Preliminary Investigation of Accelerating Molecular Dynamics Simulation on *Godson-T* Many-core Processor

Liu Peng[1], Guangming Tan[2], Rajiv K. Kalia[1], Aiichiro Nakano[1], Priya Vashishta[1], Dongrui Fan[2] and Ninghui Sun[2]

[1]Collaboratory for Advanced Computing and Simulations, University of Southern California, USA
Email: {liupeng, rkalia, anakano, priyav}@usc.edu

[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
Email: {tgm,fandr,snh}@ict.ac.cn

Abstract

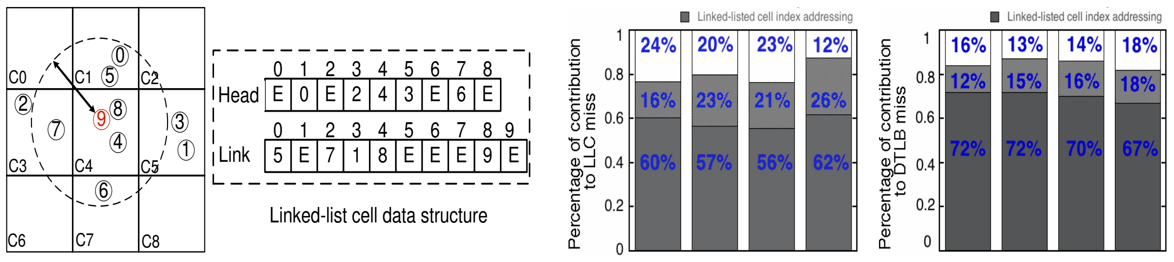
Molecular dynamics (MD) simulation is widely used in computational science, however, its irregular memory-access pattern imposes great difficulty on performance optimization. This paper presents a joint application/architecture study to accelerate MD on an emerging unconventional computing platform–*Godson-T* many-core architecture. We propose three incremental optimizations: (1) a divide-and-conquer algorithm adaptive to on-chip memory; (2) a novel data-layout to re-organize linked-list cell data structures to improve data locality; (3) an on-chip locality-aware parallel algorithm to enhance data reuse. Experiments on an event-driven, cycle-accurate *Godson-T* simulator achieve excellent speedup of 62 on 64 cores.

1 Introduction

Molecular dynamics (MD) simulation is widely used to study material properties at the atomistic level. But increasingly large computing power is needed to satisfy the spatiotemporal scale of the real world simulations. The advent of many-core paradigm has provided unprecedented computing power, and promises to enable large-scale and long-time simulation only if we can efficiently harvest the computing power. Challenges to achieve efficient parallel MD algorithm mainly on many-core platform arise from two aspects: (1) MD application is characterized by irregular memory access which imposes difficulty on locality optimization; (2) many-core hardware limitation (volume of on-chip memory, bandwidth of on-chip networking, etc.) constrains the size of working-set per core which imposes difficulty on on-chip parallelization. To address these difficulties, this paper presents a joint study from both application and architecture aspects on how to accelerating MD on *Godson-T* emerging many-core architecture, where we map an MD algorithm to architecture for achieving high on-chip parallel efficiency.

The main contribution of this paper are:(1)An adaptive divide-and-conquer (ADC) algorithm is designed to optimize the use of memory hierarchy; (2)A novel data layout is employed to re-organize linked-list cell data structures to maximize data locality;(3) An on-chip locality-aware parallel algorithm is designed to maximize data reuse;(4)Detailed experiments on an *Godson-T* simulator shows the optimized MD achieves a strong-scaling parallel efficiency 0.92 on 64 cores.

