J. Parallel Distrib. Comput. 73 (2013) 1469–1482

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Scalability study of molecular dynamics simulation on *Godson-T* many-core architecture

Liu Peng [a,*], Guangming Tan [b,*], Rajiv K. Kalia [a], Aiichiro Nakano [a], Priya Vashishta [a], Dongrui Fan [b], Hao Zhang [b], Fenglong Song [b]

[a] *Collaboratory for Advanced Computing and Simulations, University of Southern California, Los Angeles, CA, 90089, USA*

[b] *Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China*

ARTICLE INFO

Article history:
Received 13 June 2011
Received in revised form
2 June 2012
Accepted 29 July 2012
Available online 23 August 2012

Keywords:
Molecular dynamics
Many-core architecture
Scalability

ABSTRACT

Molecular dynamics (MD) simulation has broad applications, and an increasing amount of computing power is needed to satisfy the large scale of the real world simulation. The advent of the many-core paradigm brings unprecedented computing power, but it remains a great challenge to harvest the computing power due to MD's irregular memory-access pattern. To address this challenge, this paper presents a joint application/architecture study to enhance the scalability of MD on *Godson-T*-like many-core architecture. First, a preprocessing approach leveraging an adaptive divide-and-conquer framework is designed to exploit locality through memory hierarchy with software controlled memory. Then three incremental optimization strategies – a novel data-layout to improve data locality, an on-chip locality-aware parallel algorithm to enhance data reuse, and a pipelining algorithm to hide latency to shared memory – are proposed to enhance on-chip parallelism for *Godson-T* many-core processor. Experiments on *Godson-T* simulator exhibit strong-scaling parallel efficiency of 0.99 on 64 cores, which is confirmed by a field-programmable gate array emulator. Also the *performance per watt* of MD on *Godson-T* is much higher than MD on a 16-cores Intel core i7 symmetric multiprocessor (SMP) and 26 times higher than MD on an 8-core 64-thread Sun T2 processor. Detailed analysis shows that optimizations utilizing architectural features to maximize data locality and to enhance data reuse benefit scalability most. Furthermore, a hierarchical parallelization scheme is designed to map the MD algorithm to *Godson-T* many-core cluster and a simple performance model is derived, which suggests that the optimization scheme is likely to scale well toward exascale. Certain architectural features are found essential for these optimizations, which could guide future hardware developments.

Published by Elsevier Inc.

1. Introduction

Molecular dynamics (MD) simulation is widely used to study material properties at the atomistic level [1], in which interatomic forces are computed quantum mechanically to accurately describe chemical reactions [5]. Large-scale MD simulations involving multibillion atoms are beginning to address broad material problems, but an increasingly large amount of computing power is needed to satisfy the spatiotemporal scale of the real world simulations [22,21,26]. The advent of the many-core paradigm has provided unprecedented computing power, and promises to enable large-scale simulation only if we can efficiently harvest the computing power.

Recently, many-core architectures started dominating the design of computing engine in supercomputer systems, which makes the efficiency of on-chip parallelism increasingly more important. Challenges to achieve an efficient on-chip parallel MD algorithm mainly arise from two aspects: (1) MD application is characterized by irregular memory access which imposes a 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 achieve the scalability and high performance of MD on an *Godson-T*-like emerging many-core architecture, where we map an MD algorithm to the architecture for achieving high on-chip parallel efficiency. We focus on MD simulation with nonbonded n-tuple interactions, which is common in materials simulations [26] and provides a broad computational-characteristics context for algorithmic design.

The objective of this paper is not only to identify how application scientists can utilize new mechanisms provided in emerging many-core architectures to improve performance of their

* Corresponding authors.
E-mail addresses: liupeng@usc.edu (L. Peng), tgm@ict.ac.cn (G. Tan).

0743-7315/$ – see front matter. Published by Elsevier Inc.
doi:10.1016/j.jpdc.2012.07.007

applications, but also to compare the usefulness of various architectural mechanisms as evidenced by their impacts on application performance, which could guide future hardware developments. This work thus serves as an example of architecture-algorithm co-design to inform the development of future exascale computing systems [29].

The main contributions of this paper are eight-fold:

- A preprocessing approach leveraging an adaptive divide-and-conquer (ADC) framework is designed to exploit locality through explicit memory hierarchy, where an analytical formula for divide-and-conquer (DC) criterion, in terms of the size of each core's local memory, is derived to enhance locality by utilizing software controlled memory.
- A novel data layout is employed to re-organize linked-list cell data structures to maximize data locality. Data structures are designed, respectively, for atomic data in private local memory and neighbor atomic data in shared L2 cache or off-chip memory. Combined with *cell-centered* addressing, this ensures consecutive accessing of atomic data within a cell.
- An on-chip locality-aware parallel algorithm is designed to maximize data reuse and to reduce high-latency shared-data access through a preprocessing approach to collect cell pairs within the interaction cutoff radius.
- A pipelining algorithm using data transfer agent (DTA) to orchestrate computation and memory operations is designed to hide latency to shared memory.
- Detailed experiments on an event-driven, cycle-accurate, *Godson-T* simulator verify the effectiveness of the proposed optimization strategy, as the optimized MD achieves a strong-scaling parallel efficiency 0.99 on 64 cores, which is further confirmed by a field-programmable gate array (FPGA) emulator. In addition, the *performance per watt* of MD on *Godson-T* is 144 times higher than MD on a 16-cores Intel core i7 symmetric multiprocessor (SMP) and 26 times higher than MD on an 8-core 64-thread Sun T2 processor.
- Detailed analysis is done to clarify the impact of specific architectural features on applications' performance.
- Software-controlled explicit memory hierarchy and efficient on-chip communication mechanisms are identified as essential architectural features to achieve high performance on a many-core chip, which could guide future hardware developments.
- A hierarchical parallelization scheme is designed to map the MD algorithm to a many-core cluster, while a simple performance model is derived showing that the proposed optimization scheme has the potential to scale well to much larger numbers of computing elements than those in current petascale systems.

The rest of this paper is organized as follows. Section 2 briefly introduces the divide-and-conquer MD algorithm used in this work, where key performance bottlenecks are summarized. Section 3 highlights the main architectural features of *Godson-T*, and Section 4 describes our on-chip optimization strategies. Section 5 presents the experimental results and detailed analysis, and Section 6 describes our hierarchical parallelization scheme and the performance model. Section 7 compares our work with related work, and finally Section 8 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 $E(r^N)$ ($r^N = \{r_1, r_2, \ldots, r_N\}$ are positions of all atoms). The most commonly used algorithm in parallel MD simulation is DC-MD based on 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. 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 processing unit, 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.

This paper is concerned with large spatiotemporal-scale MD simulation implemented with the linked-list cell method, of which $E(r^N)$ consists of two-body $E_2(\{r_{ij}\})$ and three-body $E_3(\{r_{ijk}\})$ terms. 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 (in addition to the original cell). Fig. 1 shows a schematic of the computational kernel of MD (for simplicity, we depict the algorithm as a 2D case, while it is 3D in real implementation). A simulation domain is divided into small rectangular cells, and the linked-list data structure is used to organize atomic data (e.g. coordinates (r), velocities (v), atom type (a) and indices (id)) in each cell. In addition, we maintain a neighbor list for each atom in order to calculate the three-body interactions, which have a shorter cutoff length. The conventional summation rule to compute the three-body interaction is written as

$$E_3(r_{ijk}) = \sum_{i=1}^{N} \sum_{j=1}^{nbr(i)} \sum_{k=i}^{nbr(j)} v(r_i, r_j, r_k), \tag{1}$$

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). Traversing through the linked list, one retrieves all atom information belonging to a cell and thereby computes interatomic interactions. Although the linked-list cell method is of space efficiency, unfortunately, we observe several characteristics that prevent the program from achieving high performance on traditional parallel computers:

- *Irregular memory access*. Straightforward implementation of linked-list data structure as shown in Fig. 1 leads to irregular memory accesses in three-body interaction calculation. For instance, for atom 9, its neighbor list *nbr* is {4, 8, 7, 0, 5, 6}, and atomic data (coordinates, velocities, atom type and indices) are accessed according to this sequence, so that atom 7 is accessed after atom 8. However, in the implementation of linked-list data structure, atom 9 is stored after atom 8 instead of atom 7, which leads to irregular memory access. The locality mechanism based on traditional transparent memory hierarchy cannot optimize much for this type of memory accessing. A more efficient way is to replace the linked list with a sparse-matrix-like data structure to record only cells that are unempty, where the original list *nbr* in Fig. 1 is reorganized as {[0, 5], [2, 7], [4, 8, 9], [1, 3], [6]}, and the original *head* is re-written as {0, 2, 2, 4, 7, 9, 9, 10, 10} pointing to the end of atoms in each cell. However, the sparse-matrix-like data structure can only exploit a little locality for accessing atoms located in a cell. Thus we do not expect that this data structure could make great improvement as, to our best knowledge, it is still very difficult to efficiently optimize sparse matrix operations [27].
- *High frequency communication*. In a cellular decomposition scheme, communication is mainly induced by atom caching operations, where each processing unit gets cached atomic data from its neighbors, which are located in other processing units, for computing interatomic interaction at each MD step. This results in high frequency communication, and in turn prevents

