Computers and Electrical Engineering xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computers and Electrical Engineering

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

Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property ☆

Xiaowen Chen [a,b,*], Zhonghai Lu [b], Axel Jantsch [b], Shuming Chen [a], Shenggang Chen [a], Huitao Gu [a]

[a] School of Computer, National University of Defense Technology, 410073 Changsha, China
[b] Department of Electronic Systems, KTH – Royal Institute of Technology, 16440 Kista, Stockholm, Sweden

ARTICLE INFO

Article history:
Available online xxxx

ABSTRACT

In Network-on-Chip (NoC) based multi-core platforms, Distributed Shared Memory (DSM) preferably uses virtual addressing in order to hide the physical locations of the memories. However, this incurs performance penalty due to the Virtual-to-Physical (V2P) address translation overhead for all memory accesses. Based on the data property which can be either private or shared, this paper proposes a hybrid DSM which partitions a local memory into a private and a shared part. The private part is accessed directly using physical addressing and the shared part using virtual addressing. In particular, the partitioning boundary can be configured statically at design time and dynamically at runtime. The dynamic configuration further removes the V2P address translation overhead for those data with changeable property when they become private at runtime. In the experiments with three applications (matrix multiplication, 2D FFT, and H.264/AVC encoding), compared with the conventional DSM, our techniques show performance improvement up to 37.89%.

1. Introduction and motivation

As a general trend, on-chip computing platforms are evolving from single-core bus-based systems to multi-core network-based systems [1,2], which are referred to as multi-core Network-on-Chips (NoCs) in the paper. For instance, in 2007, Intel researchers announced their research prototype multi-core NoC architecture. It contains 80 tiles arranged as a 10×8 2D mesh network [3]. Another trend exhibits that the embedded memory content in System-on-Chips (SoCs) increases from 20% 10 years ago to 85% of the chip area today and will continue to increase in the future [4]. Technology advances, such as high density embedded memory and 3D integration, make integrating distributed memories on a chip become feasible and preferable. For instance, in 2008, Loh proposed a 3D-stacked memory architecture where each core has its own memory bank [5]; in 2010, a Japanese group presented a technique which can integrate 64 NAND Flash memory dies in a 3D-stacked chip [6]. Combining these two trends, it is a question to organize and manage so many on-chip storage resources. This should be carefully addressed. In multi-core NoCs, especially for medium and large scale system size, a distributed memory organization is more suitable to the multi-core architecture and the general on-chip communication network, since the

☆ Reviews processed and proposed for publication to Editor-in-Chief by Guest Editor Dr. Fangyang Shen.

* Corresponding author at: School of Computer, National University of Defense Technology, 410073 Changsha, China. Tel.: +86 13508481483.
E-mail address: xiaowenc@kth.se (X. Chen).

0045-7906/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

centralized memory organization has already become the bottleneck of performance, power and cost [4]. In the distributed memory organization, memories are distributed in all nodes, featuring good scalability and fair contention and delay of memory accesses. One critical question of the distributed memory organization is whether the memory space is shared or not. Keeping memory only local and private is an elegant architecture solution but ignores rather than solves the issues of legacy code and programing convenience. A lot of legacy software assumes a shared memory model. Most programmers find it easier to express themselves within a shared memory programing model rather than a model that is strictly based on explicit message passing. Consequently, we believe that it's essential to support DSM on multi-core NoCs.

The key technique of DSM is to maintain an address mapping table of translating virtual addresses into physical addresses and hence to implicitly access remote shared data. The Virtual-to-Physical address translation table (V2P table) fully reveals how the DSM space is organized. However, the Virtual-to-Physical (V2P) address translation costs time, which is the inherent overhead of DSM. Every memory access operation contains a V2P address translation, increasing the system's processing time and hence limiting the performance. We observe that different data in parallel applications have different properties (private or shared) and it is unnecessary to introduce V2P address translations for private data accesses. According to the data property of parallel applications, we can obtain two observations:

(1) During the entire execution of parallel applications, some data are shared by multiple processor cores, while other data are private and only used by the local processor core.
(2) Some data are private and only accessed by the local processor core in a certain phase of the entire program execution. However, they may change to be shared and accessible to other remote processor cores in another phase of the entire program execution.

Take 2D FFT as an example. Fig. 1 illustrates the parallelization of 2D FFT on multiple processor nodes. The 2D FFT application performs 1D FFT of all rows firstly and then does 1D FFT of all columns. There is a synchronization point between the FFT-on-rows and the following FFT-on-columns. Each node computes data in a row in step 1 and then data in a column in step 2. Parameters (e.g. loop variable, address offset) in each processor node's program are only used by that processor node, so they are private during the entire program execution. As shown in Fig. 1, data of row *n* are located in the local memory of Node *n*. For original data of each row, they are private in step 1 because they are only used by the local processor node. However, after step 1, they are updated and become shared in step 2 since they have to be accessible to other remote processor nodes. For instance, X_{01} is the 2nd element in row 1, located in the local memory of Node 0. It's accessed by Node 0 (its local processor node) in step 1. After step 1, X_{01} is updated as X'_{01} that is accessed by Node 1 (a remote processor node).

The conventional DSM [7] organizes all memories as shared ones, regardless of whether the processed data are only used by its local processor node or not. That is, each memory access operation in the conventional DSM system includes a translation of virtual address to physical address, even if the accessed object is just local. If we get rid of the address translation overhead when the processor core handles the data which only belong to it (i.e. the data are private), the system's performance is expected to improve.

Motivated by aforementioned considerations, we introduce a hybrid organization of DSM in multi-core NoCs in the paper. The local memory is partitioned into two parts: private and shared. Two addressing schemes are introduced: physical addressing and logic (virtual) addressing. The philosophy of our hybrid DSM is to support fast and physical memory accesses for private data as well as to maintain a global and single virtual memory space for shared data. There are two kinds of way to partition the hybrid DSM: static partitioning and dynamic partitioning. Within the static partitioning, the organization of hybrid DSM is fixed at system design time. It addresses observation (1) to eliminate the V2P address translation overhead of those data that are always private during the entire program execution. Regarding observation (2), the dynamic partitioning is proposed to support programmable boundary partitioning of private region and shared region of the local memory based on the data property of parallel applications. It changes the hybrid DSM dynamically at runtime. We qualitatively analyze its performance by formulating the hybrid DSM with its static and dynamic partitioning. The experimental results show that the

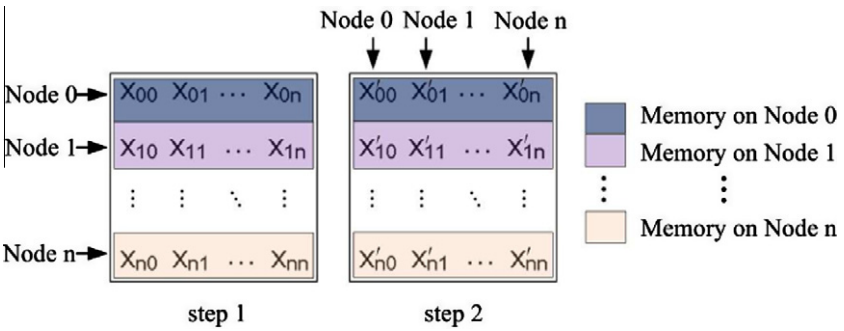

Fig. 1. Parallelization of 2D FFT.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

hybrid DSM with its static and dynamic partitioning demonstrates performance advantages over the conventional DSM counterpart. The percentage of performance improvement depends on problem size, way of data partitioning and computation/communication ratio of parallel applications, network size, etc. In our experiments, the maximal improvement is 37.89%, the minimal improvement 3.68%.

The rest of the paper is organized as follows. Section 2 discusses the background and the related work. Section 3 introduces our multi-core NoC platform, the hybrid DSM, its concurrent memory addressing flow, and its static and dynamic partitioning. Section 4 presents the dynamic partitioning of the hybrid DSM in detail. Performance analysis is conducted in Section 5. Section 6 reports simulation results with application workloads. Finally we conclude in Section 7.

2. Background and related work

As one form of memory organization, DSM has been attracting a large body of researches. Famous researches, such as Alewife [8], DASH [9], FLASH [10], TreadMarks [11] etc., addressed implementing an entire, viable and performance-optimized DSM system in multiprocessor machines. Their work organized all physical memories as shared and provides a single shared memory space with implicit message passing mechanism. In this paper, we do not treat all memories as shared. The memory addressing space is organized in a mixed way, featuring both physical addressing and logical addressing schemes. Moreover, we note that up to today there are few DSM researches on NoC based multi-core chips. In [12], Monchiero et al. explored a DSM architecture suitable for low-power on-chip multiprocessors. However, his work focused on the energy/delay exploration of on-chip physically distributed and logically shared memory address space for Multiprocessor System-on-Chips (MPSoCs). In our view, we envision that there is an urgent need to support DSM because of the huge amount of legacy code and easy programing. Monchiero also pointed out that the shared memory constitutes one key element in designing MPSoCs, since its function is to provide data exchange and synchronization support. Therefore, this paper focuses on the efficient organization and partitioning of DSM space on multi-core NoCs.

