GODSON-T: AN EFFICIENT MANY-CORE PROCESSOR EXPLORING THREAD-LEVEL PARALLELISM

GODSON-T IS A RESEARCH MANY-CORE PROCESSOR DESIGNED FOR PARALLEL SCIENTIFIC COMPUTING THAT DELIVERS EFFICIENT PERFORMANCE AND FLEXIBLE PROGRAMMABILITY SIMULTANEOUSLY. IT ALSO HAS MANY FEATURES TO ACHIEVE HIGH EFFICIENCY FOR ON-CHIP RESOURCE UTILIZATION, SUCH AS A REGION-BASED CACHE COHERENCE PROTOCOL, DATA TRANSFER AGENTS, AND HARDWARE-SUPPORTED SYNCHRONIZATION MECHANISMS. FINALLY, IT ALSO FEATURES A HIGHLY EFFICIENT RUNTIME SYSTEM, A PTHREADS-LIKE PROGRAMMING MODEL, AND VERSATILE PARALLEL LIBRARIES, WHICH MAKE THIS MANY-CORE DESIGN FLEXIBLY PROGRAMMABLE.

Dongrui Fan
Hao Zhang
Da Wang
Xiaochun Ye
Fenglong Song
Guojie Li
Ninghui Sun
Institute of Computing Technology, Chinese Academy of Sciences

Although various many-core processors, such as Tilera's TILE64,[1] IBM's Power7,[2] AMD's Opteron,[3] and the SPARC64,[4] provide tremendous computational capability, programmers still face the grand challenge of expressing and exploiting parallelism correctly and efficiently. Parallelization always requires significant programming efforts. Even when a parallel program works correctly, performance tuning can be daunting. Parallel programming brings many complex problems that are highly related to performance issues: managing conflicts in accessing shared resources, synchronizing disparate threads, and so on. In many cases, significant programming efforts can't be transformed into performance gain. This problem is unlikely to ever be perfectly resolved by software programming infrastructure alone.

Therefore, the Institute of Computing Technology (ICT) of the Chinese Academy of Sciences (CAS) has developed a many-core processor called Godson-T, which provides a widely used programming paradigm and highly efficient architectural support for multithreaded programs. Thus, it frees programmers from concentrating on efficient parallel execution and lets them focus on expressing parallelism. The Godson-T architecture supports two fundamental multithreading operations: data communication and thread synchronization. At the same time, it provides a multithread programming environment and a runtime system.

Godson-T architecture overview

As Figure 1 shows, the Godson-T processor is a many-core processor with 64 homogeneous, in-order, and dual-issue processing

