Enhance the Strong Scaling of LAMMPS on Fugaku

Jianxiong Li
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
lijianxiong20g@ict.ac.cn

Tong Zhao
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
zhaotong@ict.ac.cn

Zhuoqiang Guo
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
guozhuoqiang20z@ict.ac.cn

Shunchen Shi
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
shishunchen22z@ict.ac.cn

Lijun Liu
Department of Mechanical Engineering, Graduate School of Engineering, Osaka University
Osaka, Japan
liu@mech.eng.osaka-u.ac.jp

Guangming Tan
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
tgm@ict.ac.cn

Weile Jia*
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
jiaweile@ict.ac.cn

Guojun Yuan
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
yuanguojun@ncic.ac.cn

Zhan Wang*
State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
wangzhan@ncic.ac.cn

ABSTRACT

Physical phenomenon such as protein folding requires simulation up to microseconds of physical time, which directly corresponds to the strong scaling of molecular dynamics(MD) on modern supercomputers. In this paper, we present a highly scalable implementation of the state-of-the-art MD code LAMMPS on Fugaku by exploiting the 6D mesh/torus topology of the TofuD network. Based on our detailed analysis of the MD communication pattern, we first adapt coarse-grained peer-to-peer ghost-region communication with uTofu interface, then further improve the scalability via fine-grained thread pool. Finally, Remote direct memory access (RDMA) primitives are utilized to avoid buffer overhead. Numerical results show that our optimized code can reduce 77% of the communication time, improving the performance of baseline LAMMPS by a factor of 2.9x and 2.2x for Lennard-Jones and embedded-atom method potentials when scaling to 36, 846 computing nodes. Our optimization techniques can also benefit other applications with stencil or domain decomposition methods.

KEYWORDS

High Performance Computing, Molecular Dynamics, LAMMPS, Computational Science

ACM Reference Format:
Jianxiong Li, Tong Zhao, Zhuoqiang Guo, Shunchen Shi, Lijun Liu, Guangming Tan, Weile Jia, Guojun Yuan, and Zhan Wang. 2023. Enhance the Strong Scaling of LAMMPS on Fugaku. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3581784.3607064

1 INTRODUCTION

Molecular dynamics (MD) simulation is the *de facto* tool for investigating microscopic phenomena such as material defects, drug design, and chemical reactions on supercomputers [3]. It consumes a large part of the computational time at supercomputer centers worldwide in simulating scientific problems such as nanoscale friction and wear [34], protein folding [17], two-dimensional materials [13], and ligand binding [19]. Recent advances, for example, ReaxFF, neural network-based potential energy surfaces, and enhanced sampling methods, have flourished traditional MD in

*Corresponding author

SC '23, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0109-2/23/11…$15.00
https://doi.org/10.1145/3581784.3607064

modeling more complex phenomena with more accurate force fields [5, 7, 20, 31]. However, one major challenge is the small time step constraint imposed by numerical stability and accuracy. For example, typical MD time steps are restricted to a few femtoseconds (fs) [25], which implies that 10^9 MD steps are required to simulate physical processes, such as protein folding that occur in the microsecond timescale [25, 26]. Therefore, strong scaling, which has historically been limited by inter-node communication due to the exchange of atomic position and forces of ghost regions, is arguably the most critical issue in the MD community.

A few well-established MD software, such as GROMACS [6], NAMD [23], Amber [9], HOOMD-blue [4], and Desmond [8], are developed to support different features for multiple fields of science [24]. NAMD, for example, is known for its highly scalable performance on supercomputers, and GROMACS can achieve good performance on a single processor [8]. Among all these packages, LAMMPS [28] is one current state-of-the-art package in simulating materials with periodic boundary conditions. Several works have been introduced to improve the scaling of MD codes. For example, K. Bowers et al. proposed parallel decomposition methods and message-passing techniques to achieve faster time-to-solution on commodity clusters than Blue Gene/L [8]. J. Glaser et al. showed that 108 million atoms can be simulated with 3, 375 GPUs on the Titan supercomputer [14]. B. Acun et al. achieved 128 nanoseconds per day on 1024 nodes of Summit [1]. Another example is the work by X Duan et al., who developed a cell-list-based method and achieved strong scaling up to 65,536 cores on Sunway Taihu Light [11]. More radical work, such as the domain-specific supercomputer Anton 3, can reach unprecedented 22.4 microseconds of 10,666K atoms per day [24].

Exascale supercomputer is shifting towards many-core architecture due to power consumption considerations. To date, the top 5 supercomputers in the top500 list [30] are equipped with either heterogeneous acceleration hardware or many-core CPUs. The gap between computation, memory access, and network bandwidth is growing bigger and bigger. This poses challenges and opportunities to the scaling of MD codes. On the one hand, the memory capacity of modern supercomputers has enabled scientists to extend the limit of MD simulation of bigger physical systems. For example, weak scaling of $\sim$ 275 billion atoms has been achieved by X Duan et al [12]. On the other hand, strong scaling is a more interesting problem since the architecture shift brings more opportunities, for example, the more complex network topology and the growing number of network interface cards (NIC) in one node. In this paper, we focus on the simulation of long physical time for a physical system of moderate scale, for example, tens or hundreds of thousands of atoms. Such ability corresponds to many crucial advances in material science. However, the massive parallelization of such code on many-core supercomputers requires enormous inter-node communication. Therefore, the strong scaling of MD brings far more challenges than executing an extremely large system with short physical time.

In this paper, we present a highly scalable implementation of the state-of-the-art MD code LAMMPS on the Fugaku supercomputer. The communication pattern of MD is redesigned to fully exploit both computational power and 6D mesh/torus topology of the Tofu interconnect D (TofuD) [2]. Our major contributions are listed as the following:

- We analyze the communication pattern of LAMMPS in detail, and then we adopt coarse-grained peer-to-peer ghost-region communication to exploit the TofuD.
- We apply fine-grained thread pool to further improve the scalability of the LAMMPS code on the Fugaku supercomputer.
- RDMA primitives are utilized to avoid buffer overheads.
- Numerical results show that our optimized code can reduce up to 77 percent of the communication time, improving the performance of baseline LAMMPS by a factor of 2.9x and 2.2x for Lennard-Jones and embedded-atom method potential when scaling to 36, 846 computing nodes.

Although we only implement the scaling of LAMMPS on Fugaku, we remark that such optimization techniques can also benefit other applications with stencil or domain-decomposition methods.

The rest of this paper is organized as follows. We review the MD simulation and Fugaku supercomputer in section 2. Both computational and communication innovations are shown in section 3. The numerical results and analysis are presented in section 4, followed by the conclusion and outlook in section 5.

2 BACKGROUND

2.1 MD Simulations and LAMMPS

Classical MD based on Newtonian mechanics use potential functions to describe the interactions among particles. As one of the most classical potential functions, Lennard-Jones (L-J) potential [18] represents the pair-wise interaction of atoms using van der Waal's forces. Equation 1 shows the analytical form of the L-J potential function. Note that r_{ij} stands for the distance between particles i and j. The 12-order term stands for the repulsion force, and the six-order term is the attractive force. σ_{ij} and ε_{ij} are related to the atomic types, with σ_{ij} being the equilibrium distance and ε_{ij} being the minimum potential energy between atom i and j.

$$U_{LJ} = \sum_{i<j}^{n} 4\varepsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^{6}] \tag{1}$$

Another widely used atomic potential is the embedded-atom method (EAM) potential [10], which is based on density functional theory and particularly targeted at metallic systems. The analytical form of the EAM potential is shown in Equation 2. F_i denotes the embedding energy, and it is a function of the atomic electron density ρ_j. Φ_{ij} is the pair potential interaction between particle i and j.

$$U_{EAM} = F_i(\sum_{j\neq i} \rho_j(r_{ij})) + \frac{1}{2}\sum_{j,i,j\neq i} \Phi_{ij}(r_{ij}) \tag{2}$$

Domain decomposition method is widely used in MD software packages such as LAMMPS. As shown in Fig. 1, the physical system is evenly distributed among all MPI ranks, and each MPI rank holds a fraction of the system (colored in blue in Fig. 1(b)). A ghost region of neighboring atoms within a cutoff radius r_{cut} is set up to conduct the evaluation of pair forces. The ghost region is built during the MD simulation in the border stage by receiving neighboring atoms from adjacent MPI ranks, followed by the rebuilding of the neighbor

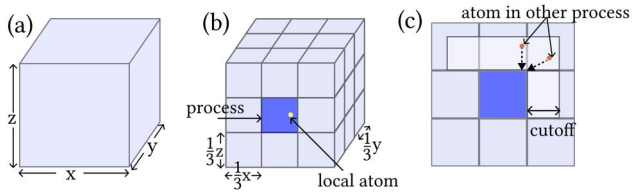

Figure 1: Data distribution of a physical system in LAMMPS. (a) A physical system with a simulation domain of X, Y, and Z. (b) The data of the physical system is divided into 27 MPI ranks, and each rank holds a sub-box of size $[\frac{1}{3}X, \frac{1}{3}Y, \frac{1}{3}Z]$. (c) The ghost region of one MPI rank in the two-dimension view. Note that only half of the ghost atoms are communicated with Newton's 3rd law enabled.