Regarding memory organization and partitioning, prior research work mainly considers partitioning algorithm [13–17] and methodology [18–22] in terms of two aspects: performance and power/area efficiency. In [21], Xue et al. explored a proactive resource partitioning scheme for parallel applications simultaneously exercising the same MPSoC system. His work combines memory partitioning and processor partitioning and revealed that both are very important to obtain best system performance. In [16], Srinivasan et al. presented a genetic algorithm based search mechanism to determine a system's configuration on memory and bus that is energy-efficiency. Both Xue and Srinivasan addressed memory partitioning in combination with other factors, e.g. processors and buses. In [19], Mai et al. proposed a function-based memory partitioning method. Based on pre-analysis of application programs, his method partitions memories according to data access frequency. Different from Xue's, Srinivasan's and Mai's work, this paper considers memory organization and partitioning according to data property of real applications running on multi-core NoCs. In [17], Macii et al. presented an approach, called address clustering, for increasing the locality of a given memory access profile, and thus improving the efficiency of partitioning and the performance of system. In the paper, we improve the system performance by partitioning the DSM space into two parts: private and shared, for the sake of speeding up frequent physical accesses as well as maintaining a global virtual space. In [15], Ozturk et. al. also considered private memory and shared memory and addressed problems of multi-level on-chip memory hierarchy design in the context of embedded chip multiprocessors by a proposed Integer Linear Programing (ILP) solution. He only used private memory as the first-level memory in his memory hierarchy design and did not take DSM into account. In [22], Suh et al. presented a general partitioning scheme that can be applied to set-associative caches. His method collects the cache miss characteristics of processes/threads at runtime so that partition sizes are varied dynamically to reduce the total number of misses. We also adopt the runtime adjustment policy, but we address partitioning the DSM space dynamically according to the data property when the system is running, for the sake of eliminating the V2P address translation overhead of accessing private data. In [23], Qiu et al. also considered optimizing the V2P address translation in a DSM based multiprocessor system. However, the basic idea of his work is to move the address translation closer to memory so that the TLBs (Translation Lookaside Buffers: supporting translation from virtual to physical addresses) do not have consistency problems and can scale well with both the memory size and the number of processors. In the paper, we address reducing the total V2P address translation overhead of the system by ensuring physical accesses on private data.

In [24], our previous work implemented a Dual Microcoded Controller (DMC) to support flexible DSM management on multi-core NoCs. It off-loads DSM management from the main-processor to the DMC that features two programmable controller in order to concurrently serve memory access requests from the local node and the remote nodes via the on-chip network. In this paper, we extend the DMC's function to provide hardware support for the proposed hybrid DSM space with static and dynamical partitioning techniques.

The Stanford FLASH uses only one programmable coprocessor (named MAGIC) per IP to support DSM. This leads to contention overhead when two requests arrive concurrently. out DMC hosts two mini-processors to allow concurrent accesses. The MAGIC takes 19 cycles to read a local word. In contrast, our DMC takes two cycles to read a local private word and 16 cycles to read a local shared word. Since our DMC partitions the memory into private and shared parts, fast private memory access and static/dynamic partitioning techniques enhance the performance.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

3. Multi-core Network-on-Chip with hybrid Distributed Shared Memory

3.1. Multi-core NoC

In our Multi-core NoC, memories are distributed at network nodes but partly shared. Fig. 2 (a) shows an example of our multi-core NoCs with hybrid DSM. The system is composed of 16 Processor-Memory (PM) nodes interconnected via a packet-switched mesh network, which is a most popular NoC topology proposed today [25]. The microarchitecture of a PM node is illustrated in Fig. 2 (b). Each PM node consists of a processor core with tightly coupled caches, a network interface, a programmable memory controller, called Dual Microcoded Controller (DMC) [24], and a local memory. As can be observed, memories are distributed in each node and tightly integrated with processors. All local memories can logically form a single global memory address space. However, we do not treat all memories as shared. As illustrated in Fig. 2 (b), the local memory is partitioned into two parts: private and shared. Two addressing schemes are introduced: physical addressing and logical (virtual) addressing. The private memory can only be accessed by the local processor core and it is physical. All of shared memories are visible to all PM nodes and organized as a DSM addressing space and they are virtual. For shared memory

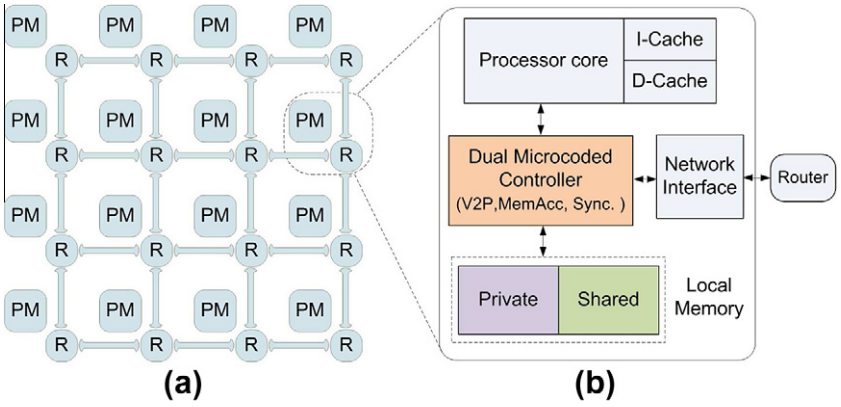

Fig. 2. (a) Multi-core Network-on-Chips, and (b) Processor-Memory node.

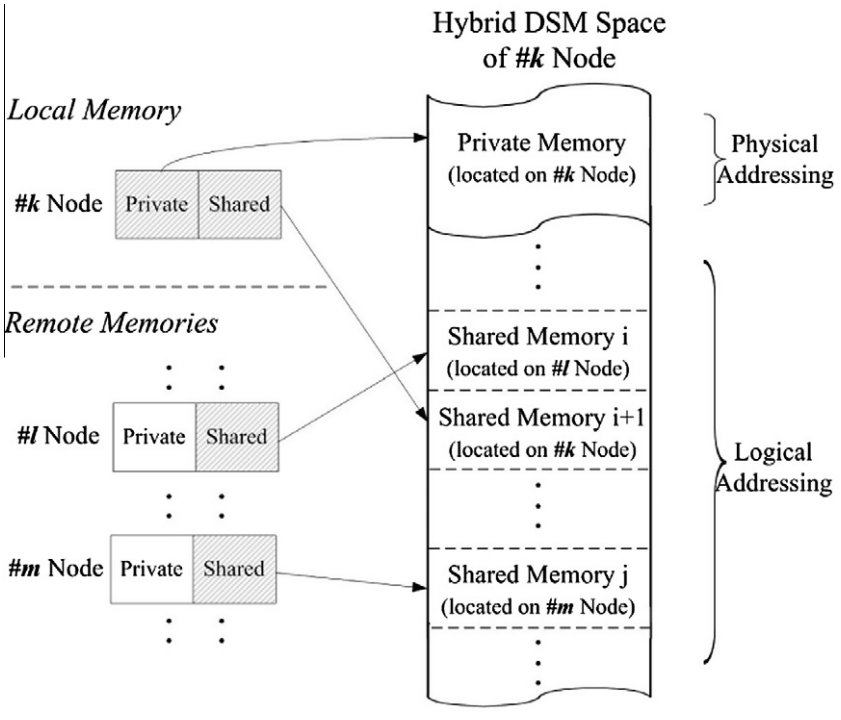

Fig. 3. Hybrid Distributed Shared Memory.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

access, there requires a V2P address translation. Such translation incurs overhead. The philosophy of our hybrid DSM is to support fast and physical memory accesses for private data as well as to maintain a global and single virtual memory space for shared data.

We extend the function of our previous DMC design [24] to organize and manage the hybrid DSM space. It can concurrently respond to memory access requests from the local processor core and remote processor cores via the on-chip network (see details in Section 3.3). It also provides synchronization support. It avoids simultaneous memory accesses to the same memory location and guarantees atomic read-and-modify operations on synchronization variables.

3.2. Hybrid DSM

We illustrate the organization and address mapping of hybrid DSM in Fig. 3. As can be seen, a PM node may use both physical and logical addresses for memory access operations. Physical addresses are mapped to the local private memory region, and logical addresses can be mapped to both local shared and remote shared memory regions. For #***k*** Node, its hybrid DSM space is composed of two parts. The first one is its private memory which is physical addressing. So the logic address is equal to the physical address in the private memory. The second part maps all shared memories. The mapping order of these shared memories is managed by the V2P table. Different PM nodes may have different hybrid DSM space. For instance, in the hybrid DSM space of #***k*** Node, its local shared memory region is mapped logically as shared memory $i + 1$ following shared memory i in #***l*** Node.

3.3. Concurrent memory addressing

The DMC is used to support the hybrid DSM. For each PM node, it maintains one V2P table and three registers: **Local PM Node No.**, **Base Address** and **Boundary Address** (see Fig. 4).

- The V2P table reveals how the hybrid DSM space is organized. The V2P table is configured at the beginning of the program execution. It records the destination Node No. and the physical address offset of a logic memory access.
- **Local PM Node No.** denotes the number of the local PM node.
- **Base Address** denotes the basic physical address of a logic memory access. It is added by the physical address offset obtained from the V2P table to compute the absolute physical address of a logic memory access. **Base Address** is configured at the beginning of the program execution.

