GODSON-3: A SCALABLE MULTICORE RISC PROCESSOR WITH X86 EMULATION

THE GODSON-3 MICROPROCESSOR AIMS AT HIGH-THROUGHPUT SERVER APPLICATIONS, HIGH-PERFORMANCE SCIENTIFIC COMPUTING, AND HIGH-END EMBEDDED APPLICATIONS. IT OFFERS A SCALABLE NETWORK ON CHIP, HARDWARE SUPPORT FOR X86 EMULATION, AND A RECONFIGURABLE ARCHITECTURE. THE FOUR-CORE GODSON-3 CHIP IS FABRICATED WITH 65-NM CMOS TECHNOLOGY. EIGHT- AND 16-CORE GODSON-3 CHIPS ARE IN DEVELOPMENT.

Weiwu Hu
Jian Wang
Xiang Gao
Yunji Chen
Qi Liu
Guojie Li
Institute of Computing Technology, Chinese Academy of Sciences

Godson-3 is the third generation of the Godson microprocessor series, a project of the Institute of Computing Technology at the Chinese Academy of Sciences. As a multicore processor, Godson-3 targets high-throughput server applications, high-performance scientific computing, and high-end embedded applications.

Godson-3's scalable and distributed on-chip network connects processor cores and globally addressed level-two (L2) cache modules. A directory-based cache-coherence protocol maintains multiple level-one (L1) copies of the same L2 block. Godson-3's MIPS64-compatible superscalar reduced-instruction-set-computing (RISC) processor core is designed for high performance and low power dissipation. It also supports efficient x86 to MIPS binary translation through dedicated hardware support.

Godson-3 adopts the scalable mesh of crossbar (SMOC) on-chip network topology. Using the SMOC architecture, a 2×2 mesh network can support a 16-core processor, and a 4×4 mesh network can support a 64-core processor. We've already defined the four-, eight-, and 16-core product chips, and we've designed and fabricated four-core Godson-3 based on 65-nm CMOS technology. The eight- and 16-core Godson-3 chips are still in physical implementation.

CPU core features

The GS464 is a general-purpose processor core, upgraded from the Godson-2 microprocessor. The GS464's four-way superscalar execution mechanism has extremely high requirements for resolving interinstruction dependency and providing instructions and data. It therefore uses out-of-order execution and aggressive cache design to improve pipeline efficiency, as other modern microprocessors do.[1-6]

The GS464 out-of-order execution scheme combines register renaming, dynamic scheduling,[7] and branch prediction. GS464 has a 64-entry physical register file for fixed-point and floating-point register mapping. In GS464, the 16-entry fixed-point reservation

0272-1732/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

station and the 16-entry floating-point reservation station issue instructions out of order, whereas the 64-entry reorder queue commits out-of-order executed instructions in program order. GS464 also implements a 16-entry branch target buffer, an 8-Kbyte entry branch history table, a 9-bit global history register, and a 4-entry return address stack for branch prediction.

GS464 has two fixed-point functional units and two floating-point functional units. Both fixed-point units execute addition, subtraction, logical, shift, and comparison instructions. In addition, the first fixed-point arithmetic logic unit (ALU1) executes trap, conditional move, and branch instructions; the second (ALU2) executes multiplication and division instructions. The first floating-point unit (FALU1) can execute all floating-point instructions; the second floating-point unit (FALU2) can execute floating-point addition, subtraction, and multiplication instructions.

The GS464 memory system supports 64-bit virtual addresses and 48-bit physical addresses, and can access a 128-bit quad word in one cycle. GS464 has a 64-Kbyte L1 instruction cache and a 64-Kbyte L1 data cache; both are four-way set associative. GS464's fully associative translation lookaside buffer (TLB) has 64 entries, each of which maps an odd page and an even page. The 24-entry memory-access queue, which contains a content-addressable memory for dynamic memory disambiguation, supports out-of-order memory access, nonblocking cache, and load speculation. GS464's memory-access pipeline includes four cycles: address calculation, TLB and cache reading, tag comparison, and write back.

GS464 implements the Enhanced JTAG (EJTAG) standard for debugging and performance tuning. We implemented error-correcting code for the data cache and parity check for the instruction cache. We optimized the CPU core for low power dissipation in the architectural, logical, and physical design stages.

Figure 1 shows the architecture of the GS464 CPU core.

We designed several CPU chips with different process technologies based on the GS464 architecture. The Godson-2F[8]—which integrates the GS464 core, a 512-Kbyte L2 cache, a 333-MHz DDR2 controller, and a PCI/PCIX controller—is based on 90-nm CMOS technology. It achieves 1 GHz with both SPECint2000 and SPECfp2000 scores of more than 500. The chip includes 51 million transistors and has a die size of 42 square millimeters. It consumes from 3 to 5 watts depending on the application.