Published by the IEEE Computer Society

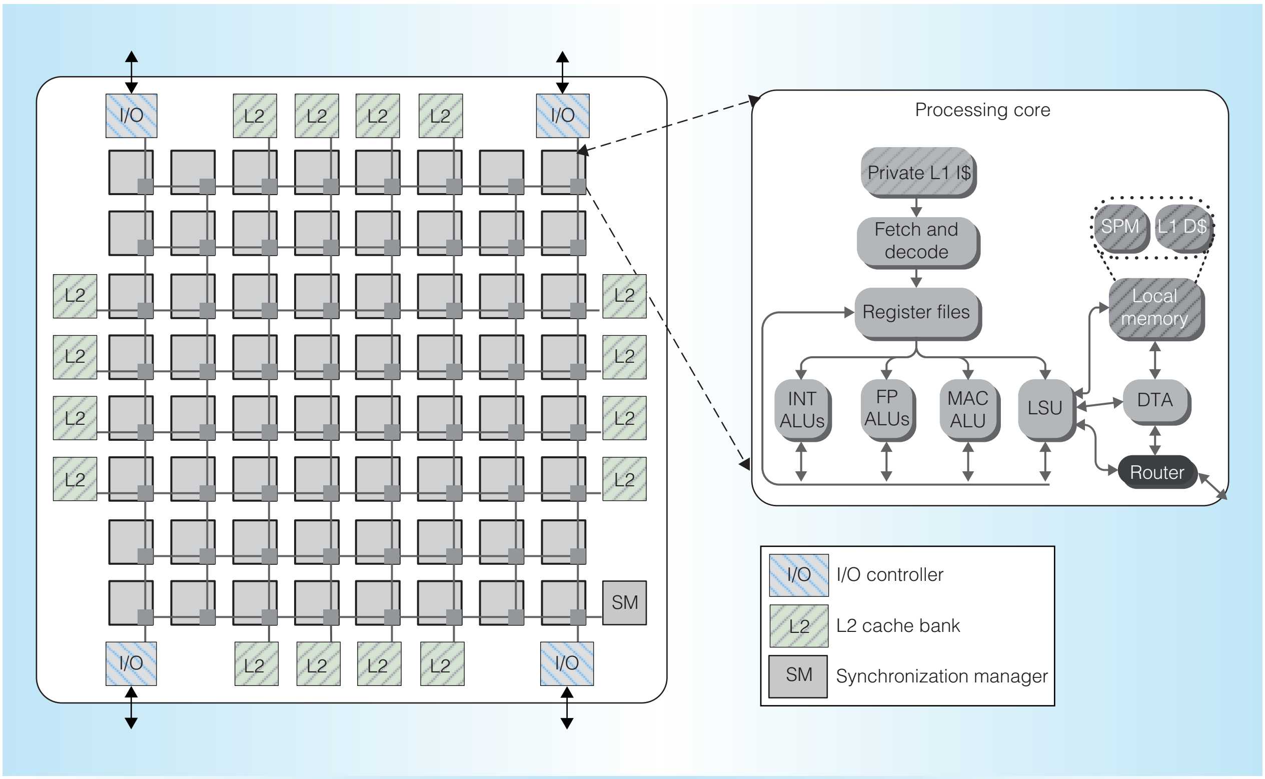

Figure 1. High-level block diagram of Godson-T. (ALU: arithmetic logic unit; D$: data cache; DTA: data transfer agent; I$: instruction cache; INT: integer; FP: floating point; LSU: load-store unit; L1: Level 1; L2: Level 2; MAC: multiply accumulate; SM: synchronization manager; SPM: scratch-pad memory.)

cores running at 1 GHz. The taped-out prototype chip is a set of 16 cores, with the same architecture and key techniques to verify the proposed solutions. The 64-core chip is now well underway. The following descriptions are based on the 64-core architecture.

Each eight-stage pipeline processing core supports the million instructions per second (MIPS) instruction set architecture and the synchronization instruction extensions. Two floating-point operations, including a multiply-accumulation, can be issued to fully pipelined function units in one cycle.

Godson-T's on-chip memory hierarchy has two levels. The first is the memory attached to each core, which contains a 16-Kbyte two-way set-associative private instruction cache and a 16-Kbyte data cache. The data cache can also be configured as scratch-pad memory (SPM) or a hybrid structure of SPM and a data cache.

The second level is the global L2 cache shared by all the cores. There are 16 address-interleaved L2 cache banks (128 Kbytes each) distributed along four sides of the chip. The L2 cache can serve up to 64 cache-accessing requests. Four L2 cache banks on the same side of the chip share a memory controller (see Figure 1). Each L2 cache bank is connected with a router.

This two-level cache uses a region-based cache coherence (RCC) protocol. Each core has a data transfer agent (DTA) for fast data communication. A dedicated on-chip synchronization manager (SM) provides architectural support for efficient mutual exclusion, barrier, and signal/wait synchronization.

A 128-bit-width packet-switching 2D mesh network connects all on-chip units. The mesh network employs deterministic X-Y routing and a wormhole switching policy and provides 4 terabytes per second (Tbytes/s) of on-chip bisection bandwidth among 8 × 8 processing cores working at 1 GHz.

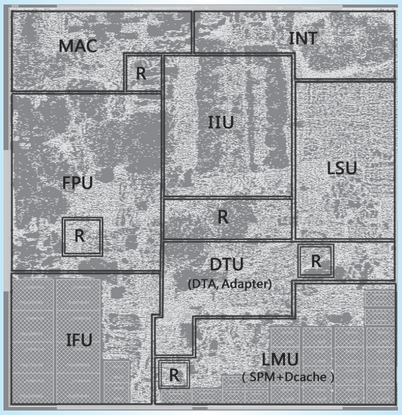

Figure 2. Godson-T processing-core floorplan with nine main regions: instruction-fetch unit (IFU), floating-point unit (FPU), multiply-accumulate unit (MAC), integer unit (INT), load-store unit (LSU), instruction issue unit (IIU), data transfer unit (DTU), local memory unit (LMU), and router (R). The DTU includes a data transfer agent (DTA) and an adapter. The IFU contains a 16-Kbyte instruction cache. The LMU is composed of a 16-Kbyte data cache and control logic.

Microarchitecture features

Figure 2 shows a Godson-T processing core's floorplan. It includes nine main functional regions: the instruction-fetch unit, floating-point unit, multiply-accumulate unit, integer unit, load-store unit, instruction issue unit, data transfer unit, local memory unit, and router.

Pipeline

In a given cycle, each Godson-T core will fetch two instructions from the instruction cache, and then decode and issue them. There are nine execution units within each core: one double-precision floating-point, multiple-add unit; three floating-point units; two branch execution units; two integer arithmetic logic units; and one load-store unit. The double-precision, floating-point multiple-add unit can also execute two single-precision floating-point operations simultaneously.

The Godson-T core has a combined 64×64 bit register file that's easy to reuse as fixed-point or floating-point registers. This improves the available number of fixed-point registers, as well as floating-point registers, which is more convenient for program optimization. The load-store unit also provides the ability of 128-bit dual-double-precision floating-point load and store in a single cycle.