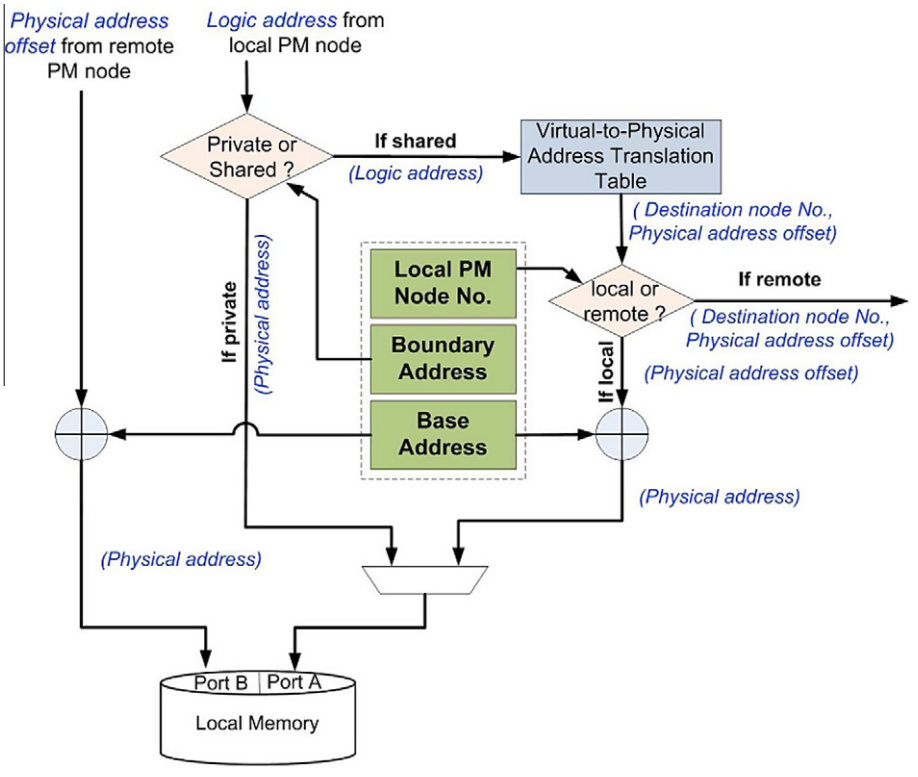

Fig. 4. Concurrent memory addressing flow.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

- **Boundary Address** denotes the address of boundary of the private region and the shared region in the local memory. Initially, **Boundary Address** is equal to **Base Address**. It can be configured at runtime to support dynamic re-organization of the hybrid DSM space.

Fig. 4 shows the concurrent memory addressing flow of each PM node. As shown in the figure, each PM node can handle two memory access requests concurrently from the local PM node and from a remote PM node via the on-chip network. In the beginning, the local PM node starts a memory access in its hybrid DSM space. The memory address is logic. The local PM node firstly distinguishes whether the address is private or shared. If private, the requested memory access hits the private memory. In the private memory, the logic address is equal to the physical address, so the address is forwarded directly to Port A of the Local Memory. If the memory access is "write", the data are stored into the Local Memory in the next clock cycle. If the memory access is "read", the data are loaded out of the Local Memory in the next clock cycle. The physical addressing is very fast. In contrast, if the memory address is shared, the requested memory access hits the shared part of the hybrid DSM space. The memory address first goes into the V2P table. The V2P table is responsible for translating the logic address into two pieces of useful information: *Destination Node No.* and *Physical Address Offset*[1]. *Destination Node No.* is used to obtain which PM node's shared memory is hit by the requested memory address. Once the PM node with the target shared memory is found, *Physical Address Offset* helps position the target memory location. *Physical Address Offset* plus **Base Address** in the target PM node equals the physical address in the target PM node. After *Destination Node No.* and *Physical Address Offset* are obtained from the V2P table, we distinguish whether the requested memory address is local or remote by comparing *Destination Node No.* with **Local Node No.**. If local, *Physical Address Offset* is added by **Base Address** in the local PM node to get the physical address. The physical address is forwarded to Port A of the Local Memory to accomplish the requested memory access. If the requested memory access is remote, *Physical Address Offset* is routed to the destination PM node via the on-chip network. Once the destination PM node receives a remote memory access request, it adds the *Physical Address Offset* by the **Base Address** in it to get the physical address. The physical address is forwarded to Port B of the Local Memory. If the requested memory access is "write", the data are stored into the Local Memory. If the requested memory access is "read", the data are loaded from the Local Memory and sent back to the source PM node.

3.4. Partitioning of hybrid DSM

In the paper, we categorize the partitioning of the hybrid DSM space into two types: static partitioning and dynamic partitioning.

- Within static partitioning, the organization of hybrid DSM is fixed as Section 3.2 describes when the system is designed. It eliminates the V2P address translation overhead of those data that are always private during the program execution. Fixing the hybrid DSM facilitates hardware design and software development.
- To further exploit the performance of the hybrid DSM system, the dynamic partitioning is proposed. Within the dynamic partitioning, the private region and the shared region of the hybrid DSM can be dynamically adjusted at runtime. It can not only eliminate the V2P address translation overhead of private data but also remove the V2P address translation overhead of those data with changeable data property when they become private during the program execution. Though it induces extra overhead of changing the hybrid DSM dynamically, the system's performance can be improved when the overhead of re-organizing the hybrid DSM is less than the V2P address translation overhead of the data with changeable data property.

In the following section, the dynamic partitioning is presented in detail. The performance of the hybrid DSM with both static and dynamic partitioning is analyzed in Section 5 and validated by experiments in Section 6.

4. Dynamic partitioning

The dynamic partitioning is achieved during runtime by the application C/C++ program. Specifically, the programmer inserts the following single line to dynamically configure the **Boundary Address** between the private and shared region.

void `Update_BADDR(unsigned int V`*alue*);

4.1. Idea Description

Fig. 5 shows an example of the dynamic partitioning of the hybrid DSM space. Assume that there are two PM nodes in a 1×2 network. The hybrid DSM space of PM Node 0 contains its private memory region, its shared memory region and the shared memory region of PM Node 1, while the hybrid DSM space of PM Node 1 is equal to its Local Memory plus the shared

[1] *Physical Address Offset* is the offset to **Base Address**. It can be positive or negative, so that the physical address can be less than, equal to, or greater than **Base Address**.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

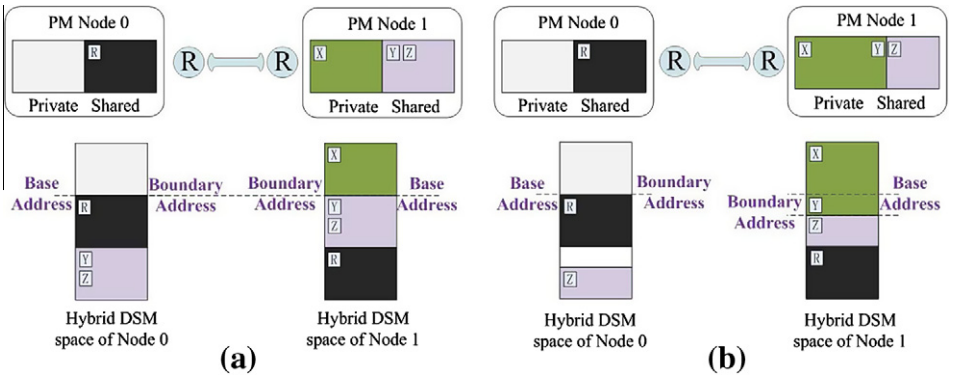

Fig. 5. An example of runtime DSM partitioning.

memory region of PM Node 0. In the beginning (see Fig. 5 (a)), datum 'X' is in the private region of the Local Memory in PM Node 1, while datum 'Y' and 'Z' are in the shared region of the Local Memory in PM Node 1. Therefore, 'X' is invisible to PM Node 0, while 'Y' and 'Z' are accessible to PM Node 0. Assume that the **Boundary Address** in PM Node 1 is re-configured to a larger value during the program execution (i.e the private memory part enlarges) such that 'Y' becomes private. In this situation (see Fig. 5 (b)), PM Node 0 cannot access 'Y'.

The procedure illustrated by Fig. 5 can be used to improve the system performance. For instance, PM Node 0 acts as a producer, PM Node 1 as a consumer. PM Node 0 produces several data that will be consumed by PM Node 1. In the beginning, the property of these data are shared, PM Node 0 firstly stores them into the shared region of the Local Memory of PM Node 1. After that, these data are only used by PM Node 1, it's unnecessary for PM Node 1 to access them in logic (virtual) addressing scheme. By changing the boundary address, we can make them be private. The V2P address translation overhead is averted so that the system performance is improved. Section 4.2 moves further to detail a producer-consumer mode.

Another way of removing V2P address translation overhead is to move data from the shared memory into the private memory when their property changes from "shared" to "private". However, it introduces overhead of data movement. If moved data are huge and the latency cannot be hidden, the system's performance may be negatively affected. Our dynamic partitioning technique overcomes this problem as it does not need to change data's actual storage locations.