Hardware support for x86 to MIPS binary translation

To support x86 emulation, Godson-3 provides hardware support for binary translation from x86 to MIPS in its GS464 core. Although the Crusoe processor supports translation from x86 to VLIW,[9] no commercial RISC processor provides dedicated support for x86 emulation because of the difference between x86 and RISC.[1-5] Because some x86-related features aren't present in MIPS (EFlags, the floating-point register stack, segment addressing mode, and so on), software-based translation from x86 binary to MIPS binary is inefficient.[10,11] In many cases, translating an x86 instruction requires tens of MIPS instructions because of the difference between the x86 and MIPS ISAs. Godson-3's x86 binary translator smoothes translation from x86 binary to MIPS binary with minimal hardware support. To achieve this, GS464 defines new instructions and runtime environments through the MIPS64 user-defined interface (UDI) to bridge the gap between the x86 and MIPS64 ISAs. It defines and implements new instructions in MIPS format for functions that are in x86 ISA but not in MIPS64 ISA.

The Godson-3 virtual machine will be compatible with x86 at both the ISA level and Linux application binary interface (ABI) level. We'll build the system- and process-level virtual machines accordingly. The ISA-level compatibility is for low-cost PC applications, where Microsoft Windows is most popular; the Linux ABI-level compatibility is for server applications, where Linux servers with x86 processors are most popular. Figure 2 shows the Godson-3 binary translation system's software architecture. The process- and system-level virtual machine monitors (VMMs) are implemented on Linux, which is improved to provide x86-compatible system calls.

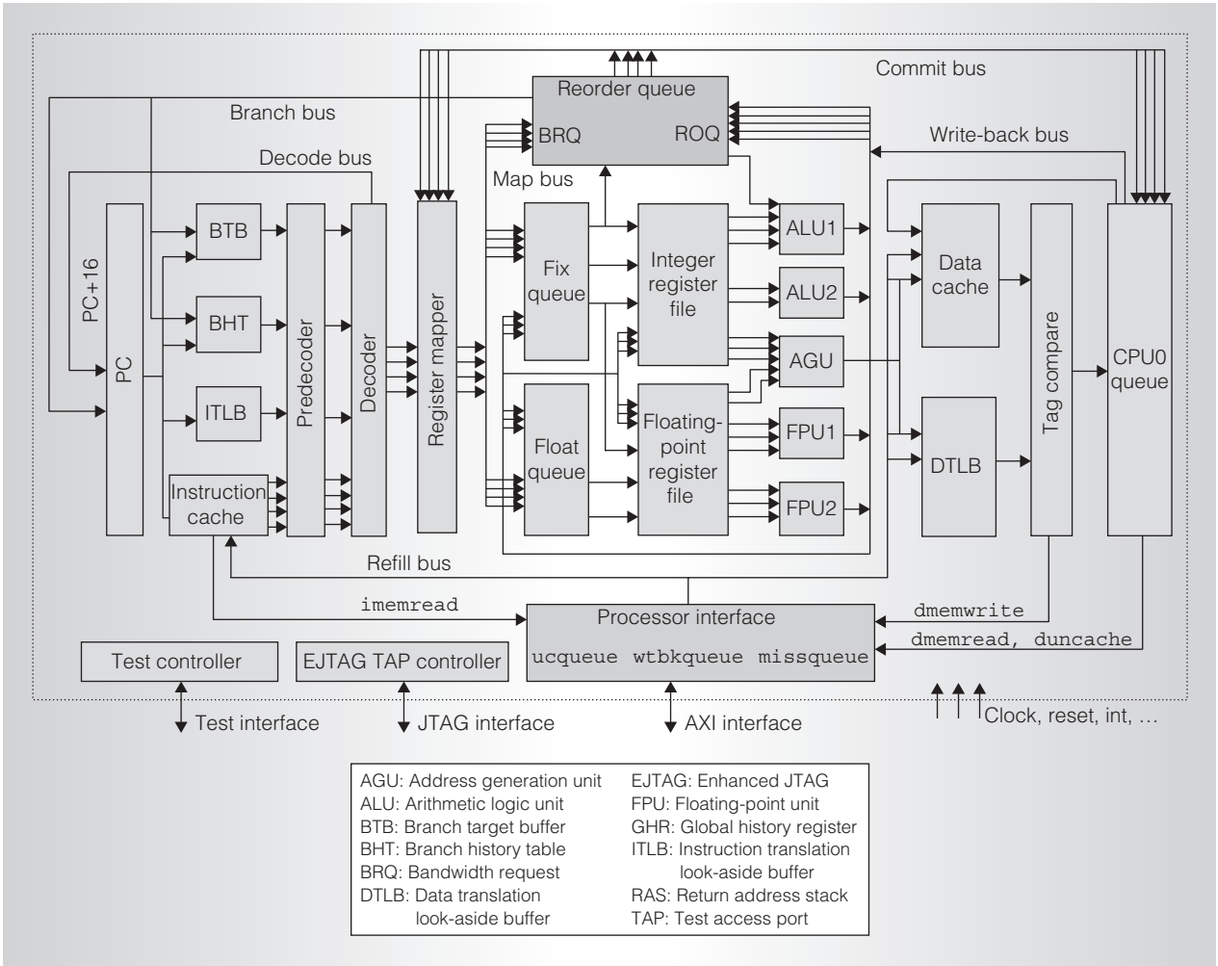

Figure 1. GS464 microarchitecture. GS464 adopts a nine-stage dynamical pipeline.