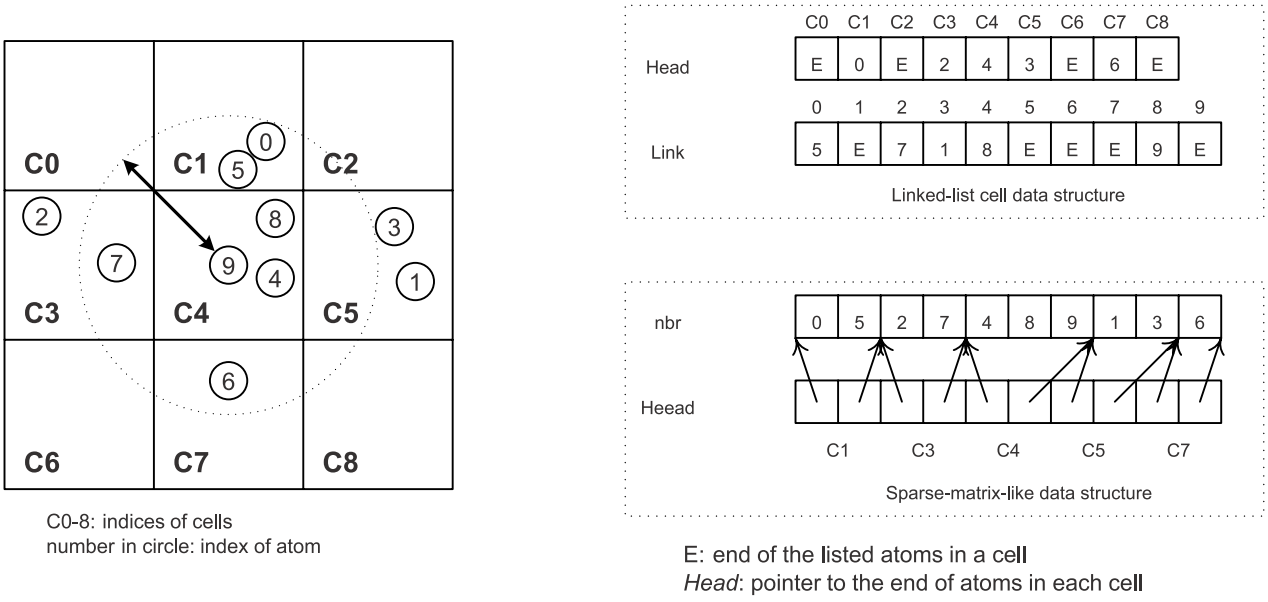

Fig. 1. 2D schematic of the linked-list cell method and data structure. Only atoms within the cutoff radius from the shaded atom perform interaction. C0-8 are cell indices, and a circled number represents an atom index.

our MD from scaling with a larger number of computing units on conventional architecture due to its cache coherence protocol and memory bandwidth restriction.

- *High latency to access shared data*. In a conventional implementation of MD algorithm, the neighbor-list *nbr* used in three-body interaction is shared by all processing units. Let N_c denote the number of cells in each Cartesian direction (we consider a cubic system), and q the atom density, i.e. the number of atoms divided by the system volume $(N_c R_c)^3$. Each cell containing qR_c^3 atoms on average is surrounded by 26 neighbor cells that involve $26qR_c^3$ atoms in total. Thus, the number of cross-cell atom pairs is $qR_c^3 \quad 26qR_c^3$ for each cell. Since there are N_c^3 cells in the entire system, the total number of cross-cell pairs in the system is given by $N_c^3 \quad qR_c^3 \quad 26qR_c^3$. The size of atomic data (e.g. millions of atoms with each one occupying 100 bytes) for interaction computation easily exceeds that of the last level cache (16 MB). Combining with the irregular memory access mentioned before, the latency problem becomes even worse.

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 128 Gflops. 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 [9]:

- *Fine-grained parallelism* [28]. 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 [28,16].
- *Locality-awareness*. Each processing core has a 16 kB 2-way set-associative private instruction cache and a 64 kB local memory (like data cache). As inspired by IBM CELL and Nvidia GPU (graphics processing unit), 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 scratch-pad memory (SPM) to further help programmer maximize locality. A 128-bit wide, 8 8 packet-switching 2D mesh network connects all on-chip units, employing deterministic X–Y routing policy, which can provide a total of 2 TB/s on-chip bandwidth among 64 processing cores. In addition, there are 16 address-interleaved L2 cache banks (256 kB 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 per cycle. The bandwidth between SPM and L2 cache is 256 GB/s, and each four L2 cache banks on the same side of the chip share a a 25.6 GB/s memory controller.
- *Latency tolerance*. 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 direct memory accessing (DMA)-like coprocessor 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 a scalable and efficient MD algorithm based on the features provided by *Godson-T*-like many-core architecture. First, we design a preprocessing approach leveraging the adaptive divide-and-conquer (ADC) framework to achieve better locality with the small on-chip memory. Then we propose three incremental optimizations: a novel data layout to re-organize linked-list cell data structures to maximize data locality; an on-chip locality-aware parallel algorithm is designed to improve data reuse; and a software pipelining algorithm using DTA to hide latency to shared memory. The following subsections discuss these techniques in detail.

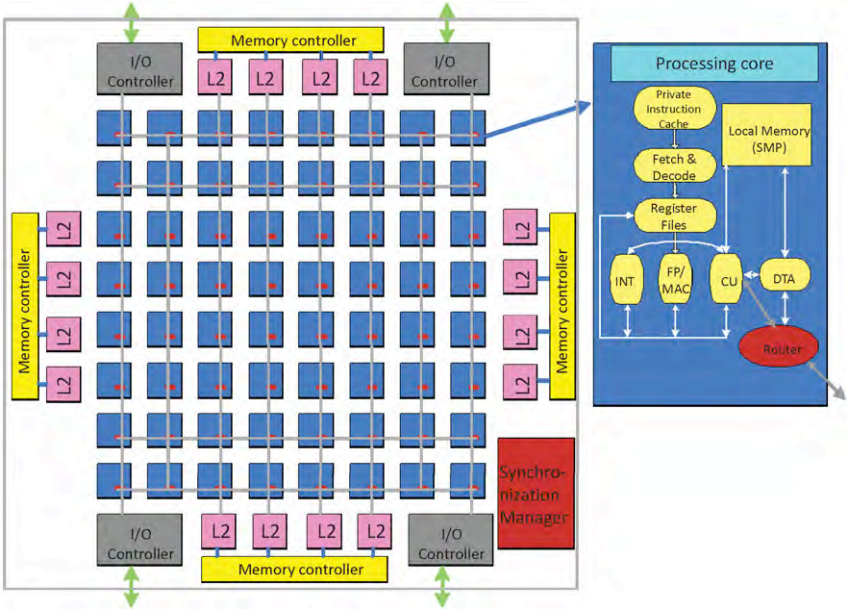

Fig. 2. *Godson-T* Architecture: INT is fixed point arithmetic unit, FP/MAC is floating point unit, and CU is communication unit.

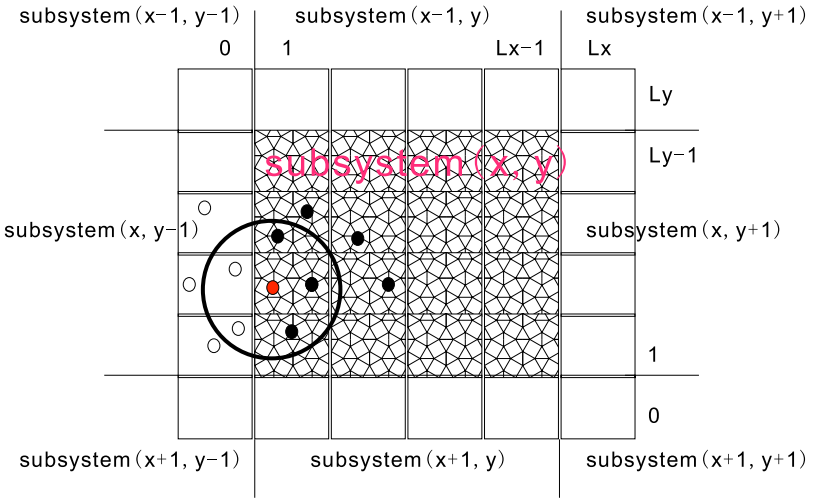

Fig. 3. Cellular decomposition scheme for on-chip parallelization. Each subsystem is further divided into cells, where calculation of atomic interaction needs the latest neighbor surface atoms (cached atoms) shown in the white box. The shaded boxes represent the resident cells.

4.1. Preprocessing

In the original DC-MD algorithm, the physical space is subdivided into spatially localized cells, with local atoms constituting subproblems [18,17]. The algorithm recursively divides a coarse cell into finer cells until some criterion is satisfied. In our 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.