4.2. Producer-consumer mode

With concurrent memory addressing (see Section 3.3), the dynamic partitioning provides the software programmer with a convenient producer-consumer mode. The producer-consumer mode features low-latency and tightly-coupled data transmission between two PM nodes. Concurrent memory addressing allows that one PM node produces data and in the meanwhile another PM node consumes the data that have been produced. During data transmission, the property of transmitted data may be changed. The dynamic partitioning dynamically adjusts the DSM according to the changed data property and hence reduce the total V2P address translation overhead so as to improve the efficiency and performance of data transmission.

Fig. 6 illustrates the producer-consumer mode. Assume that there are two PM nodes in a 2×1 network and m data blocks transferred from PM Node 0 to PM Node 1. The hybrid DSM space of PM Node 0 or 1 contains its private memory region, its shared memory region and the shared memory region of PM Node 1 or 0. The PM Node 0 produces m data blocks one by one, while the PM Node 1 consumes the m data blocks one by one once the data are ready.

In Fig. 6, the m data blocks are located in the Local Memory of PM Node 1. As shown in Fig. 6 (a), all data blocks are with the property of "shared" since all data blocks have not been produced by PM Node 0 yet. The **Boundary Address** indicates that $B_1, \ldots, B_m$ are in the shared memory region of PM Node 1 and accessible to PM Node 0. In the beginning, PM Node 0 produces B_1. The values of B_1 are written into the shared memory region of PM Node 1. After PM Node 0 finishes producing B_1, there is a synchronization point that informs PM Node 1 of the readiness of B_1. Once B_1 are ready (see Fig. 6 (b)), PM Node 1 re-configures its **Boundary Address** to make B_1 be "private". After the runtime boundary address configuration, the hybrid DSM spaces of PM Node 0 and PM Node 1 are both changed. PM Node 0 only can access $B_2, \ldots, B_m$ rather than B_1, and PM Node 1 can access B_1 with physical addressing and hence eliminate the V2P address translation overhead. The DMC (see Fig. 2) allows concurrent memory references from both the local PM node and the remote PM node via the on-chip network. As shown in Fig. 6 (b), while PM Node 0 produces B_2, PM Node 1 is consuming B_1. The rest can be done in the same manner. So while PM Node 0 produces B_i, PM Node 1 is consuming B_{i-1} ($i = 2, \ldots, m$). At last, PM Node 1 is consuming B_m, as shown in Fig. 6 (c). At this time, all data blocks become "private" and the entire Local Memory of PM Node 1 is private and invisible to PM Node 0. The hybrid DSM space of PM Node 0 only contains its Local Memory.

In Fig. 6, assume that the total transferred data size is fixed. Larger block size and hence smaller block number result in less overhead of dynamic boundary address configuration but less concurrency. Less overhead of dynamic boundary address

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

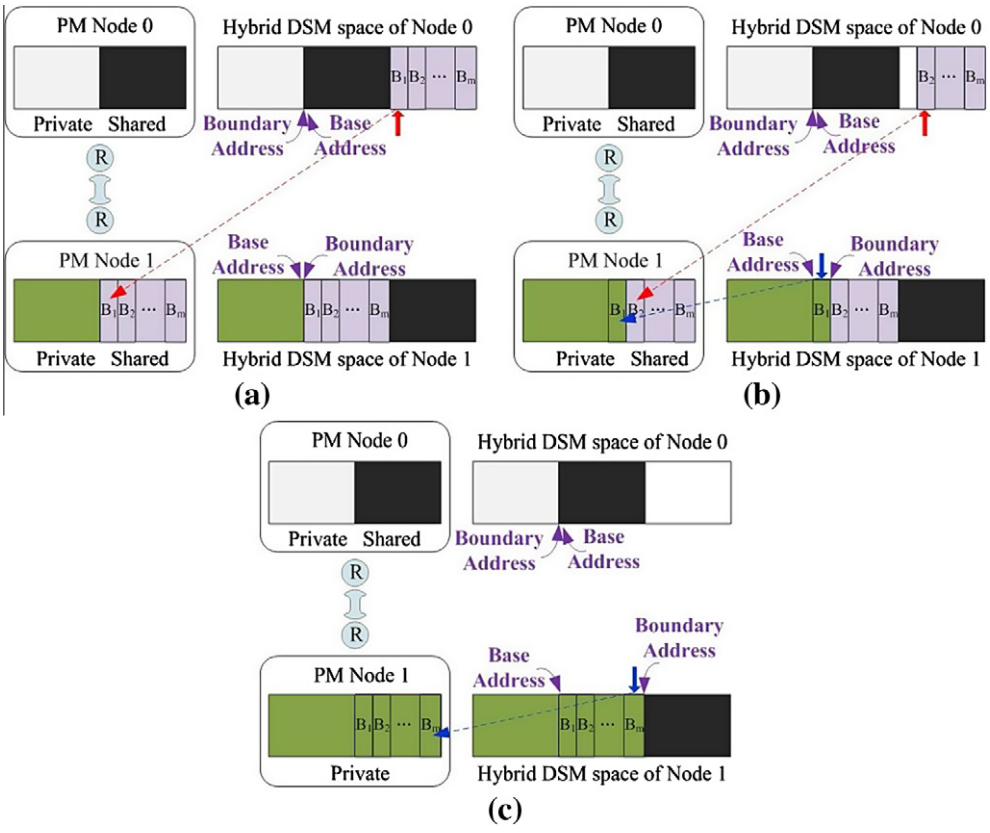

Fig. 6. Producer-consumer mode.

configuration leads to positive performance, less concurrency negative performance. Therefore, there is a tradeoff to determine an optimized block size and boundary address configuration times.

4.3. Memory consistency issue

Since the boundary address configuration is flexible to the software programmer, re-organizing the hybrid DSM space dynamically during the program execution brings the memory consistency issue. For instance, in Fig. 5, changing the boundary address must be after PM Node 0 accomplishes storing 'Y'. This can be guaranteed by inducing a synchronization point before PM Node 1 starts re-writing the **Boundary Address** register. Our multi-core NoC provides underlying hardware support for synchronization to solve this memory consistency issue. When `Update_BADDR()` is executed, it triggers the multi-core NoC architecture to perform three main actions to accomplish updating the **Boundary Address** register:

(1) Its host PM node handshakes with other PM nodes to announce that the shared region in the host PM node is going to be changed.
(2) After receiving acknowledgements from other PM nodes, it updates the **Boundary Address** register.
(3) Finally, it informs other PM nodes of the accomplishment of adjusting the hybrid DSM space in the host PM node.

5. Performance analysis

In this section, we consider the example shown in Fig. 5 and formulate the hybrid DSM with its static and dynamic partitioning to compare its performance with the conventional DSM.

5.1. Definitions and notations

To facilitate the following analysis and discussion, we first define a set of symbols in Table 1.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

Table 1
Definitions and notations.

Basic parameter:	
N	data size processed by PM Node 1 (see Fig. 5)
x	ratio of private data (e.g. 'X' in Fig. 5) to total data
y	ratio of data with changeable property (e.g. 'Y' in Fig. 5) to total data
z	ratio of local shared data (e.g. 'Z' in Fig. 5) to total data
r	ratio of remote shared data (e.g. 'R' in Fig. 5) to total data
t_{mem}	cycles of accessing the Local Memory once.
t_{v2p}	cycles of V2P address translation
t_{com}	cycles of network communication
t_p	cycles of accessing a private datum. ($t_p = t_{mem}$)
t_{ls}	cycles of accessing a local shared datum ($t_{ls} = t_{v2p} + t_{mem}$)
t_{rs}	cycles of accessing a remote shared datum ($t_{rs} = t_{v2p} + t_{mem} + t_{com}$)
t_{bp}	cycles of changing the **Boundary Address** once
m	times of boundary address configuration during program execution
Data access delay:	
T_1	data access delay in the conventional DSM
T_2	data access delay in the hybrid DSM with static partitioning
T_3	data access delay in the hybrid DSM with dynamic partitioning
Performance metric:	
α	average reduced execution time of accessing a datum in the hybrid DSM with static partitioning in comparison with the conventional DSM. ($\alpha = \frac{T_1 - T_2}{N}$)
β	average reduced execution time of accessing a datum in the hybrid DSM with dynamic partitioning in comparison with the conventional DSM. ($\beta = \frac{T_1 - T_3}{N}$)
γ	average reduced execution time of accessing a datum in the hybrid DSM with dynamic partitioning in comparison with the hybrid DSM with static partitioning. ($\gamma = \frac{T_2 - T_3}{N}$)

5.2. Analysis and discussion

In the conventional DSM, all memories are considered to be "shared" and hence the V2P address translation overhead is involved in every local or remote memory access. In Fig. 5, PM Node 1's accessing 'X', 'Y', 'Z', and 'R' includes the V2P address translation overhead. Therefore, we can obtain its data access delay in the conventional DSM by Formula (1) below.

$$\begin{aligned} T_1 &= N \cdot (x + y + z) \cdot t_{ls} + N \cdot r \cdot t_{rs} \\ &= N \cdot t_{mem} + N \cdot t_{v2p} + N \cdot r \cdot t_{com} \end{aligned} \quad (x + y + z + r = 1) \tag{1}$$

For the hybrid DSM, the V2P address translation overhead is only included when shared data are accessed. Hence, in Fig. 5, PM Node 1's accessing 'X' does not require the V2P address translation.

