Optimization and Evaluation of VLPL-S Particle-in-cell Code on Knights Landing

Dandi Ding*†, Minhua Wen*, Shan Zhou†, Min Chen†, James Lin*§

*Center for High Performance Computing, Shanghai Jiao Tong University, Shanghai, China
†Intel Asia-Pacific Research and Development Ltd. Shanghai, China
‡Department of Physics and Astronomy, Shanghai Jiao Tong University, Shanghai, China
§Tokyo Institute of Technology, Japan

{*}, {†}, {‡}, {§} \{axe, wenminhua, minchen, james\}@sjtu.edu.cn, shan.zhou@intel.com

Abstract—VLPL-S code is developed based on the particle-in-cell (PIC) algorithm, which is the mainstream algorithm of plasma behavior research. In this paper, we report our early experience on porting and optimizing the VLPL-S particle-in-cell code on the Knights Landing. By applying general optimization methods such as memory access optimization, thread level parallelism and vectorization to the code, we achieved 1.68 times speedup compared to the original code. After optimization, 1.53 times speedup is achieved on Knights Landing 7210P compared with that on a two-socket Xeon E5-2697v4 node. In this work, the performance improvement of different optimization methods on Knights Landing and the Xeon-based node is compared. The results show that most of the commonly used optimization methods are effective for the VLPL-S code on Knights Landing.

Keywords: Laser Plasma, Particle-in-Cell, Knights Landing

I. INTRODUCTION

With the rapid development of high performance computing, laser plasma simulations can be conducted in larger scale and with higher precision. Virtual Laser Plasma Laboratory (VLPL)[10] is a c++ based three dimensional relativistic PIC code, which is widely used in laser plasma interactions[5][9][11]. VLPL-S is a modified version in Shanghai Jiao Tong University and it is migrated to the Many Integrated Core architecture. VLPL-S considers the impact ionization, field ionization, and two body collision. The ionization of electrons is implemented with the virtual functions, which shows the polymorphism of object oriented language. Moving window technology is adopted in the VLPL-S code to research on the laser wakefield acceleration in GeV region. By including these processes, terahertz radiation, laser propagation in neutral gas medium, surface electron acceleration as well as GeV level electron acceleration and radiation in laser wakefield are studied[3].

In this paper, VLPL-S is optimized with different methods and then evaluated on the second generation of Intel® Xeon Phi™ processor (code named Knights Landing). Prefetch improves the performance significantly on KNL (Knights Landing) regarding the original latency of memory access bounded hotspot. The scalability of VLPL-S on KNL cluster is improved with the help of thread level parallelism. With the help of those optimizations, 1.68 times speedup is achieved compared to the original code.

This paper is organized as follows. Section II introduces the related work. Section III and Section IV briefly introduces the PIC algorithm and Intel Knights Landing. The optimization methods with details are presented in Section V. Section VI shows the evaluation of the optimizations on different platforms and then presents the discussion. Finally, Section VII concludes this paper.

II. RELATED WORK

For the last 3 decades, many implementations of PIC algorithm have been proposed. Those works have proved the possibility of exploiting parallelism of PIC algorithm. Bastrakov et al.[2] reported the evaluation of their implementation of PIC code on the multi-core system, KNC (The 1st generation of Intel® Xeon Phi™ processor, code named Knights Corner), and GPGPU recently. By their optimization that improved data locality and SIMD (Single instruction, multiple data) capability, they achieved up to 7x speedup on an 8-core Xeon E5-2690 processor with 99% strong scaling efficiency on shared memory. As for the implementation on KNC, only 1.5x speedup is achieved on the 7110X coprocessor over the 8-core Xeon E5-2690 CPU. The performance is only 20% of the peak performance of Xeon Phi 7110X (KNC). Also, their GPGPU implementation which reduces the fields data by atomic operations achieved 10x speedup on Fermi-generation GPUs over 8 CPU cores in single precision.

