2018 IEEE International Symposium on High Performance Computer Architecture

SmarCo: An Efficient Many-Core Processor for High-Throughput Applications in Datacenters

Dongrui Fan[*§], Wenming Li[*‡], Xiaochun Ye[*], Da Wang[*¶], Hao Zhang[*¶], Zhimin Tang[*], and Ninghui Sun[*]
[*]*SKL Computer Architecture, ICT, CAS, Beijing, China*
[§]*School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing, China*
[¶]*SmarCo Co., Ltd., Beijing, China*
{*fandr, liwenming, yexiaochun, wangda, zhanghao, tang, snh* }*@ict.ac.cn*

***Abstract* - Fast-growing high-throughput applications, such as web services, are characterized by high-concurrency processing, hard real-time response, and high-bandwidth memory access. The newly-born applications bring severe challenges to processors in datacenters, both in concurrent processing performance and energy efficiency. To offer a satisfactory quality of services, it is of critical importance to meet these newly emerging demands of high-throughput applications in the future datacenters in a more efficient way.**

In this paper, we propose a novel architecture, called SmarCo, which allows high-throughput applications to be processed more efficiently in datacenters. Based on the dominant characteristics of high-throughput applications, we implement large-scale many-core architecture with in-pair threads to support high-concurrency processing; we also introduce a hierarchical ring topology and laxity-aware task scheduler to guarantee hard real-time response; furthermore, we propose high-throughput datapath to improve memory access efficiency. We verify the efficiency of SmarCo by using simulators, large-scale FPGA and prototype with TSMC 40-nm technology node. The experimental results show that, compared to Intel Xeon E7-8890V4, SmarCo achieves 10.11X performance improvement and 6.95X energy-efficiency improvement with higher throughput and a better guarantee of real-time response.

1. INTRODUCTION

With the recent flood of all kinds of Internet services, datacenters are under considerable pressure to offer satisfactory Quality of Service (QoS) under ever-increasing data volumes and number of concurrent users. To provide a high quality of service, including web search, social network, E-commerce and video sharing, Google, Facebook, Microsoft, and Amazon are continuously expanding the capacities of their datacenters. As these applications are prevalent all over the word on the Internet, datacenters are required to simultaneously process ever-increasing concurrent tasks, usually on a scale of 100 million tasks. For instance, Facebook deploys more than 100000 servers to handle almost one trillion page views per month and 1.2 million photo views per second[1]. Different types of applications, together with large-scale concurrent processing requirements are often overloading servers in datacenters, while also presenting designers of microprocessors with considerable challenges.

Currently, the hearts of servers used in datacenters are mostly general-purpose processors. As the Moore's Law still plays the remaining effect, more transistors are integrated into a chip to provide higher computing performance. Increasing the density of processors currently remains the key method to deal with web service applications in datacenters, for instance, Intel's 72-core Knights Landing, IBM 96-thread POWER9 processor.

However, current general-purpose processors are more inclined to process conventional computing-intensive high-performance applications, rendering the processing of high-throughput computing (HTC) applications inefficient. A number of studies[1][2][3][4][5] strongly indicate that conventional general-purpose processors are not suitable for high-throughput computing applications in datacenters. What's worse, while we are still enjoying the benefit of Moore's Law's continuation, the power consumption appears one of the most concerns when designing future processors. For example, Xeon Phi processors consume more than 200 Watts, and E7 series processors consume more than 100 Watts. As a consequence, datacenters usually operate at tremendous cost, both for power supply, and cooling of processors.

We observed that HTC applications require high-concurrency data processing, hard real-time response, and high-bandwidth memory access. These newly emerging characteristics of HTC applications are very different from high-performance computing (HPC) applications, which pursue high computing speed of single task like Linpack. However, HTC applications pursue "throughput", i.e. they process as many tasks per time unit as possible. Therefore, the criterion for evaluating HPC processors is not suitable for HTC applications, since task throughput, not the computing speed of single task, is the most important evaluation criterion for HTC processors.

We tested three basic algorithms of HTC applications running on conventional high-performance processors (Intel Xeon E7-8890V4). As shown in **Fig. 1** (a) and (b), apart from the instruction starvation, the idle ratio goes up with the increase of thread number, which means the conventional cores are not suitable for high concurrent execution. As shown in **Fig. 1** (c) and (d), conventional multi-level caches are not suitable for HTC applications, because they suffer from high miss ratio and cause longer access latency.

[‡] Wenming Li is the corresponding author of this paper.

2378-203X/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCA.2018.00057

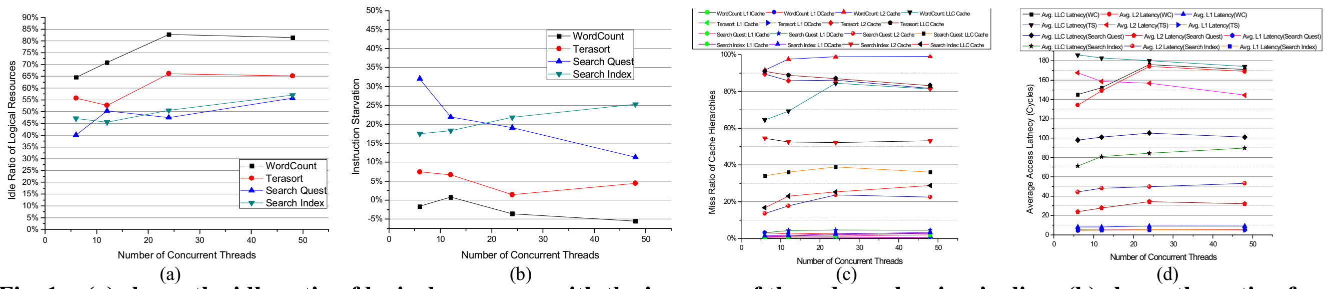


Fig. 1 : (a) shows the idle ratio of logical resources with the increase of thread number in pipeline; (b) shows the ratio of instruction starvation with the increase of thread number in pipeline; (c) shows the miss ratio of L1, L2 and LLC; (d) shows the average access latency of L1, L2 and LLC

Another important HTC application, Content Delivery Network (CDN)[6], which currently serves as part of edge computing devices, uncovers more collisions between HTC applications and conventional processors. In this study, we built a CDN system using Nginx (engine x) [7], and a 10Gbps network interface card (NIC) to provide video service.

The rate of the videos is 25Mbps and the size of videos is more than 1GB. From the experiment results (see **Fig. 2**) we can see that with the number of connection reaches the limit due to the NIC bandwidth, the utilization of CPU is commonly under 10% and the branch miss ratio exceeds 10% when the number of client approaches the limit. For cache hierarchy, the miss ratio of L1 cache is about 40%, which is higher than conventional applications on HPC processors. All the above-mentioned performance values reveal the severe mismatch between HTC applications and conventional processors.

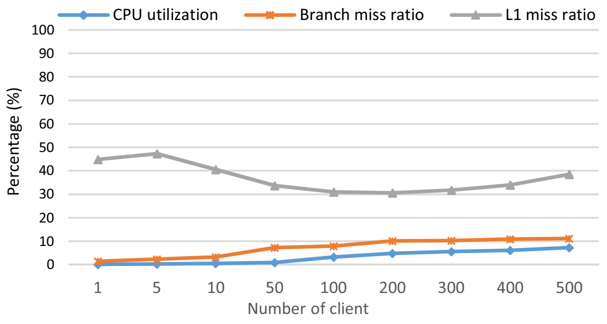


Fig. 2 : Performance of conventional processor under typical CDN application

In this paper, we designed a novel datacenter processor, called SmarCo. The microarchitecture of SmarCo is more suitable for handling HTC applications. We first verified the design of our SmarCo using a simulator and a large-scale FPGA. Furthermore, we implemented a small-scale prototype with 40-nm technology node. The SmarCo was designed as an accelerator, which can be simply integrated into existing systems with PCIe interface. Our specific contributions are as follows:

- **Analysis of HTC Applications on Conventional Processors —** We analyze representative HTC applications on current powerful processors, which gives us a deep insight of HTC requirements on architectures before designing our SmarCo.
- **SmarCo Architecture Design Method —** We design and evaluate SmarCo processor, containing in-pair threads, topology, on-chip memory, NoC, programming model and tasks scheduling, which shows better performance than conventional processors in processing HTC applications.
- **Evaluation and Verification of Architecture —** To verify the efficiency of SmarCo architecture, we build a parallel simulation platform and a large-scale FPGA verification platform for many-core architecture evaluation during the design stage. We also tape out SmarCo prototype with TSMC 40-nm technology node.