This section only addresses the issue of fine-grained parallelism within a subsystem, and the interchange of cached atoms (atoms near subsystem boundaries) has been completed by a higher-level parallelism (e.g. using message passing interface (MPI)) among multiple many-core computing processors as described in Section 6. Fig. 3 illustrates the scheme of cellular decomposition 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 $P = P_x \quad P_y \quad P_z$ cores in a many-core processor and that the number of cells in a subsystem is $L = L_x \quad L_y \quad L_z$. Then each core i processes L/P cells as Eq. (2) (since how to efficiently embed 3D mesh into 2D one has already been solved by classical algorithms [12], the 2D on-chip network on *Godson-T* is viable for this decomposition):

$$\left\{(c_x, c_y, c_z) | c_x \quad \left[\frac{iL_x}{P_x}, \frac{(i+1)L_x}{P_x}\right), c_y \quad \left[\frac{iL_y}{P_y}, \frac{(i+1)L_y}{P_y}\right),\right.$$
$$\left. c_z \quad \left[\frac{iL_z}{P_z}, \frac{(i+1)L_z}{P_z}\right)\right\}. \tag{2}$$

Let B denote the memory space for storing one atomic data. In order for all the resident atomic data to fit in the private local memory, R_c should satisfy

$$\frac{L}{P} \quad qR_c^3 \quad B \quad C_{\text{pm}} \quad R_c \quad \sqrt[3]{\frac{PC_{\text{pm}}}{LBq}}. \tag{3}$$

For the computation of three-body interactions, an extra space is needed to store the list of neighbor atoms within the three-body

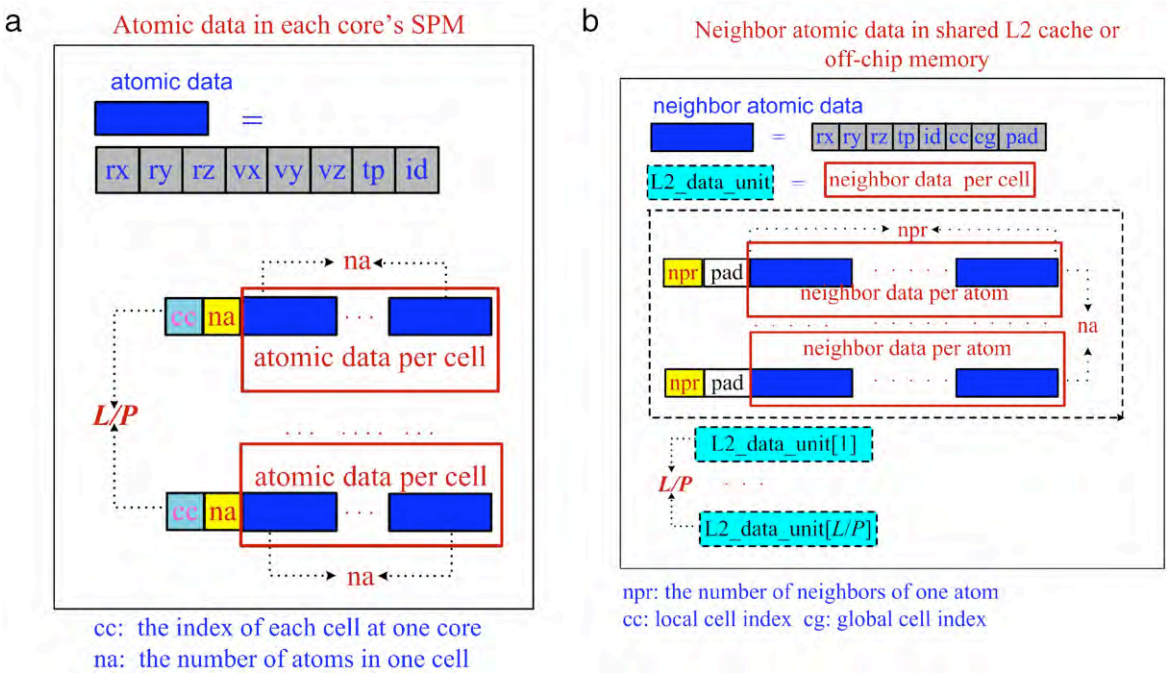

Fig. 4. (a) The atomic data of cells in a core's local memory SPM. rx, ry, rz represents atom position, vx, vy, vz represents atom velocity, tp represents atom type and id represents global atom ID; (b) The neighbor atomic data of cells in the shared L2 cache or off-chip memory. Pad is used for address 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 ($i = 1, \ldots, L/P$).

cutoff radius. Let N_b denote the number of neighbor atoms per cell, then Eq. (3) becomes

$$qR_c^3 \quad \left(N_b + \frac{L}{P}\right) \quad B \quad C_{\text{pm}} \quad R_c \quad \sqrt[3]{\frac{PC_{\text{pm}}}{(PN_b + L)Bq}}. \tag{4}$$

Our ADC algorithm performs recursive cellular decomposition until Eq. (4) is satisfied. In fact, Eq. (4) gives an upper bound of R_c. 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 to 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.

4.2. Data layout optimization

At the beginning of MD simulation, each atom is assigned an integer in $[0 \ldots N \quad 1]$, 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.

Fig. 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 $\{(cc_x + l_x, cc_y, +l_y, cc_z + l_z)|l_x, l_y, l_z \quad \{ \quad 1, 0, 1\}\}$, where the number of neighbors including itself is 27.

As was shown in Section 2, the number of cross-cell atom pairs is $qR_c^3 \quad 26qR_c^3$ per cell, and it is impossible to store all the data in each core's private local memory SPM. 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. Fig. 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*, which will be discussed in Section 4.4.

4.3. On-chip locality optimization

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 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 $ci0, \ldots, cik$ 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

```
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
      via on-chip network
9.   endif
10.  for each cell c_core_i[k] listed in PC[cj]
11.   calculate interactions between c_core_i[k] and cj
12.  endfor
13. endfor
```

a

```
1.  tag1 = tag2 = 0
2.  for each cell c_core_i[k] listed in PC[cj]
3.   if (tag1 ≠ tag2)
4.    DTA_ASYNC(spm_buf[1−tag2], L2_dta_unit[c_core_i[k]])
5.    tag2 = 1 − tag2
6.   endif
7.   calculate atomic interactions between c_core_i[k] and cj
8.   spm_buf[tag1]← cell c_core_i[k]'s neighbor atomic data
9.   tag1 = 1 − tag1
10. endfor
11. if (tag1 ≠ tag2)
12.  DTA_ASYNC(spm_buf[1−tag2], L2_dta_unit[c_core_i[k]])
13.  tag2 = 1 − tag2
14. endif
```

b

Fig. 5. (a) 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*. (b) The algorithmic pipelining for hiding latency to L2 cache and off-chip memory. spm_buf: a pointer to the SPM double-buffer, where tag_1 and tag_2 facilitate interleaving access to the double-buffer.

$PC[5] = \{4, 8\}$. 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(a). 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(a)).

The algorithm in Fig. 5(a) uses a preprocessing approach 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 $O(\frac{L}{P} \quad 26)$. 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 $O(\frac{L}{P} \quad qR_c^3 \quad 26qR_c^3)$.

4.4. Pipelining optimization

In this subsection, we present our pipelining algorithm that reduces the latency to access globally shared data, i.e., the neighbor atomic data used for three-body calculation. Specifically, we use the DTA mechanism on *Godson-T* to hide latency to L2 cache or off-chip memory: First we decouple computation with memory operations, then we assign the memory operations to L2 cache or off-chip memory to DTA while each core read/write data from/to its private local memory SPM, and finally we try to overlap computation with memory operations.

Specifically, the operations of line 11 in Fig. 5(a) generating the neighbor atomic data involve L2 cache or off-chip memory access. Since the neighbor atomic data of a cell are organized in a contiguous region, it is natural to use DTA for block transfer. In order to overlap the computation with memory operations, the for-loop of lines 10–12 in Fig. 5(a) is transformed to a pipeline. In addition, the reserved SPM space for the neighbor atomic data is implemented as a double buffer, which supports the parallel execution of one core's computation and its DTA's memory access. Lines 8–12 in Fig. 5(a) are rephrased as the algorithm in Fig. 5. Half of the double-buffer size is equal to the size of one L2_data_unit in Fig. 4(b). The runtime system on *Godson-T* implements a non-blocking system call DTA_ASYNC, which asynchronously transfers data for the next stage of the pipeline.

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. The experiments demonstrate relative usefulness of various optimization strategies based on *Godson-T* many-core architecture.

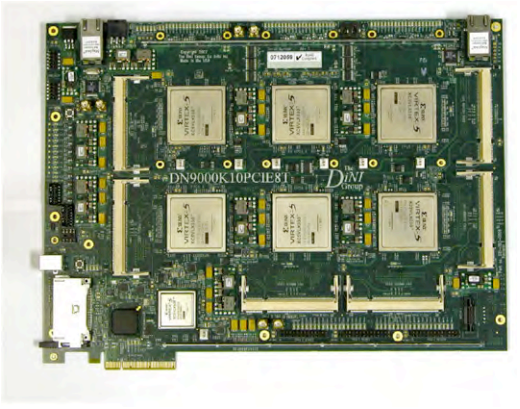

Fig. 6. FPGA emulator for *Godson-T* many-core processor.

5.1. Experimental methodology

Godson-T is an ongoing research project at the Institute of Computing Technology, Chinese Academy of Sciences for building a petaflops supercomputer, and a real chip is expected to be shipped in late 2012. In order to evaluate its performance at early stage of the architectural development, we here employ an instruction-level simulator and FPGA emulator, where all experiments on the instruction-level simulator is confirmed by the FPGA emulator. Fig. 6 is the FPGA emulator for register transfer level (RTL) verification. The *Godson-T* simulator is event-driven, cycle-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 1. 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

Table 1
Godson-T simulator parameters.