More recently, I.A. Surmin et al.[13] developed a high performance PIC code on KNC called PICADOR. After optimizations for better data locality and vectorization efficiency, they achieved 3.75x speedup on Intel Xeon E5-2600 CPU and is 7.5x on Xeon Phi 5110P (KNC). The speedup of Xeon Phi 5110P is 1.6x compared to Xeon CPU. I.A. Surmin et al. also evaluated the performance of PICADOR[14] on KNL. Simply rebuilding the code for KNL yields 2.43x speedup compared to Knights Corner. After optimizations, additional 1.89x speedup and 100GFLOPS double precision performance are achieved on KNL.

VLPL-S is widely used for ionization included processes, so a large number of particles may be generated during simulations compared to other PIC implementations, and thus more memory consumption and more communication cost inevitably happen. Though we are now in the post-Peta and Exascale
era, it is not easy to exploit the full performance of manycore processors. Hiroshi Nakashima[8] showed the toughness of manycore and SIMD-aware implementation of PIC simulation code. The new generation of Xeon Phi (KNL) provides higher peak performance and memory bandwidth with the help of MCDRAM (Multi-Channel DRAM) compared to the first-generation of Xeon Phi (KNC). Moreover, KNL provides 6 Tflops theoretical peak performance and thus makes it an interesting topic of porting and evaluating VLPL-S on KNL.

III. VLPL-S

This section briefly describes the algorithm used in VLPL and VLPL-S code. A detailed introduction of PIC algorithm is given in [6].

The simulation area is a three-dimensional space divided by uniformly aligned grids. Dynamics of the electric field \( E \) and magnetic field \( B \) is defined by Maxwell’s equations solved on the grid using the FDTD method[13]. The particles are assembled and then represented in the form of plasma which in detail is a combination of momentum \( p \), position \( r \), constant mass \( m \) and charge \( q \). The position and velocity \( v \) of the so assembled particles evolve according to the Lorentz force and equation of motion that is numerically integrated using Boris method. Electric current \( j \) is created by the movement of particles, which is used in Maxwell’s equations. The equations mentioned above compose a self-consistent system.

\[
\begin{align*}
\nabla \times B &= \frac{4\pi}{c} j + \frac{1}{c} \frac{\partial E}{\partial t} \\
\nabla \times E &= -\frac{1}{c} \frac{\partial B}{\partial t} \\

\end{align*}
\]

The brief computational scheme of the PIC method with the main equations and data dependencies is given in Fig. 1. An iteration of the main loop makes a move to the time step. There are four stages in each time step. Field solver updates the electromagnetic field values of the grid. Field interpolation computes the Lorenz force affecting particles by the grid electromagnetic values. In order to push particles one step further, the equations of particle motion are solved. The final stage is the computation of the current created by the movement of the particles. Regarding code implementation, merging the field interpolation, force computation and solving equations of particle motion into one stage may improve the data locality. The merged stage is referred as particle push. The PIC method can be extended in many ways. The VLPL-S code considers the interaction of the laser and plasma which requires another procedure to calculate the affection of the laser on the grid electromagnetic field dynamics. Among all procedures of PIC algorithm, the particle push part is the hotspot of VLPL-S code. It takes a large part of the computational time.

As for the domain decomposition method, VLPL-S divide the simulated space with uniform length both horizontally and vertically. Cells in PIC algorithm are of the same size, so each MPI domain contains the same number of cells. This method of domain decomposition generates a load balanced task partition only when the distribution of particles are balanced in the simulated space.

There are two principally different data sets in PIC algorithm: an ensemble of charged particles with continuous coordinates and values of the field and current density set on a space-discrete grid[13]. Due to the complex computation and the access of both data sets, particle push and current deposition takes most of the running time. In VLPL-S code, the assembled particles are stored with linked list. Each cell in the discrete grid maintains a linked list of the particles that reside in it. The impact of the irregular memory access on the performance will be discussed later.

IV. INTEL KNIGHTS LANDING

Knights Landing is the 2nd generation of Intel® Xeon Phi™ processor, which brings several new technological features. KNL (Knights Landing) is self-bootable which means KNL can work as a stand-alone processor. A high bandwidth on package memory called Multi-Channel DRAM (MCDRAM) in addition to the traditional DDR4 is available as RAM on KNL, which leads to three modes of MCDRAM configuration: cache, flat, and hybrid. Different configurations of MCDRAM and DDR memory partitioning between SW addressable and HW managed cache may lead to different memory access performance. Also, three types of clustering modes (All-to-all, Quadrant, SNC (Sub NUMA Clustering) ) are available which configures how memory requests are routed in the mesh.