In this paper, we first analyze the challenges general-purpose processors represent for HTC applications in section 2. Based on this analysis, the highlights of SmarCo architecture are presented in Section 3. To prove the efficiency and reliability of our SmarCo, we describe the evaluation results of large-scale simulation platform, FPGA platform, and prototype in Section 4. Section 5 is related work. We discuss our findings in section 6. The final section is the conclusion.

2. CHALLENGES BROUGHT BY HTC APPLICATIONS

HTC applications mainly composed of various workloads of Internet services, such as web search, social network, E-commerce, and video on demand. As predicted by Internet Data Center (IDC), the volume of global data will reach 35 ZB by 2020. In addition, 100 million concurrent requirements of HTC applications leaves datacenters with a heavy burden, forming the challenges of processors of datacenter servers. As shown in Fig. 1, to understand the difference between HPC and HTC more vividly, we use car racing as a metaphor for the difference. HPC processors are designed for faster processing speed of single task like Linpack or FFT. In car racing, it means the fastest car is the winner. However, HTC processors should be capable of achieving higher data throughput per time unit, which means to process as many independent tasks as possible per time unit. It means the winner is the team with more cars passing the finishing line before the deadline, not the team with the fasted cars!

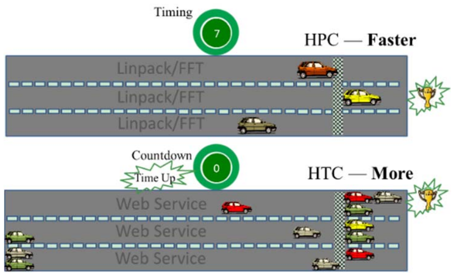


Fig. 3 : Difference between HPC and HTC

In HTC situations, the users may tolerate certain service delay when searching web pages or chatting with others. Therefore, by utilizing such feature, datacenters are capable of serving as many users as possible with satisfactory QoS before the deadline.

3. SMARCO ARCHITECTURE

In this section, we present a detailed architecture of SmarCo to meet the requirements of high throughput processing, hard real-time response, and high-bandwidth memory access.

As shown in **Fig. 4**, we integrate 256 cores to enhance SmarCo's processing ability of concurrent processing. Cores are arranged with a hierarchical ring, including main-ring and sub-ring. Each sub-ring comprises 16 cores and is connected to main-ring by routers. Four separate DDRs are connected to the main-ring with equal space. Other devices are attached to the main-ring as well.

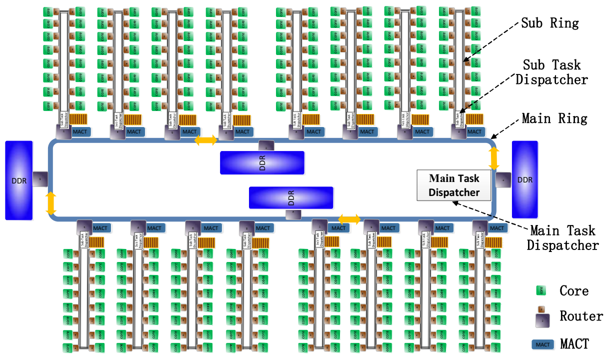


Fig. 4: Overview of SmarCo Architecture Framework

3.1 Thread Core Group

Different with HPC applications, HTC applications are usually memory intensive applications. Frequent memory accesses and long network diameter of large-scale NoC aggravate the "Memory Wall". To improve the utilization of computation resource, we need to exploit methods which can overlap memory access latency. In SmarCo, we propose Thread Core Group (TCG), which is able to support a number of threads running simultaneously. In TCG, Dispatcher, ALU, and AGU are private for each thread, I-cache and local memory are shared among paring threads. **Fig. 5** shows the overview of a TCG core (left) and hardware resources owned by a thread (right). In our SmarCo, each core contains a 4-wide issue, 8-stage depth, in-order superscalar pipeline, similar to an extension of ARM11 series processor[8] architecture. A TCG core is able to support 8 threads living simultaneously, but only 4 threads are running at the same time. Each core owns 16KB instruction cache, 16KB data cache and 128KB local memory (Scratchpad Memory).

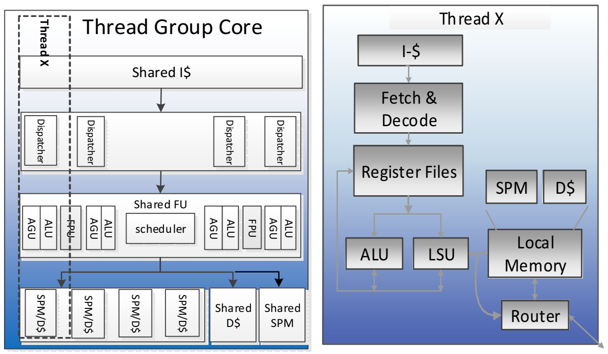


Fig. 5: Overview of a TCG core and hardware resource for a thread

3.1.1 In-Pair Threads

As discussed above, HTC applications are memory-intensive, therefore, long memory access latency normally causes lower pipeline utilization. In previous studies [1][9], SMT (simultaneous multi-threading) or similar techniques were advocated to improve the performance of cores in web service datacenters. However, although many previous studies[10][11] have proposed methods to overlap latency by using SMT or similar techniques, unlike conventional HPC applications, task threads of HTC applications are simpler in behaviors and have little relevance to each other, what's worse, they frequently suffer long memory access latency at the same time, which makes it less efficient to use previous SMT scheduling methods to overlap memory access latency for each other.

To further improve the efficiency of SmarCo, a simple but efficient dual-thread interleaved multi-thread scheduling mechanism is proposed in this paper, we call it in-pair threads. The in-pair threads mechanism combines the advantage of SMT and coarse-grained multithreading. In the proposed mechanism, each thread is coupled with a friend thread. When a thread encounters a long latency caused by cache/SPM miss, its friend thread starts running immediately. Alternate execution of the in-pair threads will hide memory access latency for each other, thus increases the utilization of pipeline, even the in-pair threads are in same behaviors. Note that only one of the in-pair threads executes at any time. Each thread runs when its friend thread encounters a memory access.

Fig. 6 shows a schematic diagram of control flow in the pipeline with dual-thread interleaved multi-thread scheduling mechanism. A hardware scheduler is applied to dispatch paired threads based on the memory access state.

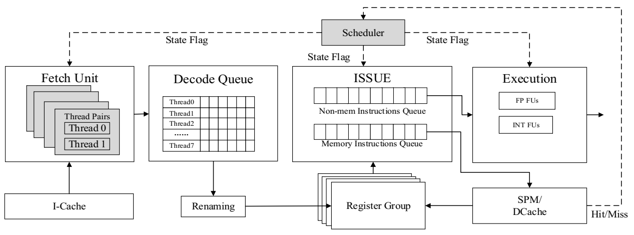


Fig. 6: Schematic diagram of dual-thread interleaved multi-thread scheduling mechanism

In addition to its own thread id, every thread also contains a thread pair ID to indicate which pair it belongs to. Each thread is working with two states: *Waiting* and *Running*. The software system first sets the initial state of in-pair threads. When a *Running* thread encounters SPM/DCache miss, its state changes into *Waiting*. Once the missed data is returned, the thread is scheduled back to *Running* when its friend thread encounters SPM/Dcache miss. Since HTC applications are memory-intensive and behave similarly, the proposed in-pair threads scheduling mechanism works well.