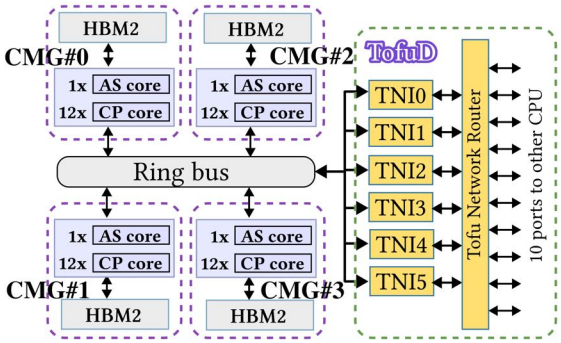

Figure 2: The Fugaku A64FX CPU features 4 CMGs, each containing 13 cores. The CMGs are composed of AS (assistant) cores, which are dedicated to operating system and I/O, and CP (computing) cores. Note that the TofuD controller has 6 TNIs, which can independently send and receive messages. Each node can directly connect with 10 CPU nodes with a bandwidth of 6.8GB/s.

list for every atom. In every MD step, the pair forces are carried out in the pair stage and exchanged back in the reverse stage. Next, the atomic position and velocity of local atoms are updated. The positions of the ghost atoms are updated in the forward stage at the beginning of the next timestep.

The most computationally intensive part lies in the evaluation of the pair force, and it scales perfectly as the number of MPI ranks grows. The bottleneck of the strong scaling stems from the communication of ghost regions that occurs during the reverse and forward stages of MD simulation. For example, when scaling to 36864 computing nodes, communication time takes up to 64 percent of the total. Note that communication is needed only among adjacent MPI ranks, as shown in Fig. 1(c). So, taking advantage of the locality characteristic can be the key to reducing communication time.

2.2 Fugaku supercomputer

The Fugaku supercomputer at the RIKEN Center for Computational Science ranks No. 2 on the top 500 list [29]. It has 158,976 computing

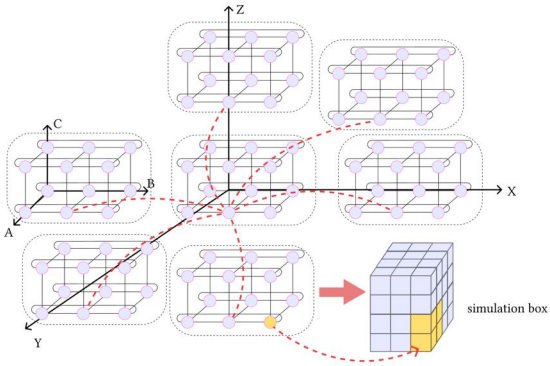

Figure 3: 6D torus topology of the Fugaku TofuD network. Each cell has 12 nodes (organized in $2 \times 3 \times 2$ 3D torus), and cells are also organized in 3D torus. 4 adjacent MPI ranks can be assigned to one node, and all MPI ranks can be directly mapped to a sub-box while preserving the original physical topology.

nodes and a peak performance of 537 PFlop/s. Each node is equipped with an A64FX CPU featuring four core memory groups (CMG). Each CMG has 13 cores (one for IO and OS, and the remaining 12 for computing) and an 8GB high bandwidth memory (HBM2) operating at 256GB/s, as shown in Fig.2 [21]. The A64FX CPU supports 512-bit SIMD extensions, and each core can execute 32 double-precision floating-point operations per cycle.

Fugaku nodes are interconnected by TofuD. Each node has a TofuD controller connected with CMGs by a ring bus network on a chip (NoC). The TofuD controller contains 6 Tofu network interfaces (TNIs) and is capable of transmitting 6 messages simultaneously. The TNI supports communication functions of remote direct memory access (RDMA) PUT/GET with minimal latency of 0.49us. It also provides a cache injection mechanism so that TNI can directly write data to the last level cache to further reduce the latency. The TofuD network router, as shown in Fig. 2, has 10 physical ports (112Gbps Bi-direction rate per port) and is connected to 10 nodes in the following fashion: all connection directions of ports are divided into 6: X-, Y-, Z-, A-, B- and C-. Among them, X-, Y-, Z- and B- directions have 2 ports each, and A- and C- directions have 1 port each. These ports form a "torus fusion" topology named 6D Torus/Mesh, shown in Fig.3. Fugaku also provides the uTofu programming interface, which allows developers to directly use low-level primitives to perform low-latency one-sided communications.

The TofuD network can benefit MD packages in three ways. Firstly, the 6D topology can be arranged to take advantage of the three-dimensional domain decomposition of MD, as illustrated in Fig. 3. MPI ranks can be directly mapped to a sub-box while preserving the original physical topology. This can effectively reduce the average communication hops and latency. Secondly, the 6 TNIs on each node make it possible to simultaneously communicate with 6 nodes, which can significantly speed up multiple MPI transactions. Finally, the RDMA primitives provide developers with low-latency

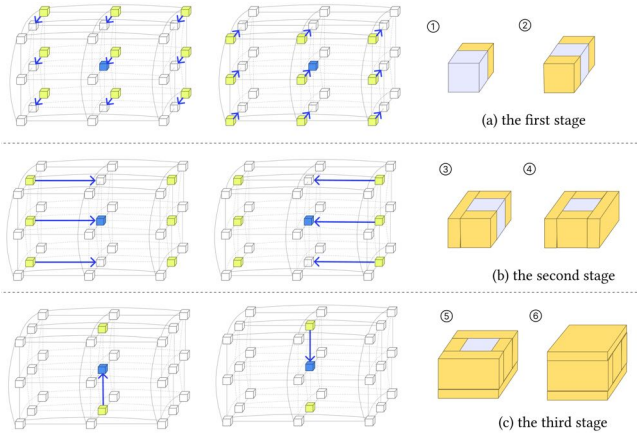

Figure 4: The 3-stage communication pattern. The rank at the center obtains atoms from the 26 ranks arranged in three-dimensional coordinates around it through three stages. In the first stage, 9 ranks in the same vertical plane as the central rank get atoms from their horizontally related ranks in the front and back planes. In the second stage, 3 ranks on the same vertical line as the central rank receive atoms from their horizontally related ranks on the left and right vertical lines. In the third stage, the central rank receives atoms from the ranks vertically above and below it. Note that all ranks acquire atoms in the same way as we described for central node behavior. Besides, each stage will carry part of the atoms received from the previous stage.

communication. In the following section, we will show how these features can be used to improve the strong scaling of LAMMPS.

3 OPTIMIZATION

We present the optimization details of scaling LAMMPS on Fugaku. Note that throughout this paper, Newton's 3rd law is enabled. And all tests in this section are performed on two typical physical systems of 65K (small system) and 1.7 million (big system) hydrogen atoms, respectively. We remark that we focus on communication in this section since we are mainly optimizing the strong scaling.

3.1 Communication Analysis for LAMMPS

Our work starts with modeling the LAMMPS communication behavior and finding the direction of optimization through quantitative analysis. In LAMMPS, communications between neighbor MPI ranks typically can be implemented via two ways: the 3-stage [8] and peer-to-peer (p2p) pattern [8].