With the static partitioning, accessing the data with changeable property (e.g. 'Y' in Fig. 5) is done through logical (virtual) addressing. We can get the data access delay in the hybrid DSM with static partitioning by Formula (2) below.

$$\begin{aligned} T_2 &= N \cdot x \cdot t_p + N \cdot (y + z) \cdot t_{ls} + N \cdot r \cdot t_{rs} \\ &= N \cdot t_{mem} + N \cdot (y + z + r) \cdot t_{v2p} + N \cdot r \cdot t_{com} \end{aligned} \quad (x + y + z + r = 1) \tag{2}$$

The dynamic partitioning makes PM Node 1's accessing 'Y' with physical addressing and hence avoids its V2P address translation overhead. However, the data access delay contains the extra overhead of changing the boundary address. We can get the data access delay in the hybrid DSM with dynamic partitioning by Formula (3) below. The maximal frequency of changing the boundary address is that the boundary address is changed for each datum access, so $m \leqslant N \cdot y$.

$$\begin{aligned} T_3 &= N \cdot (x + y) \cdot t_p + N \cdot z \cdot t_{ls} + N \cdot r \cdot t_{rs} + t_{bp} \cdot m \\ &= N \cdot t_{mem} + N \cdot (z + r) \cdot t_{v2p} + N \cdot r \cdot t_{com} + t_{bp} \cdot m \end{aligned} \quad \begin{pmatrix} x + y + z + r = 1 \\ m \leqslant N \cdot y \end{pmatrix} \tag{3}$$

To evaluate the performance of the hybrid DSM, we establish three performance metrics (α,β,γ). They are defined as average reduced execution time of accessing a datum.

α is obtained in Formula (4) and reflects the performance improvement induced by the hybrid DSM with static partitioning in comparison with the conventional DSM.

$$\alpha = \frac{T_1 - T_2}{N} = x \cdot t_{v2p} \tag{4}$$

From Formula (4), we can see that

- Compared with the conventional DSM, our hybrid DSM eliminates V2P address translation overhead of private memory accesses and hence improves performance.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

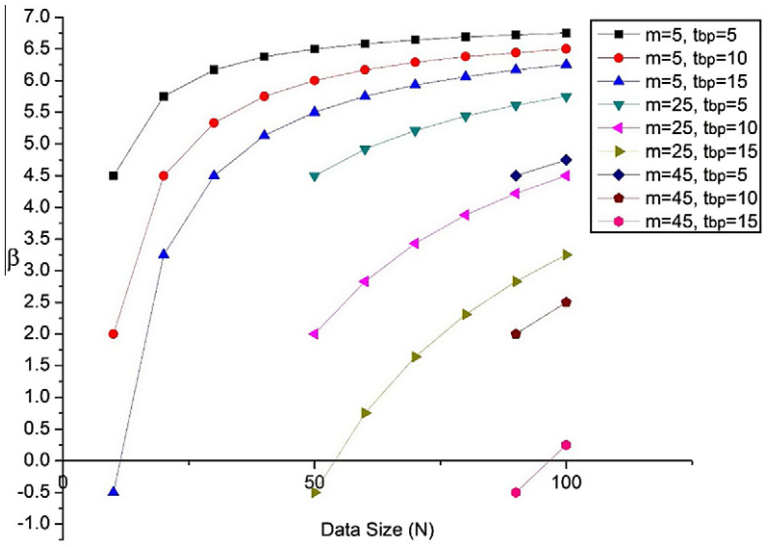

Fig. 7. Trends of performance improvement induced by the hybrid DSM with dynamic partitioning in comparison with the conventional DSM.

- If private data take a lager proportion in parallel programs (i.e. x increases), the hybrid DSM demonstrates higher performance advantage.
- For the same size of parallel program in the hybrid DSM, hardware implementation of V2P address translation obtains higher performance gain than software implementation (t_{v2p} of hardware solution is greater than that of software solution).

β is calculated in Formula (5). It compares the hybrid DSM with dynamic partitioning with the conventional DSM.

$$\beta = \frac{T_1 - T_3}{N} = (x + y) \cdot t_{v2p} - \frac{t_{bp} \cdot m}{N} \tag{5}$$

To illustrate the relation between β and the parameters, we consider a concrete case, assuming that "$t_{v2p} = 10; x = 20\%$; $y = 50\%; t_{bp} = 5,10,15$ and $m = 5$, 25, 45" and could have the trends of performance improvement induced by the hybrid DSM with dynamic partitioning versus the data size (N) from 10 to 100 (see Fig. 7).

From Formula (5) and Fig. 7, we can obtain that

- The dynamic partitioning can further eliminate the V2P address translation overhead of memory accesses of data with changeable data property, but it incurs the extra overhead of changing the boundary address.
- As the data with changeable data property take a larger proportion, the advantage of the dynamic partitioning dominates.
- Hardware implementation of V2P address translation obtains higher performance gain than software implementation.
- The re-configuration of **Boundary Address** induces negative performance. In Fig. 7, β is negative for $m = 5, t_{bp} = 15; m = 25, t_{bp} = 15; m = 45, t_{bp} = 15$. i.e. frequent configuration degrades the performance.
- However, avoiding frequently changing the boundary address (m is less) and optimizing the boundary address configuration times (t_{bp} is less) alleviates the negative effect on performance.
- As the data size (N) increases, β becomes larger, thus achieving higher performance improvement.

We introduce γ in Formula (6) to discuss that when to use the static partitioning or the dynamic partitioning in the hybrid DSM can have higher performance.

$$\gamma = \frac{T_2 - T_3}{N} = y \cdot t_{v2p} - \frac{t_{bp} \cdot m}{N} \tag{6}$$

From Formula (6), we can have that

- When $y \cdot t_{v2p} > \frac{t_{bp} \cdot m}{N}$, the hybrid DSM with dynamic partitioning is preferable. Otherwise, the hybrid DSM with static partitioning is better.
- Since $m \leqslant N \cdot y$, Formula (6) can be transformed as $\gamma \geqslant y \cdot (t_{v2p} - t_{bp})$. If the V2P address translation overhead (t_{v2p}) is greater than the boundary address configuration overhead (t_{bp}),γ is always bigger than 0. In this case, the hybrid DSM with dynamic partitioning obtains better performance than that with static partitioning.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

5.3. Time complexity

The dynamical partitioning consumes time since it configures the boundary address during program execution. As defined in Table 1, t_{bp} represents the time of changing the boundary address once and m denotes the boundary address configuration times. Therefore, the overhead of the dynamical partitioning is $t_{bp} \cdot m$. Since adjusting the boundary address is supported by hardware, t_{bp} is a constant for a given platform. Therefore, the time complexity of the dynamic partitioning is $O(m)$, and m depends on how the application program is parallelized and mapped by the programmer.

5.4. Discussion

Based on the previous analysis, we summarize and clarify advantages, limitations and application scope of our hybrid DSM.

1. In general, compared with the conventional DSM, our approach (both the static partitioning and the dynamic partitioning) improves the system performance by eliminating V2P address translation overhead of those data when they are private (see Formula (4) and (5)). If in a parallel application, data are always be exchanged (shared) by processor nodes (i.e. in Formula (4) and (5), $x = 0$ and $y = 0$, and hence $m = 0$), our hybrid DSM is equivalent to the conventional DSM, having no performance advantage. However, when a parallel application is mapped on multiple processor nodes, if the processor nodes mainly handle their own data and only have few data exchanges among them, our hybrid DSM can achieve high performance. As the private data size increases (i.e. in Formula (4) and (5), x and y become larger), better performance is obtained.
2. The static partitioning technique aims to remove the V2P address translation overhead for private data (see Formula (4)). With the static partitioning, the organization of hybrid DSM is fixed at the system design time. The performance gain is limited by the volume of the private region. The larger the region is, the higher the gain is. Thus it is more suitable for parallel applications which have more private data. However, it is simple and easy to implement, and does not bring extra overhead.
3. Based on the static partitioning, the dynamic partitioning further eliminates the V2P address translation overhead of those data with changeable property when they become private (see Formula (5)). Within the dynamic partitioning, the organization of the hybrid DSM can be dynamically configured during the program execution. The dynamic partitioning extends the application scope, covering those parallel applications that have no private data but data with changeable property. Despite this, adjusting the hybrid DSM space costs time and causes the consistency issue. If the hybrid DSM space is adjusted too frequently, the incurred overhead becomes large, possibly leading to degraded system performance. Therefore, the dynamic partitioning is not suitable to parallel applications that have most data with frequently changed property.

6. Experiments and results

In this section, we perform experiments to evaluate the hybrid DSM with its static and dynamic partitioning in terms of execution overhead in our DSM based multi-core NoC platform, applying three applications. In experiments, we first define "Speedup" in Formula (7) and "Performance Improvement" in Formula (8).

$$\text{Speedup}_m = \frac{T_{1node}}{T_{mnode}} \tag{7}$$

$$\text{Performance Improvement} = \frac{\text{Speedup}_{\text{hybrid DSM}} - \text{Speedup}_{\text{conventional DSM}}}{\text{Speedup}_{\text{hybrid DSM}}} \tag{8}$$

In Formula (7), T_{1node} is the execution time of single PM node in the conventional DSM as the baseline, T_{mnode} the execution time of m PM node(s).