3.1.2 Shared Instruction Segment

For a number of special algorithms (for example, KMP, Wordcount) in HTC applications, parallel threads are normally running on the same instruction segment, processing different dataset. Under such circumstance, threads in a TCG are able to share the same instruction segment in the instruction cache. In order to improve the efficiency of instruction fetch, we prefetch the whole instruction segment, which is shared among multiple threads, to local SPM using DMA mechanism. Thus we save the capacity of instruction cache, as well as decrease instruction fetch miss ratio. The sharing operations are transparent to programmers. Task schedulers are responsible to share instruction segment among neighboring threads.

3.2 Hierarchical Ring Topology

With so many cores integrated into a chip, an efficient topology becomes crucial to performance. Mesh (e.g., Tile64[12]), ring (e.g., Xeon Phi[13]), and bus (e.g., Cortex-A series[14]) are prevalent in state-of-the-art NoC. Compared to mesh network, ring topology has simpler routing algorithm and lower per-hop latency[15]. In the ring topology, the data transmission latency is more predictable. In addition, routers of ring topology consume less on-chip resources than mesh. Furthermore, a core running HTC application usually has little shared data with others, especially with cores far away. Therefore, to improve data throughput and predictability of packet transmission latency, as well as to reduce the probability of congestion in the central part of NoC, we choose the hierarchical ring topology to arrange 256 cores in a chip, as shown in **Fig. 4**.

In our hierarchical ring NoC, the main-ring is responsible for communicating with peripheral I/O devices, such as PCIe and DDR Controller. 16 sub-rings connect to the main-ring.

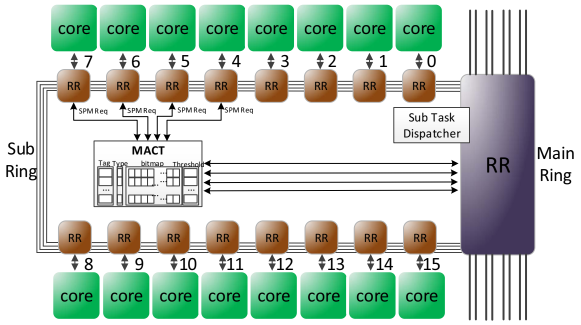


Fig. 7: Sub-ring of SmarCo hierarchical ring

Both main-ring and sub-ring are composed of routers and bidirectional channels. Cores are able to choose both directions of sub-ring to send packets based on the congestion condition, as showm in **Fig. 7**.

3.3 High-density NoC

In our SmarCo, the main-ring is composed of eight datapaths logically. Each direction of the ring is configured with three fixed datapaths, and the rest two datapaths are bidirectional datapaths. The sub-ring is made up of four datapaths, which are configured as one fixed datapath of each direction and two bidirectional datapaths. The width of each logic datapath is 64 bits, so the total bandwidth of main-ring is 512 bits and the sub-ring is 256 bits.

For many-core processors, NoC takes the responsibility to move data among different units. HTC applications bring severe challenges to NoC. **Fig. 8** shows the memory access granularity distribution of six typical HTC applications and eleven conventional applications from SPLASH2, where we see that the memory access granularity of HTC is much smaller than that of conventional applications. Transmitting small data packets through wide datapath of NoC is really a kind of waste of bandwidth, as well as power/energy. For example, suppose the datapath width of NoC is 64-bit wide when a 16-bit packet is passing through, 3/4 of the bandwidth is wasted.

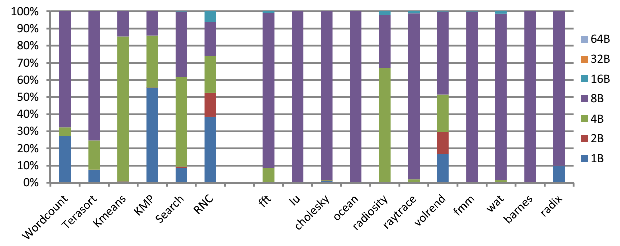


Fig. 8: Distribution of memory access granularity in HTC applications (left) and conventional applications (right)

We propose a high-density NoC to adapt to small packets in HTC applications to make full use of NoC bandwidth. **Fig. 9** illustrates the physical link between two routers. As depicted, the conventional wide link is divided into four self-

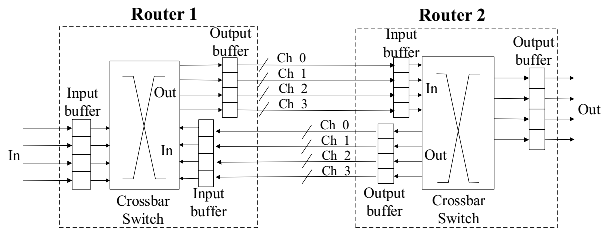


Fig. 9: Physical link between two routers of high-density NoC

governed narrow links. Packet, which is transferred by a conventional wide link, occupies all wide links during transmission, independently of the size of the packet. For our high-density NoC, small packet only occupies channels they really need, leaving the free channels to other small packets.

To support self-governing small channels, additional control logic and control algorithms are added. As illustrated in **Fig. 10**, buffer, crossbar, control logic, and channel are all divided into small granularities. We also propose greedy allocation algorithm to take full advantage of our high-density NoC, which means to choose as many packets as possible in switch allocation stage to fully utilize the whole channel to improve data throughput. When a series of flits are waiting to pass the crossbar switch, if the total size of adjacent flits is smaller or equal to the width of the link, flits are able to pass the link simultaneously. Furthermore, if free space is still available, packets from other input directions will occupy it and pass the crossbar switch simultaneously.

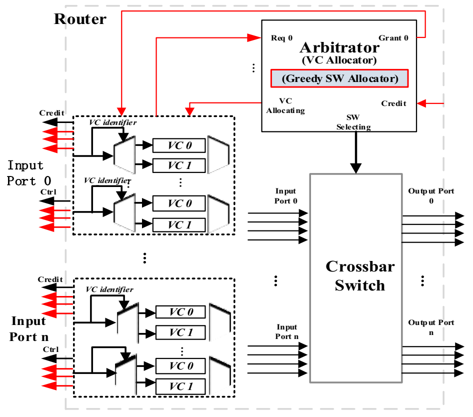


Fig. 10: Router structure of high-density NoC

3.4 Memory Access Collection Table

As analyzed in section 3.3, a large portion of memory access packets is in small granularity. Additionally, with so many cores integrated into a chip, a considerable number of unrelative, discrete, and small packets are shuttling in NoC. Although the high-density NoC in our SmarCo is proposed to handle such packet pattern, it is of little use when confronting packets in different time and different space of NoC. Thus in SmarCo, a mechanism named memory access collection table (MACT) is developed to process discrete and small grannularity packets from adjacent cores in batch. MACT collects memory access packets for in sub-rings. **Fig. 11** shows the execution mechanism of MACT.

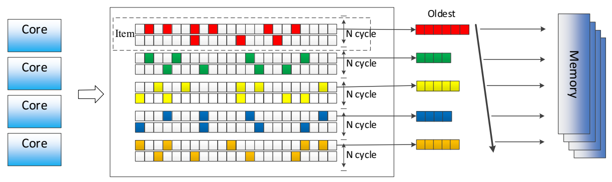
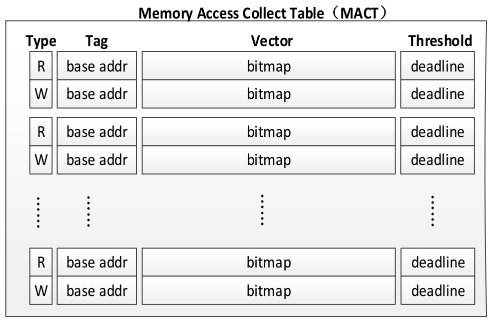


Fig. 11: Execution mechanism of MACT

Discrete memory accesses are collected into MACT from cores in a certain time limit. The memory accesses are then packed and sent to memory in batch when table capacity is full or reaching the deadline. The deadline means the longest time that every item can stay in MACT. Each item of MACT must be packaged and sent to memory in N cycles to maintain timeliness of memory access requests.