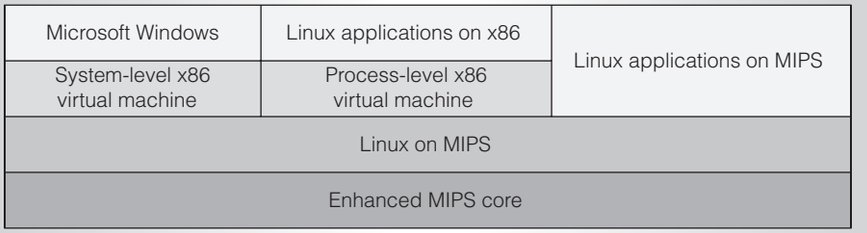

Figure 2. The GS464 virtual machine's software architecture. The x86 operating systems and applications are built on MIPS Linux system through virtual machine monitor.

Hardware support for EFlag of x86

A major difference between the x86 and MIPS ISAs is that the x86 ISA uses EFlags. Most x86 fixed-point arithmetic instructions generate 6-bit EFlags as by-products of the arithmetic calculation, and the branch directions of branch instructions are determined according to the EFlag values. MIPS fixed-point arithmetic instructions don't generate EFlags, and MIPS branch instructions decide

Number of instructions		Instruction				Comment
0		SUB	ECX	EDX		
1		JE	X86_target			

(a)

0.00		SUBU	Result	Recx	Redx	
0.01		SRL	Rsf	Result	31	/*SF=Result[31]*/
0.02		BEQ	Result	R0	L1	
0.03		ADD	Rzf	R0	R0	/*ZF=0*/
0.04		B	L2			
0.05		NOP				
0.06	L1:	ADDI	Rzf	R0	1	/*ZF=1*/
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
0.35		B	L8			
0.36		NOP				
0.37	L7:	ADDI	Rcf	R0	1	/*CF=1*/
0.38	L8:	ADD	Recx	Result	R0	
1.00		BNE	Rzf	R0	MIPS_target	
1.01		NOP				

(b)

0.0	SUBU	Result	Recx	Redx	/*Generating Sub result*/
0.1	**SETFLAG**				
0.2	**SUBU**	**Reflag**	**Recx**	**Redx**	**/*Generating EFLAGS*/**
1.0	**X86JE**	**Reflag**	**MIPS_target**		**/*Branch on EFLAGS*/**

(c)

0.0	SUB	Result	Recx	Redx	/*Generating Sub result*/
0.1	**X86SUB**	**Reflag**	**Recx**	**Redx**	**/*Generating EFLAGS*/**
1.0	**X86JE**	**Reflag**	**MIPS_target**		**/*Branch on EFLAGS*/**

(d)

Figure 3. Example of EFlag translation: Original x86 program (a); the program translated with standard MIPS code (b). We can reduce the number of instructions by adding a SetFlag prefix, which turns the SUB instruction into its EFlag counterpart (c), or by adding a new instruction to define new EFlag counterpart instructions (d). The instructions in boldface type are new instructions for x86 emulation.

branch directions according to the general-purpose registers' values.

Software simulation of the 6-bit EFlags with MIPS instructions requires tens of MIPS instructions. As the x86 program segment in Figure 3a shows, we need about 40 MIPS instructions to simulate the "SUB ecx, edx" instruction to produce the subtraction result and the four most commonly used bits of EFlag (SF, ZF, OF, and CF), shown in Figure 3b.

To reduce the cost of generating the x86 EFlags with MIPS instructions, GS464 provides an EFlag counterpart instruction for each fixed-point arithmetic instruction—either by adding a SetFlag prefix to the original instruction, or, for frequently used instructions, by defining the EFlag counterpart instruction. Using new instructions can significantly reduce the number of translated instructions. For example, adding a SetFlag prefix to the SUB R1, R2, R3 instruction turns the SUB instruction into its EFlag counterpart, which performs the same calculation as the original SUB instruction but generates x86 EFlags instead of the difference

of R2 and R3 (as Figure 3c shows). We can also define the new instruction x86SUB for generating EFlagS of the subtraction, as Figure 3d shows.

An instruction's EFlag counterpart can reuse most of the original instruction's data paths, such as register renaming logic, reorder logic, issue logic, and write-back logic. We only need to slightly adjust the decode logic and execution unit.

GS464 also defines a set of branch instructions corresponding to x86 EFlag-based branch instructions, such as x86JE.

Hardware support for the floating-point format and register

The x87 FPU differs from the FPU of RISC processors in that it is accessed in a stack-based way and supports 80-bit floating-point numbers.

The x87 FPU instructions (math-related instructions for the x86 architecture) treat the eight x87 FPU data registers as the register stack. It addresses the data registers relative to the register on the top of the stack, and it stores the current top-of-stack register's number in the 3-bit TOP field in the x87 FPU status word. Maintaining the TOP pointer and calculating the absolute register number from the relative register number through software at runtime is costly. To solve this problem, GS464 maintains a TOP pointer dynamically. GS464 adds the TOP value to the floating-point register number in the decode stage. It uses the new register number as a logical register number to look up the physical register number in the register renaming stage. We define some instructions to modify the TOP pointer. GS464 uses a hardware flag in the MIPS floating-point control register to indicate whether a register number for floating-point instructions is relative to TOP. The TOP pointer only affects MIPS instructions translated from x86 instructions. With the support of the GS464 floating-point register, the TOP pointer reduces more than 10 instructions in each x86 floating-point instruction translation. It also reduces the number of switches between the translator and the generated code.