The rest of this paper is organized as follows. Section 2 briefly introduces the DC-MD algorithm used in this work, where key performance bottlenecks are summarized. Section 3 highlights the main archi-



(a) (b) (c)

Figure 1: (a)2D schematic of the linked-list cell method and data structure. C_{0-8} are the indices of cells; number in a circle represents index of an atom; E represents the end of the listed atoms in a cell;(b)and (c) are Percentages of events that cause last level cache (LLC) miss and data translation look aside buffer (DTLB) miss for the original DC-MD algorithm on an Intel quadcore core i7 920 platform measured by Intel VTune Performance Analyzer.

tectural features of *Godson-T* , and section 4 describes our optimization strategies. Section 5 presents the experimental results and detailed analysis. Finally, section 6 concludes the paper.

2 A Divide-and-conquer MD Algorithm

MD simulation follows the phase-space trajectories of an N-atom system, where force fields describing the atomic force laws between atoms are spatial derivatives of a potential energy function $E(r^N)$ ($r^N = \{r_1, r_2, ..., r_N\}$ are positions of all atoms). The most commonly used algorithm in parallel MD simulation is spatial decomposition, where the simulation system is partitioned into subsystems of equal volume, and atoms located in a particular subsystem are assigned to one of the processors in a parallel computer, which are logically arranged according to the topology of the simulation subsystems (e.g. 3D mesh). In order to compute interatomic interaction with cutoff radius r_c at each MD step, atomic coordinates of 26 neighbor subsystems, which are located within r_c from the subsystem boundary, are copied to each processor, where data coherence is maintained by copying the latest neighbor surface atoms every time before atomic accelerations are computed. The periodic boundary condition is applied to the system in three Cartesian dimensions.

We have previously proposed a space-time multiresolution MD (MRMD) algorithm to reduce the $O(N^2)$ time complexity of potential evaluation to $O(N)$[1]. In the MRMD, $E(r^N)$ consists of two-body $E_2\{r_{ij}\}$ and three-body $E_3\{r_{ijk}\}$ terms within a cutoff radius r_c. In the linked-list cell method, the dimension R_c of the cells is usually chosen to be larger than r_c. For a given atom in a cell, the search space for interacting neighbor atoms is limited to the 26 nearest neighbor cells. Figure 1(a) shows a schematic of the computational kernel of MD in 2D. Conventional summation rule to compute the three-body interaction is written as $E_3(r_{ijk}) = \sum_{i=1}^{N} \sum_{j=1}^{nbr(i)} \sum_{k \neq i}^{nbr(j)} v(r_i, r_j, r_k)$, where r_i is the coordinate of the $i - th$ atom and $nbr(i)$ is the list of neighbor atoms within the three-body cutoff length from atom i, which acts as the center of atomic triplet (j, i, k). To maximally exploit parallelism in a multi-core cluster, our EDC-STEP-HCD scheme has employed a multi-level parallelization strategy[2, 3]. Although this scheme has achieved internode parallel efficiency well over 0.95 for 218 billion-atom MD simulation on $212,992$ BlueGene/L processors[2], it suffers inefficient on-chip parallelism with on-chip parallel efficiency only 0.65 for 8 threads on a dual Intel quadcore Xeon SMP platform[3]. We observe several features that prevent the program from achieving high performance on conventional multi-core architectures: (1)*Irregular memory access*. Straightforward or sparse-matrix-like implementation of linked-list data structure as shown in Fig. 1(a) leads to irregular mem-

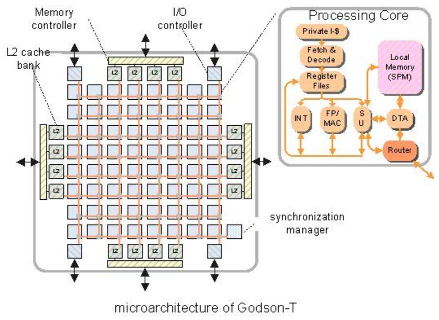


Figure 2: *Godson-T* architecture: INT is fixed point arithmetic unit, FP/MAC is floating point unit, and CU is communication unit.