Fig. 12 shows the logic structure of MACT, with each line containing four domains, namely *Type*, *Tag*, *Vector*, and *Threshold*. *Type* indicates the type of memory access, Write or Read. *Tag* stores the base address of this line. *Vector* comprises bitmaps, with each bit of bitmaps representing one byte value in the address. The address equals base address plus bitmap vector. *Threshold* is a timer, which indicates the configurable deadline of each MACT line. Once each line reaches the deadline, the memory accesses must be sent to memory in batch to guarantee the QoS. On the other hand, once the bitmap is full, the line will be also packed and sent to memory.


Memory Access Collect Table（MACT）

Type	Tag	Vector	Threshold
R	base addr	bitmap	deadline
W	base addr	bitmap	deadline
R	base addr	bitmap	deadline
W	base addr	bitmap	deadline
⋮	⋮	⋮	⋮
R	base addr	bitmap	deadline
W	base addr	bitmap	deadline

Fig. 12: MACT logic structure

In case of applications of superior real-time priority, memory access requests are filtered to judge whether they are suitable to be collected into MACT. The requests, which are marked of superior real-time priority, bypass MACT and flow to memory in an ordinary way.

3.5 Memory Hierarchy

Memory hiearchy palys a crucial role in coping with "Memory Wall". Optimized memory hierarchy will lead to higher performance, especially for memory-intensive applications. In SmarCo, we spent great efforts to think about the design of on-chip memory.

3.5.1 ScratchPad Memory (SPM)

ScratchPad Memory (SPM), which is characterized by faster access speed, higher energy, and silicon area efficiency compared to ordinary caches, is being widely used as programmer visible local memory in current processors. As previous studies revealed, SPM reduces the consumption of energy by 40% and reduces the consumption of silicon area by 34%[16]. Additionally, the access latency of SPM is more predictable than caches[17], it is, therefore, more suitable for use in HTC architectures, especially in processing applications that require the hard real-time response.

In SmarCo, each core is configured with an SPM, which can also be shared among cores in sub-ring and initialized of unified addressing with main memory. Programmers are responsible to manage the SPM by allocating data in the range of SPM address space. LSQ units check the address and judge whether to send the requirement to the cache or to the SPM when issuing load/store instructions. **Fig. 13** shows the data flow and control flow of memory write operation in SmarCo. When occurring an access miss in SPM or executing data prefetch, data are exchanged between SPM and memory. MACT collects memory access requests from SPM and sends them to memory in batch. Note that data are not cached in MACT, only memory access requests are necessary, including address, request type, and so on.

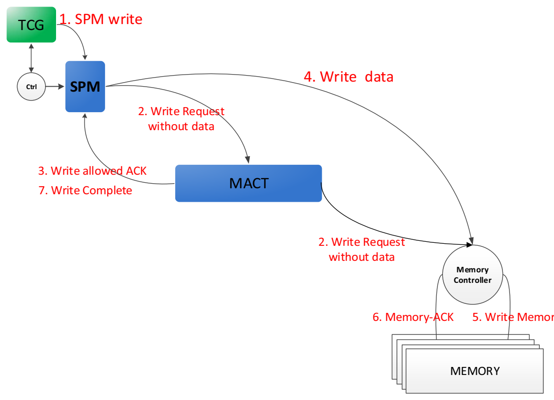


Fig. 13: Datapath of memory write operation in SmarCo with SPM local memory and MACT structure

Because SPM can be shared among sub-ring cores, a certain portion of shared data may be transmitted among neighboring cores, therefore, it is necessary to support data transmission between SPMs. To reduce transmission latency while keeping cores computing simultaneously, we apply DMA to transmit data between SPMs. Thus SPMs spare top 256 bytes space to act as control registers, such as DMA source address, DMA destination address, and data size.

3.5.2 Direct datapath for memory access

"Memory Wall" represents a considerable challenge in processor design. The latency of memory access increases along with the number of cores integrated into a chip, further aggravating the problem of the "Memory Wall". 256 cores are arranged in hierarchical rings in our SmarCo. Memory access requirements need to pass through sub-ring and main-ring before they reach the memory. To decrease memory access latency, we implemented a fast memory access datapath for each sub-ring in the star shape, As shown in **Fig. 14**. Memory accesses are able to directly reach memory without passing sub-ring and main-ring. The direct datapaths are used for control messages and memory read requirements, which are marked with high real-time priority, especially when the ring network is in heavy congestion.

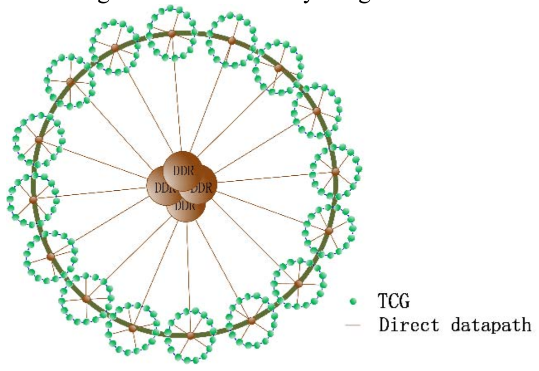


Fig. 14: Logic diagram of direct datapath

3.5.3 DDR Memory

Four memory controllers are connected to the main-ring with the same interval between the controllers, as shown in **Fig. 4**. Each controller is connected to a 16G 128bits DDR4-2133 memory to serve as the main memory for 256 cores, the total memory bandwidth can reach 136.5GB/s.

3.6 Programming Model

In SmarCo, we implemented the basic programming model based on POSIX threads. Programmers can easily create and terminate threads by calling library functions, such as *pthread_create()*, and *pthread_exit()*.

To facilitate the construction of target HTC applications, we further implement a widely used MapReduce programming model. Three computing nodes are needed to support MapReduce programming, namely Master node, Map node and Reduce node. The master node may be host CPU, Map node and Reduce node are processing cores of our SmarCo. Thanks to this framework, it will be possible for programmers to concentrate on the development of target applications, rather than complex management problems of parallel programming, such as distributed storage, tasks scheduling, load balancing and network communication.

Fig. 15 shows an example of MapReduce running model in SmarCo. Firstly, input dataset is sliced into equal stacks by MapReduce framework based on the hardware resources of SmarCo. Secondly, Master node mappings Map tasks to SmarCo cores, for example, subring *0* to *N* are chosen to execute Map tasks, as shown in **Fig. 15**. Thread tasks are attached to TCG cores. If the capacity of TCG SPM is sufficient, the dataset is stored in the SPM, otherwise, the

SPM will exchange data with main memory. The results will be stored into the SPM of those subrings used, which are chosen as Reduce nodes, for example, subring *K1* to *Km*, as shown in **Fig. 15**. Thirdly, Reduce nodes execute *Reduce()* function to process results of Map nodes. Finally, the output of Reduce nodes will be sent to Master node to finish *Merge()* operation and export final output.