The x87 FPU supports 80-bit floating-point numbers, whereas MIPS supports 64-bit floating-point numbers. Transferring between 80-bit and 64-bit floating-point numbers requires more than 40 integer instructions. GS464 defines one instruction to transfer an 80-bit floating-point number stored in two 64-bit registers to a 64-bit number kept in one register, and two instructions to transfer a 64-bit floating-point number to an 80-bit floating-point number that occupies two registers. Figure 4 shows an example of floating-point format translation. Without any hardware support, about 40 MIPS instructions in Figure 4b are needed to simulate the three x86 instructions in Figure 4a, which GS464 can emulate with four instructions.

The x87 FPU has a 16-bit tag word to indicate the contents of each of the eight registers in the x87 FPU data-register stack (one 2-bit tag per register). The tag codes indicate whether a register contains a valid number, zero, a special floating-point number, or is empty. The x87 FPU uses the tag values to detect stack overflow and underflow. GS464 provides dedicated instructions to simulate the x87 tag with general-purpose registers and defines a new exception to reflect stack overflow or underflow exceptions in tag simulation. Except when turned on explicitly, the binary translator can ignore the tag simulation because correct programs never raise the stack overflow or underflow exception.

Hardware support for x86 multimedia instructions

The x86 ISA defines powerful multimedia instruction sets such as MMX, streaming SIMD extensions (SSE), and SSE2, which differ from the MIPS digital media extension (MDMX) in application-specific extensions of MIPS. GS464 emulates x86 multimedia instructions using its own multimedia instructions with little additional hardware cost.

In GS464, multimedia instructions have similar functions to those in x86 SSE2. GS464 defines and implements its multimedia instructions by extending the *fmt* field in the floating-point operations. Like MIPS floating-point instructions, GS464 internal floating-point operations use a 5-bit *fmt* field to specify data types. The *fmt* values of 16, 17, 20, 21, and 22 represent single-precision floating point, double-precision floating point, fixed-point

Number of instructions		Instruction			Comment
0	FLD	*%R10			
1	FMUL	*16(%R10)			
2	FSTP	*%R10			

(a)

0.00	LD	Rtmp1	12(R8)		/*convert 1st operand*/
0.01	LD	Rtmp2	4(R8)		
0.02	ANDI	Rsign	Rtmp1		/*get sign bit and sign bit of exp*/
0.03	DSLL32	Rsign	Rsign	16	/*get biased exponent
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
0.23	DMTC1	F8	Rfp2		
1.00	MUL.d	F9	F7	F8	/*64-bit multiply*/
2.00	DMFC1	Rres	F9		
2.01	DSRL32	Rsign	Rres	31	/*get sign bit*/
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
2.12	SD	Rres1	12(R8)		/*write back result*/
2.13	SD	Rres2	4(R8)		

(b)

0.0	**GSLQC1**	**F4**	**4(R8)**		**/*128-bit load to F4 and F5*/**
0.1	**CVT.d.ld**	**F7**	**F4**	**F5**	**/*80-bit to 64-bit convert*/**
0.2	**GSLQC1**	**F2**	**20(R8)**		**/*128-bit load to F2 and F3*/**
0.3	**CVT.d.ld**	**F8**	**F2**	**F3**	**/*80-bit to 64-bit convert*/**
1.0	MUL.d	F9	F7	F8	/*64-bit multiplication*/
2.0	**CVT.ud.d**	**F7**	**F9**		**/*64-bit to high part of 80-bit*/**
2.1	**CVT.ld.d**	**F8**	**F9**		**/*64-bit to low part of 80-bit*/**
2.2	**GSSQC1**	**F7**	**4(R8)**		**/*128-bit store*/**

(c)

Figure 4. Example of 80-bit floating-point operation translation. The original x86 program (a), translated with standard MIPS code (b), and translated with GS464 floating-point conversion instructions (c). The instructions in boldface are new instructions for x86 emulation.

word, fixed-point long, and paired single-precision floating point, respectively. GS464 SIMD multimedia operations extend the *fmt* fields in floating-point operations to define eight 8-bit or four 16-bit fixed-point data units in the 64-bit floating-point data path. For example, the ADD.fmt operation represents ADD.single, ADD.double, or ADD.PS in MIPS. GS464 extends the operation to represent ADD.8 × 8 and ADD.4 × 16. To further facilitate emulating x86 multimedia instructions, GS464 extends MIPS-style unaligned memory-access instructions to floating-point registers.

Other hardware support for x86 binary translation

In addition to support for x86 EFlags, floating-point instructions, and multimedia instructions, GS464 adopts many other techniques to further facilitate x86 emulation.

New addressing mode. The x86 ISA has more flexible addressing modes than MIPS. It supports the SIB addressing mode in the form

of "(base) + (index) × scale + disp," whereas MIPS only supports the "(base) + disp" for both fixed-point and floating-point load and store instructions, and the "(base) + (index)" addressing mode for floating-point load and store instructions. To ease the translation of x86 addressing modes, GS464 supports the "(base) + (index) + disp8" addressing mode for both fixed-point and floating-point load and store instructions.