An overview of KNL for developers is given in [12]. Detailed stream bandwidth[7] of MCDRAM and DDR with comparison to Xeon is shown in Table I.

<table>
<thead>
<tr>
<th>Knights Landing</th>
<th>Xeon E5-2697v4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory type</td>
<td>MCDRAM</td>
</tr>
<tr>
<td>Copy (GB/s)</td>
<td>376.1</td>
</tr>
<tr>
<td>Scale (GB/s)</td>
<td>331.6</td>
</tr>
<tr>
<td>Add (GB/s)</td>
<td>421.4</td>
</tr>
<tr>
<td>Triad (GB/s)</td>
<td>419.6</td>
</tr>
</tbody>
</table>

TABLE I: Measured stream bandwidth of Xeon and Xeon Phi. Xeon: two-socket E5-2697v4; Xeon Phi: 7210 in Flat memory mode and Quadrant cluster mode

As for the chip, there are 36 tiles interconnected with 2D mesh on the chip. Each tile possesses 2 cores and a 1MB L2 cache. There are 2 VPU in each core. The detailed architecture of KNL is shown in Figure 2(b). OPA (Intel® Omni-Path Architecture) is integrated on package as optional high-speed fabrics.
The virtual function is a fantastic feature that C++ provided as an object-oriented programming language. However, the virtual function is implemented with virtual table, which calls the functions depending on the value of the function pointer at runtime. The inline functions are good to performance because it eliminates the process of saving the context (saving the value of registers to memory). On the contrary, the virtual function can not be inlined, and the compiler can not predict the registers used in the virtual function. All the used registers in the current context are saved before the virtual function is called. Virtual function brings more operation than the general function[4]. So all the virtual functions in the particle push part of VLPL-S were changed into general functions.

### V. Methodology

In this section, we first introduce the key features and the performance bottleneck of the VLPL-S code, then show the details of how we apply the optimizations to VLPL-S code in each subsection. The used tools that help identifying the bottleneck of VLPL-S are shown in Table II.

#### A. Basic Optimization

As we all know, square root operation takes much more clock cycles than other operations. Also, square root operation on 512-bit register is more expensive than it on 256-bit register. The original code is wholly developed with single precision floating points. There are several square root operation in the kernel of the VLPL-S code. We find that the single precision variable is extended to double precision before square root operations in the assembly code generated by Intel compiler. The cause is the precision extending policy of the compiler.

As mentioned above, one particle may contribute to different cells surrounding it after every iteration. With OpenMP, particles from two different thread that resides in two adjacent cells may write to the same memory location at the same time. As we mentioned above, particles are stored with linked list in the VLPL-S code with the help of OpenMP. The particles are stored with linked list in the VLPL-S code with the help of OpenMP. We find that the single precision variables and thus refrain the use of the mixed precision expression. To avoid the extending of the single precision variable, we use the single precision static variables and thus refrain the use of the mixed precision expression.

The virtual function is a fantastic feature that C++ provided as an object-oriented programming language. However, the virtual function is implemented with virtual table, which calls the functions depending on the value of the function pointer at runtime. The inline functions are good to performance because it eliminates the process of saving the context (saving the value of registers to memory). On the contrary, the virtual function can not be inlined, and the compiler can not predict the registers used in the virtual function. All the used registers in the current context are saved before the virtual function is called. Virtual function brings more operation than the general function[4]. So all the virtual functions in the particle push part of VLPL-S were changed into general functions.

#### B. Memory Access Optimization

As mentioned above, particles are stored with linked list in the VLPL-S code. Since an iteration over all the particles at each time step is required by the PIC algorithm, the linked list of particles is traversed over and over again during the simulation procedure. The memory of each node of linked list is allocated separately, So the memory access pattern of the traverse over linked list is unpredictable.