Cache hierarchy

To ease large-scale parallelism, we prefer an RCC protocol to traditional ones.[5] The principle of RCC is that the coherence of data should be lazily guaranteed just upon request. The shared access and private access are expected to be distinguished. The cache hierarchy benefits from differentiating types of accesses and taking appropriate consistency operations to maximize the benefits. In our terminology, a region is a code sequence marked by an open-region primitive and a closed-region primitive. Memory accesses inside the region are regarded as shared accesses and guaranteed coherence, whereas accesses outside the region aren't guaranteed coherence. Ideally, regions embrace only the shared accesses. The RCC's flexibility lets users declare either coarse- or fine-grained regions, and makes it possible to ensure the program's correctness at first, and then make incremental performance improvements by reducing the regions' sizes. It's also convenient to correctly port a parallel program to Godson-T by just inserting these primitives at synchronization points, such as mutually exclusive locks.

In Godson-T, the L1 data cache is configured as a hybrid structure containing a cache and an explicitly controlled and globally addressed SPM to speed up data access. Setting a border register defines the SPM and data cache capacity. Each core can configure its SPM separately. The local SPM's latency is only one cycle, and the remote SPMs' latencies depend on the distance and network congestion. The SPM is globally addressed for all cores. Each core's SPM maps to an interleaving address space of more than 3 Gbytes; thus, the SPM won't clash with data or instruction memory. Processing cores can access any SPM through its mapped address, whether it is local or remote.

L2 cache banks are placed symmetrically around the edge of the core array. Each L2 bank supports four requests and two DTA operations. To meet the need of lock-free programming, this L2 cache supports two extra atomic operations included in the four requests: fetch and add (FAA) and test and set (TAS).

Scratch-pad memory

One of SPM's most important features is that it provides eight full/empty bits for each 32-byte cache line to synchronize fine-grained threads. The Godson-T design extends synchronization instructions to support full/empty bit operations.

When an on-chip private memory is configured as SPM, each cache line's 8-bit write mask works as a set of full/empty bits (see Figure 3). Once the processing core issues a memory access operation, its memory address is used to choose a target from local and remote SPMs.

Data coherence should be maintained by software via new instructions, such as sync_load, sync_store, load_future, and store_future. The sync_load instruction waits until the full/empty bit is set to full in order to read, and then sets it to empty. Conversely, sync_store waits until the full/empty bit is set to empty in order to write, and then sets it to full. The load_future instruction waits until the full/empty bit is set to full in order to read, and then leaves this bit untouched. Similarly, store_future waits until the full/empty bit is set to full in order to write, and then leaves this bit untouched. These two instructions can be used to handle the situation for multiple producers and multiple consumers, together with sync_load and sync_store.

Whether a sync_store instruction stores a new value depends on the relevant full/empty bit's value. When its full/empty bit's value is 0, it stores this new value and sets the full/empty bit to 1. Otherwise, if the full/empty bit's value is 1, the current sync_store should wait for a sync_load instruction to set it to 0 by loading the data, and then the sync_store can be executed. In an analogy to a sync_store instruction, a sync_load instruction can load the requested data when the full/empty bit is 1, and then sets it to 0. If the requested data's full/empty bit is 0, the current sync_load instruction should be stalled in the pipeline until a sync_store sets it to 1 by storing the data. Once the full/empty bit is set to 1, the load_future and store_future instructions can load or store to the corresponding blocks directly and leave their respective full/empty bits to 1. Otherwise, they should be stalled in the pipeline until a sync_store instruction sets the full/empty bit to 1. Figure 4 shows a state machine of full/empty bits for explanation.

Efficient fine-grained synchronization is implemented by a tagged memory or register, in which data can be carried to transfer synchronization information. Compared to the traditional full/empty bit mechanism, the proposed synchronization implements a counter; therefore, various synchronization schemes can be implemented on the basis

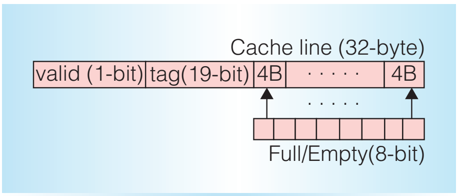

Figure 3. SPM full/empty bit configuration. When an on-chip private memory is configured as SPM, each cache line's 8-bit write mask works as a set of full/empty bits.

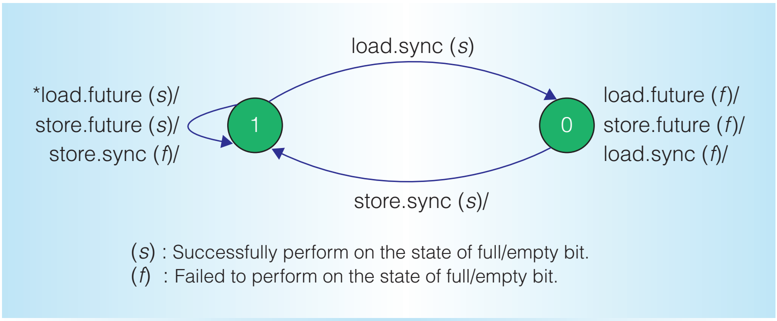

Figure 4. State machine of full/empty bits. Via value transformation of full/empty bits, a synchronization of multiple producers and multiple consumers can be implemented by a sync_store instruction with some store_future instructions and a sync_load instruction with some load_future instructions.