Bounded load and store. GS464 supports bounded load and store instructions, which read the bound register as the memory-access boundary in addition to the normal base register and value register. The bounded load and store instructions have the same behavior as normal load and store instructions, except an address exception is raised if the memory-access address of a load and store instruction exceeds the boundary address. The bounded load and store instructions help ease the translation of segment address mode instructions in x86.

Fixed-point multiplication and division. MIPS fixed-point multiplication and division instructions use the special Hi/Lo registers as destination registers, and MIPS provides instructions to move data between Hi/Lo registers and general-purpose registers. GS464 implements fixed-point multiplication and division instructions, which use general-purpose registers as destination registers to ease the translation of fixed-point multiplication and division x86 instructions.

Byte insertion and extraction. The x86 ISA supports 8-, 16-, 32-, and 64-bit operations, whereas RISC processors normally support 32- and 64-bit operations. To close this gap, GS464 implements flexible byte insertion instructions that can insert a byte, half word, or word from any location of a register to any location of another register; and byte extraction instructions that extract a byte, half word, or word from any register location and store the result to another register after zero or sign extension.

Hardware supports for binary translation mechanism

Binary translation dynamically generates binary codes on the target machine. The binary codes generated during runtime are the data results of the binary translator. Because the binary translator stores these codes in the data cache, executing them requires flushing them from the data cache and loading them into the instruction cache. However, flushing the data cache through software to keep coherence between the data and instruction caches is time consuming. Therefore, GS464 keeps coherence between the data and instruction caches, as well as the L2 cache, through hardware.

Translation of indirect branch instructions is costly because the binary translator must look up the MIPS branch target dynamically according to the x86 branch target from some mapping mechanism, such as a hash table. GS464 implements a 64-entry content-associated memory (CAM) to speed up the translation and execution of indirect branch instructions. Each CAM entry includes a process ID field, an address field, and a data field. GS464 provides instructions to read, write, and probe the CAM. Figure 5 shows an indirect branch target translation. GS464 CAM instructions greatly reduce the complexity of translation results. Although we've added more codes to handle the CAM miss situation, they're rarely executed.

Binary translation system

The Godson-3 binary translator is built on top of the Linux operating system. We improved the Linux operating system based on Godson-3's MIPS ISA to provide a system call compatible with x86 Linux and increase binary translator efficiency. We implemented a process-level binary translator on Linux to run x86 applications. We can also implement a system-level binary translator to achieve x86 compatibility at the ISA level.

Godson-3's binary translation system is an improvement upon the open-source binary translator QEMU.[12] In addition to implementing a new interpreter and translator under the QEMU framework, we redesigned the intermediate representation to allow optimizations, such as fixed register allocation and lazy conditional code evaluation. Like other traditional binary translators, the Godson-3 VMM initially interprets x86 instructions on the MIPS processor and monitors the behavior during interpretation. When the VMM finds hot spots, it translates

Number of instructions	Instruction			Comment
0	MOV	%RAX	%R11	
1	JMPQ	%*R11		

(a)

0	MOVE	Rr11	Rrax	
1.0	**CAMPV**	**Rtmp**	**Rr11**	**/* Look up the first level indirect jump address */**
1.1	**CAMPV**	**Rtgt**	**Rtmp**	**/* Look up the final jump address */**
1.2	JR	Rtgt		

(b)

Figure 5. Example of indirect branch target translation: The original x86 program (a), and the program translated with Godson-3 content-associated memory (CAM) instructions (b). The boldface text indicates new instructions for x86 emulation.

their x86 codes to MIPS codes for execution and optimizes them at different levels according to the hot degree. The multicore Godson-3 can also perform parallel optimization for very hot spots. The Godson-3 hardware support for x86 instruction translation makes the translation process straightforward, not only improving the translated code's efficiency, but also simplifying the binary translator's implementation.

System optimization also improves the Godson-3 binary translator's performance. To fully use the 64-bit processor resources, such as registers, we implemented the N32/N64 tool chain. The Godson-3 system uses GNU binutils, the GNU compiler collection (GCC), and the GNU C library (GLIBC) as the basic compilation environment. We modified the binutils' GNU Assembler (Gas) to help GCC recognize all the new instructions. We added new instruction templates and pipeline descriptions to GCC to improve the generated code quality. We also implemented autovectorization for the 128-bit memory-access instructions and multimedia instructions to fully utilize the Godson-3 processor resources. We profiled GLIBC and rewrote the frequently executed routines.

Preliminary performance results

The first four-core Godson-3 design was taped out in 2008. Before the chip returned from fabrication, we carried out the Godson-3 performance analysis on two platforms: a register-transfer-level (RTL) simulation platform and a field-programmable gate array (FPGA) prototyping platform. In the RTL simulation environment, we set the core clock frequency to 1 GHz, the DDR2/DDR3 clock frequency to 333 MHz, and the HyperTransport clock frequency to 800 MHz. To speed up the simulation, we used Cadence's Xtreme-3[13] simulation accelerator, which can achieve a speed of 200,000 to 400,000 cycles per second. Because of the difficulty of building a full-scale Godson-3 FPGA prototype system, we built a partial-scale prototype to evaluate a single processor core's performance. The prototype system includes one processor core, a 1-Mbyte L2 cache, one DDR2/DDR3 controller, and one HyperTransport controller. FPGA prototyping speed is 50 MHz, which is much faster than RTL simulation. Because the ratios between the FPGA prototype's core and I/O clocks differ from those in a real system, we carefully adjusted the FPGA prototyping system's I/O latency to obtain accurate performance results.