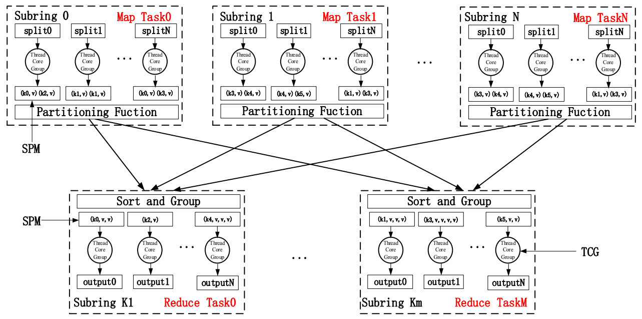
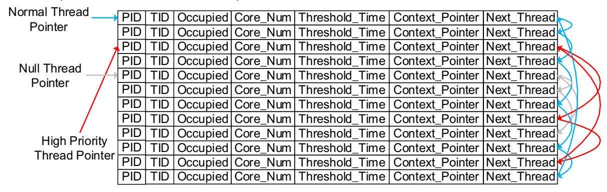


Fig. 15: Map and Reduce stages of MapReduce model running in SmarCo

3.7 Laxity-Aware Task Scheduler

To ensure load-balance and QoS, we implemented a laxity-aware task scheduler, including main-scheduler and sub-scheduler. As shown in **Fig. 4**, the main-scheduler is attached to the main-ring, which dispatches tasks received from the host CPU to ensure that the whole SmarCo chip is running with good load-balance. The sub-scheduler are attached to every sub-ring to dispatch tasks to TCG threads of the sub-ring to make sure the QoS of thread tasks.

To further improve the ability of QoS service, we implemented a hardware scheduler in each sub-ring. The hardware scheduler includes three different types of chain table, namely null thread chain table, normal thread chain table and high priority thread table, as shown in **Fig. 16**. Thread tasks of different priorities will append to different chain tables, waiting for their execution according to their execution laxity, which is calculated based on current time and their deadline. To save area and power consumption, while reducing the complexity of implementation, we use RAM, instead of CAM, to build the hardware schedulers.



PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread
PID	TID	Occupied	Core_Num	Threshold_Time	Context_Pointer	Next_Thread

Fig. 16: Hardware-based task scheduler of sub-rings

4. SmarCo VERIFICATION

In this section, we performed a series of experiments to test the efficiency of the proposed architecture of our SmarCo. Firstly, before we taped out the chip, we simulated and evaluated the SmarCo design on the simulator. Secondly, we built a 256-core FPGA verification platform to verify the efficiency of SmarCo architecture, especially for NoC design. Finally, we taped out a small-scale SmarCo prototype using TSMC 40-nm technology node.

4.1 Benchmarks

We extracted six representative microbenchmarks from HTC applications to evaluate our SmarCo in different dimensions of high-throughput applications requirements, namely high-concurrency processing, hard real-time response, and high-bandwidth memory access. We chose Wordcount, Terasort, Search, K-means, KMP, and RNC as representative HTC benchmarks. WordCount and TeraSort were ported from Phoenix++ [18], which is a typical MapReduce framework. WordCount is used to count the number of words from files. Terasort is used to sort large-scale datasets according to the keys. Search was extracted from Xapian project[19], which is a search engine used in web search. K-means is one of the unsupervised learning algorithms that solve the well-known clustering problem. KMP is a string matching algorithm which is a basic algorithm used in bigdata processing. RNC is a governing element in the UMTS radio access network (UTRAN) [20], which is able to respond in real-time.

4.2 Simulator Verification

To verify the feasibility and rationality of our SmarCo architecture, we construct a parallel based on PDES (Parallel Discrete Event Simulation) mechanism. The simulator is divided into two parts: framework and function modules. The framework is responsible for synchronization, communication, as well as parallel acceleration. Function modules, such as core, router, memory, and NoC, are integrated on the framework to build a complete target system. We built a 256-core hierarchical ring SmarCo simulation model to verify our SmarCo design.

4.2.1 Performance of Thread Core Group

To evaluate the efficiency of TCG architecture, we analyzed IPC of cores with the increase of threads number from 1 to 8. As shown in **Fig. 17**, since TCG is a 4-issue superscalar pipeline, IPC increases rapidly from 1 thread to 4, almost "growing linearly". As the thread number doubled from 4, IPC increases slowly except for search benchmark,

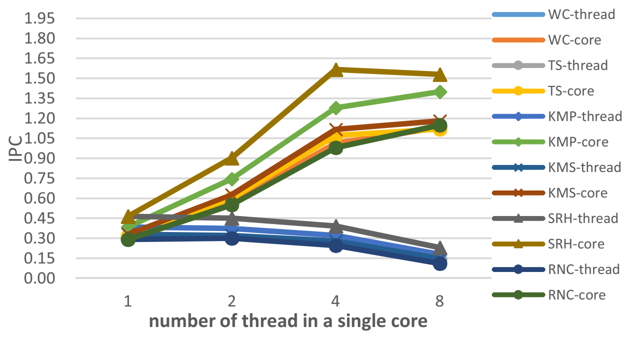


Fig. 17: IPC of each core under different thread configuration

which even decreases slightly. Because search benchmark is characterized by lower memory instruction, it can not take full advantage of our pairing threads mechanism.

4.2.2 High-density NoC performance evaluation

In the hierarchical ring, the bandwidth of main-ring is 512 bits and the sub-ring is 256 bits. In this experiment, we sliced the bandwidth of hierarchical ring into 2 bytes ones, 4 bytes ones, 8 bytes ones and 16 bytes ones separately in our comparison experiments. **Fig. 18** indicates that when the bandwidth of sliced links decreases from 16 bytes to 2 bytes, the throughput rate (packets per time unit) increases. For benchmarks that are characterized by a larger portion of small granularity memory access packets, e.g. KMP and RNC (see **Fig. 8**), they still benefit when the bandwidth of sliced links decrease from 4 bytes to 2 bytes. K-means contains few 1 Byte or 2 Bytes memory access packets, so almost nothing benefits can be got when decreasing from 8 bytes to 2 bytes.

From the results, we can see that high-density NoC perform better for HTC applications than conventional NoC. The higher portion of small granularity, the greater it will benefit from high-density NoC.

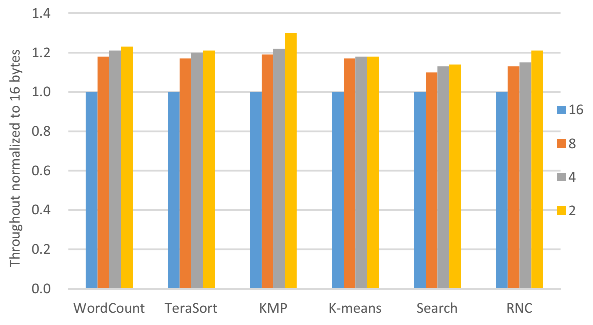


Fig. 18: Improvement of throughput rate under different channel width (Byte)

4.2.3 MACT performance evaluation

MACT structures are implemented to collect discrete memory accesses from adjacent cores in a sub-ring. Since memory accesses are waiting to be sent to memory in MACT until reaching time threshold, therefore, the time threshold is of great importance for the timeliness of memory accesses. As shown in Fig. 19, we can see that a threshold is set to 16 cycles is best for most of the benchmarks, thus we use 16 cycles as the threshold for subsequent experiments.

Fig. 20 shows the execution results with MACT in our SmarCo. We evaluated four aspects influencing performance, namely execution speedup, memory access request latency, bandwidth utilization of NoC, and the number of memory access requests. Results reveal that applications, which are characterized by a larger proportion of memory access instructions and small granularity of data transfer, achieve greater execution speedup. However, we also note that the speedup of K-means is less than 1, it is because that K-means suffers from the increase of memory access latency. MACT collects memory accesses and sends to memory either when the capacity is full or time up, which leads to the increase of the memory access latency. The number of request decreases because of the collection and processing in batch. Since thread tasks with the high priority of real-time may bypass MACT, QoS of these tasks can be guaranteed. The utilization of bandwidth increases because of the speedup of execution.