Function unit	Parameter
Core	64 cores running at 1 GHz, dual-issue load-to-use latency = 3 cycles, FMAC latency = 4 cycles
SPM	64 kB, 16 64-bit-width SRAM sub-banks with 2 memory ports each (1 for read, 1 for write) 1 cycle for load and store.
L2 cache	16 banks, 4 MB in total, 8-way set-associative, 64 B/cacheline 4 cycles for contentionless hit request.
Memory controller	4 memory controllers, running at 1 GHz. 64-bit FSB. Each memory controller controls one DRAM bank.
Off-chip memory	4 GB. DDR2-800 DRAM clock is 400 MHz tCAS = 5, tRCD = 5, tRP = 5, tRAS = 15, tRC = 24 measured in memory clock.
Network	2-D mesh. Wormhole routing. 2 cycles contentionless latency per hop.
Synchronization (all cores)	6–66 cycles.

by

$$S(P) = \frac{\text{Time}_{\text{one\_core}}}{\text{Time}_{P\_\text{cores}}} \tag{5}$$

where $\text{Time}_{P\_\text{cores}}$ represents the executing time on P cores and $\text{Time}_{\text{one\_core}}$ represents that on one core. The strong-scaling parallel efficiency $E(P)$ is then defined as

$$E(P) = \frac{S(P)}{P}. \tag{6}$$

Since power becomes increasingly more important in multi-core/many-core processor design [14] and Godson architecture features energy efficiency [4], we also consider power consumption as a performance measure. We employ speedup of *performance per watt (PPW)*, SP_{PPW}, to compare relative MD performance across different multi-core/many-core architectures. Let Time_i and Power_i denote the executing time and power consumption on two platforms p_1 and p_2. The PPW speedup of p_1 to p_2, $SP_{\text{PPW}}(p_1, p_2)$ is calculated as

$$SP_{\text{ppw}}(p_1, p_2) = \frac{\text{Time}_2}{\text{Time}_1}\,\frac{\text{Power}_2}{\text{Power}_1}. \tag{7}$$

In the following experiments, the MD simulation tested is for a silica system. Within the ADC framework, the whole system is divided into subsystems each containing 24,000 atoms as the fixed problem size for strong scalability analysis.

5.2. Experimental results on Godson-T many-core processor

The experiments compare four incrementally improved versions of our MD algorithm:

- *baseline:* Implementation of preprocessing.
- *optimization*-1: Implementation of preprocessing and data layout optimization.
- *optimization*-2: Implementation of preprocessing, data layout optimization and on-chip locality optimization.
- *optimization*-3: Implementation of preprocessing, data layout optimization, on-chip locality optimization, and pipelining.

First we test the scalability of our optimized MD on *Godson-T*. Fig. 7 compares the speedup of our fully optimized MD (*optimization*-3) with that of the *baseline* as a function of the number of cores/threads. Although the preprocessing leverages the ADC framework to achieve locality through memory hierarchy [10,11], the scalability of *baseline* begins to deteriorate when the number of cores exceeds 32. Additional optimizations, which take advantage of architectural features to maximize data locality and exploit data reuse, make *optimization*-3 scale almost linearly up to 64 cores

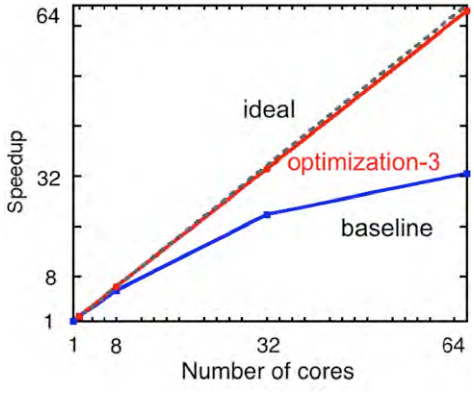

Fig. 7. Speedup as a function of the number of cores.

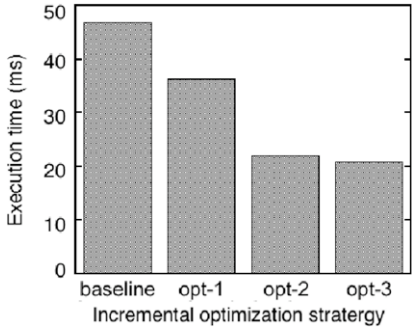

Fig. 8. Execution time in milliseconds on 64-core *Godson-T*. Here, opt-*i* represents optimization-*i* ($i = 1, 2, 3$).

with an on-chip strong-scaling parallel efficiency 0.99 on 64 cores. In fact, given a larger number of cores, each core needs less space for its resident atomic data, and can spare more SPM space to hold more neighbor atomic data, which can be used for further optimization. Therefore, our optimized algorithm is expected to scale well on more cores.

Fig. 8 plots an incremental reduction of execution time for the four versions of parallel MD on fully-configured 64-core *Godson-T*. The result demonstrates the efficiency of the locality optimization methods of re-organizing data layout and exploiting data reuse. Their combination, *optimization*-2, reduces the execution time by a factor of 2.

To understand the reason behind the performance gain by the data layout optimization *optimization*-1, Fig. 9 compares the number of L2 cache events in *baseline* with that in *optimization*-1. The numbers of L2 cache accesses, L2 cache misses and L2 cache

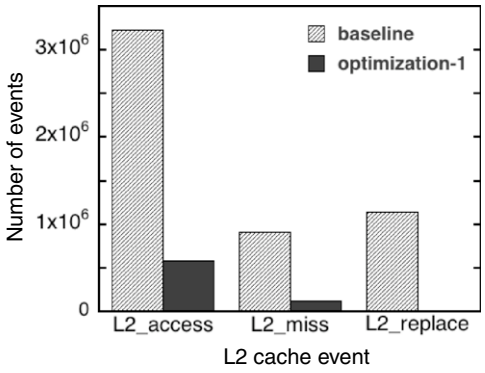

Fig. 9. L2 cache performance with statistic data for all 16 banks in total. L2_access: the number of L2 cache accesses. L2_miss: the number of L2 cache misses. L2_replace: the number of requesting for replaced L2 cache lines (the number is 1976 for *optimization*-1).

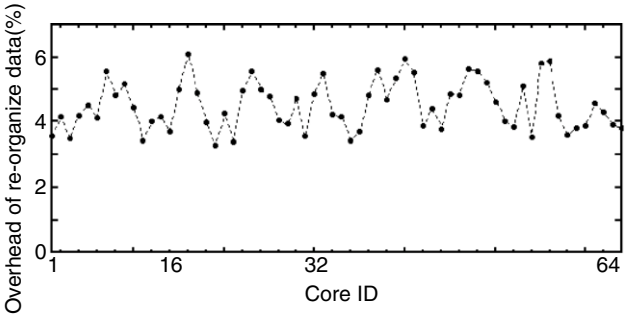

Fig. 10. The overhead of re-organizing data layout on each core of *Godson-T*, which at most accounts for 6% of the entire execution time.

replaces are all much smaller with *optimization*-1 than those with *baseline*. 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 as shown in Fig. 9. Another reason behind the improved L2 cache usage is explicitly controlled private memory SPM. Since SPM is configured with explicit control by the programmer, the new data layout puts a contiguous block of the atomic data into a core's SPM. Thus the operations on its own atomic data only involve accesses to local SPM, instead of the shared L2 cache.

In order to estimate the overhead for traversing the linked-list cell data involved in the data layout re-organization, Fig. 10 plots the percentage of the data layout re-organization time out of the entire execution time for each core. The result shows that the optimization scheme has a low overhead, with the overhead lower than 6%.

In order to quantify the advantage of achieving on-chip locality, the simulator collects statistical data of remote SPM accessing, in which a core reads from or writes to a memory address in another core's SPM. Fig. 11 shows that *optimization*-2 reduces the number of remote SPM accesses to around 7% of that of *optimization*-1. This explains the significant decrease of the execution time by *optimization*-2 in Fig. 8. For given atomic data, the amount of reuse can be estimated as follows. Assume that the size of pair set $PC[cj]$ in Fig. 5(a) 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 2, each cell contains qR_c^3 atoms. Therefore, one atomic data may be reused by mqR_c^3 atoms, i.e., the algorithm may reduce the number of remote SPM accesses by mqR_c^3 1.

After the strategies to improve locality are applied, the pipelining algorithm to hide latency achieves only a minor improvement as shown in Fig. 8. In fact, it should be noted that previous

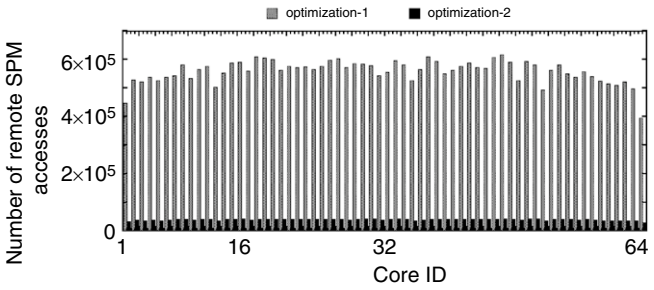

Fig. 11. The number of remote SPM accesses on each core of *Godson-T*.

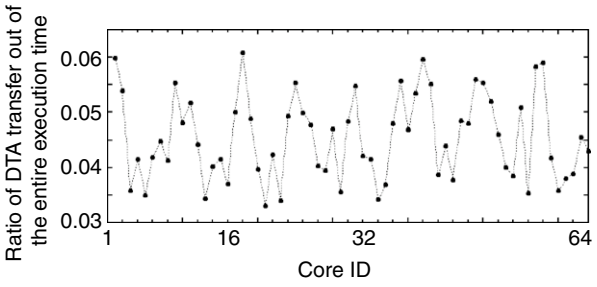

Fig. 12. Ratio of the cost of DTA transfer to the entire execution time on each core of *Godson-T*.

Table 2
Execution time and power performance on *Godson-T*, 16-core Intel Nehalem SMP, and 8-core 64-thread SUN T2 platforms.

	Godson-T	Intel Nehalem 16-core SMP	SUN T2
Power (W)	10	360	130
Time (ms)	20.8	82.6	42.6
SP_{PPW} ($p_{Godson\text{-}T}$, p_x)	1	144	26

optimization strategies have a side effect of reducing the latency, i.e. through block transfers and less network congestion. Fig. 12 plots the ratio of the cost of DTA transfer to the entire execution time on each core. The cost of DTA operations is less than 6% of the total time, and thus it is not surprising that the pipelining algorithm improves the performance only slightly.

