement HDL-based FPGA accelerators.

## 2 Molecular dynamics simulation

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

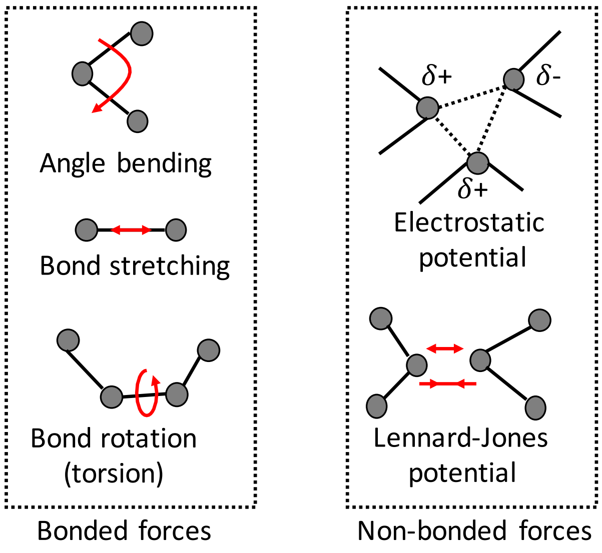

Figure 2: Bonded and non-bonded forces consider in MD simulations.

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.

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” [22, 23].

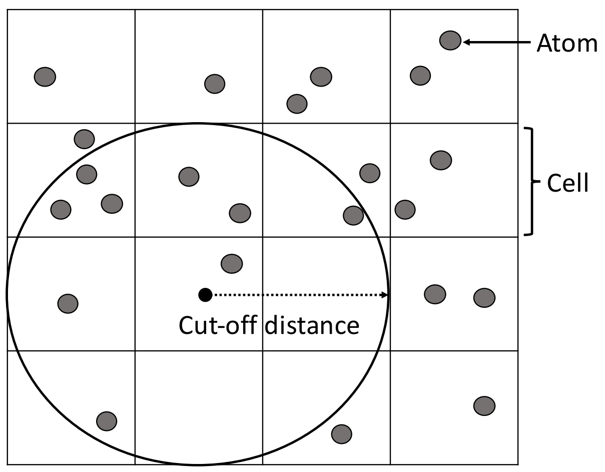

Figure 3: Division of the simulation box in to cells.

## 3 Implementation of the FPGA accelerator using OpenCL

### 3.1 Implementing the same c-code used in CPU

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 [23] 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.4. 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 [22]. 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 iteration, 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.

### 3.2 Implementing a pipelined architecture

To solve this problem, work in [21] proposes a pipelined architecture designed using OpenCL. We briefly explain it in this section. We focus on removing the nested-loop structure of the OpenCL code. For this purpose, 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

```
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
    }
  }
}
```

Figure 4: Algorithm of the non-bonded force computation method using cell-pair list.

accelerate the computation.

Fig.5 shows the flow-chart of the 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.6 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.7 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.8 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