1. 3-stage communication . Fig. 4 shows the default 3-stage communication in LAMMPS. Each MPI rank communicates in turn with 26 neighboring MPI ranks in 3 directions (X, Y, and Z) via sending 6 messages. However, this communication pattern can not fully exploit the TofuD network for the following reasons: (1) computation and communication can not be overlapped; (2) an MPI barrier is mandatory between stages. Especially, Fugaku has 6 TNIs equipped on each node, and the 3-stage can not fully exploit the hardware capability.

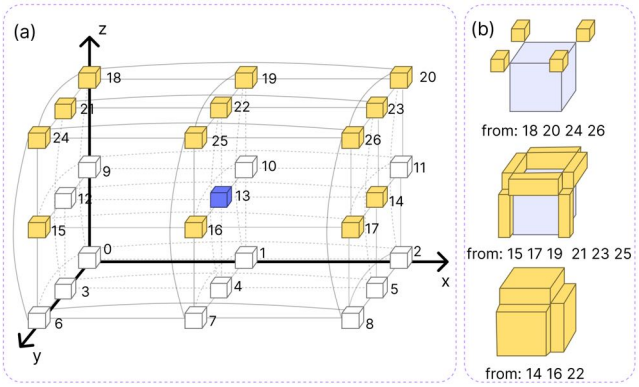

Figure 5: The p2p pattern with Newton's 3rd Law enabled. (a) The blue MPI rank communicates with the yellow neighbors to receive ghost atoms for the calculation of pairwise interaction. The resulting forces are then sent back to the yellow neighbors. The forces with the white neighbors are calculated by the white ones and then sent back. The blue MPI rank sends atoms and receives forces from the white neighbors without receiving atoms from them. (b) The yellow region shows the ghost regions obtained from the yellow neighbors.

2. **Peer-to-Peer (p2p) pattern**.

In the p2p pattern, each MPI rank communicates directly with its 26 neighbors, as shown in Fig.5. This approach offers several advantages over the 3-stage pattern. Firstly, it readily accommodates optimizations that enable Newton's 3rd law to save half of the communication volume. Secondly, p2p allows computation and communication to be overlapped easily. Finally, since communication with each neighbor is independent, parallel communication can be implemented. Furthermore, since more messages are sent and received in the p2p communication, it is possible to fully utilize the multiple TNIs provided by the Fugaku supercomputer.

Note that Table 1. is the communication pattern of the central MPI rank13 (marked in blue) in Fig. 5. Column $from\_neighbor$ denotes the MPI ranks with which the blue MPI rank communicates. a represents the length of the border side of each sub-box, and r represents r_{cut}. Note that message size can be expressed as the volume of the ghost region, and we define the sub-box as a cube to simplify the calculation. hop stands for the number of network hops required to reach the target MPI rank in the 3D torus topology. msg denotes the total number of messages in the corresponding row. T_{0-5} in the last column represents the peer-to-peer communication time required to send a message of the given size and hop count to the corresponding neighbor.

We introduce another time parameter, denoted by T_{inj}, which represents the time interval of 2 continuous messages reaching the network from the sending node. Since message transmission in hardware can be fully pipelined and the time spent in the software network stack for a message can be overlapped with the hardware time of the previous one, the time interval mainly depends on the CPU time.[33] In the case of small message sizes, we do not consider message blocking in the network. Therefore, the total time of the

Table 1: Communication patterns analysis

pattern	*msg_size*	hop	msg	*from_neighbor*	time
3-stage	a^2r	1	2	10,16	T_0
	$a^2r + 2ar^2$	1	2	12,14	T_1
	$(a+2r)^2r$	1	2	4,22	T_2
	$total\_atom = 8r^3 + 12ar^2 + 6a^2r$, $total\_msg = 6$				
p2p	a^2r	1	3	14,16,22	T_3
	ar^2	2	6	15,17,19,21,23,25	T_4
	r^3	3	4	18,20,24,26	T_5
	$total\_atom = 4r^3 + 6ar^2 + 3a^2r$, $total\_msg = 13$				

[1] a stands for the side length of each sub-box, and r stands for the r_{cut}.
[2] The neighbor ids of two patterns are shown in Fig. 5
[3] *hop* stands for the number of hops to the destination node in the logical topology during message transmission.

two communication patterns can be expressed as follows. T_{last} is the time taken by the last of the 13 messages.

$$T_{3stage-naive} = 2T_0 + 2T_1 + 2T_2 \tag{3}$$

$$T_{p2p-naive} = 12T_{inj} + T_{last} \tag{4}$$

As the two messages in the same stage of the 3-stage pattern can be transmitted simultaneously, the Equation (3) can be optimized as below:

$$T_{3stage-opt} = 3T_{inj} + T_0 + T_1 + T_2 \tag{5}$$

Meanwhile, in the p2p pattern, since the network transmission time of the earlier sent messages can overlap with T_{inj}, the p2p pattern can prioritize the message with the shortest transmission time for final transmission. Therefore, Equation (4) can be modified as follows:

$$T_{p2p-opt} = 12T_{inj} + Min(T_3, T_4, T_5) \tag{6}$$

Because Fugaku can send six messages simultaneously and support parallel communication, the communication time can be further optimized. As the two messages on the same stage can be sent simultaneously in the 3-stage pattern, we eliminate the T_{inj} on each stage and get the Equation (7). And it's the same in Equation (8).

$$T_{3stage-parallel} = T_0 + T_1 + T_2 \tag{7}$$

$$T_{p2p-parallel} = 2T_{inj} + Min(T_3, T_4, T_5) \tag{8}$$

Because the T_{inj} depends on the time consumed by the CPU at the receiving and sending nodes [33], and is much smaller than the peer-to-peer time, and T_3 is equal to T_0, p2p pattern theoretically can take less communication time than 3-stage in Fugaku supercomputer.

3.2 Coarse-grained P2P

We first adopt a coarse-grained parallelization by invoking 4 MPI ranks on each Fugaku node (each MPI rank has 12 threads to fully use the 48 cores on A64FX). Since each Fugaku node comprises 4 NUMA domains, MPI rank count that can not be divided by this factor will result in cross-NUMA memory access. Simultaneously,

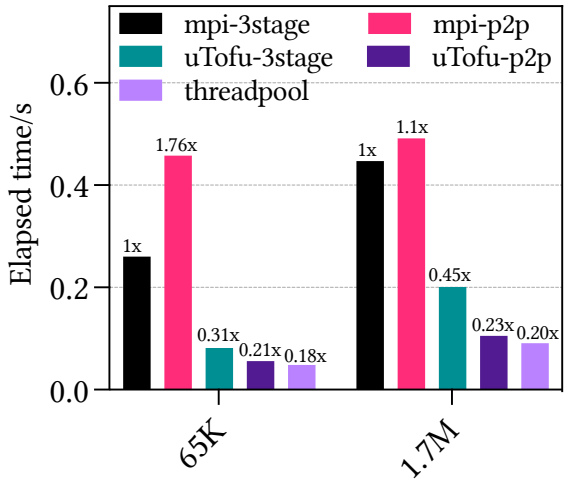

Figure 6: The message transmission time was evaluated over 10k iterations on 768 nodes. The result excludes data-packing time. Due to the high overhead of MPI, the p2p pattern exhibits weaker performance than the 3-stage pattern with the MPI interface. Conversely, uTofu has low communication overhead and small T_{inj}. Consequently, the p2p pattern with the uTofu interface (uTofu-p2p) achieves shorter communication time, which is reduced by 79% compared to the 3-stage pattern with the MPI interface.

MPI rank count exceeding 4 will amplify inter-node communication [22]. Therefore, launching 4 ranks is an optimal balance for both computation and communication needs. Although we have proved that p2p should be faster compared to the 3-stage communication in section 3.1, a naive MPI-p2p implementation on 65K and 1.7 million hydrogen atoms is slower compared with baseline LAMMPS, as shown in Fig. 6. This can be attributed to the high communication overhead of MPI caused by its heavy software stack, such as message fragmentation and tag-matching.

In order to fully leverage the advantages of the p2p pattern and 6D torus network, we implement both 3-stage and p2p communication via uTofu, which is a low-overhead one-sided communication interface that allows users directly control TofuD network hardware.