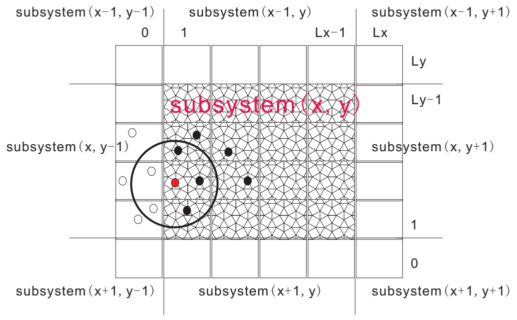


Figure 3: Cellular decomposition scheme for on-chip parallelization. Cached atoms and resident atoms are shown in white and shaded boxes.

ory accesses in three-body interaction calculation; (2)*High latency to access shared data*. Let N_c denote the number of cells in each Cartesian direction, q the atom density(number of atoms divided by system volume $(N_cR_c)^3$, then the total number of cross-cell pairs in the system is given by $N_c^3 \times qR_c^3 \times 26qR_c^3$. Thereby the size of atomic data for interaction computation easily exceeds that of the last level cache. Combining with the irregular memory access mentioned before, the latency problem becomes even worse, as evidenced by the memory accessing performance in Fig. 1(b) and Fig. 1(c) conducting on Intel quadcore core i7 920 platform measured by the Intel VTune Performance Analyzer. Therefore, it is of great significance to optimize the memory accessing to improve the performance and scalability of our MD application on many-core platforms.

3 *Godson-T* Many-core Architecture

Godson-T is a low-power many-core architecture developed by Institute of Computing Technology, Chinese Academy of Sciences to serve as a dedicated petaflops computing engine. As shown in Fig. 2, *Godson-T* has 64 homogeneous, dual-issue and in-order processing cores running at 1 GHz, where a floating-point multiply-accumulate operation can be issued to a fully-pipelined function unit in each cycle, resulting in a peak floating-point performance of 128Gflops. The 8-pipeline processing core supports 32-bit MIPS ISA (64-bit ISA will be supported in latter version) with synchronization instruction extensions. Key architectural features for achieving decent scalability and high performance include the following[4]:

- *Fine-grained parallelism*[5]. Each core works as a lightweight hardware thread unit executing in a non-preemptive manner. A dedicated synchronization manager (SM) is a centralized unit to collect and handle synchronization requests, which provides architectural support for fast mutual exclusion, barrier and signal/wait synchronization. In addition, an extremely efficient thread execution runtime system has been developed to manage thread execution[5, 6].
- *Locality-awareness*. Each processing core has a 16KB 2-way set-associative private instruction cache and a 64KB local memory (like data cache). As inspired by IBM CELL and Nvidia GPU, an explicit memory hierarchy is implemented for user to exploit better locality with less complex hardware implementation compared to that of the traditional transparent memory hierarchy. Moreover, in *Godson-T* , each local memory is configured as explicitly-controlled, globally-addressed scratched-pad memory (SPM) to further help programmer maximize locality. An 8×8 packet-switching 2D mesh network connects all on-chip units with a 128bit bandwidth employing deterministic X-Y routing policy, which can provide a total of 2TB/s on-chip bandwidth among 64 processing cores. In addition, there are 16 address-interleaved L2 cache banks (256KB each) distributed along the perimeter of the chip,

which are shared by all processing cores and can serve up to 64 cache accessing requests in total. The bandwidth between SPM and L2 cache is 256GB/s, and each four L2 cache banks on the same side of the chip share a memory controller with a 25.6GB/s memory-accessing bandwidth.

- *Latency tolerance*[7]. Since there may exist intensive contention on on-chip network and memory controller, latency to L2 cache will possibly become primary obstacle to achieve decent performance. To address this issue, a DMA (direct memory accessing)-like coprocessor Data Transfer Agent (DTA) is built in each core to do fast data communication, that is, when one core is doing calculations, DTA can be programmed to manage various data communications at backend in parallel.