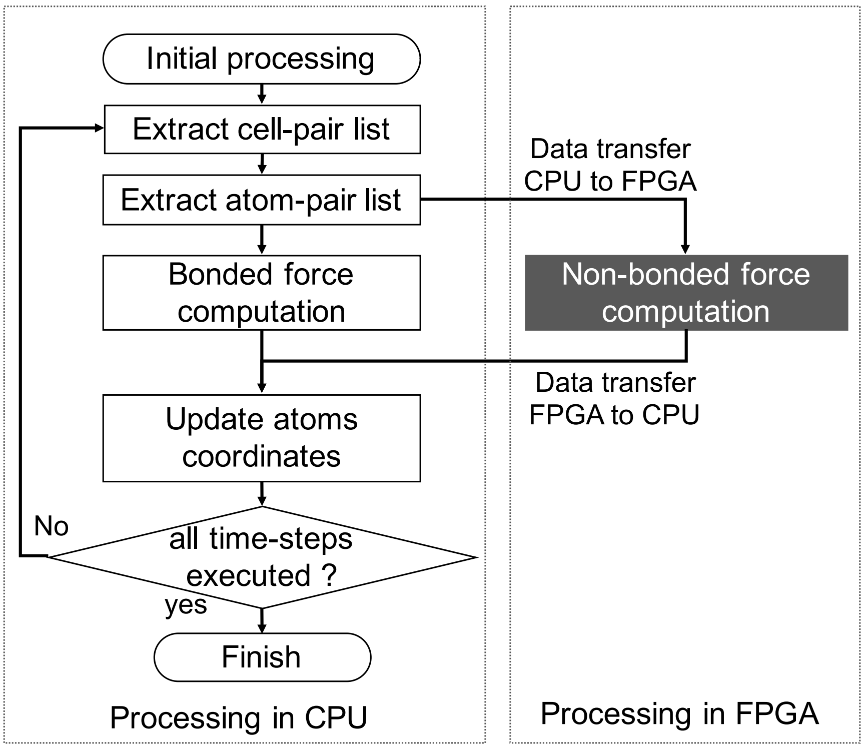

Figure 5: Flow-chart of the CPU-FPGA heterogeneous processing.

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 [24]. Similarly, electrostatic potential computation requires divisions, additions, subtractions and multiplications [22]. 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.

## 3.3 Memory access optimization

Data such as atom numbers, 3-D atom coordinates, charge of the atom, are required for the computation in each loop-iteration. We can store those data in different arrays on the global memory of the FPGA. However, this requires multiple memory access transactions, since we have to access different areas of the memory. We can improve the performance by coalescing the memory access. For this purpose, we have to pack all data required in one loop-iteration into one data packet. We use an structure shown in Listing 1 to pack the data. Therefore, all the required data are available in a contiguous memory region. Then we declare and array-of-structure to store all input data.

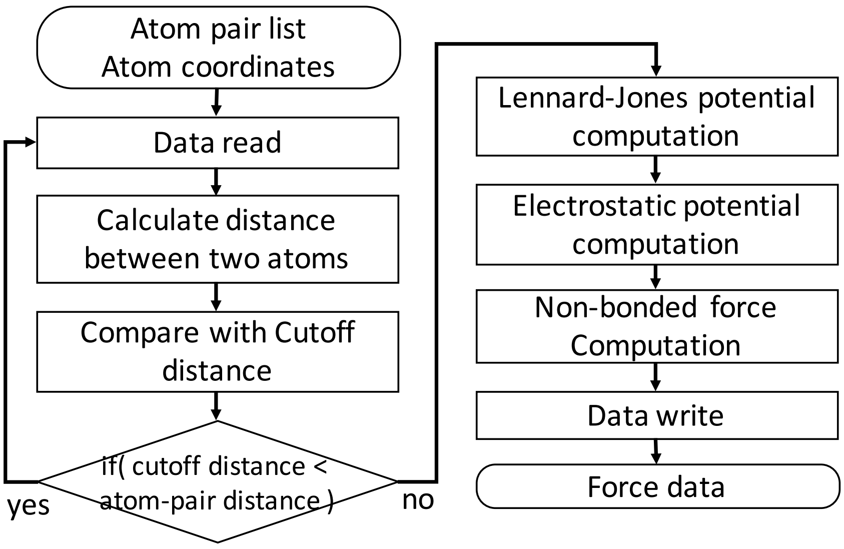
Atom pair list
Atom coordinates
Data read
Calculate distance between two atoms
Compare with Cutoff distance
if( cutoff distance < atom-pair distance )
yes
no
Lennard-Jones potential computation
Electrostatic potential computation
Non-bonded force Computation
Data write
Force data

Figure 6: Flow-chart of the processing in FPGA accelerator.

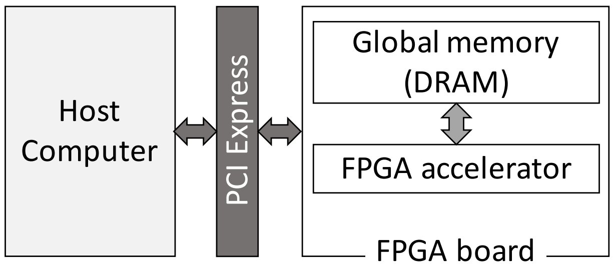

Figure 7: CPU-FPGA heterogeneous processing system for molecular dynamic simulations.

```
struct atom_pair
{
  int atom1;
  int atom2;
  float crd1[3];
  float crd2[3];
  float charge1;
  float charge2;
  short atom_type1;
  short atom_type2;
  int image;
};
```