As shown in Fig. 7, each TNI has 9 control queues (CQs) to manage message transmit and receive queues. A virtual control queue (VCQ) is created within the application and bound to the CQ for one-sided communication.

In the coarse-grained parallelization, we bind 4 MPI ranks to 4 individual TNIs. And we implement both 3-stage and p2p communication with uTofu. The network injection time T_{inj} can be significantly reduced once uTofu is used to replace MPI. Testing results on 768 nodes show that both 3-stage and p2p communication has decreased considerably. uTofu-p2p communication is 1.5 times faster compared to the uTofu-3-stage implementation, as shown in Fig. 6. This agrees with our analysis in section 3.1.

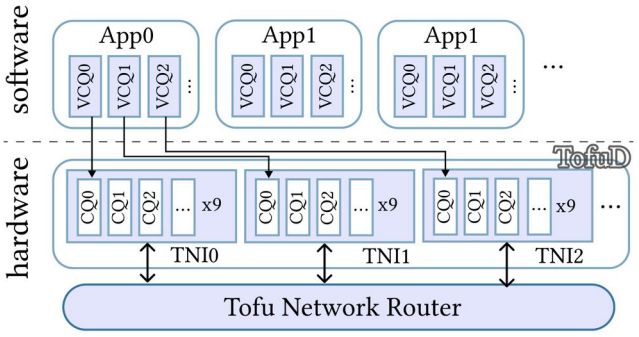

Figure 7: In hardware, each TNI has 9 CQs. All CQs in the same TNI share the same message-processing engine. In software, when using 4 TNIs, each MPI rank creates a VCQ binding the CQ in the specified TNI and uses the VCQ for one-sided communication. When using 6 TNIs, each MPI rank creates 6 VCQs, and each VCQ binds one CQ of each TNI. Each VCQ is independent and can be accessed simultaneously by different threads. The figure shows using 6 TNIs.

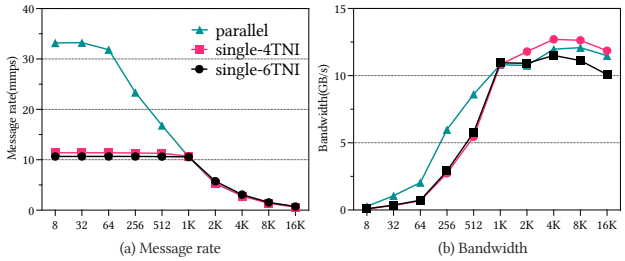

Figure 8: The performance of p2p pattern communication was evaluated on 768 nodes. single-4TNI and single-6TNI represent communication with a single thread and using 4 and 6 TNIs, respectively. *Parallel* represents communication using 6 threads and 6 TNIs. For the message size less than 512 bytes, the parallel method has a higher message rate.

3.3 Fine-grained P2P

In this section, we show a fine-grained p2p communication to further exploit the TofuD network. According to the analysis in the previous section, parallel communication is the key to taking full advantage of the p2p pattern. Note that in the above coarse-grained p2p communication, uTofu is utilized with single thread binding to a single CQ, and we use only 4 out of all the 6 TNIs. To efficiently use all 6 NTIs, more threads are required to launch VCQs and bind to individual CQ. We remark that CQ is not thread-safe when performing one-sided communication, and each MPI rank can only allocate one CQ on each TNI by default. For example, MPI rank0 can only hold all the CQ0 from TNI0 to TNI5, as shown in Fig. 7. In this fashion, the four MPI ranks in a node can query 24 individual CQs, allowing each MPI rank to use 6 threads for communication. We implement the fine-grained parallel communication in our optimized LAMMPS.

Fig. 8 shows the message rate and bandwidth of one node with respect to the message sizes. We find that when communicating with a single thread, the message rate using 6 TNIs is lower than that of 4TNIs due to the contention in one TNI, which further reduces the message rate. Communication in parallel can be faster when the message size is small (under 1KB). When simulating 65K atoms on 768 nodes, each MPI rank contains only 22 atoms, and the size of each message is less than 528B. Hence, we can boost the message-sending rate by at least 50%.

However, the communication process in LAMMPS is divided into multiple stages, and using OpenMP for parallel communication can introduce additional overhead due to thread startup and synchronization in each stage. Our experiments indicate that the communication time increases after adding the overhead of OpenMP. Therefore, we choose to use the spin lock thread pool with lower overhead to implement our parallel communication. In the case of small messages, using thread pool resulted in an 14% improvement in performance, making it better suited for strong scaling scenarios. The communication time of the thread pool is shown in Fig. 6.

We further conducted tests to measure the overhead of OpenMP and thread pool for thread startup and synchronization, which resulted in 5.8us and 1.1us, respectively. In the case of small atom numbers, the original LAMMPS still uses OpenMP to speed up certain functions in the pair and modify stages. However, we found that using OpenMP in some stages can negatively impact optimization. For instance, the Lennard-Jones potential uses the microcanonical ensemble (NVE) to update the position and velocity of atoms. Enabling OpenMP causes the modify stage to take ten times longer. Therefore, we opted to use thread pool in all stages of LAMMPS to make parallel computing more efficient.

For load balancing, each MPI rank is limited to 6 threads for communication, whereas each sub-box has 13 neighboring sub-boxes. These neighboring boxes have varying message sizes and numbers of hops. As indicated by the analysis presented in Table 1, the neighbors in the positive direction of X, Y, and Z typically have the longest message size but a short number of hops, whereas the neighbor farthest away has a small message size but a long number of network hops. Consequently, we distributed the load appropriately for each thread, as depicted in Fig. 10.

3.4 Pre-registered address

In LAMMPS, the receive and send buffers are dynamically expanded during the simulation if they run out of space. However, with the uTofu RDMA communication, the receive and send buffers need to be registered before one-sided communication can occur, which incurs significant overhead for the requirement of falling into the kernel state.

To minimize address registration overhead, we pre-calculate the theoretical upper limit of atoms to be exchanged during communication and allocate an appropriate buffer size in the setup stage based on data such as the cutoff range, sub-box size, and atom density. This reduces the need for frequent buffer expansions during the simulation. During the forward stage, updated atom positions are directly written into the position array of the remote MPI rank. Meanwhile, during the reverse stage, the forces of ghost atoms are

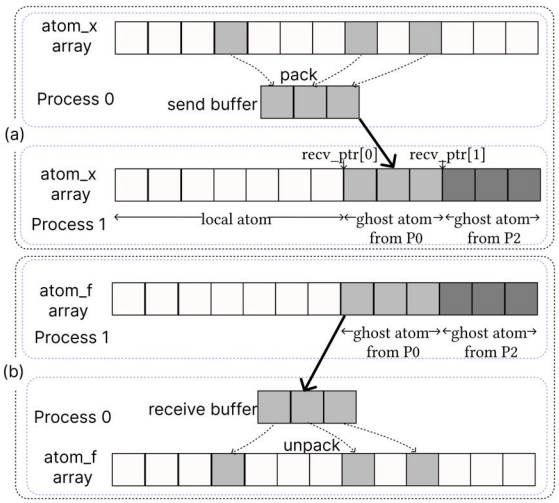

Figure 9: (a) In the forward stage, MPI rank0 packs the positions of local atoms and directly sends the data to the atom position array of MPI rank1. MPI rank1 stores the offset of the ghost atom position from P0 in recv_ptr and informs MPI rank0 of the offset in advance during the border stage. (b) In the reverse stage, MPI rank1 sends the forces of ghost atoms to MPI rank0. Then MPI rank0 unpacks the receive buffer and combines the forces with local ones.