Prefetch is a widely used method to reduce the stalls in the pipeline of CPU. Hardware prefetchers and software prefetcher work together to feed the arithmetic processing unit. However, hardware prefetchers can not recognize the pattern of traversing linked list and neither can the Intel Compiler. In order to improve the efficiency of memory access, we prefetch the next particle in the linked list into the cache with the help of Intrinsics. Due to the randomicity of the memory locations of linked list, we do not know the address of next node until we receive the data of the current node. Owing to this feature of linked list, we maintain a Look Ahead Table for each cell. In the VLPL-S code, each cell maintains a linked list of particles that reside in this cell. Thanks to the Look Ahead Table, prefetching with an adjustable distance is possible.

#### C. Thread Level Parallelization

When compared with Xeon, the advantages of KNL is that it has more cores than Xeon though the frequency of KNL is lower than Xeon CPUs. More cores require more processes to fully utilize the computing power. So, higher parallelism is necessary for the state-of-art performance.

The main hotspot of the VLPL-S code is particle push part of the PIC Algorithm which takes more than 80% of wall time. The hotspot function, every particle can be processed in parallel since the calculation only depends on the data of particle itself and the cell around it. The current generated by the particle moving process is accumulated to the cells around the particles. One particle may write to several different cells. So, data hazard should be taken care of.

After each time step of the iteration, a synchronization is necessary since particles may move outside the owning process. The data of fields on the boundary of each process also needs to be transferred to its neighbors. So, a lot of MPI communication should be done after each time step. With the help Intel ITAC profiling tool, we found that a significant portion of time is consumed by the MPI package. That implies the communication or the load imbalance becomes the bottleneck of the program. To reduce the communication overhead on MPI level and improves the load balance on thread level, we apply TLP (Thread Level Parallelism) to VLPL-S code with the help of OpenMP.

As mentioned above, one particle may contribute to different cells surrounding it after every iteration. With OpenMP, particles from two different thread that resides in two adjacent cells may write to the same memory location at the same time. We proposed three methods to deal with the data hazard.

<table>
<thead>
<tr>
<th>Package Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ compilers</td>
<td>Intel® Trace Analyzer and Collector</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier 2016</td>
<td>Intel® Advisor</td>
</tr>
</tbody>
</table>

Table II: Compiling tools and performance analyzing tools
1) **Atomic operation**: The OpenMP package provides us the atomic directive that specifies a memory location must be updated atomically rather than letting multiple threads attempt to write to it. In essence, this directive provides a mini-CRITICAL section[1]. As the particles only contribute to the surrounding cells, the non-boundary cells of each thread do not suffer the data hazard. The atomic operation is applied on the boundary only. The peeling scheme of atomic operations is shown in Figure 3(a).

![Atomic operations with peeling](image)

Fig. 3: Scheme of the methods avoiding data hazard of thread level parallelism

2) **Passive contribution**: In the original code of VLPLS, the currents generated by the movement of particles are directly writing to the destination. This method is called active contribution in terms of the particles. With the active contribution method, the current is accumulated to the field data right after it is computed for each cell. With the passive contribution method, the current is stored in the particle data and then waiting for another loop over the cells to pull the current from all of the possible particles to the field data. So, a buffer storing the current is necessary for each particle. An illustration of the active and passive contribution method is shown in Figure 3(b).

![Active and passive contribution](image)

3) **Buffering and reduction**: This method is developed based on the Passive contribution method. In order to store the intermediate value of the current generated by the movement of particles, 3 variable of 32 bit are added to the particle struct. As there are millions of particles simulated, those variables and the corresponding operations on them lead to a significant increasement of the total memory consumption and the total memory access operations. The buffering and reduction method improves the passive contribution method by accumulating the generated current to the cell data. Thus the buffering variable is only necessary to be added to the cell struct. The number of cells is much smaller than the number of particles. Thus the total number of memory access is reduced compared to the Passive contribution method. A brief illustration of the buffering and reduction method is shown in Figure 4.