4 Optimizations on *Godson-T*

In this section, we describe how to design an efficient MD algorithm based on the features provided by *Godson-T* many-core architecture.

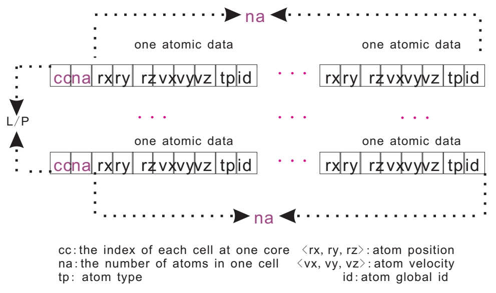
4.1 Adaptive Divide-and-Conquer Algorithm

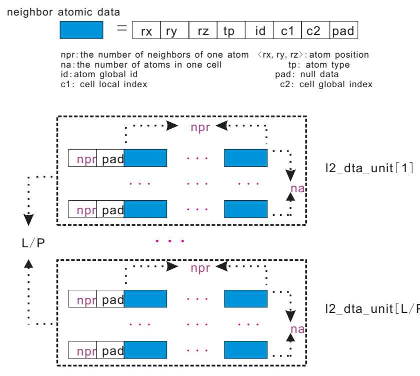
In the original DC-MD algorithm, the physical space is subdivided into spatially localized cells, with local atoms constituting subproblems. The algorithm recursively divides a coarse cell into finer cells until some criterion is satisfied. In our adaptive divide-and-conquer (ADC) algorithm specially designed for many-core architectures, we use the size of the first level memory (i.e. private local memory), C_{pm}, as a critical factor of the criterion.

Since this paper only addresses the issue of fine-grained parallelism within a subsystem, we assume that the interchange of cached atoms (atoms near subsystem boundaries) has been completed by a higher-level parallelism (e.g. using MPI) among multiple many-core computing processors[2]. Figure 3 illustrates of the cells in a subsystem. The algorithm divides the subsystem consisting of the resident and cached atoms into small cells of equal size. Assume that there are $P = P_x \times P_y \times P_z$ cores in a many-core processor and that the number of cells in a subsystem is $L = L_x \times L_y \times L_z$. Then each core i processes $\frac{L}{P}$ cells as Eq. **??** (since how to efficiently embed 3D mesh into 2D one has already been solved by classical algorithms [8], the 2D on-chip network on *Godson-T* is viable for this decomposition): $\{(c_x, c_y, c_z) | c_x \in [\frac{iL_x}{P_x}, \frac{(i+1)L_x}{P_x}), c_y \in [\frac{iL_y}{P_y}, \frac{(i+1)L_y}{P_y}), c_z \in [\frac{iL_z}{P_z}, \frac{(i+1)L_z}{P_z})\}$. Let B denote the memory space for storing one atomic data, N_b denote the number of neighbor atoms per cell. In order for all the resident atomic data to fit in the private local memory, R_c should satisfy

$$qR_c^3 \times (N_b + \frac{L}{P}) \times B \leq C_{pm} \Rightarrow R_c \leq \sqrt[3]{\frac{PC_{pm}}{(PN_b + L)Bq}}. \quad (1)$$

ADC algorithm performs recursive cellular decomposition until Eq. 1 is satisfied. When the atomic data are distributed into each core's local memory, they are reused in both two- and three-body force calculations, as the software controlled SPM provides a mechanism for user to decide what data locate in the private local memory and when. However, a direct implementation based on the linked-list cell data structure is not efficient enough because of MD's irregularity. In the next subsection, we propose a novel data layout optimization to address this problem.


na
one atomic data
one atomic data
cc na rx ry rz vx vy vz tp id
rx ry rz vx vy vz tp id
L/P
na
cc:the index of each cell at one core
na:the number of atoms in one cell
tp: atom type
<rx, ry, rz>:atom position
<vx, vy, vz>:atom velocity
id:atom global id
(a)



(b)

Figure 4: (a) The atomic data of cells in a core's SPM. (b)The neighbor atomic data of cells in the shared L2 cache or off-chip memory. Pad is used for address alignment. Each L2_data_unit[i] represents a contiguous block of all neighbor atomic data for the $i-th$ cell in L2 cache or off-chip memory $(i = 1, ..., L/P)$.