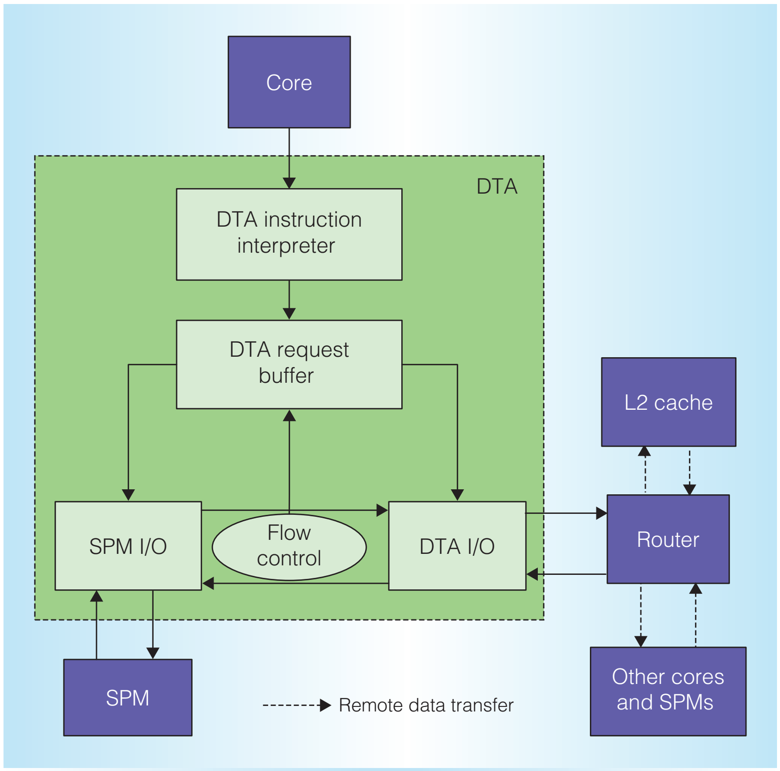

Figure 5. Block diagram of the DTA, which includes five modules: the DTA instruction interpreter, DTA request buffer, SPM I/O, flow controller, and DTA I/O.

of it, such as in a multiple-producer, single-consumer situation. We use two microbenchmarks to evaluate the efficiency of Godson-T's full/empty-bit mechanism: the 2D Wavefront and Livermore Loop 6. The result of fine-grained Livermore Loop 6 gains 70 percent speedup over the coarse-grained synchronization version.[5]

Orchestrating data movement

As an execution core coprocessor, the DTA is the main component of our data management framework. Through the DTA, we can transfer data blocks with various strides among SPMs and the L2 caches. Figure 5 shows the overview of the DTA block with five modules. The DTA instruction interpreter receives the instructions issued from the core and translates DTA instructions to operating actions. The request buffer has three entries, and each can keep one DTA request until completion. Therefore, a DTA can deal with three requests concurrently. The DTA request buffer contains the entire network information.

The DTA can assemble the flits to be transferred to local SPM, remote L2 cache banks, and remote SPMs. If the request is to put local data to a remote L2 cache bank or an SPM, the DTA request buffer sends assembled operation instructions to the SPM I/O and then sends data to the DTA I/O module. If the request is to fetch remote data to the local SPM, the DTA request buffer first sends data requests to the DTA I/O. Then, it puts the return data into the local SPM. A flow controller limits and balances the traffic load of the network-on-chip (NoC) traffic load. A flow controller records the number of issued and received flits. When the difference is above some threshold, the controller sends a stall signal to the request buffer to stop successive requests.

Figure 6 shows the data block pattern that the DTA supports. The shadowed area indicates the packets that must be moved. The block is constructed by packets that are smaller than 16 bytes. Figure 6a indicates the pattern in which blocks and packets are both continuous; Figure 6b indicates the pattern in which packets are continuous but blocks are noncontinuous; and Figure 6c indicates the pattern in which both packets and blocks are noncontinuous.

Godson-T provides DTA-related APIs that are constructed with reduced-instruction-set computing (RISC)-like DTA instructions to implement explicit data movement, which is critical to improving on-chip memory utilization and on-chip network bandwidth efficiency. Our evaluation results show that the performance can be dramatically accelerated by 2 to 3 times when using on-chip DTA operations.

Synchronization manager

There is a hardware SM on chip, which can handle both mutual-exclusion and barrier synchronization. The SM implements fast hardware synchronization to replace a pure software synchronization method. The evaluation results show that the overhead of mutual-exclusion and barrier synchronization on Godson-T is far lower than software-based methods, ranging from tens to hundreds of times lower.