![Buffering and reduction](image)

Fig. 4: Scheme of buffering and reduction which avoids the data hazard

On the MPI level of parallelization, different processes hold the same number of cells. Due to the frequent movement across cells, particles may distribute disproportionally among the cells. On the thread level of parallelization, the nested loop over x and y axes are unrolled, and the cells containing a different number of particles are distributed to different threads dynamically with the help of OpenMP dynamic directive. Due to the randomness of the particle distribution, the method of peeling in the atomic method is no longer valid.

**D. Vectorization**

One of the most attractive features of KNL is that there are two VPU (Vector processing unit) of 512bit in each core. Applications targeting the state-of-art performance on KNL never lack efficient vectorization. In this section, we show an interim method of vectorization of the linked list data structure. The evaluation of this method and further discussion of efficient vectorization will be shown in Section VI.

The linked list data structure is designed for a list of objects that are inserted and deleted frequently. The advantage it brings is the O(1) cost of inserting and deleting operation at any position of the linked list. However, the randomness of memory access it brings is fatally expensive in HPC applications.

![Linked List](image)

![Packed data](image)

Fig. 5: An example of the packing of 4 linked particles

To take full advantage of the 512-bit wide vector processor and have a quick evaluation of the performance gain, the
interim method of vectorization which reads and packs the data of linked list is adopted. An example of packing four nodes of linked list is shown in Figure 5. Originally, though \( m, q \) and \( p \) values of each particle are stored one after another, \( m \) values of the four particles are separated in the memory which is not capable of vectorization. 32-bit float point value is fetched from linked particles until the vector of enough width is formed. After reading the data of four particles, \( m \) values as well as \( q \) and \( p \) values are packed together and stored in an aligned array. Therefore, 512-bit vector instructions can be used to process the packed vector of data. The VLPL-S code uses single precision floating points, so 16 particles are processed on KNL in each iteration of the inner loop rather one particle of the linked list.

### VI. RESULTS AND DISCUSSION

#### A. Workload statement and machine configuration

Two different workloads are involved in the evaluation part. The detailed information about the workloads is shown in Table III. The testA is a workload designed mainly for performance evaluation. The distribution of the particles in testA is uniform, so testA shows good load balance. Also, the simulated space is a square, thus makes testA suitable for most of the domain decomposition configuration. The testB is a workload used for practical researches. The distribution of the particles in testB is uniform at the beginning of the simulated time and then become imbalanced after several time steps. This workload suffers load imbalance which turns the performance comparison between platforms into the comparison of single core performance. Also, the simulated space is short in height and thus brings higher communication overhead when the simulated space is partitioned into more than 60 parts vertically.

The VLPL-S code is evaluated on three platforms which are shown in detail in Table IV. Different memory mode and cluster mode of the Knight Landing is used in the evaluation part. The detailed configuration will be shown in company with running time of VLPL-S later. This workload suffers load imbalance which turns the performance comparison between platforms into the comparison of single core performance. Also, the simulated space is short in height and thus brings higher communication overhead when the simulated space is partitioned into more than 60 parts vertically.

The VLPL-S code is evaluated on three platforms which are shown in detail in Table IV. Different memory mode and cluster mode of the Knight Landing is used in the evaluation part. The detailed configuration will be shown in company with running time of VLPL-S later. The compiling option used for KNL is “-O3 -xMIC-A VX512”, and “-O3 -xCORE-A VX2” is used for CPUs.

#### B. Evaluation of Knights Landing with original VLPL-S code

One improvement of Knights Landing compared with Knights Corner is its binary compatibility with Xeon. As recommended by Intel compiler team, we recompile the original program with -xMIC-A VX512 for best optimization. In order to evaluate the performance of primitive applications on KNL, a brief evaluation of the original VLPL-S code on KNL is shown in Figure 6.

![Fig. 6: A preview of performance on KNL](image)

The performance of different cluster mode and different memory mode is almost the same for original VLPL-S code. The performance gap of VLPL-S running on MCDRAM or only DDR is obvious. When scaling from 128 processes to 256 processes, speedup is considerable on MCDRAM no matter with which clustering mode. However, VLPL-S does not scale with more than 128 processes on DDR. The cause is that MCDRAM provides much higher bandwidth than DDR. The performance is bounded by the bandwidth of DDR when running with 256 processes.