4.2 Data Layout Optimization

At the beginning of MD simulation, each atom is assigned an integer in [0...$N-1$], which is used as an identifier for the linked-list based algorithm to access atomic data. We refer to this method as *global-ID-centered* addressing. However, during the simulation, the identifiers cannot be kept contiguous due to atom migration between computing nodes/processors. In the ADC algorithm, it is expected that the atomic data is distributed among different cores, where each cell only interacts with 26 neighbor cells. Therefore, if all the atomic data within one cell were grouped together, they would be easily reached through its cell index. Here, we propose a new strategy—*cell-centered* addressing.

Figure 4(a) depicts the data structure designed for the atomic data in SPM, where na denotes the maximum number of atoms in one cell. Since all the atomic data for each cell are grouped, the cell index (cc) can be used to search the neighbor cells, and then the atomic data in each cell can be touched contiguously. Moreover, considering the sequential mapping between cores and cells, the searching of neighbor cells is completed in $O(1)$ time: The scalar value of cc is transformed into a vector (cc_x, cc_y, cc_z), then the neighbor cell index is calculated by the combination of $\{(cc_x + l_x, cc_y, +l_y, cc_z + l_z)|l_x, l_y, l_z \in \{-1, 0, 1\}\}$, where the number of neighbors including itself is 27.

As was shown in section 2, the number of cross-cell atom pairs is $qR_c^3 \times 26qR_c^3$ per cell, and it is impossible to store all the data in each core's private memory. Since the three-body interaction calculation also involves atoms in one cell and its 26 neighbor cells, it can also benefit from the *cell-centered* addressing for contiguous accessing of atomic data in a cell. Similarly, we group the neighbor atoms as well. Figure 4(b) depicts the data layout of neighbor atoms located in L2 cache or off-chip memory, where we group all the neighbor atomic data for the i-th cell together as L2_data_unit[i], which can be transferred to the SPM through a DMA like operation that utilizes high bandwidth provided in *Godson-T* .

4.3 On-chip Locality Aware Parallel Algorithm

In this subsection, we present our on-chip locality aware parallel algorithm to enhance data reuse and further to alleviate the long latency to access the shared neighbor atomic data in L2 cache or off-chip memory. The two-body force calculation involves core-core communication, and it may cause on-chip network congestion. Moreover, the access to L2 cache also goes through the on-chip network, which may introduce more congestion. Here, we propose a solution to enhance the data reuse and to reduce the remote shared-data memory accessing, thereby alleviating the long latency. Suppose that the atoms in a cell at core i interacts

```
1.  for each cell $c_{core\_i}[k]$
2.    for each interacting cell $cj$ with cell $c_{core\_i}[k]$
3.      append $c_{core\_i}[k]$ to set $PC[cj]$
4.    endfor
5.  endfor
6.  for each pair set $PC[cj]$
7.    if $cj$ is not at core $i$
8.      load the atomic data of $cj$ into core $i$'s local memory
        through intercore communication via on-chip network
9.    endif
10.     for each cell $c_{core\_i}[k]$ listed in $PC[cj]$
11.       calculate atomic interactions between $c_{core\_i}[k]$ and $cj$
12.   endfor
13. endfor
```

Figure 5: Algorithm for calculating interatomic interactions while achieving on-chip locality for core i. Here $c_{core\_i}[k]$ is the k-th cell assigned to core i.