To evaluate power related performance, we compare the PPW speedup of our fully optimized MD on *Godson-T* with those of DC-MD on a 16-core Intel core i7 Nehalem SMP platform and an 8-core 64-thread SUN T2 processor. The *Godson-T* chip is targeted at 10 W, while the 16-core Intel core i7 Nehalem SMP is around 360 W (each quadcore core i7 Nehalem processor is around 90 W, and there are four quadcore processors in the 16-core SMP) and SUN T2 processor is about 130 W. Table 2 shows that the PPW of our optimized MD algorithm on *Godson-T* is improved 144 times and 26 times, respectively, compared to the DC-MD algorithm on the 16-core Intel core i7 Nehalem SMP platform and SUN T2 platform. It should be noted that our proposed optimizations take advantage of special architectural features of *Godson-T* and as such they cannot be implemented on the Intel or SUN T2 platform.

To further understand the performance at the instruction level, Fig. 13 summarizes executed instructions per core. It illustrates the total number of floating-point (fpu), arithmetic logic (alu) (excluding branch operations), load, store and branch instructions. In order to evaluate a core's utilization, here we employ CPI (cycles per instruction) by calculating the ratio of the number of instructions over the number of computation cycles. The measured CPI is 1.69, which is about three times the theoretical minimum value of 0.5 (as each core is configured as dual-issue).

Moreover, in order to study floating-point performance, CPF (cycles per floating-point instruction) is introduced, which is

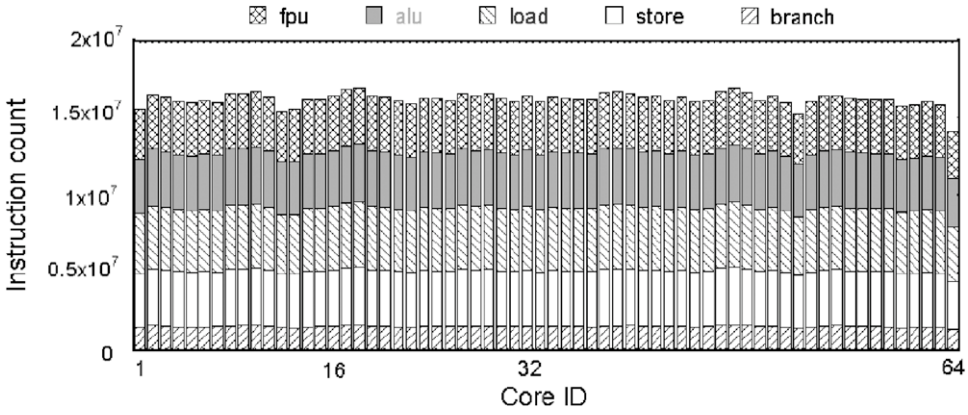

Fig. 13. Instruction histogram on each core of *Godson-T*.

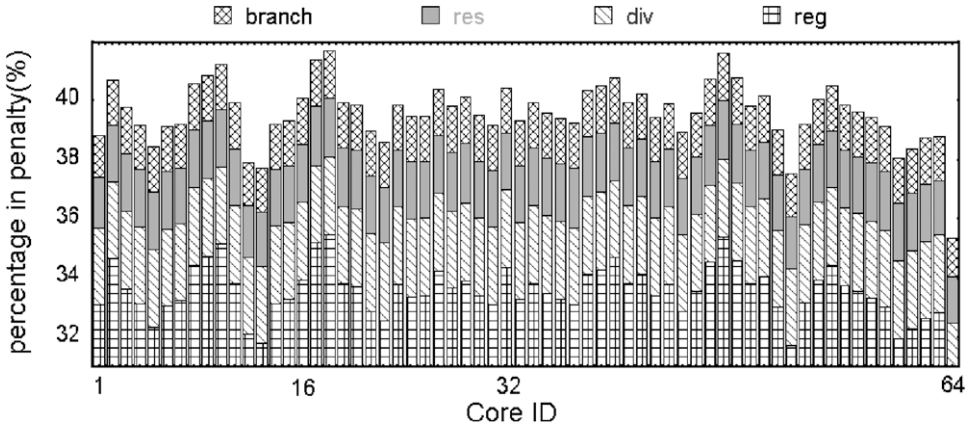

Fig. 14. The penalty of low instruction level parallelism on each core of *Godson-T*. *reg*: register dependency, *div*: latency of divide branch, *res*: resource conflict, *branch*: branch miss predication.

defined as the average number of cycles per floating-point instruction. According to Fig. 13, we get the CPF of 6.89, which shows a relatively low floating-point utilization. There are mainly two factors leading to this low utilization:

- Low proportion (27.39%) of the floating-point instructions out of the total instructions, which was also reported by a previous work [2].
- Lack of deep instruction level parallelism (ILP) in each core. To achieve the objective of low power, each processing core is simply designed (i.e. in-order execution, short pipelining, etc.). Fig. 14 details the penalty introduced by register dependency, resource conflict, branch miss predication and divide operations, which in total consume over 40% of the entire execution cycles. To further estimate the effect of ILP, an experiment is conducted by comparing with the ILP on an Intel Nehalem processor, where kernel code is extracted to generate a synthetic micro-benchmark representing the main computational features in an MD algorithm running on one core/thread, and input data set is selected to totally fit into L1 cache on Nehalem. Results show that the micro-benchmark on Nehalem is about six times faster than that on *Godson-T*. Besides the architectural ILP support, compiler optimization also contributes significantly to achieve efficient ILP: While the code on Nehalem is highly optimized by the Intel compiler, it is not fully optimized on *Godson-T* as the gcc-3.3 compiler is not able to generate an optimized code best utilizing *Godson-T*'s MIPS ISA, such as generating multiply–add/sub to fill the dual-issued core. In order to improve ILP on *Godson-T*, there are two approaches: Architecture and compiler approaches. Between the two, the latter solution – smarter compiler with simple cores – has several advantages: First is low power consumption; and second is the likelihood of compiler-based optimization for achieving high performance as demonstrated in this paper.

5.3. Impact of architectural features

To gain a better understanding of the proposed performance enhancement from architectural viewpoints, the following summarizes architectural impacts on applications' performance.

- *ILP vs. Power*. Extracting high ILP on a conventional heavy core (e.g. Intel Nehalem) with aggressive clock frequency is an effective approach to improve applications' performance. However, it is not an energy-efficient way to integrate such many heavy cores to build many-core computing engines for petascale or exascale supercomputer because of the extremely high power consumption. Table 3 shows that power consumption of Intel Nehalem core i7 920 2.66 GHz heavy-core based parallel system to attain 1 petaflop/s and 1 exaflop/s with various efficiency factor η, where η is defined as the ratio of the maximum performance of a real application $R_{\max}$ to the theoretical peak performance R_{peak}.[1] It shows that extremely large amounts of energy are needed: Even in the ideal case with full efficiency, 34 MW and 34 GW of power are needed to achieve 1 petaflop/s and 1 exaflop/s, respectively, which demonstrates the impracticality of using heavy cores for future

[1] The estimated power only includes processor power. No network, memory or air conditioning power is considered.

Table 3
Power consumption of *Godson-T* simple-core based parallel system ($Power_{Godson\text{-}T}$) and Intel Nehalem core i7 920 heavy-core based parallel system ($Power_{Nehalem}$) to attain 1 petaflop/s and 1 exasflop/s. The 64-core *Godson-T* 1 GHz is targeted at 10W and the quadcore intel Core i7 920 2.66 GHz is about 90W. η is the efficiency factor defined as the ratio of the real max performance R_{max} to the theoretical peak performance R_{peak}.

$\{R_{max}, \eta\}$	$Power_{Godson\text{-}T}$ (W)		$Power_{Nehalem}$ (W)	
{1 petaflop/s, 0.1}	0.78	10^6	338	10^6
{1 petaflop/s, 0.125}	0.64	10^6	272	10^6
{1 petaflop/s, 0.25}	0.32	10^6	136	10^6
{1 petaflop/s, 0.5}	0.16	10^6	68	10^6
{1 petaflop/s, 1}	0.08	10^6	34	10^6
{1 exaflop/s, 0.1}	0.78	10^9	338	10^9
{1 exaflop/s, 0.125}	0.64	10^9	272	10^9
{1 exaflop/s, 0.25}	0.32	10^9	136	10^9
{1 exaflop/s, 0.5}	0.16	10^9	68	10^9
{1 exaflop/s, 1}	0.08	10^9	34	10^9

parallel systems. Compared with the 90 W Intel quadcore Nehalem core i7 920 heavy-core based processor, the 10 W 64-core *Godson-T* simple-core based processor is more suitable to serve as the computing engine for future parallel system, as it only requires 0.2% of the power of the Intel quadcore Nehalem heavy-core based parallel system as shown in Table 3. Although MD on the single core of Intel Nehalem runs 6 times faster than MD on the single core of *Godson-T* (see Section 5.2), simper-core based *Godson-T* architecture is much more power efficient and accordingly more promising for future petascale and exascale parallel systems. There should be a tradeoff between performance and power for future parallel systems, and here our design choice is the *Godson-T*-like simple-core based many-core architecture for the low power consumption. We expect to shorten the single-core performance gap via compiler-based optimization for achieving high performance as demonstrated in this paper.