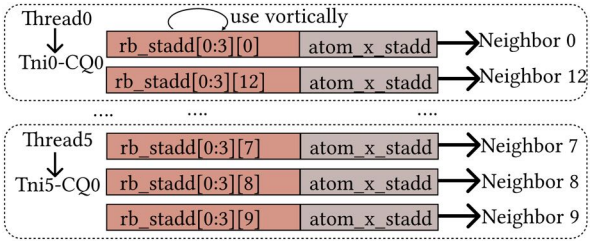

Figure 10: Four receive buffers are allocated for each neighbor and used in a round-robin fashion. During the setup stage, the addresses of these registered buffers and the registered atom position array addresses are sent to their respective neighbors. To achieve load balancing, each thread communicates with different MPI ranks, and the load division is based on the size of the messages and the number of hops involved.

directly sent back to the remote MPI rank. This approach avoids copying data from the receive buffer to the target array. However, the position and force arrays also need to be registered and dynamically expanded. To solve this issue, we create the position and force arrays with the theoretical maximum value, enabling us to complete the entire simulation process with only one-time address registration. See Fig. 9 for a visual representation of this process.

LAMMPS stores the positions and forces of both local and ghost atoms in contiguous memory. During the border stage, the offset of the ghost atoms is marked by a local index called recv_ptr (as shown in Fig. 9a). During the forward stage, the positions of local atoms are packed and sent to the corresponding neighbor MPI ranks. However, a challenge arises when the sender does not know the offset of the receiver's ghost atoms in the position array and cannot specify the send address. To overcome this challenge, the receiver needs to make another communication to inform the sender of the offset of ghost atoms. As the offset is only an 8B value, we can use the piggyback mechanism to minimize communication overhead, which can embed data in the descriptor for low-latency communication. Additionally, we overlap the data unpacking process with this extra communication to further reduce communication overhead.

The communication type of uTofu is RDMA put/get, which writes data directly to the remote buffer during put communication. However, if only one receive buffer is used, it may cause memory overwriting with the previous communication when two communication stages are close to each other. To avoid this issue, we create multiple receive buffers and use them in a round-robin fashion. After analyzing the dependencies of the LAMMPS communication workflow, we decide to create four buffers to ensure that the front and back communication stages do not conflict. During the setup stage, all the registered addresses of receive buffers and atom position arrays are sent to neighbors for subsequent communication, shown in Fig. 10.

3.5 Other Optimizations

3.5.1 message combine. Due to the constraints of the MPI interface, transmitting an array with an unknown length requires sending its length first, followed by its content in another message, which incurs additional communication overhead. To address this problem, we compress the two-step communication as follows: when sending an array, we set the first element of the message as the array length, and the remaining parts are the array content. The receiver can determine the packet length by parsing the first field of the message.

3.5.2 border bin. Local atoms may be in several neighbor MPI ranks' ghost regions. To avoid the overhead of going through all neighboring sub-boxes to determine which sub-boxes the local atoms should be sent to, we employ a faster algorithm. In the setup stage, we divide the sub-box into 27 bins arranged in a 3x3x3 pattern based on the cutoff and calculate the borders of each bin. We then assign bins to neighbor MPI ranks according to their borders. When packing atoms, we identify which bin an atom belongs to, and the target neighbors can be directly determined. This algorithm speeds up the decision-making process and helps to improve efficiency.

3.5.3 topo map. Molecular Dynamics simulations with 3D topology can be seamlessly integrated into the Fugaku 6D-torus network. When submitting the task, users can specify the required number and shape of nodes in the job script. The job management software then assigns a group of nodes that fulfill the requirements. At runtime, MPI ranks can obtain the physical topology coordinates of their nodes through the mpi-extend library provided by Fugaku and calculate their respective sub-boxes. Therefore, the number of hops required for communication can be reduced.

4 EVALUATION

We evaluated our optimized code of LAMMPS on Fugaku by launching four MPI ranks per node.

We selected two commonly used potentials in the field of materials: L-J and EAM. The potential parameters were provided by LAMMPS benchmarks, with Newton's law enabled for both potentials. Further details on the configuration can be found in Table 2. The Fig. 11 shows the atomic configuration of the hydrogen and silicon systems.

The simulation process for the L-J potential is the same as section 2.1, while the EAM potential requires two additional communications during the pair stage to send the density and derivative of the embedding energy of each particle for force calculation. Additionally, the EAM potential uses a neighbor list building method that involves checking whether any particles have moved beyond half of the neighbor skin at regular intervals and communicating the results to other MPI ranks by MPI_Allreduce. If any particle has moved beyond half of the neighbor skin, all MPI ranks rebuild the neighbor list.

Our optimization is based on the stable version of LAMMPS released on December 22, 2022. The reference results shown in the figures below are obtained from the original version of LAMMPS.

4.1 Accuracy

Our optimized version of LAMMPS does not modify the force calculation of any potentials and only focuses on optimizing communication and parallel performance. Therefore, we retain the original precision of the LAMMPS. We evaluate the pressure of 65K atoms for both L-J and EAM potentials. As shown in Fig. 11, the results of the optimized LAMMPS agree with the original code perfectly.

4.2 Step-by-step results

We evaluated the performance of the two potentials of 65K and 1.7M particles on 768 nodes. The result is shown in Fig. 12. The thread pool version creates 12 threads, 6 of which are responsible for communication in addition to computing. Others use OpenMP with 12 threads for computing, and only the master thread is responsible for communication.

Table 2: potential configuration

parameter	L-J	EAM
Units	lj	metal
Lattice	0.8442 FCC	3.615 FCC
Cutoff	2.5	4.95
Skin	0.3	1.0
Timestamp	0.005 tau	0.005 psec
Newton	on	on
Neigh_modify	20	5
	check no[1]	check yes
Fix	NVE	NVE
Potential Parameters	sigma:1	$Potential file: Cu\_u3.eam$[2]
	epsilon:1	

[1] The neighbor list needs to be rebuilt at specified intervals. If the check flag is set to yes, each rank scans itself regularly, and if any atom moves out of the 1/2 neighbor skin, all ranks are notified to rebuild the neighbor list. In this process, an all-reduce is required to collect global information. But if the check flag is set to no, the scan is not required, and the neighbor list is forced to be updated regularly.

[2] Cu_u3.eam is provided by the LAMMPS package.

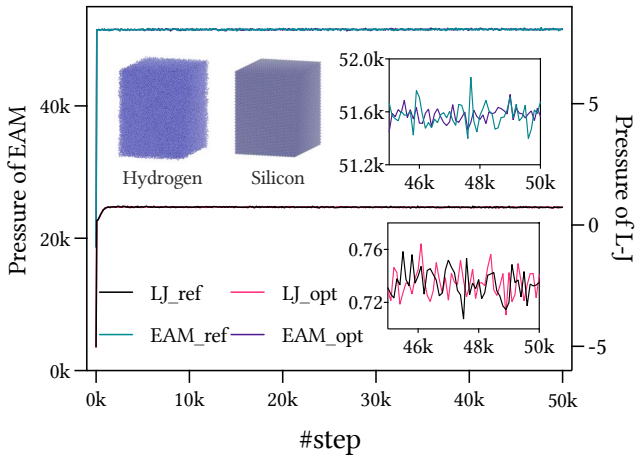

Figure 11: The experiment evaluates the pressure of 65K atoms system with 50K steps for L-J and EAM potentials. The two northwest subgraphs are the atom system of hydrogen and silicon that we evaluated. The eam_ref and lj_ref are the results of the original code of LAMMPS. The lj_opt and eam_opt are the results of our optimized code. The northeast subgraph is the final trend of pressure for EAM potential, and the southeast subgraph indicates the final trend of L-J potential.

Our optimized version demonstrated a significant performance improvement over the original code, with 3.01x and 2.45x speedup for 65K particles, as shown in Fig. 12a. However, for 1.7M particles, due to the large number of particles in each MPI rank, the simulation time is dominated by the pair stage, resulting in a relatively less improvement of 1.6x and 1.4x, respectively.

The performance of the p2p pattern using 6 TNIs with single thread is abnormally poor compared with 4 TNIs. This result can be attributed to the significant time overhead incurred by the software function call during one-sided communication. Therefore, one TNI per MPI rank is sufficient to meet the communication requirements when there is only one thread for communication per MPI rank. Using 6 TNIs, on the other hand, can cause contention among MPI ranks that attempt to use the same TNI simultaneously.