with those in another cell at core j. In order to achieve on-chip locality for core i, we maximize the data reuse of cells from core j, and vice versa for core j. Our solution is first to construct a set of cell pairs $PC[cj] = \{ci0, ci1, ., cik\}$, where cj is the global index of the cell interacting with cells $ci0, ..., cik$ in core i. For example, suppose that core i has cells $\{1, 4, 8\}$, and core j has cells $\{2, 5\}$. Also assume that cell 2 interacts with $\{1, 4\}$ and that cell 5 interacts with $\{4, 8\}$. Then we construct two cell pairs $PC[2] = \{1, 4\}$ and $PC[5] = \{4, 8\}$. We then use the core-core communication to transfer the atomic data according to the cell pairs from core j to core i. The algorithm running on each core i is shown in Fig. 5. If more than one cell are assigned to a core, then some neighbor cells are located in the same core, and thus no core-core communication is required (see lines 7-9 in Fig. 5).

The algorithm in Fig. 5 uses a preprocessing to collect the set of cell pairs. Since each cell only interacts with its 26 surrounding neighbors, the size of set PC is expected to be less than $O(\frac{L}{P} \times 26)$. Since the calculation of neighbor's indices is done in $O(1)$ time using *cell-centered* addressing, the preprocessing requires $O(\frac{L}{P})$ time and $O(\frac{L}{P})$ space, which is negligible compared with the two-body interaction time $O(\frac{L}{P} \times qR_c^3 \times 26qR_c^3)$.

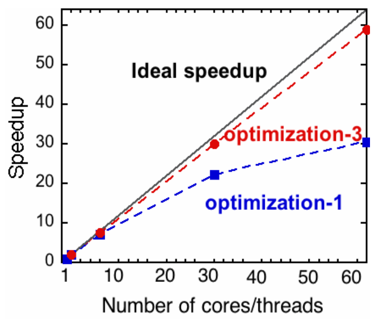
5 Evaluation

In this section, we present the experimental results and detailed analysis of the proposed MD optimization on a *Godson-T* many-core simulator.

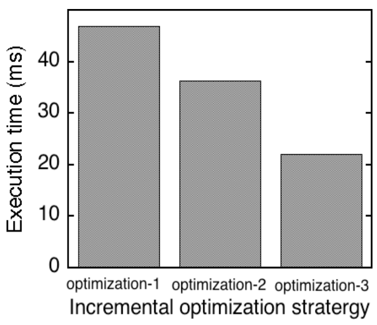
Godson-T is an on-going research project for building a petaflops supercomputer, and a real chip is expected to be shipped in late 2010. In order to evaluate its performance at early stage of the architectural development, we here employ an instruction-level simulator. The *Godson-T* simulator is event-driven, cycle-

Table 1: *Godson-T* simulator parameters

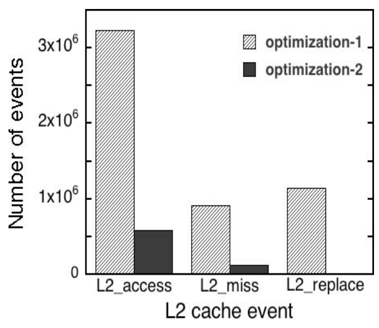
function unit	parameter
core	64 cores running at 1GHz, dual-issue, load-to-use latency=3 cycles, FMAC latency=4 cycles
SPM	64KB, 16 64-bit-width SRAM sub-banks with 2 memory ports each 1 cycle for load and store.
L2 cache	16 banks, 4MB in total, 8-way set-associative, 64B/cacheline, 4 cycles for contentionless and hit request.
memory controller	4 memory controllers, running at 1GHz. 64-bit FSB. Each memory controller controls one DRAM bank.
Off-chip memory	4GB. DDR2-800 DRAM clock is 400MHz. tCAS = 5, tRCD = 5, tRP = 5, tRAS=15, tRC = 24.
network	2-D mesh. Wormhole routing. 2 cycles contentionless latency per hop.
synchronization	$6 \sim 66$ cycles.