- *Multithreading vs. DMA (Direct Memory Access)*. Multithreading and DMA are two major approaches to latency hiding. On one hand, our highly optimized MD algorithm attempts to take advantage of *Godson-T*'s DTA (DMA-like) asynchronous block transfer mechanism to hide latency, but it achieves only minor performance enhancement. While the low ratio (6%) – the required data transfer time divided by the entire executing time – restricts DTA's effectiveness, another major factor is MD's irregular memory access pattern, which cannot be handled by current DTA protocol. On the other hand, the SUN T2 processor, a well-known processor for multithreading latency hiding [25], achieves a better performance than Intel Nehalem platform shown in Table 2. Although MD on SUN T2 is about twice as slow as MD on *Godson-T*, we should take into consideration that the major performance enhancement is brought by *Godson-T*'s software controlled explicit memory hierarchy (see Fig. 8), which is not viable for SUN T2 platform. Furthermore, with respect to multithreading's potential benefit to irregular pattern and comparably easy-programmability, it should be a good alternative solution for latency hiding.
- *Single-Instruction Multiple-Thread (SIMT)*. SIMT [19,15] is an emerging technique proposed by GPU architects for easily building many-core processors. However, the SIMT execution is extremely sensitive to branches, i.e. the branching restriction in the thread warp of compute unified device architecture (CUDA) [19]. As a result, it is not suitable for our MD algorithm, as branch instructions account for more than 13% of the total instructions as shown by the profiling data in Fig. 14.
- *Explicit memory hierarchy*. Explicit local memory and DMA-like mechanism have already been used in IBM CELL computing engine [13], and software controlled local memory mechanism has demonstrated its advantages in several emerging computing architectures like IBM CELL [13], IBM Cyclops64 [20] and GPU [15]. Here, our experimental results show that the software controlled explicit memory benefits performance most, and thus explicit memory hierarchy should be well suited in a many-core processor.
- *Fine-grained synchronization*. In a fine-grained parallel algorithm, the lightweight workload of each thread makes the overhead of synchronization non-negligible. From the early IBM Blue Gene cellular architecture, to the new IBM Cyclopes 64, and to the emerging CUDA GPU, efficient barrier synchronization logic and barrier are implemented. Similarly, the *Godson-T* many-core architecture also embeds a great synchronization manager which supports efficient barrier and lock operations, and it greatly contributes to its superior performance over Intel and Sun T2 architectures, illustrating the importance and significance of efficient synchronization in many-core architecture.

6. Hierarchical parallelization and performance model of MD on many-core parallel systems

In order to gain a deeper understanding and better evaluation of the proposed MD optimization scheme on many-core based parallel system, we first propose a hierarchical parallelization scheme mapping MD application to a many-core cluster. We then introduce a simple performance model to assess the scalability of the optimization scheme on emerging parallel systems, which have much larger numbers of computing elements than those in current petascale systems.

6.1. Hierarchical parallelization scheme of MD on many-core parallel systems

To take advantage of the multi-level feature of emerging many-core parallel systems, we propose a hierarchical parallelization scheme mapping an MD application to a many-core parallel system via inter-node and inter-core level parallelization.

- *Inter-node level parallelism*. Our inter-node level parallelism is based on spatial decomposition, similar to DC-MD, where the physical system is partitioned into subsystems of equal volume. Atoms located in a particular subsystem are assigned to one of the compute nodes in the cluster, which are logically arranged according to the topology of the physical subsystems (specifically, we use 3D mesh). In parallel MD, two communication operations are implemented using message passing. The first is atom caching: In order to compute interatomic interaction with cut-off 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 node, where data coherence is maintained by copying the latest neighbor surface atoms every time before atomic accelerations are computed. The second communication operation is atom migration: After the atomic coordinates are updated according to the time-integration algorithm, some resident atoms may have moved out of the subsystem boundary, and such atoms are moved to appropriate nodes. We implement the inter-node spatial decomposition using the MPI standard.
- *Inter-core level parallelism*. After the inter-node level parallelism, a finer inter-core level decomposition, cellular decomposition, is employed to explore parallelism among cores within each node – we further decompose each subsystem in each node into small chunks and assign each chunk to a core by multithreading implemented with the POSIX thread standard. To improve the inter-core parallel efficiency, similar optimizations as those described in Section 4 can be implemented. In addition, the followings optimization techniques can be employed:

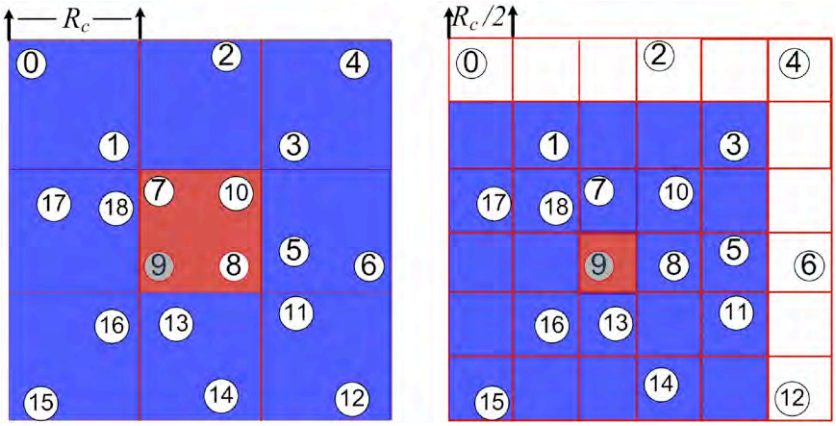

Fig. 15. 2D example of the effect of cell dimension in cell decomposition. Assume that we calculate the interaction for shaded atom 9. The blue areas in the left and right panels represent interacting neighbors with decomposition-cell size R_c and $R_c/2$, respectively. By employing a finer cell dimension than the interaction cutoff, it is possible to reduce unnecessary computation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

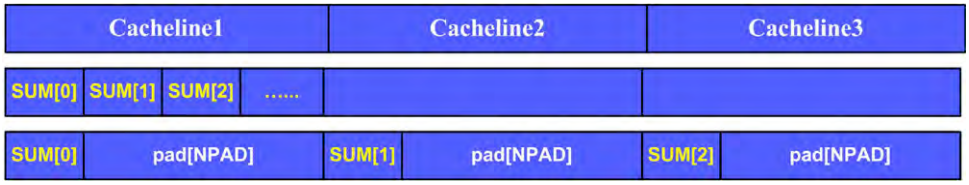

Fig. 16. Illustration of a padding technique to eliminate false sharing among threads for $N_T = 3$. The second and third lines are the cacheline layout without and with padding, respectively. NPAD is the number of padding to fill the cacheline.

– *Decomposition-cell size tuning.* As described in Section 4.1, we can determine the upper bound of decomposition-cell size from Eq. (4). For better inter-core parallelism, we analyze the effect of inter-core decomposition cell size on pair interactions, which is representative of most of the computation. In the conventional inter-core cellular decomposition method, the cell size R_c is chosen to be larger than the interaction cut-off length r_c, so that, for a given computation unit (denoted by the shaded circle 9 in Fig. 15) in a cell, the search space for interacting neighbors is limited to the 26 nearest-neighbor cells (in addition to the original cell). As noted in Section 2, the total number of cross-cell unit-pairs in each node N_c^3 qR_c^3 $26qR_c^3$. It is possible to reduce the computation by choosing a finer cell size, so that neighbor cells better approximate the surrounding volume within r_c. Specifically, consider a cell reduced in size by half, i.e. $R_c/2$. The number of cells in each direction doubles to $2N_c$, and the average number of computation units per cell becomes $q(R_c/2)^3$. The number of interacting neighbor units in the neighbor cells is $q((5R_c/2)^3$ $q(R_c/2)^3)$, and thus the number of interacting pairs for each cell is $q(R_c/2)^3$ $q((5R_c/2)^3$ $q(R_c/2)^3)$. Finally, multiplying with the number of cells $(2N_c)^3$, the total number of examined unit-pairs is $15.5N_c^3$ qR_c^3 qR_c^3, which is less than 60% of the original number.

– *Cacheline false sharing elimination.* Our multithreading scheme also takes account of cacheline false sharing conditions among threads. A typical example is an array sum[N_T] (N_T is the number of threads) that provides separate accumulators to different threads for global sum. Though this eliminates a critical section at data level, cache-level race condition still occurs when multiple threads simultaneously modify sum[i] laid in the same cacheline. Here, we employ a padding technique that separates sum[i] to different cachelines (see Fig. 16). Furthermore, better performance is expected by placing frequently used variables for individual threads together in the same cacheline.

6.2. Performance model of MD on many-core parallel systems

In this subsection, a performance model, based on the hierarchical parallelization scheme, is derived to analyze the scalability of MD on many-core based parallel systems.

- *System configuration and assumptions.* As we mainly focus on the efficiency regarding strong-scaling parallelism, here we fix the simulation size N and assume that the N atoms are distributed uniformly. Also we assume that our simulation is for solid-state material characterized by negligible atomic diffusion so that atom migrations are not done every step. To account for the atom migration at intermediate steps, the decomposition cell size is chosen to be r_c + margin so that the communication cost for atom migrations is negligible. For simplicity, we also assume a cubic system.
- *Machine configuration and assumption.* As is the case for most supercomputers today, we assume that the many-core based parallel machines will use wormhole routing to send flits on the network. In addition, an intelligent topology-aware mapping is employed to limit the number of links traversed, so that the routing time can be ignored. In addition, we assume that our parallel machine only sends/receives large messages (via message packing) so that the start time is negligible compared with the transfer time. Let t_w denote the per-word transfer time: If the network bandwidth is B_w GB/s and the size of a word is 4 bytes, each word spends $t_w = 4/B_w$ time to traverse the network.
  Then if an application sends M messages each containing L words, the communication time is:

$$T_{\text{comm}} = M \quad L \quad t_w. \tag{8}$$

Let $T(N, p)$ be the execution time of N-atom MD simulation on p processing units. Then speedup of p processing units over one processing unit, $S(p)$, can be calculated as

$$S(p) = \frac{T(N, 1)}{T(N, p)} \tag{9}$$

and the corresponding parallel efficiency, $E(p)$, can be calculated as

$$E(p) = \frac{S(p)}{p} = \frac{T(N,1)}{T(N,p)\quad p}. \tag{10}$$

The following describes our hierarchical parallelization model (HPM): which is our performance model for inter-node parallelization and performance model for inter-core parallelization.