6.1. Experimental platform

We constructed a cycle-accurate multi-core NoC experimental platform at the RTL level as shown in Fig. 2. Except the DMC realized in Verilog, all other modules are implemented in VHDL. We detail the main modules as follows:

- The LEON3 [26] is used as the processor core in each PM node. The LEON3 core is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. It implements an advanced 7-stage pipeline with separate instruction and data cache buses (Harvard architecture).
- The DMC [24] manages the hybrid DSM space, supporting both static partitioning and dynamic partitioning. The V2P table and **Local PM Node No.**, **Base Address** and **Boundary Address** are implemented in the DMC. The DMC is connected to the LEON3's AHB bus, the Network Interface (NI) and the Local Memory.
- The Network Interface (NI) allows all PM nodes to communicate over the on-chip network. It is connected to the DMC.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

- The Local Memory (LM) is managed by the DMC. It is partitioned into two parts: private and shared, and adopts two addressing schemes: physical addressing and logic (virtual) addressing. All LMs form a hybrid DSM space that consists of the private region and all shared regions.
- An instance of Nostrum NoC [27] is used as the on-chip network. It is a packet-switched 2D mesh network with configurable size. It employs deflection routing to minimize the buffering cost in the router. Transmitting a packet in one hop takes one cycle. Upon contention, packets with a lower priority indicated by a hop count are deflected to a non-minimal path.

On the platform, application programs are written in C++, running on the LEON3 processors.

6.2. Application 1: Matrix Multiplication

In this section, we use Matrix Multiplication to evaluate the performance advantage of the hybrid DSM with the static partitioning over the conventional DSM.

The Matrix Multiplication calculates the product of two matrix, A[64, 1] and B[1, 64], resulting in a C[64, 64] matrix. We consider both integer matrix and floating point matrix to reflect the impact of computation/communication ratio on performance improvement. As shown in Fig. 8 (a), matrix A is decomposed into p equal row sub-matrices which are stored in p PM nodes, respectively, while matrix B is decomposed into p equal column sub-matrices which are stored in p PM nodes, respectively. The result matrix C is composed of p equal row sub-matrices which are respectively stored in p PM nodes after multiplication. Fig. 8 (b) shows the conventional DSM and all data of matrix A, C and B are shared. Fig. 8 (c) shows the hybrid DSM. The data of matrix A and C are private and the data of matrix B are shared. Because the sub-matrices of matrix A and C are only accessed by their own host PM node and matrix B are accessed by all PM nodes. It's unnecessary to re-organize the hybrid DSM dynamically and the hybrid DSM space is fixed.

Fig. 9 shows speedups of integer Matrix Multiplication in both the conventional DSM and the hybrid DSM. When the system size increases, the speedup is from 1 to 1.983, 3.938, 7.408,10.402, 19.926 and 36.494 for the conventional DSM and from 1.516 to 2.826, 5.593, 10.403, 13.993, 25.799 and 45.633 for the hybrid DSM. In comparison with the conventional DSM, the hybrid DSM obtains 34.05%, 29.82%, 29.59%, 28.79%, 25.66%, 22.77% and 20.03% performance improvement. Fig. 10 shows speedups floating-point Matrix Multiplication. As the system size is scaled up, the speedup is from 1 to 1.998, 3.985, 7.902, 13.753, 27.214 and 52.054 for the conventional DSM and from 1.083 to 2.097, 4.181, 8.281, 14.345,

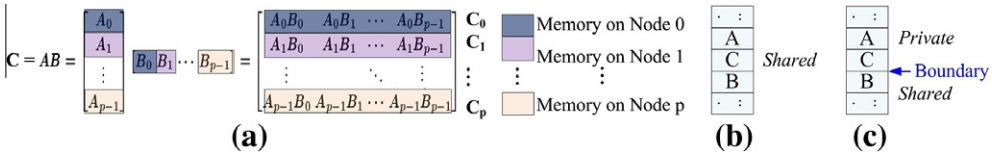

Fig. 8. (a) Memory allocation for matrix multiplication, (b) conventional DSM, and (c) hybrid DSM with static partitioning.

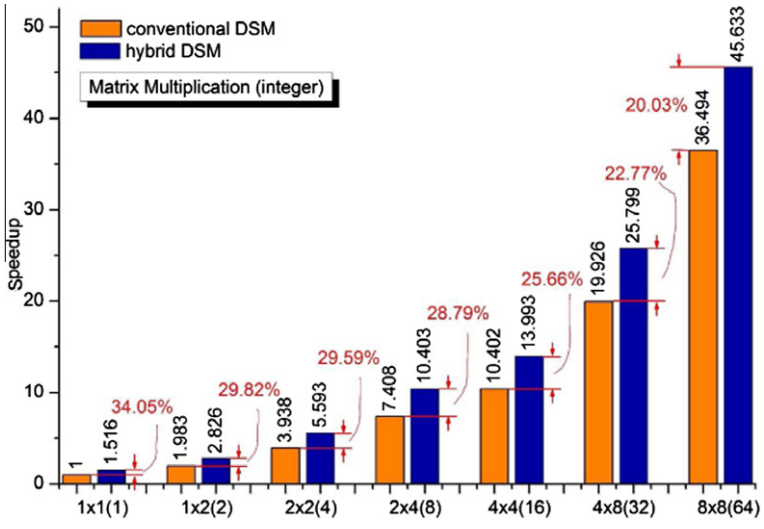

Fig. 9. Speedup and performance improvement of integer matrix multiplication.

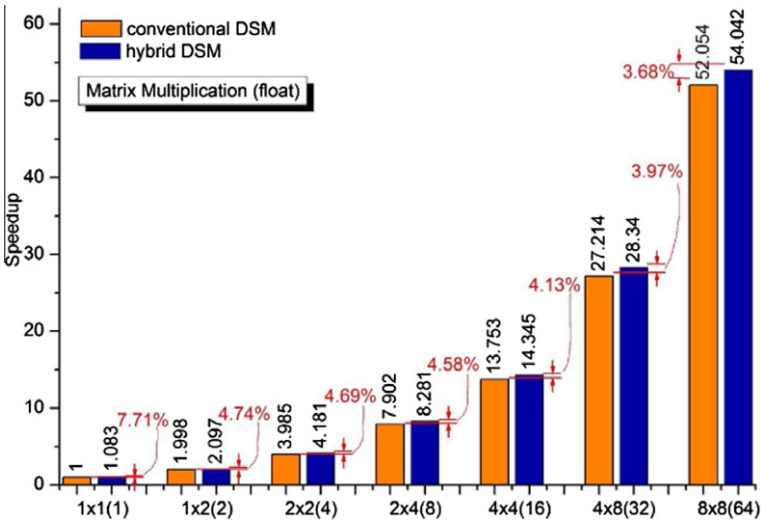

Fig. 10. Speedup and performance improvement of floating-point matrix multiplication.

28.340 and 54.042 for the hybrid DSM. In comparison with the conventional DSM, the hybrid DSM obtains 7.71%, 4.74%, 4.69%, 4.58%, 4.13%, 3.97% and 3.68% performance improvement with increase of the system size. From Figs. 9 and 10, we can see that

- The value of performance improvement is decreasing as the system size is scaled up, since the communication delay becomes larger.
- The relative speedup for the floating point multiplication is higher than that for the integer computation. This is as expected because when increasing the computation time, the portion of communication delay becomes less significant, thus achieving higher speedup.
- The improvement for the floating point multiplication is lower because the floating point has a larger percentage of time spent on computation, thus reducing the communication time in memory accesses achieves less enhancement.
- The single PM node case achieves the highest performance improvement because all data accesses are locally shared for the conventional DSM and private for the hybrid DSM.

6.3. Application 2: 2D FFT

2D FFT is applied to demonstrate performance improvement induced by the hybrid DSM with the dynamic partitioning in comparison with the conventional DSM.

We implement a 2D radix-2 DIT FFT. As shown in Fig. 11 (a), the FFT data are equally partitioned into *n* rows, which are stored on the *n* PM nodes, respectively. According to the 2D FFT algorithm, the application first performs FFT on rows (step 1). After all nodes finish row FFT (synchronization point), it starts FFT on columns (step 2). We experiment on the two DSMs. One is the conventional DSM, as shown in Fig. 11 (b), for which all FFT data are shared. The other is the hybrid DSM, as illustrated in Fig. 11 (c). The data used for row FFT calculations at step 1 are located locally in each PM node. After step 1, they are updated and their new value are to be used for column FFT calculations at step 2. We can dynamically re-configure the boundary address at runtime, such that, the data are private at step 1 but become shared at step 2.