In the case of 65K particles, the parallel-p2p optimization reduces the communication time by 77% compared to the original code, as shown in Fig. 12b. Since each MPI rank only has 22 particles, the message size is relatively small, at most 528B in the forward and reverse stages. Thus, our parallel strategy achieves lower communication time. Besides, the parallel strategy can also speed up the data packing and unpacking process. Because the number of particles is small, the pair stage time is relatively short, and the OpenMP overhead is more significant. As shown in Fig. 12c, the pair stage time of the first four tests in L-J potential is almost identical. However, after we change the parallel method to the thread pool, the pair stage time reduces by 43%. For EAM potential, the optimized version also

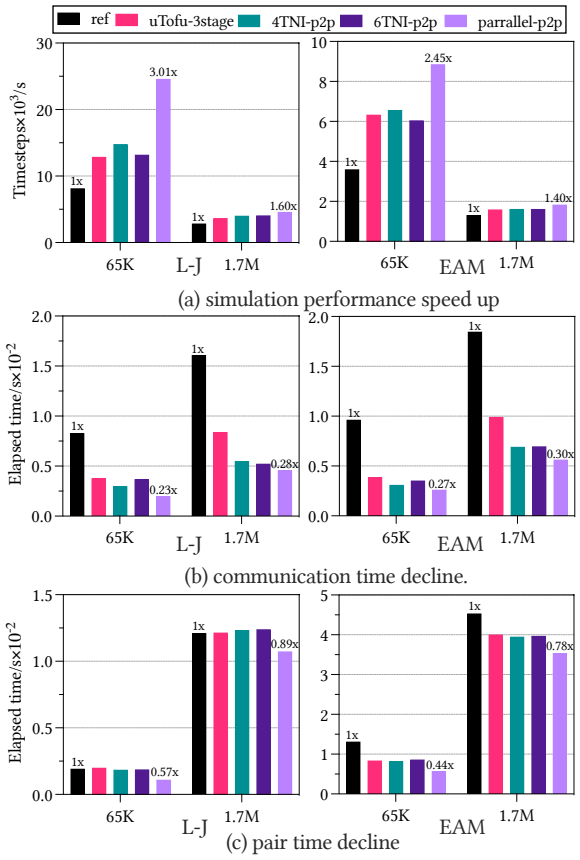

Figure 12: The step-by-step performance of optimization. Ref stands for the original code of LAMMPS. uTofu-3stage stands for communication using the 3-stage pattern with the uTofu interface. 4TNI-p2p and 6TNI-p2p represent the p2p pattern with the uTofu interface and a single thread, using 4 TNIs and 6 TNIs, respectively. Parallel-p2p represents using thread pool with the uTofu interface and 6 TNIs.

improves communication performance in the pair stage, resulting in a reduction of 56% in the pair stage time.

When simulating 1.7M particles, all implementations of the p2p pattern show significantly lower communication time than the 3-stage pattern. Each message is much larger in this case, resulting in longer time to pack and unpack the data and slower message transmission. The strict sequential nature of the 3-stage pattern magnifies this negative effect threefold. However, the p2p pattern can send messages simultaneously, effectively covering the extra overhead caused by the larger message size. Moreover, the pack and unpack process can overlap with the message transmission, further reducing the overall time.

4.3 Scaling results

4.3.1 Strong scaling. Because the job system allocates resources to users on a 2x3x8 (a shelf) unit, forming the torus network on a large scale. So we apply for nodes as integral multiples of the shelf. The scale of our strong scaling evaluation is 768, 2160, 6144, 18432, and 36864, corresponding to 8x12x8, 12x15x12, 16x24x16, 24x32x24, 32x36x32, respectively. The particle count is 4,194,304 and 3,456,000 for each potential. At the last point, the average number of particles on each core was 2.3 and 1.9, respectively. Our optimized version achieved a 2.9x and 2.2x improvement compared to the original version, as shown in Fig. 13a, resulting in simulation performance of 8.77M tau/day and 2.87 us/day for the L-J and EAM potentials, respectively.

The improvement achieved by the EAM potential is relatively less because it needs an additional atomic cross-border scan and an all-reduce operation is performed every 5 timesteps to collect the global neighbor list status. The time taken for this process is shown in the "Other" column of Table 3. We can see that the "Other" stage takes over 31.84% of the total time, which is greater than the time taken for communication. The overhead of the all-reduce operation is high in large-scale evaluations. We can reduce the frequency of the all-reduce operation by increasing the neighbor skin.

The comparison of the pair stage time is shown in Fig. 13b. Our optimizations not only speed up communication but also improve computing performance. At the first point, the computing time is relatively long because of the large number of particles. Therefore, the overhead brought by OpenMP is not apparent, and the time of the pair stage is similar before and after optimization. However, at the last point of strong scaling, the pair stage time of the optimized version drops by 40% and 57% compared to the original version. There are two reasons for this. Firstly, in the pair stage, the overhead of thread startup and synchronization by OpenMP is significant, which is greatly reduced after using the thread pool. Secondly, two additional communications are needed during the pair stage of the EAM potential. We also optimize the communication time in the pair stage, resulting in a reduction in the pair stage time for both potentials.

Table 3: Breakdown of the strong scaling in the last point

potential	$stages^{12}$				
	Pair	Neigh	Comm	Modify	Other
Origin-L-J[3]	2.17	0.2125	9.196	1.3272	1.2756
	15.3	1.5	64.85	9.36	8.99
Opt-L-J	1.3019	0.18073	2.1287	0.4986	0.7641
	26.71	3.71	43.67	10.23	15.68
Origin-EAM	14.266	0.75542	11.001	1.2637	5.56
	43.44	2.3	33.5	3.85	16.91
Opt-EAM	6.143	0.61705	3.0115	0.47991	4.79
	40.85	4.1	20.02	3.19	31.84

[1] These five stages are the basic workflow of LAMMPS. Pair is to compute particle force and also contains the communication in the process of solving force. Neigh is to create neighbor list. Comm is to exchange ghost atom data. Modify is to update the velocity and position of local atoms. Other contains any remaining time including the output stage .

[2] The first row of each type of evaluation is the elapsed time of running 99 steps, and the unit is 0.01 seconds. The second row is the percentage of the stage in the entire running time.

4.3.2 Weak Scaling. We also evaluated the weak scaling performance of the L-J and EAM potentials. We initialized each core with 100K particles for L-J and 72K for EAM and expanded from

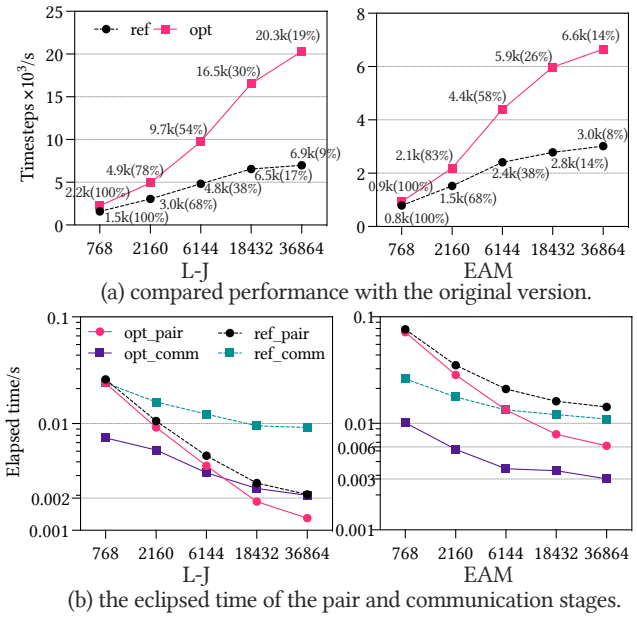

Figure 13: We evaluate the strong scaling performance from 768 to 36864 nodes with 4,194,304 and 3,456,000 particles for both potentials. The percentage in Fig.(a) is the parallel efficiency [12], and the baseline is the performance of the first point, 768 nodes. The elapsed time of the pair and Communication stages in Fig.(b) is derived from the result statistics of LAMMPS. The communication time only includes the forward, reverse, border, and exchange stages. The elapsed time of communication in the pair stage of the EAM potential is counted into the pair stage time.