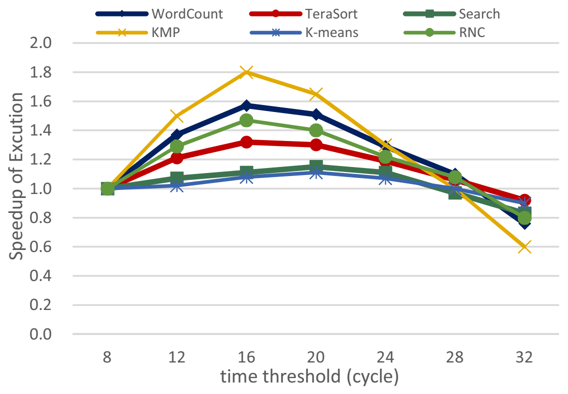


Fig. 19: Speedup with different time threshold (all values are normalized to the speedup value of 8 cycles)

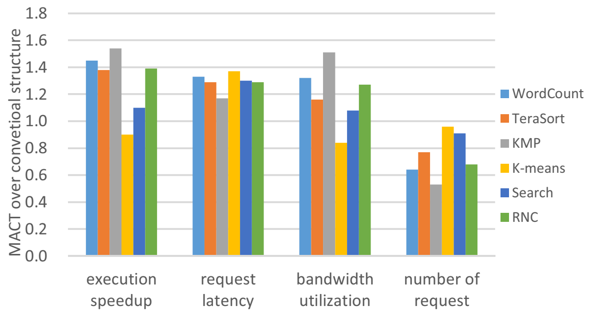
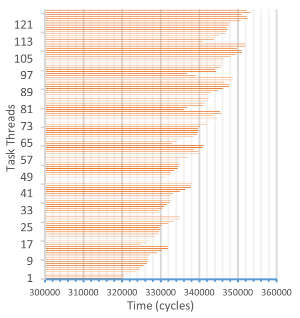


Fig. 20: Comparison between MACT and conventional structure

4.2.4 Laxity-aware task scheduler evaluation

We used RNC benchmark to evaluate the performance of laxity-aware task scheduler in our SmarCo, because RNC benchmark requires the ability of hard real-time response. Once the task thread exceeds the response deadline, the task is considered to be failed. As shown in **Fig. 21**, both figures show the execution result of one of the sub-rings, 128 threads in total. The deadline of all task threads is set to 340000 cycles. In the left figure, we used the Deadline Scheduler[21], which dynamically schedules tasks based on the remaining time of each task to improve the success rate of all the tasks.

As a result, the exit time of all the thread tasks ranges from 320000 cycles to 354000 cycles. The right figure is the execution result using our laxity-ware task scheduler based on hardware. The exit time of all the task threads ranges from 334000 cycles to 342000. Although the execution time of the earliest exit thread is greater than that of the left figure, we improved the success rate of overall task threads.



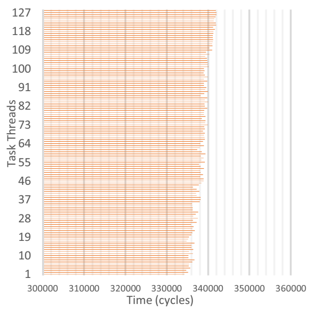


Fig. 21: Execution time of each task thread in a sub-ring, the left uses software scheduler, the right uses laxity-aware task scheduler based on hardware

4.2.5 Area overheads and power consumptions

In this section, we use McPAT[22], CACTI 6.0[23], and Orion 2.0[24] to estimate the area overheads and power consumptions of our SmarCo processor design. The operating frequency of SmarCo is 1.5GHz. **Table 1** shows the estimation results at 32nm technology node (considering the supporting of these evaluation tools).

Table 1: Area overheads and power consumptions of SmarCo with 32nm technology nodes

Main Components	Area (mm^2)	Power (Watt)
Cores	634.32	209.91
Hierarchy Ring	57.43	14.55
MACT	1.43	0.14
SPM+Cache	44.90	1.84
MC+PHY	12.92	13.65
Total	**751.00**	**240.09**

4.2.6 Performance and energy-efficiency comparison

We chose a high-performance processor, Intel Xeon E7-8890V4, to compare with our SmarCo. **Table 2** illustrates the hardware cofigurations of these two processors.

Table 2: Parameters of Xeon E7-8890V4 and SmarCo

	Xeon E7-8890V4[25]	**SmarCo**
Core	24 cores, 48 threads 2.2GHz~3.40GHz	256 cores, 2048 threads 1.5GHz
Cache & SPM	0.77MB L1 I$, 0.77MB L1 D$, 6MB L2 $, 60 MB LLC	4MB L1 I$, 4MB L1 D$, 32MB SPM
NoC	9.6 GT/s QPI Intel QPI	Hierarchy Ring, sub-ring 256-bit width main-ring 512-bit width
Memory	256GB, 85 GB/s	64GB, 136.5GB/s
Process	14 nm	32 nm
Power	165 Watt	240 Watt
Die Area	-	751 mm²

We compared both performance and energy-efficiency, the results are shown in **Fig. 22**. All the six typical HTC applications are programmed with MapReduce model. From the results, we can see that our SmarCo achieves a 4.86X to 18.57X speedup compared to Intel Xeon E7-8890V4. The average speedup is 10.11X. Our SmarCo also obtains a 3.34X to 12.77X improvement for energy-efficiency, as presented in **Fig. 22**, with 6.95X average speedup.

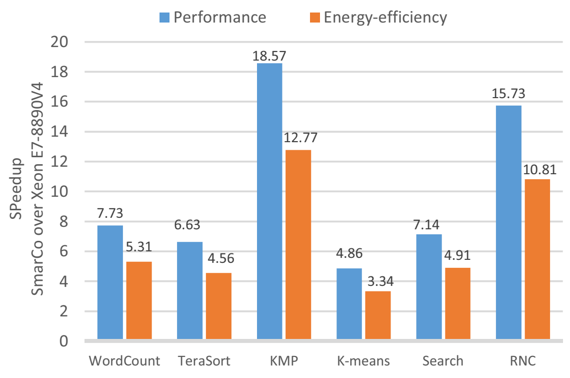


Fig. 22: Comparison of performance and energy-efficiency (SmarCo over Intel Xeon E7-8890V4)

As presented in **Fig. 23**, we evaluated the scalability of our SmarCo using KMP benchmark. With the number of threads goes up, the performance of Xeon E7-8890V4 increases and reaches the best point around 32 to 64 threads, then the performance goes down because of threads creating and threads scheduling, which consumes a lot of CPU time. Although SmarCo begins with very poor performance, it scales well with the number of threads goes up. When the number of threads exceeds 64, the performance of SmarCo is better than Xeon E7-8890V4 and continues to rise.

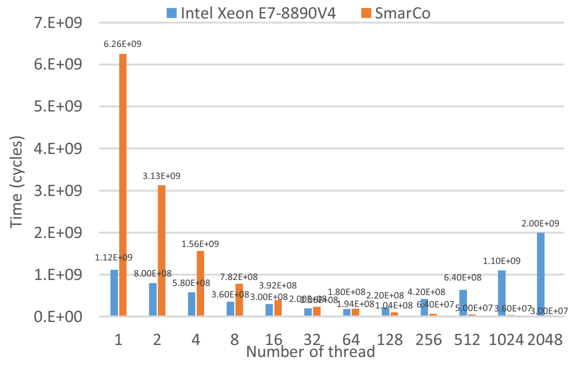


Fig. 23: Scalability evaluation of SmarCo and Intel Xeon E7-8890V4

4.3 FPGA Verification

To improve the accuracy of our simulator, as well as analyze the efficiency of different topologies of large-scale many-core processors, we built a 256-core FPGA verification platform comprised of 64 Xilinx XC7VX690T chips. In this design, four cores were integrated into one chip, with each sub-ring containing 4 chips and 16 cores in total. This FPGA verification platform allowed us to verify different topologies by changing interconnection among chips, as well as the interconnection of the cores in each chip.