Listing 1: A misaligned structure

FPGA access 64 byte data per a memory access. However, the structure in Listing 1 contains only 48 bytes. As a result, this structure is not aligned properly. Fig.9 shows how the data of the structure are stored. When the structure is not aligned, multiple transactions may required to access data in some loop-iterations. This problem can be solved by aligning the structure to 64 byte boundary. It is done by using "packed" and "aligned" attributes in OpenCL as shown in Listing 2. This will increase the structure size to 64 bytes. However,

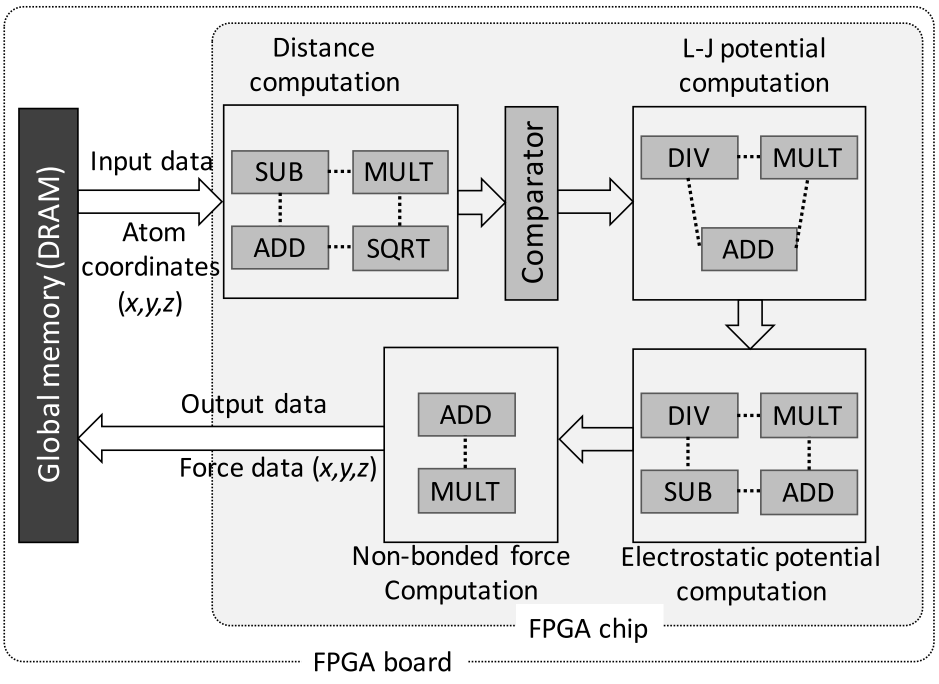

Figure 8: FPGA accelerator architecture proposed in [21].

each structure in the array fits on to the 64 byte boundary, so that one memory transaction is enough to access data.

```
__attribute__((packed))
__attribute__((aligned(64)))
struct atom_pair
{
  int atom1;
  int atom2;
  float crd1[3];
  float crd2[3];
  float charge1;
  float charge2;
  short atom_type1;
  short atom_type2;
  int image;
};
```

Listing 2: Aligned structure

In this implementation, we have to access extra 16 bytes that we do not use. As a result, OpenCL compiler may generate a wider 64bit data-path. Such wider data-paths may increase the complexity of the accelerator. To reduce this unnecessary data access, we can divide the structure into two, where one has 32 bytes and the other has 16 bytes. This will increase the amount of memory transactions. However, now we require only 32bit and 16bt data-paths instead of a 64bit data path. As a result, we can expect the OpenCL compiler to generate a much simplified accelerator, probably with an increased clock frequency. Therefore, we can expect a reduced processing time. Unfortunately, we have to compile both implementations to find out the best design.

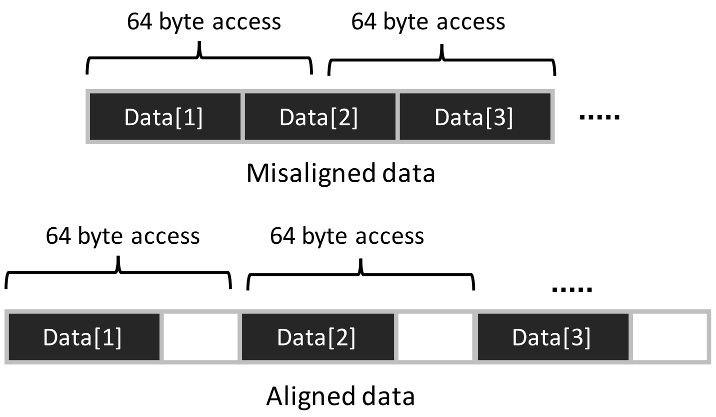

Figure 9: Aligning of structures.