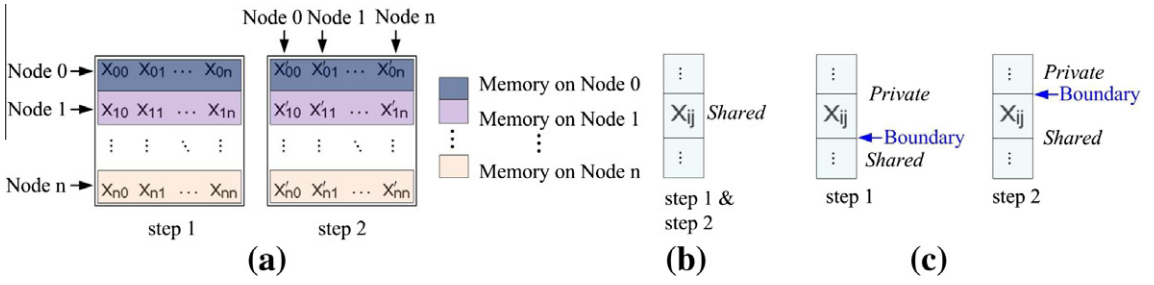

Fig. 11. (a) Memory allocation for 2D FFT, (b) conventional DSM, and (c) hybrid DSM with dynamic partitioning.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

Fig. 12 shows speedups of the FFT application in both the conventional DSM and the hybrid DSM with dynamic partitioning. As we can see, when the system size increases, the speedup with the conventional DSM goes up from 1 to 1.905, 3.681, 7.124, 13.726, 26.153 and 48.776 and the speedup with the hybrid DSM is from 1.525 to 2.269, 4.356, 8.373, 16.091, 30.279 and 55.070. The performance improvement is 34.42%, 16.04%, 15.48%, 14.92%, 14.70%, 13.63% and 11.44%. From Fig. 12, we can see that

- Using the dynamic partitioning in the hybrid DSM, fast physical addressing is performed in step 1 of 2D FFT and the entire V2P address translation overhead is reduced so that the system performance improves.
- As the system size is scaled up, larger communication delay leads to the decrease of performance improvement.
- The single PM node case achieves the highest performance improvement because all data accesses are locally shared for the conventional DSM and private for the hybrid DSM, and there is no synchronization overhead.

6.4. Application 3: H.264/AVC Encoding

As a general and realistic application, H.264/AVC Encoding is mapped to validate the producer-consumer mode and to demonstrate the performance advantage of the hybrid DSM with the dynamic partitioning over the conventional DSM.

As the latest generation of international video standard, H.264/AVC [28] addresses high coding efficiency and good picture quality. In this case study, we parallelize the H.264/AVC encoding algorithm by exploiting its data-level parallelism where a macroblock is treated as the base grain (see Fig. 13). To encode the current macroblock of the current frame, the Motion Vectors of the neighbors to the left, above-left, above and above-right are required to predict the Search Center in the reference frame. Therefore, encoding a frame in parallel is in a wavy way. To parallelize encoding a frame, the rows of the frame are assigned to PM nodes in a round-robin fashion. With this static scheduling policy, to encode a macroblock, only the availability of its above-right neighbor needs to be checked (synchronized). For instance, PM Node 0 encodes the

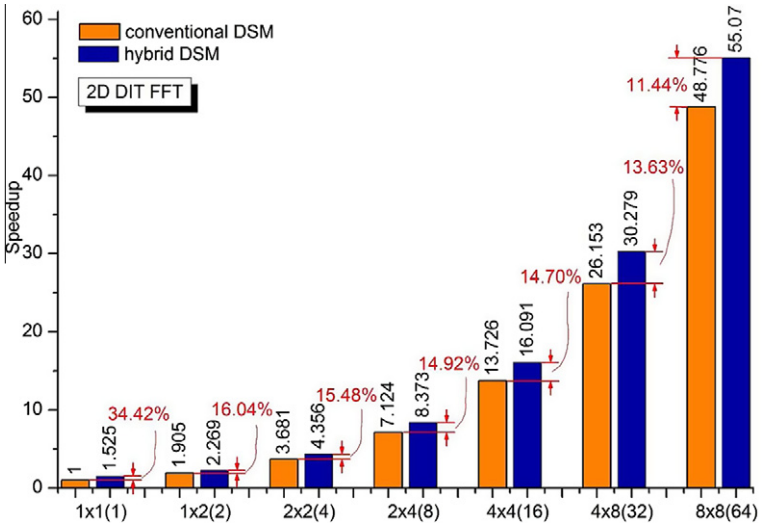

Fig. 12. Speedup and performance improvement of 2D FFT.

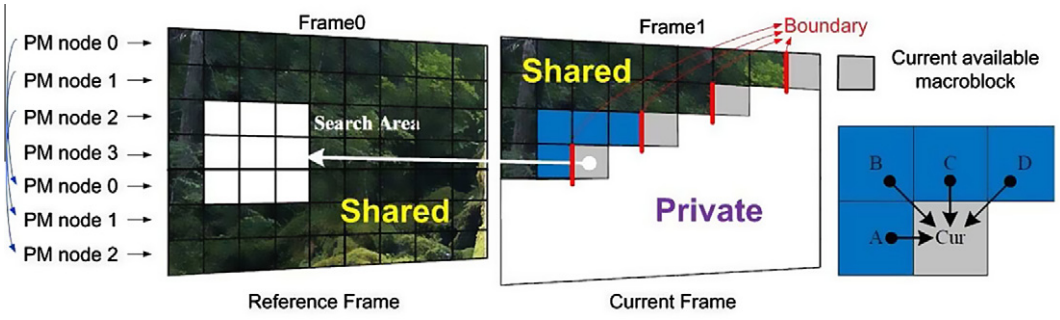

Fig. 13. Macroblock-level parallelism of H.264/AVC encoding using wavefront algorithm.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

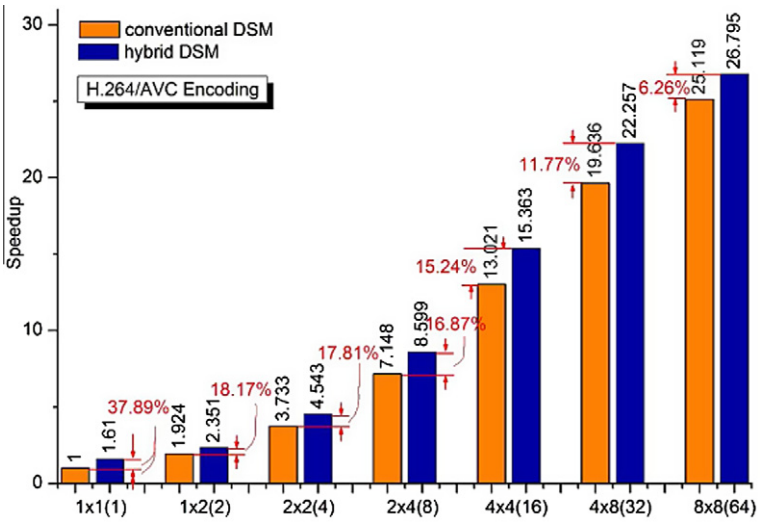

Fig. 14. Speedup and performance improvement of H.264/AVC encoding.

macroblocks in row 1. PM Node 1 cannot encode the macroblocks in row 2 until the corresponding macroblocks in row 1 have been computed by PM node 0. After finishing the computation in row 1, PM Node 0 continues to encode the macroblocks in row 4 according to the round-robin scheduling policy. In our experiment, the frame size is 1024×768, the macroblock size is 16×16, and the block size is 4×4. We run the H.264/AVC encoding application in both the conventional DSM and the hybrid DSM. For the conventional DSM, all block data are shared and memory accesses on them incur V2P address translation overhead. For the hybrid DSM, as shown in Fig. 13, all encoded block data are shared and the block data, which have not been computed by their host PM node yet, are private. The producer-consumer mode described in Section 4.2 is adopted. The host PM node accesses them with fast physical addressing. After the host PM node accomplishes computation on them, it re-configures the boundary address to make them be shared so that other PM node could access them. Compared with aforementioned Matrix Multiplication and 2D FFT, H.264/AVC encoding has more boundary address adjustments.

Fig. 14 shows speedups and the performance improvement of H.264/AVC encoding. As the system size is scaled up , the speedup with the conventional DSM goes up from 1 to 1.924, 3.733, 7.148, 13.021, 19.636 and 25.119, the speedup with the hybrid DSM is from 1.610 to 2.351, 4.543, 8.599, 15.363, 22.257 and 26.795, and we can obtain 37.89%, 18.17%, 17.81%, 16.87%, 15.24%, 11.77% and 6.26% performance improvement. From Fig. 14, we can observe that

- When encoding the current frame, all PM nodes act as both a producer and a consumer. As shown in Fig. 13, PM Node 0 acts as a producer who encodes the macroblocks in row 1. PM Node 1 acts as a consumer who uses the macroblocks in row 1. Meanwhile PM Node 1 also acts as a producer who encodes the macroblocks in row 2 which are used by PM Node 0 when it encodes the macroblocks in row 3. This procedure fits the producer-consumer mode described in Section 4.2. Using the producer-consumer mode, we can reduce and hide the system's total V2P address translation overhead by fast physical addressing and concurrent memory addressing, respectively.
- As the system size is scaled up, larger communication delay slows down the speedups and leads to the decrease of performance improvement. This is because synchronization overhead (it is mainly the waiting time) and network communication delay become more significant.
- The single PM node case achieves the highest performance improvement because all data accesses are locally shared for the conventional DSM and private for the hybrid DSM, and there is no synchronization overhead.