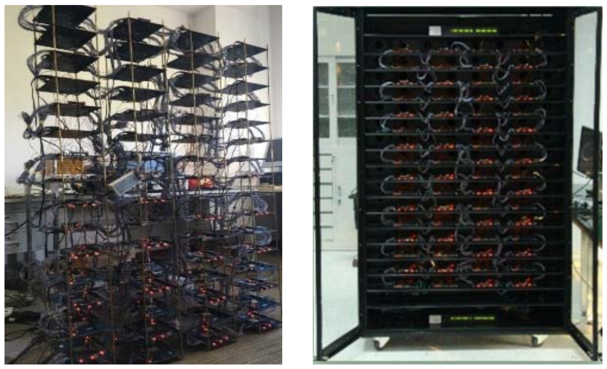


Fig. 24: 256-core FPGA verification platform

4.4 Prototype Verification

To prove the efficiency and feasibility, we taped out a SmarCo with TSMC 40-nm technology node. The SmarCo can support 256 threads at most. We also implemented two different acceleration cards, one integrated one SmarCo processor, the other integrated two SmarCo processors. The acceleration cards work with host CPU using PCIe interface, as shown in **Fig. 25**.

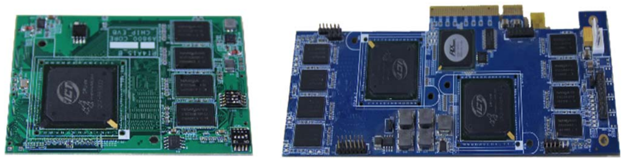


Fig. 25: Two acceleration cards with SmarCo processors, the left integrates one SmarCo processor, the right integrates two SmarCo processors

We evaluated the performance and energy-efficiency of our taped out SmarCo, the results are shown in **Fig. 26**. Our SmarCo achieves a 2.05X to 6.84X improvement for energy-efficiency, with 3.85X average speedup.

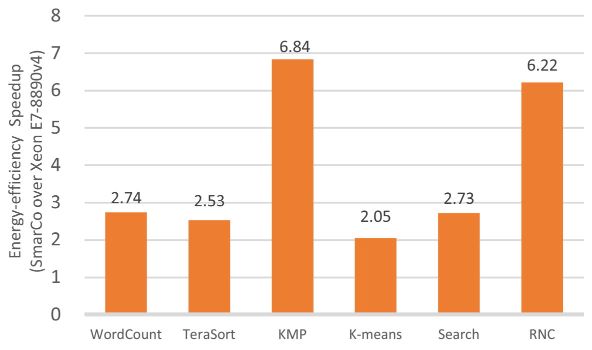


Fig. 26: Comparison of energy-efficiency (SmarCo over Intel Xeon E7-8890V4)

5. RELATED WORK

As web services bring an ever-increasing burden to datacenters, a vast number of studies have advocated various design methods for increasing the efficiency of datacenter processors, mainly based on characteristics of applications.

Applying large-scale many-core architecture represents anefficient solution for HTC applications in datacenters, which has already been proposed and extended by Hardavellas[26], Lotfi-Kamran[3][27]. In this paper, we followed the same conception, e.g. we integrated simpler cores to provide higher concurrent processing ability. In addition, we observed and concluded some typical characteristics of HTC applications, namely high throughput processing, hard real-time response and high-bandwidth memory access. Based on these characteristics, we designed and optimized a more suitable many-core architecture for HTC applications.

To satisfy high concurrency and data throughput in datacenters, more and more threads are implemented in a single core, which has advocated and shows benefit in previous studies[1][9][28]. In our SmarCo design, we implemented TCG cores to improve the utilization of core resources and offer processing capability of high concurrency. We also proposed in-pair thread mechanism, which is more suitable for threads with similar behaviors under HTC applications compared to previous studies[10][11].

Many mature commercial processors[13][29] and previous studies[30][31][32] about processors have applied ring NoC to arrange cores on the chip. Compared to other NoCs, ring is characterized by simple control logic, routing algorithm, and more predictable transmission latency, which makes it more suitable for HTC applications. In addition, we extended ring to hierarchical ring to integrate more cores, and we also added direct memory access datapath in our SmarCo design to speedup the accesses to memory, as well as reducing the risk of congestion in NoC.

Memory access datapath plays critical roles in processors performance, especially for memory-intensive or delay sensitive applications. In studies [33], [34] and [35], the physical link was split to achieve higher utilization of datapath. In comparison, we introduce an optimized split physical link mechanism, called high-density NoC, by applying a self-governing mechanism for each small granularity physical channels with greedy allocation algorithm to take full advantage of physical links. In addition, other studies focused on improving the efficiency of discrete and small granularity memory access requests. The study [36] exploited and merged the same destination address among adjacent memory access requests in GPU to reduce the amount of memory access requests, as well as reduce the memory access collision. The study [37] proposed memory access scheduling algorithm to protect its locality that may be destroyed by NoC transmission policy to improve the efficiency of memory accesses.

Other previous studies have discussed different aspects of processor architecture design for datacenters. For instance, Rhythm framework[1] exploited and merged same execution control paths across requests to improve server performance. The study [38] explored heterogeneous architectures to improve both welfare and energy efficiency by combining server-class processors and mobile-class processors together.

Conservation Cores[39] used application-specific hardware circuits to reduce energy consumption. ASIC Clouds[40] studied ASIC accelerator based ASIC Cloud datacenter to optimize the total cost of ownership (TCO). All the related work have important enlightenment and reference significance for the future design of high-throughput processors.

6. DISCUSSION

In this paper, we design and implement the SmarCo, a large-scale many-core architecture, by introducing innovative and efficient structures based on the deep analysis of HTC applications. We have some important experiences and findings to be discussed, which may be meaningful for future work.

Large-scale parallel execution: Highly concurrent execution is one of the key characteristics of HTC applications. Improving the utilization of pipeline resources and overlap memory access latency benefit processors for HTC applications. Conventional threads scheduling strategies, which makes use of the different behaviors of threads, are not very efficient for multi-threads that behaves similarly during the execution.

Simple and Latency-predictable NoC: Complex control logic of routers and complex routing algorithms may lead to higher latency and the unpredictability of memory access latency, which is unsuitable for HTC applications. Simpler control logic and simpler routing algorithms can better meet the requirements of high-concurrency processing, hard real-time response and high memory bandwidth of HTC applications.

Memory Access Datapath Optimization: As depicted in **Fig. 8**, HTC applications are characterized by larger portion of small granularity memory accesses, which is different from conventional HPC applications. In addition, large-scale parallel execution generates tremendous discrete memory accesses shuttling in NoC. Therefore, memory access datapath should be made changes to be better adapted to such pattern to build high-efficient processors.

7. CONCLUSION

With the explosion of HTC applications, datacenters have to expand their scales to provide satisfactory QoS. However, the new characteristics of HTC applications present considerable challenges to conventional processors of datacenters. Therefore, it is of critical importance to building high-efficiency processors based on the characteristics of HTC applications.

In this paper, we present SmarCo, a feasible high efficient processor architecture for HTC applications. In our SmarCo processor, we built hierarchical ring topology to arrange large-scale TCG cores. In each TCG core, we applied in-pair thread mechanism to overlap memory access latency, as well as to improve the ability of high-concurrency processing. Based on the memory access pattern of HTC applications, we proposed a series of innovative datapath structures, including MACT, high-density NoC, and direct memory access datapath, to improve memory access efficiency. Furthermore, we developed laxity-aware task scheduler based on hardware to guarantee the hard real-time response. We implemented and evaluated a practical design method of high-efficiency processors for HTC applications. Compared to Intel high-performance general-purpose processor, Xeon E7-8890v4, SmarCo achieves 10.11X performance improvement and 6.95X energy-efficiency improvement. Based on the work we have done, we hope that the computer-architecture researchers will benefit from this work, and we also thank researchers of those previous studies, which enlighten and help us a lot.