- *Performance model of inter-node parallelization.* For inter-node level parallelization, the N-atom simulation system is divided into p subsystems executing independently on p computing nodes, where each computing node deals with N/p atoms on average, and the corresponding computation time can be estimated as

$$T_{\text{comp}}(N,p) = O\left(\frac{N}{p}\right). \tag{11}$$

Here the dominant overhead of inter-node parallelization is communication cost induced by the following two operations:

– *Atom caching.* In atom caching operations, atoms near subsystem boundaries within a cutoff radius, r_c, are copied from the nearest neighbor processors for interatomic force calculation, whose communication time $T_{\text{atom\_comm}}$ scales with the surface of each spatial subsystem:

$$T_{\text{atom\_comm}}(N,p) = \left(\left(\frac{N}{p}\right)^{\frac{2}{3}}\right). \tag{12}$$

– *Global reduction.* For certain reduction operation such as the global energy calculation, global summation is employed which incurs a cost of

$$T_{\text{global\_comm}} = O(\log p). \tag{13}$$

The entire execution time of the parallel MD program is then

$$\begin{aligned} T(N,p) &= T_{\text{comp}}(N,p) + T_{\text{atom\_comm}}(N,p) + T_{\text{global\_comm}} \\ &= O\left(\frac{N}{p} + \alpha\left(\left(\frac{N}{p}\right)^{\frac{2}{3}} + \log p\right)\right) \end{aligned} \tag{14}$$

where α is a communication factor accounting for network transfer time per atom. The corresponding strong-scaling parallel efficiency can be calculated as

$$\begin{aligned} E(p,N,\alpha) &= \frac{S(p)}{p} = \frac{T(N,1)}{T(N,p)p} \\ &= \frac{N}{\left(\frac{N}{p} + \alpha\left(\left(\frac{N}{p}\right)^{\frac{2}{3}} + \log p\right)\right)p} \\ &= \frac{1}{1 + \alpha\left(\left(\frac{p}{N}\right)^{\frac{1}{3}} + \frac{p\log p}{N}\right)}. \end{aligned} \tag{15}$$

- *Performance model of inter-core parallelization.* Our HPM further models the finer inter-core level optimization to exploit parallelism among computing cores within a processor/computing node as was done on *Godson-T*. The HPM is expected to achieve decent scalability, as the communication factor, which acts as a primary factor against high strong-scalability (see Eq. (15)), is much smaller compared with that of processor-level network communication: For example, core–core communication via fast on-chip network, such as *Godson-T* is configured with 0.03 μs communication factor, while node–node communication using high performance communication cards, such as Infiniband has a typical 2 μs MPI point-to-point communication factor. Assume that the whole simulation system is uniformly distributed among p_1 nodes, while each subsystem is further distributed uniformly among p_2 cores in each node, and then the number of all computation units in the parallel system is $p = p_1 \quad p_2$. Let α_1 and α_2 denote the latency factor for node-level and core-level (on-chip level) communication, respectively. Thus, the strong-scaling parallel efficiency of the parallel MD under HPM, E_{hpm}, can be estimated as

$$E_{hpm}(p,N) = E(p_1,N,\alpha_1) \quad E(p_2,N/p_1,\alpha_2). \tag{16}$$

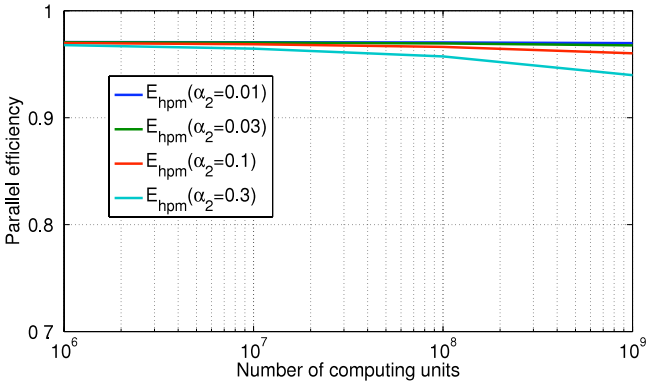

Fig. 17. Performance of parallel MD algorithm on many-core based parallel system under hierarchical parallelization model.

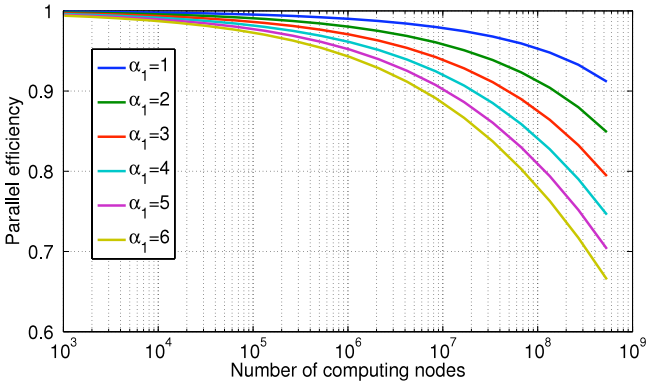

Fig. 18. Performance of parallel MD algorithm on a traditional parallel system under a pure node-level parallelization model.

To evaluate the performance of MD on many-core based parallel system under HPM, Fig. 17 plots the strong scalability of the parallel system with various configurations: $N = 10^{12}$, $p_1 = 10^6$, $p_2 \quad \{1, 10, 10^2, 10^3\}$ representing different number of cores per node, $\alpha_1 = 3$ μs, $\alpha_2 \quad \{0.01$ μs, 0.03 μs, 0.1 μs, 0.3 μs$\}$ representing different core–core communication factors. The horizontal axis represents the number of computing units (calculated by $p_1 \quad p_2$), while the vertical axis shows the strong scaling parallel efficiency and different line-color denotes different core–core communication latency α_2. Fig. 17 shows that the strong-scaling parallel efficiency remains over 0.93 up to 10^9 processing units, which shows the potential scalability of the proposed MD optimization on many-core based parallel system with much larger numbers of computing elements than those in current petascale systems.

In order to evaluate the potential of *Godson-T*-like many-core architecture for emerging parallel systems, we compare the scalability of the many-core based parallel system under HPM with that of the traditional parallel system under pure node-level parallelization. Fig. 18 shows the strong-scalability of MD on traditional parallel system under pure node-level parallelization with various node–node communication latency ranging in {1 μs, 2 μs, 3 μs, 4 μs, [0]5 μs, 6 μs} and different number of

computing nodes ranging from 10^3 to 10^9. The horizontal axis represents the number of computing nodes while the vertical axis shows the strong scaling parallel efficiency and different line-color denotes different node–node communication latency α_1. Fig. 18 shows that the parallel efficiency drops down to around 0.65 with 10^9 computing units compared to 0.93 in many-core based supercomputers under the same configuration. This illustrates the effectiveness and significance of many-core architecture in improving scalability, and consequently indicates the potential of adopting many-core architecture as computing engines in emerging parallel system: Enhancing the scalability of parallel system through many-core's on-chip parallelism to alleviate the sharp decrease of strong-scaling efficiency when the number of computing units exceeds millions.

7. Related work

Molecular dynamics simulation is one of the most prominent applications driving the development of new computer architectures and supercomputing systems. Accordingly, there have been great efforts on accelerating MD simulation. For example, special hardware accelerators including MD-GRAPE [20], Anton [24] and reconfigurable computers [23] promise to reach millisecond-level simulations. Erez et al. [7,8] implemented MD application, GROMACS, on Stanford's streaming supercomputer, Merrimac [6]. Our goal is instead to investigate how to improve MD performance and scalability on a general low-cost platform, which is available to public research groups. A. George et al. [2] conducted a similar research work at the initial stage of IBM BlueGene cellular architecture but did not discuss memory hierarchy optimization. Other projects like NAMD [21,3] mainly target supercomputing systems composed of conventional processors. However, there are few to our knowledge that reports fine-grained strong scalability of on-chip parallelism on many-core (64) processors.

8. 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 designed a preprocessing approach leveraging the EDC framework to achieve better locality of 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, and a pipelining algorithm to hide latency to shared memory. 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. Furthermore, we have derived a simple performance model, which shows that the proposed optimization of MD is expected to scale well to parallel systems with much larger numbers of computing elements than those in current petascale systems. Also the model demonstrates the potential of many-core architectures for future exascale parallel systems. 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.

Acknowledgments

The work at USC was supported by NSF–CMMI/PetaApps/EMT, DOE-SciDAC/SciDAC-e/BES/EFRC/INCITE, and DTRA. The design and development of *Godson-T* was supported by National Natural Science Foundation of China (No. 60803030, No. 61033009, No. 60921002, No. 60925009, No. 61003062), and 973 Program (No. 2011CB302500 and No. 2011CB302502).