Table 1 shows the benchmarks we tested on the Xtreme-3 and FPGA platforms. We selected nine typical kernels or full applications to evaluate the hardware improvements and software translation efficiency. We ran all benchmarks in three modes:

- native MIPS mode, in which benchmarks are directly compiled into MIPS binary and run on MIPS hardware;

Table 1. Specifications of benchmark kernels tested on Godson-3 prototypes.

Name	Source	Language	Experiment platform	Note
Floating-point IDCT	Microbench	C	Xtreme-3	x86 floating-point
Floating-point FFT	Microbench	C	Xtreme-3	x86 floating-point
General control	Microbench	C	Xtreme-3	x86 control complicated programs
Fixed-point IDCT	EEMBC	x86 assembly	FPGA	x86 SIMD
Fixed-point FFT	EEMBC	x86 assembly	FPGA	x86 SIMD
Operating system startup code	Microbench	C and x86 assembly	Xtreme-3/FPGA	
164.gzip (train)	SPEC 2000	C	FPGA	
197.parser (train)	SPEC 2000	C	FPGA	
179.art (train)	SPEC 2000	C	FPGA	

- basic translator mode, in which benchmarks are compiled into x86 binary and run on MIPS hardware using the basic QEMU binary translator; and
- improved translator mode, in which benchmarks are compiled into x86 binary and run on MIPS hardware using the improved QEMU binary translator with x86 binary translation acceleration on RISC processors (XBar) hardware support.

We compiled all programs with the GCC-O3 flag.

Figure 6 shows the relative performance of basic and improved translator modes compared to native MIPS mode. Godson-3's x86 emulation hardware support significantly accelerates binary translation from x86 to MIPS, and on average achieves performance that is nearly 70 percent of the ideal mode.

The performance results in Figure 6 are still preliminary. Much more work must be done to further improve the binary translator.

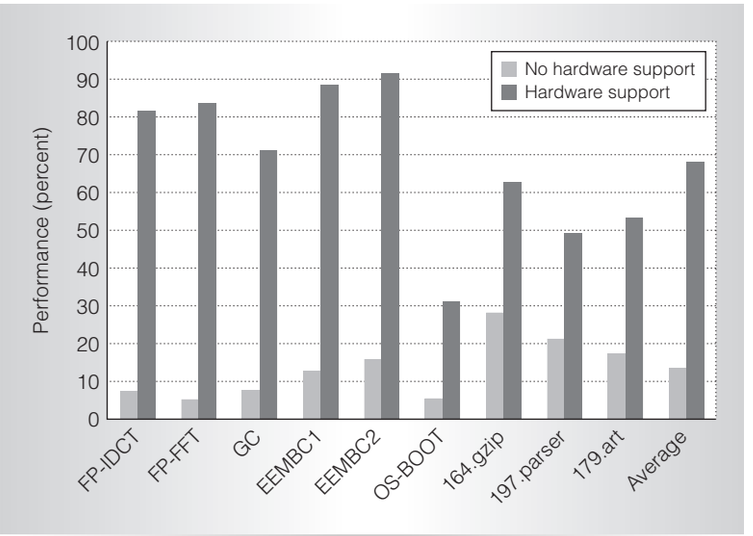

Figure 6. Experimental results for the nine benchmarks. 100 percent performance represents the execution of native MIPS code. Columns represent the execution of x86 emulation with and without hardware support. Higher is better.

Scalable multicore interconnection

Figure 7 shows the Godson-3 overall architecture. Each node in the mesh includes an 8 × 8 crossbar connecting four processor cores as four masters, four shared L2-cache banks as four slaves, and four adjacent nodes in the east, south, west, and north directions as four masters and four slaves. A second-level crossbar inside the node connects the DDR2/DDR3 memory controllers to L2-cache banks. The Godson-3 HyperTransport I/O controller is connected to the free crossbar ports of boundary nodes.

The four-core, eight-core, and 16-core Godson-3 chips take one, two, and four nodes in the mesh, respectively. Figure 8 shows the architecture of the four-core and eight-core Godson-3. Both have two DDR2/DDR3 controllers, two HyperTransport controllers, and other necessary peripheral interfaces. The four-core and eight-core Godson-3 have the same I/O package.

The Godson-3 interconnection network takes the 128-bit AXI standard interface, which is simple, efficient, and open. We extended the L1 crossbar AXI interface to support cache coherence in Godson-3.