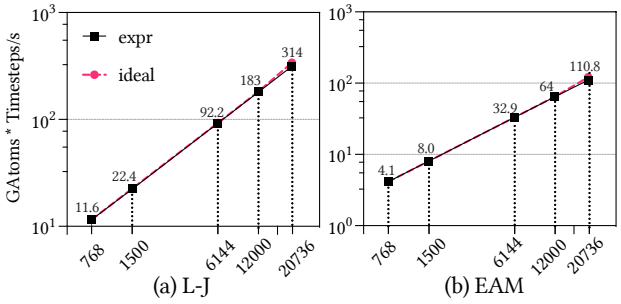

Figure 14: We evaluate weak scaling performance from 768 to 20,736 nodes with 100K and 72K particles per core for two potentials. At last, we have reached 99 and 72 billion particles, respectively. The simulation performance increases almost linearly.

768 nodes to 20,736 (24x36x24) nodes. In the end, each potential reached 99 billion and 72 billion particles, respectively. As shown in Fig. 14, nearly linear scaling can be achieved.

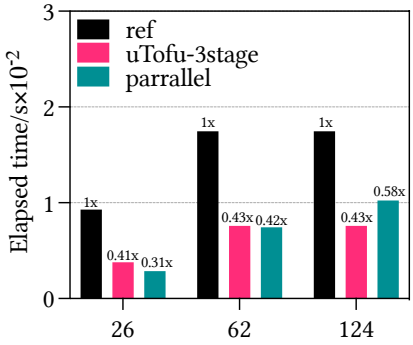

Figure 15: The scenarios of sending 26, 62, and 124 messages per communication stage are evaluated on 768 nodes. They correspond to the following force potentials. Potentials with Newton's 3rd law disabled or needing a full neighbor list have to communicate with 26 neighbors. When the cutoff is large and the sub-box size is smaller than the cutoff, MPI ranks need to communicate with 62 or 124 neighbors based on whether Newton's 3rd laws are enabled. The result shows that the optimized version works well in the first two cases but worsens in the third.

4.4 Extended Experiment

Our optimized version only evaluates potentials with half neighbor lists and Newton's 3rd law enabled. However, some potentials, such as Tersoff [27] and DeePMD [15, 16, 32], require a full neighbor list to calculate atom forces. This necessitates each MPI rank to communicate with 26 neighbors simultaneously. We evaluated this scenario, and the results are shown in Fig. 15. The results of the extended experiment demonstrate that our optimizations are also applicable to these potentials.

For most potentials, the optimal number of particles per MPI rank is achieved when the size of each sub-box approaches the cutoff value. In order to fully utilize the available hardware resources, it is necessary to ensure that each core is assigned at least one particle to compute. However, for certain potentials with a longer cutoff range, the sub-box size may be smaller than the cutoff value in the strong scaling test, resulting in each MPI rank having to communicate with a larger number of surrounding MPI ranks, i.e., 62 and 124 neighbors for each type of neighbor list. We also evaluated this scenario and found that the p2p pattern takes more time than the 3-stage pattern when communicating with 124 surrounding MPI ranks, as shown in Fig. 15. This is because the 3-stage scales linearly, while p2p is an n-squared extension.

5 CONCLUSION

In this paper, we present an efficient and scalable implementation of LAMMPS on the Fugaku supercomputer. We employed a combination of optimizations, including utilizing the TofuD 6D-torus network and the manycore architecture, to enhance the strong scaling of LAMMPS. We implemented an efficient coarse-grained and fine-grained p2p neighbor communication to reduce communication time and improve parallel efficiency. Additionally, we use the

pre-registrated address to reduce buffer overhead. Our experiments show that our optimized LAMMPS reduces communication time by 77% compared to the original code. Our implementation achieved simulation speeds of 8.77M tau/day and 2.87 us/day for L-J potential and EAM potential on 36864 nodes, respectively.

Modern supercomputers such as Fugaku, Frontier, the new Sunway, etc., have provided multiple NICs with high message rates, yet few applications can efficiently utilize these devices to reduce their corresponding time-to-solution. This work is a demonstration of exploiting the performance of network hardware to further extend the scalability of LAMMPS, a current state-of-the-art MD code. We also remark that this work demonstrates the potential of the software-hardware codesign, and our idea can also be adapted to other applications with the similar communication pattern, such as domain decomposition and stencil computation.

ACKNOWLEDGMENTS

This work is supported by the following funding: National Key Research and Development Program of China (2021YFB0300600), National Science Foundation of China (T2125013, 62032023, 61972377, 92270206, 61972380), CAS Project for Young Scientists in Basic Research (YSBR-005) and Network Information Project of Chinese Academy of Sciences (CASWX2021SF-0103), and Huawei Technologies Co., Ltd.. This work used computational resources of the supercomputer Fugaku provided by RIKEN through the HPCI System Research Project (Project ID: hp220310). The numerical calculations in this study were partially carried out on the ORISE Supercomputer. We thank Xiaohui Duan (Shandong University), Junmin Xiao (ICT) and Fan Yang (ICT) for their helpful discussions.

REFERENCES

[1] Bilge Acun, David J Hardy, Laxmikant V Kale, Keqin Li, James C Phillips, and John E Stone. 2018. Scalable molecular dynamics with NAMD on the summit system. *IBM journal of research and development* 62, 6 (2018), 4–1.

[2] Yuichiro Ajima, Takahiro Kawashima, Takayuki Okamoto, Naoyuki Shida, Kouichi Hirai, Toshiyuki Shimizu, Shinya Hiramoto, Yoshiro Ikeda, Takahide Yoshikawa, Kenji Uchida, et al. 2018. The tofu interconnect d. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 646–654.

[3] Maral Aminpour, Carlo Montemagno, and Jack A Tuszynski. 2019. An overview of molecular modeling for drug discovery with specific illustrative examples of applications. *Molecules* 24, 9 (2019), 1693.

[4] Joshua A Anderson, Jens Glaser, and Sharon C Glotzer. 2020. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science* 173 (2020), 109363.

[5] Christopher M Baker. 2015. Polarizable force fields for molecular dynamics simulations of biomolecules. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 5, 2 (2015), 241–254.

[6] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. 1995. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer physics communications* 91, 1-3 (1995), 43–56.

[7] Rafael C Bernardi, Marcelo CR Melo, and Klaus Schulten. 2015. Enhanced sampling techniques in molecular dynamics simulations of biological systems. *Biochimica et Biophysica Acta (BBA)-General Subjects* 1850, 5 (2015), 872–877.

[8] Kevin J Bowers, Edmond Chow, Huafeng Xu, Ron O Dror, Michael P Eastwood, Brent A Gregersen, John L Klepeis, Istvan Kolossvary, Mark A Moraes, Federico D Sacerdoti, et al. 2006. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. 84–es.

[9] David A Case, Thomas E Cheatham III, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M Merz Jr, Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J Woods. 2005. The Amber biomolecular simulation programs. *Journal of computational chemistry* 26, 16 (2005), 1668–1688.

[10] Murray S Daw and Michael I Baskes. 1984. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Physical Review B* 29, 12 (1984), 6443.

[11] Xiaohui Duan, Ping Gao, Meng Zhang, Tingjian Zhang, Hongsong Meng, Yuxuan Li, Bertil Schmidt, Haohuan Fu, Lin Gan, Wei Xue, et al. 2020. Cell-list based molecular dynamics on many-core processors: a case study on sunway TaihuLight supercomputer. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[12] Xiaohui Duan, Ping Gao, Tingjian Zhang, Meng Zhang, Weiguo Liu, Wusheng Zhang, Wei Xue, Haohuan Fu, Lin Gan, Dexun Chen, et al. 2018. Redesigning LAMMPS for peta-scale and hundred-billion-atom simulation on Sunway TaihuLight. In *SC18: International conference for high performance computing, networking, storage and analysis*. IEEE, 148–159.

[13] Ping Gao, Xiaohui Duan, Jiaxu Guo, Jin Wang, Zhenya Song, Lizhen Cui, Xiangxu Meng, Xin Liu, Wusheng Zhang, Ming Ma, et al. 2021. LMFF: Efficient and scalable layered materials force field on heterogeneous many-core processors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[14] Jens Glaser, Trung Dac Nguyen, Joshua A Anderson, Pak Lui, Filippo Spiga, Jaime A Millan, David C Morse, and Sharon C Glotzer. 2015. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications* 192 (2015), 97–107.