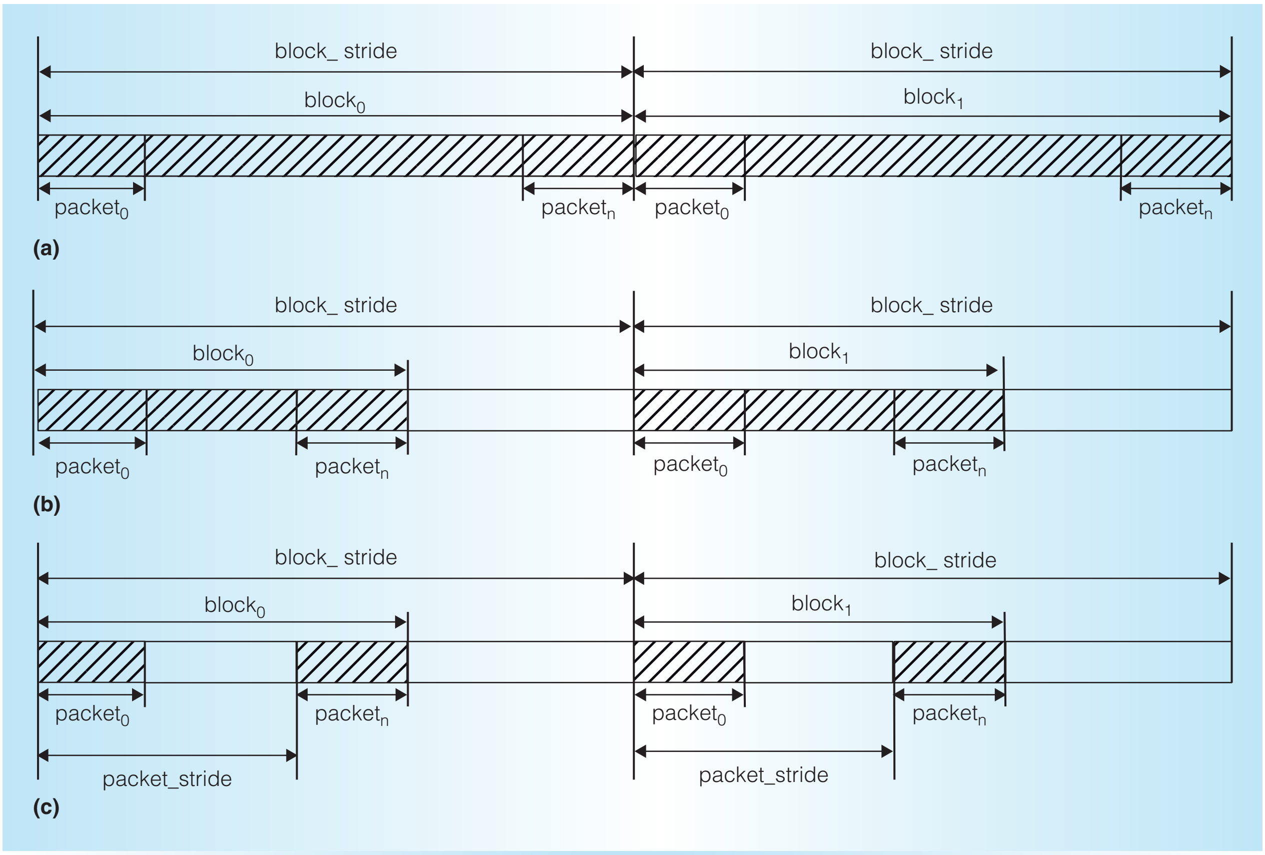

Figure 6. Bulk data patterns supported by the DTA. The figure indicates three patterns: one in which packets and blocks are both continuous (a), one in which packets are continuous but blocks are noncontinuous (b), and one in which packets and blocks are both noncontinuous (c).

As Figure 7 shows, the SM is a 128-entry and four-way set-associative table in which each entry records a 57-bit synchronization request. *Core_ID* represents the core, which sends the synchronization requests. *State* represents the state of the synchronization, such as lock_acquired or lock_waiting. *Sync_ID* is a unique identifier for a synchronization operation. *Bar_Count* is used only for barrier synchronization and records the number of threads that haven't arrived at the barrier, so this field differs in each linked entry. *Next* is a pointer to the next entry, which represents another core waiting for the same lock or barrier.

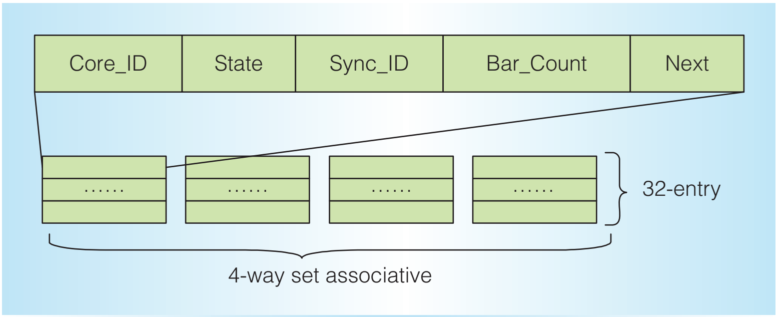

Figure 7. The SM structure. A centric on-chip synchronization manager helps to implement faster synchronization without accessing off-chip shared memory.