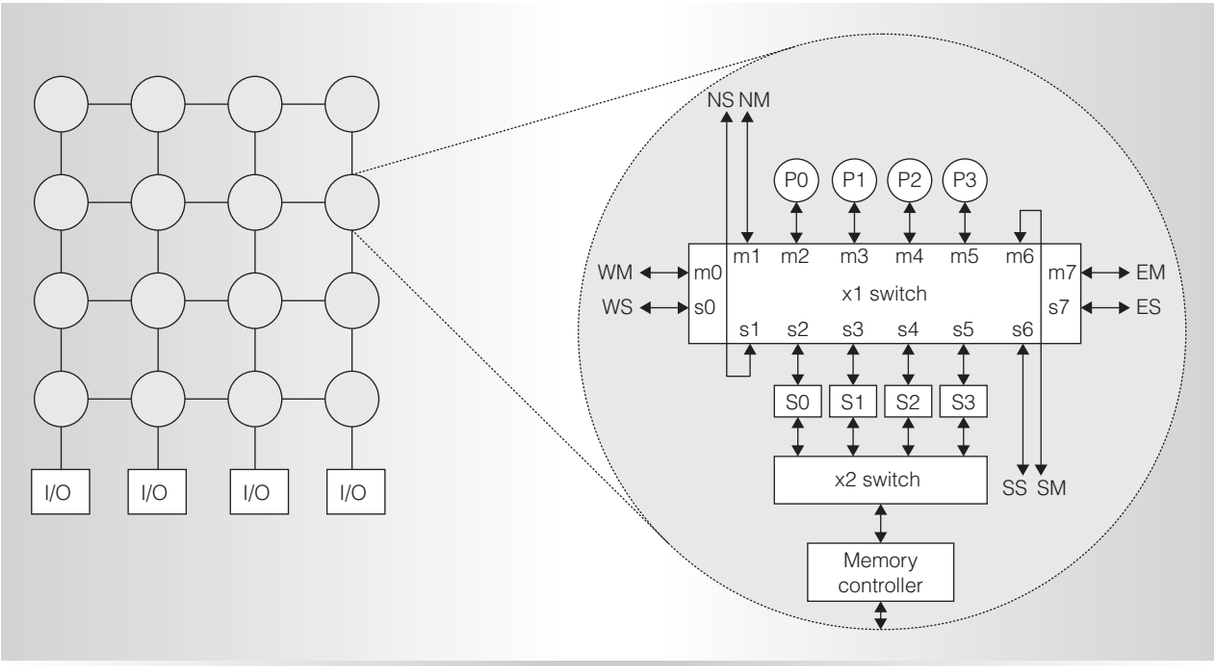

Figure 7. Godson-3 interconnection topology. A 2D mesh connects nodes, and each node connects cores and L2 modules through a crossbar; memory controllers are connected in nodes, and I/O controllers are connected in the 2D mesh boundary.

Figure 9 shows the crossbar's architecture. An 8 × 8 multiplexing matrix connects eight AXI master link (AML) modules and eight AXI slave link (ASL) modules. Each link has five channels according to the AXI protocol:

- write address channel (AW),
- write data channel (W),
- write response channel (B),
- read address channel (AR), and
- read data channel (R).

The AML routes read and write requests (AW, W, and AR channels) according to the access address. Each AW or AR channel has eight address windows, each of which assigns a destination ASL port to a matching request. The ASL routes read and write replies (R and B channels) according to the corresponding request's AML port number. Both AML and ASL have two pipeline stages—that is, the crossbar has a latency of four hops. Two buffers for each pipeline stage prevent the pipeline stalling signal from propagating across stages.

Reconfigurable architecture

Godson-3's reconfigurability involves three features:

- reconfigurable processor cores,
- dynamic L2 cache migration, and
- reconfigurable DMA engine.

Because Godson-3's interconnection network adopts the AXI protocol, we can insert any processor cores complying with the AXI protocol into the network slots. Depending on the application, we can configure the Godson-3 chips' AXI ports to contain either general-purpose GS464 cores or special-purpose cores.

A distributed shared memory system's performance depends heavily on memory-access locality. In a nonuniform cache access (NUCA) system such as Godson-3, accesses to nonlocal L2 cache blocks suffer from high L2 access latency. Godson-3 can