#### C. Evaluation of the optimizations

The performance gain of the optimizations is shown in Figure 7. Memory mode is Flat, and cluster mode is Quadrant. 256 processes are used on Xeon Phi 7210 and 36 processes are used on Xeon E5-2699v3. VLPL-S running with 256 processes achieves the best performance on KNL because there is enough memory bandwidth provided by MCDRAM and there are two VPU’s in each core. The single precision scalar program VLPL-S won’t bring very high pressure to the VPU. So, VLPL-S scales on single-node KNL with the help of hyper-threading. When it comes to CPU, since the memory bandwidth is limited and there is only 1 VPU for each core, the best performance is achieved with 36 processes just as the number of cores.
Fig. 7: The performance gain of the optimizations on different platforms compared to the baseline version. The former optimizations are applied to all the latter versions. (Xeon: E5-2699v3, Xeon Phi: 7210)

1) Basic Optimization: As shown in Figure 7, the speedup of basic optimization is higher on KNL than on CPU because KNL benefits more from the optimization of removing the virtual functions due to the higher cost of context switch on KNL. Also, the speedup from basic optimization is greater for testA than for testB. The reason is particles moves less frequent in testA than testB, thus makes testA focus more on computing which benefits from the basic optimization.

2) Memory Access Optimization: With the help of Intel VTune™, hardware events are collected during the running. As shown in Figure 8 (Evaluated on the 2-socket E5-2699v3 node), cache miss rate is reduced. The prefetch optimization works both on CPU and KNL. For testA, the performance gain on KNL is 1.21x speedup which is higher than it on Xeon because far more processes are used on KNL and MCDRAM provides higher memory bandwidth. For testB, prefetch mainly helps the heavier loaded cores due to the load imbalance issue. So the performance gain of testA on KNL compared to CPU is higher than it of testB.

3) Thread Level Parallelism: As for the Optimization of TLP, the performance gain (shown in Figure 7) on single node is considerable. For testA, 1.14x speedup is achieved on KNL and 1.05x speedup on CPU (E5-2699v3). For testB, 1.27x speedup is achieved on KNL and 1.11x speedup on CPU. Higher performance gain is achieved on KNL compared to CPU for both workloads. Without OpenMP, the optimal performance is achieved with 256 MPI processes with hyper-threading enabled on KNL and with 36 MPI processes on two-socket CPU. The thread level parallelism shares less hardware with hyper-threading and thus more MPI overhead is reduced on KNL. Also, higher performance gain is achieved for testB than testA for both platforms. The cause is that dynamic scheduling is applied on the thread level, so the load imbalance issue of testB is relieved on thread level.

Nevertheless, the MPI + OpenMP hybrid mode greatly improves the scalability of VLPL-S on multiple KNL nodes. The evaluation was performed at the Intel Endeavor system which is connected with OPA fabrics. Both BDW nodes and KNL nodes are connected with OPA Switch. As shown in Figure 9, the strong scalability of VLPL-S on BDW cluster is higher than linear. The more nodes the application scales, the higher memory bandwidth is provided and the less memory consumption is required on each node. Therefore, data locality is improved when VLPL-S scales to more nodes.

For the testA used here, the simulated space contains 1200 × 1200 cells and there are 16 particles in each cell. The total memory consumption is 1200 × 1200 × (sizeof(Cell) + 16 × sizeof(Particle)), which is 1735.83GB in total. When VLPL-S scales to 32 two-socket E5-2697v4 nodes, 2880MB (32 × 2 × 45MB) of L3 cache is provided in total which is larger than the memory requirement. This explains the excellent scalability of VLPL-S on Broadwell cluster.