7. Conclusion

In the paper, we have introduced the hybrid DSM with the static and dynamic partitioning techniques in order to improve the system performance by reducing the total V2P address translation overhead of the entire program execution. The philosophy of our hybrid DSM is to provide fast memory accesses for private data using physical addressing as well as to maintain a global memory space for shared data using virtual addressing. The static partitioning eliminates the V2P address translation overhead of those data that are always private during the program execution. The dynamic partitioning supports changing the hybrid DSM as parallel programs are running. Although the dynamic partitioning incurs time overhead of reconfiguring the boundary address, it eliminates the V2P address translation overhead for private data accesses and can thus improve the system performance. We have developed formulas to account for the performance gain with the proposed techniques, and discussed their advantages, limitations and application scope. In experiments, real applications are conducted

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

and the results show that the hybrid DSM demonstrates significant performance advantages over the conventional DSM counterpart.

In the future, we will exploit the temporal and spatial property of workload in combination with our techniques to further enhance the system performance. Another important direction is to extend our work to investigate power savings besides performance. As our hybrid DSM techniques eliminate the V2P address translation overhead of accessing private data, we expect reduction of power and energy consumption for memory accesses.

Acknowledgment

The research is partially supported by the National Natural Science Foundation of China (No. 61070036 and No. 61133007).

References

[1] Horowitz M, Dally W, How scaling will change processor architecture. In: Proceedings of the International Solid-State Circuits Conference (ISSCC'04), Digest of Technical Papers; 2004. p. 132–133.
[2] Borkar S, Thousand core chips: a technology perspective. In: Proceedings of the Design Automation Conference (DAC'07); 2007. p. 746–749.
[3] Vangal S, Howard J, Ruhl G, Dighe S, Wilson H, Tschanz J, Finan D, Iyer P, Singh A, Jacob T, Jain S, Venkataraman S, Hoskote Y, Borkar N, An 80-tile 1.28 tflops network-on-chip in 65nm CMOS. In: Proceedings of the International Solid-State Circuits Conference (ISSCC'07), Digest of Technical Papers; 2007. p. 98–100.
[4] Marinissen E, Prince B, Keltel-Schulz D, Zorian Y, Challenges in embedded memory design and test. In: Proceedings of the conference on Design, Automation and Test in Europe (DATE'05); 2005. p. 722–727.
[5] Loh G, 3D-stacked memory architectures for multi-core processors. In: Proceedings of the International Symposium on Computer Architecture (ISCA'08); 2008. p. 453–464.
[6] Sugimori Y, Kohama Y, Saito M, Yoshida Y, Miura N, Ishikuro H, Sakurai T, Kuroda T, A 2Gb/s 15pJ/b/chip inductive-coupling programmable bus for NAND flash memory stacking. In: Proceedings of the International Solid-State Circuits Conference (ISSCC'09), Digest of Technical Papers; 2009. p. 244–245.
[7] Hennessy J, Patterson D. Computer architecture: a quantitative approach. 4th ed. Morgan Kaufmann Publishers; 2007.
[8] Agarwal A, Bianchini R, Chaiken D, Johnson K, Kranz D, Kubiatowicz J, Lim B.-H., Mackenzie K, Yeung D, The MIT Alewife machine: architecture and performance. In: Proceedings of the International Symposium on Computer Architecture (ISCA'95); 1995. p. 2–13.
[9] Lenoski D, Laudon J, Gharachorloo K, Weber W-D, Gupta A, Hennessy J, et al. The Stanford DASH multiprocessor. IEEE Comput 1992;25(3):63–79.
[10] Kuskin J, Ofelt D, Heinrich M, Heinlein J, Simoni R, Gharachorloo K, Chapin J, Nakahira D, Baxter J, Horowitz M, Gupta A, Rosenblum M, Hennessy J, The stanford FLASH multiprocessor. In: Proceedings of the International Symposium on Computer architecture (ISCA'94); 1994. p. 302–313.
[11] Amza C, Cox A, Dwarkadas S, Keleher P, Lu H, Rajamony R, et al. TreadMarks: shared memory computing on networks of workstations. IEEE Comput 1996;29(2):18–28.
[12] Monchiero M, Palermo G, Silvano C, Villa O, Exploration of distributed shared memory architecture for NoC-based multiprocessors. In: Proceedings of the international conference on embedded computer systems: architectures, modeling and simulation (IC-SAMOS'06); 2006. p. 144–151.
[13] Benini L, Macii A, Poncino M, A recursive algorithm for low-power memory partitioning. In: Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'00); 2000. p. 78–83.
[14] Kim N, Peng R, A memory allocation and assignment method using multiway partitioning. In: Proceedings of the international conference on SoC (SoCC'04); 2004. p. 143–144.
[15] Ozturk O, Kandemir M, Irwin MJ, Tosun S, Multi-level on-chip memory hierarchy design for embedded chip multiprocessors. In: Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'06); 2006. p. 383–390.
[16] Srinivasan S, Angiolini F, Ruggiero M, Benini L, Vijaykrishnan N, Simultaneous memory and bus partitioning for SoC architectures. In: Proceedings of the International Conference on SoC (SoCC'05); 2005. p. 125–128.
[17] Macii A, Macii E, Poncino M, Improving the efficiency of memory partitioning by address clustering. In: Proceedings of the conference on Design, Automation and Test in Europe (DATE'03); 2003. p. 18–23.
[18] Mai S, Zhang C, Zhao Y, Chao J, Wang Z, An application-specific memory partitioning method for low power. In: Proceedings of the International Conference on ASIC (ASICON'07); 2007. p. 221–224.
[19] Mai S, Zhang C, Wang Z, Function-based memory partitioning on low power digital signal processor for cochlear implants. In: Proceedings of the Asia Pacific Conference on Circuits and Systems (APCCAS'08); 2008. p. 654–657.
[20] Kandemir M, Ramanujam J, Irwin M, Vijaykrishnan N, Kadayif I, Parikh A. A compiler-based approach for dynamically managing scratch-pad memories in embedded systems. IEEE Trans Comput Aided Des Integr Circuits Syst 2004;23(2):243–60.
[21] Xue L, Ozturk O, Li F, Kandemir M, Kolcu I, Dynamic partitioning of processing and memory resources in embedded MPSoC architectures. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE'06); 2006. p. 690–695.
[22] Suh G, Rudolph L, Devadas S. Dynamic partitioning of shared cache memory. J Supercomput 2004;28(1):7–26.
[23] Qiu X, Dubois M. Moving address translation closer to memory in distributed shared-memory multiprocessors. IEEE Trans Parallel Distrib Syst 2005;16(7):612–23.
[24] Chen X, Lu Z, Jantsch A, Chen S, Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10); 2010. p. 39–44.
[25] Pande P, Grecu C, Jones M, Ivanov A, Saleh R. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. IEEE Trans Comput 2005;54(8):1025–40.
[26] Leon3 processor. Available from: http://www.gaisler.com.
[27] Nostrum network-on-chip. Available from: http://www.ict.kth.se/nostrum.
[28] Advanced video coding for generic audiovisual services, ITU Recommendation H.264; 2007.

Xiaowen Chen received two bachelor degrees in both Microelectronic and Computer Science from the University of Electronic Science and Technology of China in 2005 and received a Ph. D. degree from the National University of Defense Technology in 2011. His research interests include VLSI design, System-on-Chips, Network-on-Chips, Distributed Shared Memory.

Zhonghai Lu received a bachelor degree from the Beijing Normal University in 1989 and received a master degree and a Ph. D. degree from the KTH – Royal Institute of Technology in 2002 and 2007, respectively. He is currently an associate professor at KTH. He has published over 25 papers in journals, book chapters and international conferences. His research interests include computer systems and VLSI architectures, interconnection networks, system-level design and HW/SW co-design, reconfigurable and parallel computing, system modeling.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009

Axel Jantsch received a Dipl. Ing. degree and a Dr. Tech. degree from the Technical University Vienna in 1988 and 1992, respectively. Since December 2002, he is a full professor in Electronic System Design at KTH – Royal Institute of Technology. He has published over 140 papers in international conferences and journals. At the Royal Institute of Technology, he is heading a number of research projects involving a total number of 10 Ph. D. students, in the areas of system level specification, design, synthesis, validation and networks on chip.

Shuming Chen received a bachelor degree, a master degree and a Ph. D. degree from the National University of Defense Technology in 1982, 1988 and 1993, respectively. Since then, he is with the School of Computer, National University of Defense Technology. Currently, he is a full professor in microprocessor design. His research interests include processor architecture, high performance circuits, custom design for reliability, and System-on-Chips. He has published over 110 papers in conferences and journals.

Shenggang Chen received a bachelor degree, a master degree and a Ph. D. degree from the National University of Defense Technology in 2004, 2006 and 2010, respectively. Since then, he works in the School of Computer, National University of Defense Technology. His main research interests include VLSI design, Digital Signal processor and video encoding.

Huitao Gu received a bachelor degree, a master degree and a Ph. D. degree from the National University of Defense Technology in 2004, 2006 and 2011, respectively. His main research interests include high performance microprocessor design and video encoding.

Please cite this article in press as: Chen X et al. Reducing Virtual-to-Physical address translation overhead in Distributed Shared Memory based multi-core Network-on-Chips according to data property. Comput Electr Eng (2012), http://dx.doi.org/10.1016/j.compeleceng.2012.04.009