References

[1] M. Allen, D. Tildesley, Computer Simulation of Liquids, Oxford Science Publication, Oxford, 1987.
[2] G.S. Almasi, C. Caşcaval, J.G. Castaños, M. Denneau, W. Donath, M. Eleftheriou, M. Giampapa, H. Ho, D. Lieber, J.E. Moreira, D. Newns, M. Snir, H.S. Warren Jr., Demonstrating the scalability of a molecular dynamics application on a petaflop computer, in: ICS'01: Proceedings of the 15th International Conference on Supercomputing, ACM, New York, NY, USA, 2001, pp. 393–406.
[3] A. Bhatelé, L.V. Kalé, S. Kumar, Dynamic topology aware load balancing algorithms for molecular dynamics applications, in: ICS'09: Proceedings of the 23rd International Conference on Supercomputing, ACM, New York, NY, USA, 2009, pp. 110–116.
[4] J. Calamia, Chinese chip wins energy-efficiency crown, IEEE Spectrum (2011) 14–16.
[5] R. Car, M. Parrinello, Unified approach for molecular dynamics and density-functional theory, Physical Review Letters 55 (22) (1985) 2471–2474.
[6] W.J. Dally, F. Labonte, A. Das, P. Hanrahan, J.-H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T.J. Knight, U.J. Kapasi, Merrimac: supercomputing with streams, in: SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2003, p. 35.
[7] M. Erez, J.H. Ahn, A. Garg, W.J. Dally, E. Darve, Analysis and performance results of a molecular modeling application on Merrimac, in: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2004, p. 42.
[8] M. Erez, J.H. Ahn, J. Gummaraju, M. Rosenblum, W.J. Dally, Executing irregular scientific applications on stream architectures, in: ICS'07: Proceedings of the 21st Annual International Conference on Supercomputing, ACM, New York, NY, USA, 2007, pp. 93–104.
[9] 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, L. Liu, *Godson-T*: an efficient many-core architecture for parallel program executions, Journal of Computer Science and Technology 24 (6) (2009) 1061–1073.
[10] M. Frigo, C.E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: FOCS'99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 1999, p. 285.
[11] M. Frigo, V. Strumpen, The cache complexity of multithreaded cache oblivious algorithms, in: SPAA'06: Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, New York, NY, USA, 2006, pp. 271–280.
[12] A. Grama, G. Karypis, V. Kumar, A. Gupta, Introduction to Parallel Computing, Benjamin Cummings, 2003.
[13] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, Synergistic processing in cell's multicore architecture, IEEE Micro (2006) 10–24.
[14] R.H. Katz, D.E. Culler, S. Sanders, S. Alspaugh, Y. Chen, S. Dawson-Haggerty, P. Dutta, M. He, X. Jiang, L. Keys, A. Krioukov, K. Lutz, J. Ortiz, P. Mohan, E. Reutzel, J. Taneja, J. Hsu, S. Shankar, An information-centric energy infrastructure: the Berkeley view, Sustainable Computing: Informatics and Systems 1 (1) (2011) 7–22.
[15] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, Nvidia tesla: a unified graphics and computing architecture, IEEE Micro 28 (2) (2008) 39–55.
[16] W. Lin, D. Fan, H. Huang, N. Yuan, 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, IEEE Computer Society, Washington, DC, USA, 2009, pp. 69–75.
[17] A. Nakano, R.K. Kalia, K. Nomura, A. Sharma, P. Vashishta, F. Shimojo, A.C.T. van Duin, W.A. Goddard, R. Biswas, A divide-and-conquer/cellular-decomposition framework for million-to-billion atom simulations of chemical reactions, Computational Materials Science 38 (2007) 642–652.
[18] A. Nakano, R.K. Kalia, P. Vashishta, T.J. Campbell, S. Ogata, F. Shimojo, S. Saini, Scalable atomistic simulation algorithms for materials research, in: Supercomputing'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, CDROM, ACM, New York, NY, USA, 2001, p. 1.
[19] Cuda programming guide, version 2.2, Tech. Rep., NVIDA Corp., 2008.
[20] Y. Ohno, E. Nishibori, T. Narumi, T. Koishi, T.H. Tahirov, H. Ago, M. Miyano, R. Himeno, T. Ebisuzaki, M. Sakata, M. Taiji, A 281 tflops calculation for *x*-ray protein structure analysis with special-purpose computers mdgrape-3, in: SC'07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, ACM, New York, NY, USA, 2007, pp. 1–10.
[21] J.C. Phillips, G. Zheng, S. Kumar, L.V. Kalé, Namd: biomolecular simulation on thousands of processors, in: SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, IEEE Computer Society Press, Los Alamitos, CA, USA, 2002, pp. 1–18.
[22] S.J. Plimpton, Fast parallel algorithms for short-range molecular dynamics, Journal of Computational Physics 117 (1995) 1–19.
[23] R. Scrofano, V.K. Prasanna, Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers, in: SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, ACM, New York, NY, USA, 2006, p. 90.

[24] D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, M.P. Eastwood, J. Gagliardo, J.P. Grossman, C.R. Ho, D.J. Ierardi, I. Kolossváry, J.L. Klepeis, T. Layman, C. McLeavey, M.A. Moraes, R. Mueller, E.C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, S.C. Wang, Anton, a Special-Purpose Machine for Molecular Dynamics Simulation, Vol. 35, ACM, New York, NY, USA, 2007, pp. 1–12.
[25] Niagara2: a highly threaded server-on-a-chip, Tech. Rep., SunCorp.
[26] P. Vashishta, M.E. Bachlechner, A. Nakano, T.J. Campbell, R.K. Kalia, S. Kodiyalam, S. Ogata, F. Shimojo, P. Walsh, Multimillion atom simulation of materials on parallel computers—nanopixel, interfacial fracture, nanoindentation, and oxidation, Applied Surface Science 18 (2001) 258–264.
[27] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, Optimization of sparse matrix–vector multiplication on emerging multicore platforms, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC'07, ACM, New York, NY, USA, 2007, pp. 38:1–38:12.
[28] L. Yu, Z. Liu, D. Fan, F. Song, J. Zhang, 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, IEEE Computer Society, Washington, DC, USA, 2009, pp. 524–529. http://dx.doi.org/10.1109/SNPD.2009.61.
[29] T. Zacharia, J. Kinter, R. Pennington, R. Arimilli, R. Cohen, L. Davis, T.D. Matteo, B. Harrod, G. Karniadakis, R. Landau, M. Macy, D. McCombie, D. Randall, S. Scott, H. Simon, T. Sterling, T. Windus, National science foundation advisory committee for cyberinfrastructure task force on high performance computing, 2011 March, http://www.nsf.gov/od/oci/taskforces/TaskForceReport_HPC.pdf.

Liu Peng received the B.S. degree in Computer Science from Huazhong University of Science and Technology, Wuhan, China, in 2004 and the M.S. degree in Computer Science from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2007. She obtained the Ph.D. degree in Computer Science in 2011 from University of Southern California, Los Angeles, CA, USA. Her research interests include high performance computing, performance analysis and optimization.

Guangming Tan received the Ph.D. degree in Computer Science from the Chinese Academy of Science, Beijing. He is an associate professor in the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science. From 2006 to 2007, he was a visiting researcher in the Computer Architecture and Parallel Systems Laboratory (CAPSL), University of Delaware. His main research interests include parallel algorithm and programming, performance modeling and evaluation, and computer architecture. He has published several papers in important conferences and journals, such as SC, ICS, SPAA, TPDS. He is a member of the ACM and CCF.

Rajiv K. Kalia is a professor in the Department of Physics & Astronomy with joint appointments in Chemical Engineering & Materials Science, Computer Science, and the Collaboratory for Advanced Computing and Simulations at the University of Southern California. His expertise is in the area of multiscale materials simulations involving atomistic, mesoscale and continuum approaches on parallel supercomputers. He received a Ph.D. in physics from Northwestern University in 1976.

Aiichiro Nakano is a professor of Computer Science with joint appointments in Physics & Astronomy, Chemical Engineering & Materials Science, and the Collaboratory for Advanced Computing and Simulations at the University of Southern California. His research interests include scalable scientific algorithms, high-end parallel and distributed computing, and computational materials science. He received a Ph.D. in Physics from the University of Tokyo, Japan, in 1989.

Priya Vashishta is a professor in the Department of Chemical Engineering & Materials Science with joint appointments in Physics & Astronomy, Computer Science, and the Collaboratory for Advanced Computing and Simulations at the University of Southern California. His research interests include high performance computing to carry out very large multiscale simulations of novel materials and processes on massively parallel and distributed computers. He received a Ph.D. in physics from Indian Institute of Technology, Kanpur, India, in 1967.

Dongrui Fan received Ph.D. degree in Computer Architecture in 2005 from the Institute of Computing Technology, Chinese Academy of Sciences, and has been an associate professor of the institute since 2006. Dongrui participated Godson-1 and Godson-2 micro-architecture designs from 2000. Currently, his research interests focus on multi-core/many-core architecture and low-power micro-architecture design. He leads the AMS research Lab and designed the new processor models—Godson-X, *Godson-T* and Godson-D, which are researches on the new generation Godson series chips. Dongrui is a technical committee member of computer architecture and system software of the Chinese Computer Federation (CCF) and is also HiPEAC/IEEE/ACM member.

Hao Zhang is an assistant professor at ICT. He is the associate chief architect of the Godson-T many core processor. His research interests include high throughput CPU microarchitectures. Zhang received a Ph.D. in Computer Science from ICT in 2008.

Fenglong Song received a Ph.D.degree from the major of Computer Architecture at the Institute of Computing Technology, Chinese Academy of Sciences. He is currently an assistant professor of ICT, CAS. His research interests focus on high performance computer architecture, on-chip memory hierarchy design and optimization, parallel computing, and highly energy-efficient computing.