The mutual-exclusion (lock) requests are organized in the form of queues in the SM. Entries belonging to the same queue are linked with Next pointers. When a lock request is sent to the SM through the on-chip network, the SM checks the state of entries with the same Sync_ID to decide whether the request will be satisfied or should be put in the waiting queue. Queuing avoids sending too many messages on the network, which is a major disadvantage of traditional test and set lock.

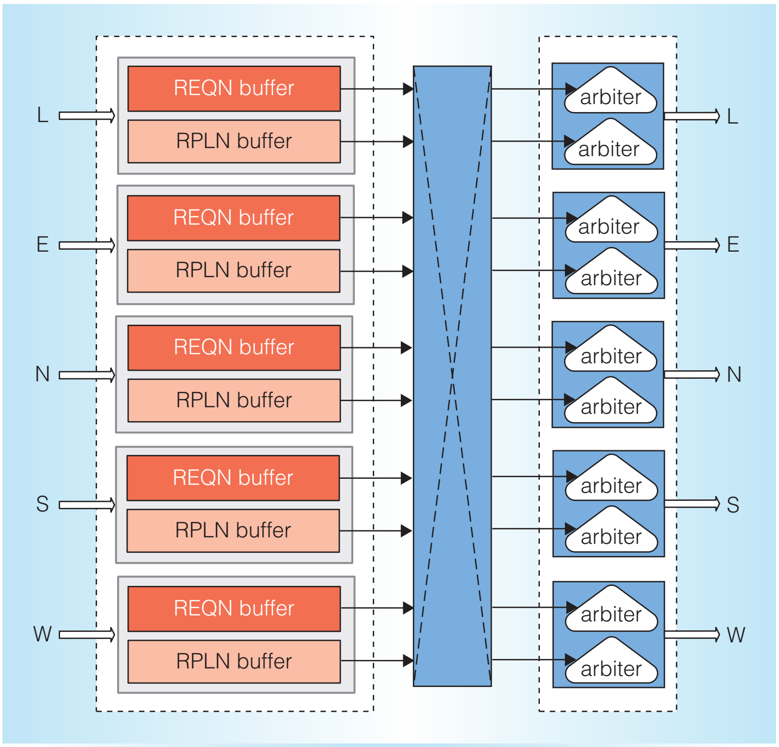

Figure 8. Overview of a wormhole router used in two independent on-chip networks. Each router connects in five directions: east, west, north, south, and to the local core. Each link contains two 128-bit unidirectional links. (REQN: request network; RPLN: reply network; L: local core; E: east; N: north; S: south; W: west; X: crossbar.)

The barrier synchronization can be implemented similarly: Barrier requests with the same Sync_ID are organized as a queue. When all the barrier requests have arrived at the SM (Bar_Count changes to 0), the ACK messages are sent to all threads participating in the barrier synchronization.

To reduce the search overhead, all 128 entries are organized into a four-way set-associative table. Both Core_ID and Sync_ID are used to index the entry number. A software handler is triggered if the SM table overflows.

Network on chip

Godson-T's on-chip interconnect network is called gMesh, which is a departure from the traditional bus-based on-chip interconnect structure shown in Figure 1. Instead of using buses or rings, gMesh connects the processing cores using two 2D mesh networks, which provide the transport channels for cache access, coherence information, I/O access, and other communication activity. Because of the deadlock avoidance strategy, two networks are designed: the request network (REQN) and the reply network (RPLN). In Godson-T, on-chip messages are orchestrated into two classes: request messages and reply messages. For example, cache-miss messages travel on REQN, whereas cache-refill messages travel on RPLN. Such classification isolates memory request and reply messages into different networks, so they won't interfere with each other at the core's memory port or the L2 caches.

For each processing core connected to the NoC, there's a dedicated router, as shown in Figure 8. In the router, five ports are connected by a fully connected crossbar, which allows all-to-all five-way communication. Each router port contains a two-entry input buffer for each link. Having the two mesh networks leverages the on-chip wiring resources to provide massive on-chip communication bandwidth. When Godson-T works at 1 GHz, the gMesh networks can afford a 4-Tbyte/s bisection bandwidth for an 8 × 8 mesh network. By using mesh networks, the architecture can support any number of cores with few modifications on the fabric. In fact, the router resource can remain unchanged, even for a fairly large scale. Growing the mesh size provides more connecting bandwidth. The design provides the essential scalability for the Godson-T many-core architecture.

Software environments

Given a many-core processor like Godson-T, one challenge is efficiently utilizing the large on-chip computing capability. Programming on Godson-T is based on multithreading. Providing a simple and reasonable programming interface is undoubtedly critical to programmability.

The Godson-T programming environment adopts conventional C programming and the necessary tool chain. We also provide a Pthreads-like C library for task management. Therefore, a large amount of parallel program sources written with Pthreads can be conveniently ported onto Godson-T. The library provides a rich set of APIs for task management, including batch thread management, which can significantly reduce tasking overhead by grouping identical operations in batches. The programmer is responsible for creating, terminating, and synchronizing tasks by using appropriate Pthreads-like APIs.