(a)



(b)



(c)

Figure 6: (a)Speedup as a function of the number of cores; (b)Execution time in milliseconds on 64-core *Godson-T* ; (c)L2 cache performance for all 16 banks in total. the L2_replace is 1976 for *optimization-2*.

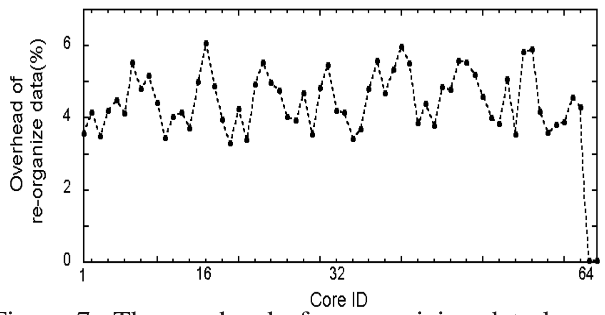


Figure 7: The overhead of re-organizing data layout on each core of *Godson-T* , which at most accounts for 6% of the entire execution time.

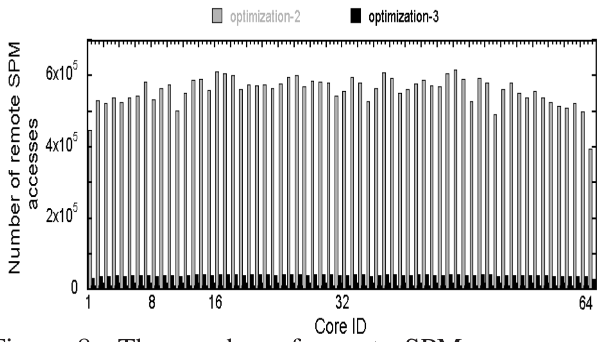


Figure 8: The number of remote SPM accesses on each core of *Godson-T* .

accurate, executing both kernel and application codes, and has modeled all architectural features introduced previously. Since it is an instruction-level simulator, it can produce detailed traces and instruction mix of all executed instructions for any given application after execution. The toolchain on *Godson-T* consists of a gcc-3.3 compiler and a thread execution runtime system, which provides a POSIX thread-like API. The configuration of the simulator is summarized in Table 5. Moreover, since our ultimate objective is to build a large-scale parallel computer, where on-chip parallelism is of critical importance, we mainly focus on on-chip parallelization with a fixed problem size, and the speedup on p cores is calculated by $S(p) = \frac{Time_{one\_core}}{Time_{p\_cores}}$, where $Time_{p\_cores}$ represents the executing time on p cores and $Time_{one\_core}$ represents that on one core. And strong-scaling parallel efficiency $E(p)$ is then defined as $E(p) = \frac{S(p)}{p}$ for a fixed problem size. In the following experiments, the MD simulation tested is for a silica system [2]. Within the DC framework, the whole system is divided into subsystems each containing $24,000$ atoms as the fixed problem size for strong scalability analysis.

The experiments compare three incrementally improved versions of MD algorithm:(1)*optimization-1*–Implementation of ADC algorithm; (2)*optimization-2*–Implementation of ADC algorithm and data layout optimization; (3)*optimization-3*–Implementation of ADC algorithm, data layout optimization and on-chip locality-aware algorithm.

First we test the scalability of the parallel algorithms. Figure 6(a) shows that *optimization-3* makes MD scale excellently with an on-chip strong-scaling parallel efficiency 0.92 on 64 cores while *optimization-1* begins to deteriorate when the number of cores exceeds 32. It tells that optimizations which take advantage of architectural features to maximize data locality and exploit data reuse benefit scalability most, which is also evidenced by the running time in Fig. 6(b) as *optimization-3*reduces the execution time by a factor of 2.

To understand the reason behind the performance gain, Fig. 6(c) compares the number of L2 cache events: the numbers of L2 cache accesses, L2 cache misses and L2 cache replaces are all much smaller with *optimization-2* than those with *optimization-1*. Compared to the original linked-list cell data structure, the data layout optimization leads to more contiguous memory access, which greatly improves L2 cache usage.