Table 1: Comparison of the non-bonded force computation time on CPU and FPGA.

| **Implementation** | **Processing time ($s$)** |
|---|---|
| CPU | 0.68 |
| FPGA (using the same CPU code) | 88.03 |
| FPGA (using the proposed method) | 0.17 |

## 4 Evaluation

For the evaluation, we used "DE5a-Net Arria 10 FPGA Development Kit" [25] that contains an Arria 10 10AX115N3F45I2SG FPGA. The FPGA is configured using Quartus Prime 16.1 pro edition with OpenCL SDK [16]. Heterogeneous system contains an Intel Xeon E5 CPU with 64GB RAM. The operating system is CentOS 7. 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 shows the processing time comparison of the CPU-based and FPGA-based implementations of the non-bonded force computation. When the FPGA accelerator is based on the same c-code that used to implement the algorithm in Fig.4, we have not achieved any acceleration. In fact, the processing time is increased by more than 129 times compared to the CPU implementation. However, when we use the proposed atom-pair list based implementation on FPGA, the processing time is reduced to 0.17 seconds. It is a massive speed-up compared to the previous implementation. The processing speed of the FPGA implementation is 4 times larger than that of the CPU's.

As explained in section 3.3, we can reduced the processing time on FPGA by optimizing the memory access. Table 2 shows the processing time comparison of different memory access methods. The processing time of coalesced access using structures (method 2 in Table 2) is larger than that of the non-coalesced access in method 1. This is because the structure is misaligned. This problem is solved in method 3 by aligning the structures, and the processing time is also reduced significantly. In method 4, the structure is divided in to two perfectly aligned structures. Although this method increases the amount of memory access, it simplifies the data-path. Therefore, the clock frequency is increased and the

Table 2: Processing time comparison of different memory access methods.

| Method | Processing time (*ms*) | Clock frequency (MHz) |
|---|---|---|
| 1: not coalesced | 171.3 | 218.75 |
| 2: coalesced, misaligned | 244.7 | 210.41 |
| 3: coalesced, aligned to 64byte | 141.1 | 209.38 |
| 4: coalesced, aligned to 32byte and 16byte | 138.5 | 221.67 |

Table 3: Resource usage of the FPGA accelerator.

| Resource | Usage | Percentage used (%) |
|---|---|---|
| Logic (ALMs) | 68,163 | 16.0 |
| Registers | 120,168 | 7.0 |
| Memory (Mbits) | 3.35 | 3.5 |
| DSPs | 57 | 3.8 |

processing time is reduced.

We achieved 4.9 times speed-up compared to CPU by using method 4 to access memory. The work in [15] reports 11.1 times speed-up using a single FPGA, while the work in [26] 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. Moreover, if we increase the degree of parallelism by unrolling the outer-loop of the OpenCL code, we can increase the processing speed further. However, the maximum speed-up is limited by the memory access bandwidth of the FPGA board. Bandwidth of our DE5a-Net FPGA board is 25.6 GBps, and the FPGA accelerator requires 9.3 GBps throughput. Therefore, we can unroll the loop by a factor of 2 to achieve 9.8 times speed-up compared to the CPU.

Table 3 shows the resource usage of the FPGA accelerator. In our implementation, we used only 16% of the FPGA resources. If we used upto 80% of the FPGA resources and the bandwidth is around 50 GBps, we can theoretically increase the speed-up to 24.5 times. Fortunately, there are many recent FPGA boards that have over 100 GBps bandwidth. Therefore, it is possible to achieve a large speed-up using such an FPGA board.

Although off-loading computation to FPGA reduces the processing time, the PCIe based data transfer between the CPU and FPGA increases. Table 4 shows the processing times of different tasks in the heterogeneous computing system. The non-bonded force computation in FPGA takes the same processing time compared to the bonded force computation in CPU. During an iteration, both computations can be done in parallel in CPU and FPGA. Therefore, we can reduce 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. Similarly, force data are transferred from the FPGA to CPU. These data transfers take 68% of the total processing time.