GodRunner

We developed a software runtime system named GodRunner to provide efficient abstraction of a large number of hardware thread units and dynamic load balancing. The GodRunner task model (in our terminology, a task resides in software, whereas a thread resides in hardware) adopts a create-join method inspired by Pthreads. GodRunner tasks don't support preemptive execution, because frequent context-switching incurs save-restore overhead and cache thrashing. Therefore, a task will keep executing on a thread unit until it terminates. Nonpreemptive execution can make task initialization simple and fast: for example, the stack is statically allocated to each thread unit, which avoids a time-consuming dynamic allocation. GodRunner lets programmers create more tasks than hardware thread units, and transparently maps them to hardware thread units at runtime. GodRunner swaps the completed task out and schedules a new one into a thread unit.

GodRunner can flexibly incorporate different task-scheduling algorithms. So far, we implemented two well-known task-scheduling algorithms for efficient dynamic load balancing: work-stealing[6] and conditional spawning.[7] Godson-T supports fine-grained threads very well. For example, GodRunner can create a new thread in just tens of cycles. Figure 9 illustrates a GodRunner task-scheduling example.

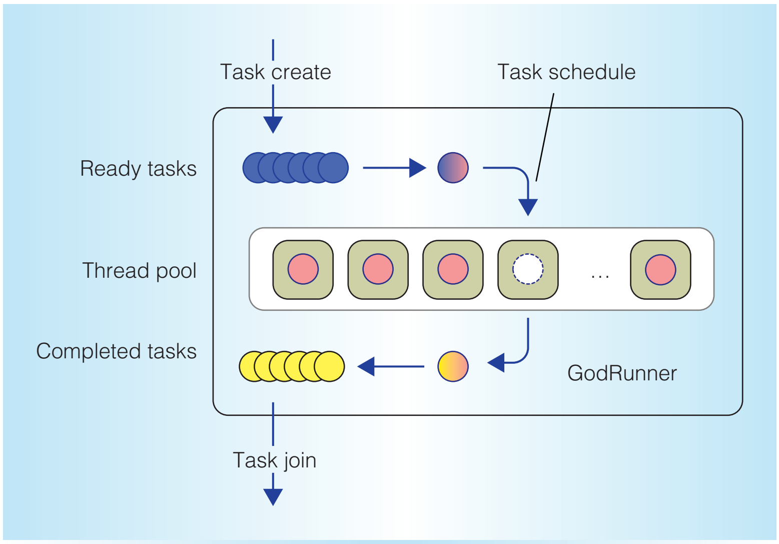

Figure 9. GodRunner lets programmers create more tasks than hardware thread units, and transparently maps them to hardware thread units at runtime. It also swaps the completed task out and schedules a new one in a thread unit.

Compiler optimization

Computer architects and system software designers face a unique opportunity to bring together new architectural features as well as corresponding compiler technology, so as to achieve a strong impact on system performance. Using the Godson-T platform, we performed some compiler technology studies. Using a pattern-making methodology, we encapsulated algorithm-specific optimizations into optimization patterns expressed in terms of preprocessor directives so that simple annotations could result in significant performance improvements. To validate this new methodology, we developed a framework named EPOD (Extendable Pattern-Oriented Optimization Directives) to map such directives to the underlying optimization schemes. Our experimental results show that a pattern-guided compiler can outperform the state-of-the-art compilers and even achieve performance as competitive as hand-tuned code. Thus, such a pattern-making methodology represents an encouraging direction for integrating domain experts' experience and knowledge into general-purpose compilers.[8]

Many-core fast simulation

The Godson-T Architecture Simulator (GAS) is a cycle-accurate and event-driven simulator. As a sequential simulator, it also suffers from slow execution speed. To speed up this many-core processor simulator, we introduced a parallel discrete event simulation (PDES) method. Our goal was to speed up the sequential simulator with multithreading, while maintaining its cycle-level accuracy. The basic idea is to divide the global event queue into separate queues for each logic process, while maintaining time

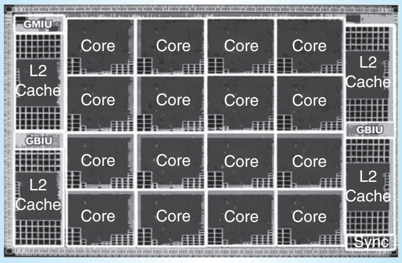

Figure 10. Floorplan of Godson-T 16-core prototype chip. Taped out with SMIC 130-nm, 1P8M CMOS process, working at 300 MHz, with an area of 230 mm^2. (GBIU: Godson-T bus interface unit; GMIU: Godson-T memory interface unit; Sync: synchronization manager.)

synchronization between these queues using PDES. Evaluation against the sequential version shows that the parallelized simulator achieves an average speedup of 10.9× (up to 13.6×) when running the Splash-2 kernel on four quad-core AMD Opteron 8347HE processors running at 1.9 GHz with 64 Gbytes of DRAM.