Further Fig. 7 plots data layout re-organization overhead: the scheme is light overhead, with the overhead lower than 6%.

Finally, to quantify the advantage of achieving on-chip locality, the simulator collects statistical data of remote SPM accessing. Figure 8 shows that *optimization-3* reduces the number of remote SPM accesses to around 7% of that of *optimization-2*. This explains the significant decrease of the execution time by *optimization-3* in Fig. 6(b). For a given atomic data, the number of reuse can be estimated as follows. Assume that the size of pair set $PC[cj]$ in Fig. 5 is m, then the algorithm needs to calculate interactions between one atom from cell cj and all atoms in m local cells. According to the notation in section 1, each cell contains $q{R_c}^3$ atoms. Therefore, one atomic data may be reused by $mq{R_c}^3$ atoms, i.e., the algorithm may reduce the number of remote SPM accesses by $mq{R_c}^3 - 1$.

6 Conclusion

The emergence of many-core architecture has provided unprecedented computing power to computational scientists, and it is of great significance to exploit the computational power of such new platforms to improve the performance and scalability of large-scale scientific applications. In this paper, we have described our investigation of accelerating MD simulation on representative many-core architecture *Godson-T* . We have proposed a divide-and-conquer algorithm adaptive to the memory hierarchy to facilitate the on-chip local memory, a novel data layout to improve data locality to alleviate the irregular memory accessing, an on-chip locality-aware parallel algorithm to enhance data reuse to amortize the long latency to access shared data. These techniques have made the parallel MD algorithm scale nearly linearly with the number of cores. Also we have found that the data locality and data reuse schemes taking advantage of explicit memory architecture and high-bandwidth on-chip network are essential to achieve high scalability. The contribution of this work lies not only in giving application scientists advice on how to optimize their applications utilizing architectural mechanisms, but also in guiding future hardware developments.

References

[1] A. Nakano, R. K. Kalia, P. Vashishta, T. J. Campbell, S. Ogata, F. Shimojo, and S. Saini, “Scalable atomistic simulation algorithms for materials research,” in *Supercomputing ’01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM, 2001, pp. 1–1.

[2] K. Nomura, R. Seymour, W. Wang, H. Dursun, R. K. Kalia, A. Nakano, P. Vashishta, F. Shimojo, and L. H. Yang, “A metascalable computing framework for large spatiotemporal-scale atomistic simulations,” in *IPDPS ’09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–10.

[3] L. Peng, M. Kunaseth, H. Dursun, K. ichi Nomura, W. Wang, R. K. Kalia, A. Nakano, and P. Vashishta, “A scalable hierarchical parallelization framework for molecular dynamics simulation on multicore clusters,” in *PDPTA*, 2009, pp. 97–103.

[4] D. R. Fan, N. Yuan, J. C. Zhang, Y. B. Zhou, W. Lin, F. L. Song, X. C. Ye, H. Huang, L. Yu, G. P. Long, H. Zhang, and L. Liu, “Godson-t: An efficient many-core architecture for parallel program executions,” *Journal of Computer Science and Technology*, vol. 24, no. 6, pp. 1061–1073, November 2009.

[5] L. Yu, Z. Liu, D. Fan, F. Song, J. Zhang, and N. Yuan, “Study on fine-grained synchronization in many-core architecture,” in *SNPD ’09: Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 524–529.

[6] W. Lin, D. Fan, H. Huang, N. Yuan, and X. Ye, “A low-complexity synchronization based cache coherence solution for many cores,” in *CIT ’09: Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 69–75.

[7] X. Wang, G. Gan, J. Manzano, D. Fan, and S. Guo, “A quantitative study of the on-chip network and memory hierarchy design for many-core processor,” in *ICPADS ’08: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 689–696.

[8] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introdution to Parallel Computing*. Benjamin Cummings, 2003.