Great efforts are needed to further improve the efficiency of our SmarCo. In the future, we will concentrate on data penetration and prefetch from memory to SPM to further improve efficiency and fairness of memory accesses. In addition, we are working hard to apply in-memory computing techniques to handle those simple and fixed computing patterns, such as string matching, to further reduce data volume that needs to be transferred between memory and cores.

8. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful comments and suggestions. This work was supported by the National Key Research and Development Program (2017YFC0803401), the National Natural Science Foundation of China (61332009, 61521092), the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (2016A04), and the Innovation Project Program of the State Key Laboratory of Computer Architecture (CARCH3303, CARCH3407, CARCH3502, CARCH3505).

Reference

[1] Sandeep R. Agrawal, Valentin Pistol, Jun Pang, John Tran, David Tarjan, and Alvin R. Lebeck. Rhythm: harnessing data parallel hardware for server workloads. In Proceedings on Architectural support for programming languages and operating systems (ASPLOS), 2014, 19-34.

[2] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[3] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi. Scale-out processors. In Proceedings of the International Symposium on Computer Architecture (ISCA), 2012.

[4] Cansu Kaynak, Boris Grot, and Babak Falsafi. Confluence: unified instruction supply for scale-out servers. In Proceedings of the 48th International Symposium on Microarchitecture (MICRO). ACM, New York, NY, USA, 166-177. 2015.

[5] C. Kozyrakis, A. Kansal, S. Sankar and K. Vaid. Server Engineering Insights for Large-Scale Online Services, in IEEE Micro, vol. 30, no. 4, pp. 8-19, July-Aug. 2010.

[6] Content delivery network. https://en.wikipedia.org/wiki/Content_delivery_network. 2017-09-10.

[7] Nginx, http://nginx.org/en/. 2017-09-10.

[8] ARM11 processors. http://infocenter.arm.com/help/index.jsp?lang=en.

[9] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, David Brooks. Profiling a warehouse-scale computer. 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, 2015, pp. 158-169.

[10] F. J. Cazorla, A. Ramirez, M. Valero, and E. Fernandez. Dynamically Controlled Resource Allocation in SMT Processors. Proceedings of International Symposium on Microarchitecture (MICRO). pp.171–182. December 2004.

[11] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In Proceedings of the 1996 International Symposium on Computer Architecture (ISCA), Philadelphia, May 1996.

[12] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, John Zook. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, 2008, pp. 88-598.

[13] Parallel Programming and Optimization with Intel Xeon Phi Coprocessors. A. Vladimirov and V. Karpusenko, 2013.

[14] CORTEX-A SERIES. http://www.arm.com/products/processors/cor tex -a. 2017-9-11.

[15] J. Kim and H. Kim. Router microarchitecture and scalability of ring topology in on-chip networks. International Workshop on Network on Chip Architectures (NoCArc), New York, NY, 2009, pp. 5-10.

[16] R. Banakar, S. Steinke, Bo-Sik Lee, M. Balakrishnan and P. Marwedel. Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES), Estes Park, CO, USA, 2002, pp. 73-78.

[17] S. Gu, E. H. M. Sha, Q. Zhuge, Y. Chen and J. Hu. A Time, Energy, and Area Efficient Domain Wall Memory-Based SPM for Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 35, no. 12, pp. 2008-2017, Dec. 2016.

[18] ustin Talbot, Richard M. Yoo, Christos Kozyrakis. Phoenix++: Modular MapReduce for Shared-Memory Systems. Proceedings of the Second International Workshop on MapReduce and its Applications (MAPREDUCE), New York: ACM, 2011:9-11

[19] An Open Source Search Engine Library. http://xapian.org/

[20] Radio Network Controller. https://en.wikipedia.org/wiki/Radio_Net work_Controller. 2017-09-10.

[21] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade and, M. Steinder, and I. Whalley. Performance-driven task co-scheduling for mapreduce environments. In Network Operations and Management Symposium (NOMS), IEEE, 2010, pp. 373 –380.

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. The 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), New York, NY, 2009, pp. 469-480.

[23] N. Muralimanohar, R. B alasubramonian, and N. P. Jouppi, " CACTI 6.0: A Tool to Model Large Caches," HP Laboratories, Tech. Rep. HPL-2009-85, 2009.

[24] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A powerarea simulator for interconnection networks," IEEE Trans. Very Large Scale Integr. Syst., vol. 20, no. 1, pp. 191–196, Jan. 2012.

[25] Xeon E7-8890V4. https://www.intel.cn/content/www/cn/zh/products/ processors /xeon/e7-processors/e7-8890-v4.html. 2017-9-20.

[26] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. IEEE Micro, 31(4):6–15, Jul-Aug 2011

[27] P. Lotfi-Kamran, B. Grot and B. Falsafi. NOC-Out: Microarchitecting a Scale-Out Processor. 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Vancouver, BC, 2012, pp. 177-187.

[28] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju G. Mancheril, Anastassia Ailamaki, Babak Falsafi. Database Servers on Chip Multiprocessors: Limitations and Opportunities. Proceedings of Biennial Conference on Innovative Data Systems Research (CIDR), At Asilomar, CA. pp. 79–87, 2007.

[29] H. P. Hofste. Power efficient processor architecture and the cell processor. 11th International Symposium on High-Performance Computer Architecture (HPCA), 2005, pp. 258-262.

[30] Ayan Mandal, Sunil P. Khatri, Rabi N. Mahapatra. Architectural simulations of a fast source-synchronous ring-based Network-on-Chip design. Computer Design (ICCD) 2012 IEEE 30th International Conference on, pp. 482-483, 2012, ISSN 1063-6404.

[31] S. Bourduas and Z. Zilic. A Hybrid Ring/Mesh Interconnect for Network-on-Chip Using Hierarchical Rings for Global Routing. First International Symposium on Networks-on-Chip (NoCS), Princeton, NJ, 2007, pp. 195-204.

[32] A. Mandal, S. P. Khatri and R. N. Mahapatra. A fast, source-synchronous ring-based network-on-chip design. 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2012, pp. 1489-1494.

[33] Lei Wang, Poornachandran Kumar, Ki Hwan Yum, Eun Jung Kim. APCR: an adaptive physical channel regulator for on-chip interconnects. Proceedings of the 21st international conference on Parallel Architectures and Compilation Techniques (PACT). New York, USA. 2012: 87-96.

[34] M. H. Cho, Mieszko Lis, Keun Sup Shim, Michel Kinsy, Tina Wen, Srinivas Devadas. Oblivious Routing in On-Chip Bandwidth-Adaptive Networks. In Proc. of the 18th Intl Conf on Parallel Architecturess and Compilation Techniques (PACT), 2009: 181–190.

[35] R. Hesse, J. Nicholls, N. Jerger. Fine-grained bandwidth adaptivity in networks-on-chip using bidirectional channels. in Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS), May 2012: 132–141.

[36] Eunhyeok Park, Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, and Sunggu Lee. Memory fast-forward: a low cost special function unit to enhance energy efficiency in GPU for big data processing. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). San Jose, CA, USA, 1341-1346.

[37] Yuan George L, Ali Bakhoda, Tor M Aamodt. Complexity effective memory access scheduling for many-core accelerator architectures. Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). New York: IEEE/ACM, 2009: 12-16.

[38] M. Guevara, B. Lubin and B. C. Lee. Navigating heterogeneous processors with market mechanisms. 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), Shenzhen, 2013, pp. 95-106.

[39] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: reducing the energy of mature computations. In Proceedings on Architectural support for programming languages and operating systems (ASPLOS). 2010. ACM, New York, NY, USA, 205-218.

[40] I. Magaki, M. Khazraee, L. V. Gutierrez, M. B. Taylor. ASIC Clouds: Specializing the Datacenter. The 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 178-190.