Physical implementation

Figure 10 shows the floorplan of the Godson-T 16-core prototype chip. It passes all standard tests and achieves the corresponding performance discussed in previous work.[5] Considering cost and performance tradeoffs, we reduced the 16 data cache banks to four in this 16-core prototype chip. The 64-core prototype chip will be manufactured in 65-nm CMOS technology, targeting an operating frequency of 1 GHz, and with an approximate area of 200 mm^2.

Godson-T will form the basis for ICT's future high-performance computing system. In the future, we will continue to investigate architectural innovations for high-performance and high-throughput computing. Once integrated processing cores increase to above 1,000, the scalability of on-chip interconnect networks and memory subsystems becomes an emergent and open problem. One of our future works will be finding a way to make many-core architecture more extendable. Another challenge is how to face the dark silicon problem. We will devote efforts to energy-efficient architecture design with the support of the state-of-art manufacturing technologies. MICRO

Acknowledgments

Godson-T represents the combined efforts of many diligent ICT researchers, engineers, and students across multiple teams in the State Key Laboratory of Computer Architecture of China. We thank professor Paolo Ienne for his Godson-T related research. This work is supported by the National Basic Research 973 Program of China (under grant 2011CB302501), the National Natural Science Foundation of China (under grants 60921002, 60925009, 61173007, 61100013, and 61100015), and the National Major Science & Technology Program of China (under grant 2011ZX01028-001-002).

References

1. S. Bell et al., ''TILE64 Processor: A 64-Core SoC with Mesh Interconnect,'' *Proc. IEEE Int'l Solid-State Circuits Conf.,* IEEE CS Press, 2008, doi:10.1109/ISSCC.2008.4523070.
2. R. Kalla et al., ''Power7: IBM's Next-Generation Server Processor,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 7-15.
3. P. Conway et al., ''Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 16-29.
4. T. Maruyama et al., ''Sparc64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 30-40.
5. D. Fan et al., ''Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions,'' *J. Computer Science and Technology,* vol. 24, no. 6, 2009, pp. 1061-1073.
6. R.D. Blumofe and C.E. Leiserson, ''Scheduling Multithreaded Computations by Work Stealing,'' *J. ACM,* vol. 46, no. 5, 1999, pp. 720-748.
7. P. Palatin, Y. Lhuillier, and O. Temam, ''CAPSULE: Hardware-Assisted Parallel Execution of Component-Based Programs,'' *Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* IEEE CS Press, 2006, pp. 247-258.
8. H. Cui et al., ''Extendable Pattern-Oriented Optimized Directives,'' *Proc. 9th Ann. IEEE/ACM Int'l Symp. Code Generation and Optimization,* IEEE CS Press, 2011, pp. 107-118.

Dongrui Fan is an associate professor in the State Key Laboratory of Computer Architecture of China at the Institute of Computing Technology, Chinese Academy of Sciences, and the chief architect of the Godson-T many-core processor. His research interests include computer microarchitecture design, low-power technologies, and VLSI design. Fan has a PhD in computer science from ICT.

Hao Zhang is an assistant professor in the State Key Laboratory of Computer Architecture of China at the Institute of Computing Technology, Chinese Academy of Sciences, and an architect of the Godson-T many-core processor. His research interests include high-throughput CPU microarchitecture. Zhang has a PhD in computer science from ICT.

Da Wang is an assistant professor in the State Key Laboratory of Computer Architecture of China at the Institute of Computing Technology, Chinese Academy of Sciences, and a member of the Godson-T research team. Her research interests include reconfigurable computing, reliability design, and VLSI design and test. Wang has a PhD in computer science from ICT.

Xiaochun Ye is an assistant professor in the State Key Laboratory of Computer Architecture of China at the Institute of Computing Technology, Chinese Academy of Sciences, and a member of the Godson-T research team. His research interests include multicore and many-core design, parallel computing, and simulation. Ye has a PhD in computer science from ICT.

Fenglong Song is an assistant professor in the State Key Laboratory of Computer Architecture of China at the Institute of Computing Technology, Chinese Academy of Sciences, and a member of the Godson-T research team. His research interests include high-performance and throughput microprocessors and memory subsystems design. Song has a PhD in computer science from ICT.

Guojie Li is the chief scientist of the Godson-T project at the Institute of Computing Technology, Chinese Academy of Sciences, and a fellow of the Chinese Academy of Engineering. His research interests include CPU microarchitecture, social computing, grid computing, and high-performance computer architecture. Li has a PhD in electrical engineering from Purdue University.

Ninghui Sun is a professor and the president of the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer architecture, operating systems, and parallel algorithms. Sun has a PhD in computer science from the Institute of Computing Technology.

Direct questions and comments about this article to Da Wang, State Key Laboratory of Computer Architecture of China, Institute of Computing Technology, Chinese Academy of Sciences, PO Box 2704, Beijing, China, 100190; wangda@ict.ac.cn.