The total processing time of the proposed and the conventional (CPU-based) methods are 0.47 seconds and 0.82 seconds per iteration. This gives only a small speed-up of 1.74

Table 4: Processing time of different tasks in the heterogeneous system per one iteration.

| **Task** | **Processing time (s)** |
|---|---|
| Non-bonded force computation in FPGA | 0.14 |
| Bonded force computation in CPU | 0.14 |
| Data transfer: CPU to FPGA | 0.25 |
| Data transfer: FPGA to CPU | 0.08 |
| Total processing time (parallel processing on CPU and FPGA) | 0.47 |
| Total processing time in conventional method (using CPU only) | 0.82 |

times. The reason for this is the large data transfer time. We can use data compression or redundant data removal to reduce the data amount and data transfer time. For example, three atom-pairs [atom1-atom2], [atom1-atom3] and [atom1-atom4] can be represented as [atom1:atom2,atom3,atom4] by removing the redundant data of atom1. Another way to solve this problem is to use SoC (system-on-chip) based OpenCL capable FPGAs. 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. Moreover, if the memory is shared between CPU and FPGA, we can completely eliminate the data transfers. This would give us a large speed-up.

## 5 Conclusion

We propose an FPGA Accelerator for MD simulations using OpenCL. The proposed architecture is completely designed by OpenCL, so that the same 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, CPU oriented c-based code is not suitable for FPGAs. OpenCL offline compiler unable to implement loop-pipelining for the nested-loops with data dependent boundaries. We used an atom-pair list based algorithm that requires a single loop to represent the force computation in OpenCL. This allows to automatically generate an efficient pipelined architecture. We achieved over 4.9 times speed-up compared to CPU implementation by using only 16% of the FPGA resources. Maximum of 24.5 times speed-up is possible by assuming an 80% resource utilization and a larger memory access bandwidth. Therefore, we can achieve better performance compared to previous works that implement HDL-based FPGA accelerators

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, we can use faster on-board data transfers. Moreover, if we can use a unified memory space for CPU and FPGA, we can completely eliminate the data transfer time.

## 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, vol. 18 no. 6, 1997, pp. 516-529.

[4] V. Daggett, ”Protein folding-simulation,” Chemical reviews, vol. 106 no. 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, vol. 26 no. 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,” 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., G”romacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit,” Bioinformatics vol. 29, no. 7, 2013, pp. 845-854.

[8] S. Plimpton, P. Crozier, and A. Thompson, LAMMPS-large-scale atomic/molecular massively parallel simulator, Sandia National Laboratories, vol. 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, vol. 4 no. 2, 1983, 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,” NIC Workshop, vol. 34, 2006, 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, vol. 51, no. 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,” Proc. International Conference for High Performance Computing, Networking, Storage and Analysis, 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,” Parallel and Distributed Processing and Applications, Springer, 2007, pp. 256-267.

[14] M. Chiu and M. C. Herbordt, ”Molecular dynamics simulation high-performance reconfigurable computing systems,” ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 3 no. 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] Intel FPGA 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] 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,” Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA’16), 2016, pp. 16-25.

[19] H.M. Waidyasooriya, Y.Takei, S.Tatsumi and M.Hariyama, ”OpenCL- Based FPGA-Platform for Stencil Computation and Its Optimization Methodology,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no.5, 2017, pp.1390-1402.

[20] Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Kota Kasahara, ”Architecture of an FPGA Accelerator for Molecular Dynamics Simulation Using OpenCL,” Proc. 15th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2016), 2016, pp. 115-119.

[21] Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Kota Kasahara, ”An FPGA Accelerator for Molecular Dynamics Simulation Using OpenCL,” International Journal of Networked and Distributed Computing, Vol. 5, No. 1, 2017, pp.52-61.

[22] 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, vol. 9 no. 12, 2013, pp. 5599-5609.

[23] mypresto, `http://presto.protein.osaka-u.ac.jp/myPresto4/index.php?lang=en`, 2015.

[24] Alan Hinchliffe, Molecular modelling for beginners, John Wiley & Sons, 2005.

[25] Terasic, DE5a-Net Arria 10 FPGA Development Kit, `https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=231&No=970&PartNo=2`

[26] S.Kasap and K.Benkrid, ”Parallel processor design and implementation for molecular dynamics simulations on a FPGA-based supercomputer,” Journal of Computers, vol. 7 no. 6, 2012, pp. 1312-1328.