With the help of Intel Trace Analyzer and Collector, detailed time of each MPI operation is collected. The MPI_Bcast which is the only collective operation in VLPL-S takes far more time than other MPI operations especially on more than 4 KNL nodes. Since the MPI_Bcast in VLPL-S is used to synchronize a 32-bit integer which indicates the running status of each MPI process, removing this broadcast operation does not affect the...
correctness of the VLPL-S. The performance of VLPL-S code after removing MPI_Bcast as a test is far higher than it with the MPI_Bcast as shown in Figure 9. A detailed performance evaluation of MP_Bcast on KNL nodes with the comparison to it on Xeon nodes is shown in Figure 11. Both the CPU and KNL clusters are connected with OPA switch. The average time of MPI_Bcast on KNL cluster is far higher than it on Xeon cluster. Also, the performance of MPI_Bcast drops greatly with hyper-threading enabled as shown in Figure 10. The cause might be the hardware sharing with hyper-threading enabled.

4) Vectorization: The interim vectorization method reduces the performance on all platform especially for testA on KNL. The cause is that the memory access time and the computing time is no longer overlapped after applying the pack operation. To identify the performance issue, the cache miss rate is measured and shows an increase after the vectorization optimization. Originally, the optimization of prefetch improves the performance significantly for testA on KNL. 16 particles are packed into the vectorized dataset before computing and therefore the hidden latency of memory access is drawn out. So, the performance of testA on KNL, which benefits most from prefetch, drops most after the vectorization.

Clearly, the performance of the current version of VLPL-S is far from the state-of-art performance on KNL. Changing the data structure to Struct of Array (SOA) is necessary to overlap the latency of memory access and fully utilize the 512bit VPU.

D. Summary

In summary, for the performance dedicated workload testA, the performance on Knights Landing 7210 is 1.77x faster than it on two-socket Xeon E5-2699v3 and 1.53x faster than it on two-socket Xeon E5-2697v4 after applying the optimizations mentioned in Section V. The performance of the optimized code is shown in Figure 12. 64 MPI processes each with 4 threads achieves the best performance on KNL and 18 processes each with 2 threads achieves the best performance on CPU. However, the performance on Xeon Phi 7210 is lower than it on two-socket E5-2697v4 for testB. The cause is the load imbalance of testB which turns the comparison of the different platform into the comparison of single core performance. The frequency of Xeon Phi is much lower than the Xeon node, thus explains the performance gap.

Though the detailed speedups are different for different workloads and platforms, the optimizations work on both Xeon CPU and KNL. Since the performance of MPI on KNL seems lower than it on Xeon with OPA fabrics, the scalability of VLPL-S on one KNL node is still acceptable even with hyper-threading enabled. Regarding the lower performance of MPI, thread level parallelism is a must for VLPL-S to achieve higher performance.

In conclusion, the commonly used optimization methods work on the new generation of Xeon Phi™ as well as Xeon. The MCDRAM greatly improves the performance of VLPL-S since it provides much higher memory bandwidth. The

VII. CONCLUSION AND FUTURE WORK

In conclusion, the commonly used optimization methods work on the new generation of Xeon Phi™ as well as Xeon. The MCDRAM greatly improves the performance of VLPL-S since it provides much higher memory bandwidth. The
MPI+OpenMP hybrid mode achieves the best performance on single node and multiple nodes.

Since the interim vectorization method does not improve the performance, a complete rewriting of linked list to SOA (Struct Of Array) is necessary to achieve the state-of-art performance. This part of work is almost finished, and it will be discussed after evaluated and analyzed. Based on the SOA version of VLPL-S, more interesting works targeting the MCDRAM and 512-bit VPU are ready to go.

One interesting thing is the performance of Intel MPI implementation on KNL is lower than it on CPU nodes especially for the collective MPI operations. The cause might be the immature implementation of MPI library for KNL. Detailed research and discussion will be done in future.

VIII. ACKNOWLEDGEMENTS

Thanks to Prof. Pukhov for the original VLPL code and his help during this work. Also, thank Zhe Wang and other members of the HPC group in Intel Asia-Pacific Research and Development Ltd. for their suggestions on the optimization methods.

This research is partially supported by China National High-Tech R&D Plan (863 plan) 2014AA01A302, 2016YFB0201800. James also greatly acknowledges support from Japan JSPS RONPAKU (Thesis PhD) Program.

REFERENCES