[15] Zhuoqiang Guo, Denghui Lu, Yujin Yan, Siyu Hu, Rongrong Liu, Guangming Tan, Ninghui Sun, Wanrun Jiang, Lijun Liu, Yixiao Chen, et al. 2022. Extending the limit of molecular dynamics with ab initio accuracy to 10 billion atoms. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 205–218.

[16] Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, E Weinan, and Linfeng Zhang. 2020. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *SC20: International conference for high performance computing, networking, storage and analysis*. IEEE, 1–14.

[17] Martin Karplus and John Kuriyan. 2005. Molecular dynamics and protein function. *Proceedings of the National Academy of Sciences* 102, 19 (2005), 6679–6685.

[18] John E Lennard-Jones. 1931. Cohesion. *Proceedings of the Physical Society* 43, 5 (1931), 461.

[19] Kai Liu and Hironori Kokubo. 2020. Prediction of ligand binding mode among multiple cross-docking poses by molecular dynamics simulations. *Journal of Computer-Aided Molecular Design* 34, 11 (2020), 1195–1205.

[20] Sergei Manzhos, Richard Dawes, and Tucker Carrington. 2015. Neural network-based approaches for building high dimensional and quantum dynamics-friendly potential energy surfaces. *International Journal of Quantum Chemistry* 115, 16 (2015), 1012–1020.

[21] Ryohei Okazaki, Takekazu Tabata, Sota Sakashita, Kenichi Kitamura, Noriko Takagi, Hideki Sakata, Takeshi Ishibashi, Takeo Nakamura, and Yuichiro Ajima. 2020. Supercomputer Fugaku Cpu A64fx realizing high performance, high-density packaging, and low power consumption. *Fujitsu Technical Review* (2020), 2020–03.

[22] Szilárd Páll, Mark James Abraham, Carsten Kutzner, Berk Hess, and Erik Lindahl. 2015. Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In *Solving Software Challenges for Exascale: International Conference on Exascale Applications and Software, EASC 2014, Stockholm, Sweden, April 2-3, 2014, Revised Selected Papers 2*. Springer, 3–27.

[23] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. 2005. Scalable molecular dynamics with NAMD. *Journal of computational chemistry* 26, 16 (2005), 1781–1802.

[24] David E Shaw, Peter J Adams, Asaph Azaria, Joseph A Bank, Brannon Batson, Alistair Bell, Michael Bergdorf, Jhanvi Bhatt, J Adam Butts, Timothy Correia, et al. 2021. Anton 3: twenty microseconds of molecular dynamics simulation before lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[25] David E Shaw, Ron O Dror, John K Salmon, JP Grossman, Kenneth M Mackenzie, Joseph A Bank, Cliff Young, Martin M Deneroff, Brannon Batson, Kevin J Bowers, et al. 2009. Millisecond-scale molecular dynamics simulations on Anton. In *Proceedings of the conference on high performance computing networking, storage and analysis*. 1–11.

[26] David E Shaw, JP Grossman, Joseph A Bank, Brannon Batson, J Adam Butts, Jack C Chao, Martin M Deneroff, Ron O Dror, Amos Even, Christopher H Fenton, et al. 2014. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 41–53.

[27] Jerry Tersoff. 1988. New empirical approach for the structure and energy of covalent systems. *Physical review B* 37, 12 (1988), 6991.

[28] Aidan P. Thompson, H. Metin Aktulga, Richard Berger, Dan S. Bolintineanu, W. Michael Brown, Paul S. Crozier, Pieter J. in 't Veld, Axel Kohlmeyer, Stan G. Moore, Trung Dac Nguyen, Ray Shan, Mark J. Stevens, Julien Tranchida, Christian Trott, and Steven J. Plimpton. 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications* 271 (2022), 108171. https://doi.org/10.1016/j.

cpc.2021.108171
[29] TOP500.org. 2023. SUPERCOMPUTER FUGAKU - SUPERCOMPUTER FUGAKU, A64FX 48C 2.2GHZ, TOFU INTERCONNECT D. https://www.top500.org/system/179807/.
[30] TOP500.org. 2023. top500. https://www.top500.org/.
[31] Adri CT Van Duin, Siddharth Dasgupta, Francois Lorant, and William A Goddard. 2001. ReaxFF: a reactive force field for hydrocarbons. *The Journal of Physical Chemistry A* 105, 41 (2001), 9396–9409.
[32] Han Wang, Linfeng Zhang, Jiequn Han, and E Weinan. 2018. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications* 228 (2018), 178–184.
[33] Rohit Zambre, Megan Grodowitz, Aparna Chandramowlishwaran, and Pavel Shamis. 2019. Breaking band: a breakdown of high-performance communication. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.
[34] Zongxiao Zhu, Shi Jiao, Hui Wang, Linjun Wang, Min Zheng, Shengyu Zhu, Jun Cheng, and Jun Yang. 2022. Study on nanoscale friction and wear mechanism of nickel-based single crystal superalloy by molecular dynamics simulations. *Tribology International* 165 (2022), 107322.

Appendix: Artifact Description/Artifact Evaluation

ARTIFACT DOI

10.5281/zenodo.7839729

ARTIFACT IDENTIFICATION

In this paper, we implemented an efficient coarse-grained and fine-grained peer-to-peer neighbor communication to reduce communication time and improve parallel efficiency for LAMMPS. Additionally, we use pre-registered address to reduce buffer overhead. The above contributions are all in our optimized code. The experimental results of the paper can be reproduced.

The following is the experimental code we provide:

(1) **opt**: p2p pattern with utofu is with utofu interface and 6 TNIs using thread pool. And it is also our optimized code.
(2) **ref**: the original code of LAMMPS.
(3) **6tni_p2p**: p2p pattern with utofu interface and a single thread to communicate with 6 TNIs.
(4) **4tni_p2p**: p2p pattern with utofu interface and a single thread to communicate with 4 TNIs.
(5) **utofu_3stage**: communication using 3-stage pattern with the utofu interface.

The above projects are corresponding to the five step-by-step experiments in our paper, and the opt is also our optimized code in the scaling experiment. Each project has a compile script and a run script. Besides, the experimental results in the paper are also in the corresponding project.

The software architecture is the same as LAMMPS and we mainly changed the communication part and switched OpenMP to the thread pool. The projects can be access in https://doi.org/10.5281/zenodo.7839729.

REPRODUCIBILITY OF EXPERIMENTS

Below is the compilation process of the thread pool version in our paper:

First, we enter the opt folder: *cd opt*

Then we compile the project: *./compile.sh*

After compiling, the executable file lmp_threadpool will be copied to the BIN_threadpool_lj and BIN_threadpool_eam folders. These two folders have the input files of Lennard-Jones and embedded-atom method potentials respectively. Then enter one of the folders, such as BIN_threadpool_lj. There are three files in BIN_threadpool_lj.

(1) **in.threadpool.lj**: LAMMPS input file of Lennard-Jones potential, default is 60k atoms. The script provides 60k, 1.7M and 5M atomic scales that appear in the paper, which can be selected according to requirements.
(2) **lmp_threadpool.pjsub**: Fugaku job script.
(3) **run.sh**: execution script

Then we go into one of the potential evaluation folders:
cd BIN_threadpool_lj
and execute run.sh:
./run.sh

The generated log file is stored in output/output_result/output.xxxxxxxx/0/1/stdout.1.0.

The execution file, lmp_threadpool, will also be copied to BIN_scaling_test_lj and BIN_scaling_test_eam for scaling evaluation, and the test process is the same as above.

For the test of 6tni_p2p, 4tni_p2p, utofu_3stage and in the paper, it is the same as the above process.

The execution time is within 2 minutes. In the log file, we mainly focus on two indicators: performance and mpi task timing breakdown. They correspond to the performance and the time of different stages of LAMMPS.

ARTIFACT DEPENDENCIES REQUIREMENTS

Our code has to run on the Fugaku supercomputer without any extra libraries. The input scripts are provided by Lammps benchmarks.

ARTIFACT INSTALLATION DEPLOYMENT PROCESS

The process of compilation and execution is outlined in AD, with compiling taking approximately 3 minutes and execution less than 2 minutes.