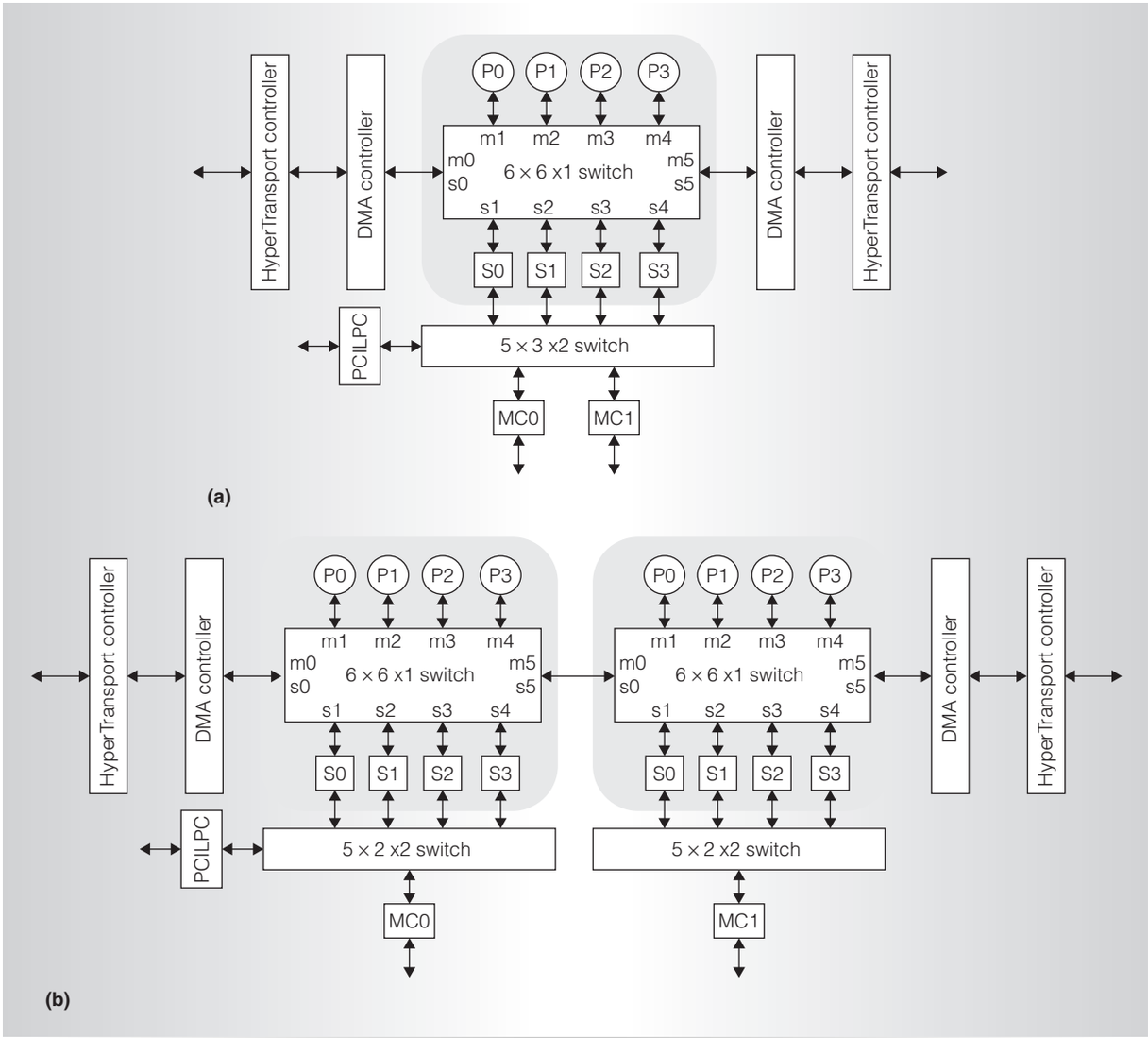

Figure 8. The Godson-3 architecture: four-core version (a) and eight-core version (b).

increase the locality of L2 cache access by migrating shared L2 cache blocks. Each AW or AR channel in the AXI crossbar has eight reconfigurable address windows, which let the software dynamically bind memory addresses to cache block locations. With the reconfigurable address windows, the software can migrate home blocks across cache banks, or set cache blocks as private or shared blocks according to the locality of memory accesses.

We can also configure Godson-3's DMA engine to achieve high performance. Software can decide whether the DMA data is to or from main memory or to or from the L2 cache directly. The DMA engine maintains cache coherence automatically when transferring data between I/O and memory. In addition, we can configure the DMA engine to prefetch data from memory to L2 cache, or to transpose a matrix in the memory without intervention from the processor core.

The four-core physical implementation

We built the four-core Godson-3 using the seven-metal 65-nm CMOS process. We designed it using cell-based flow with

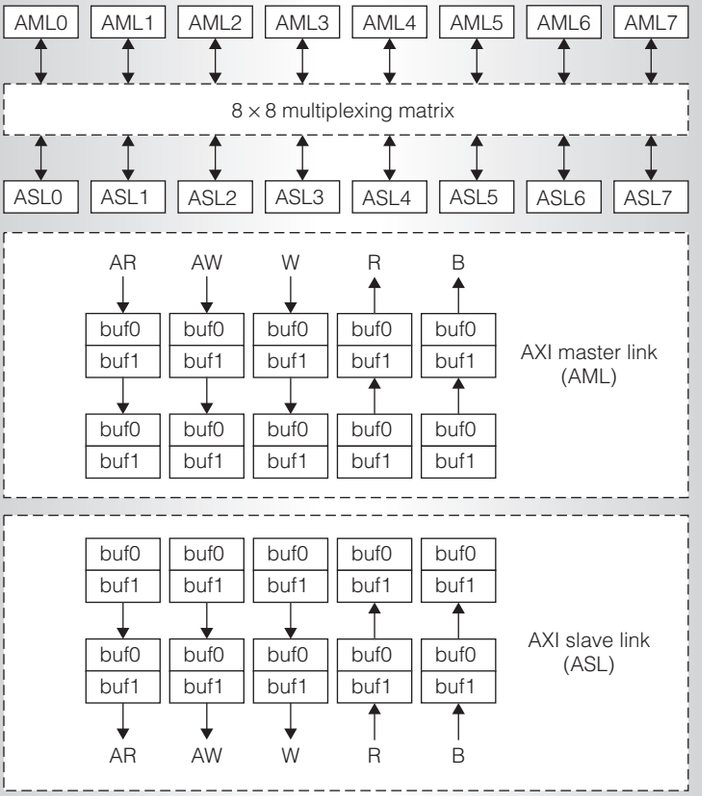

Figure 9. Crossbar architecture. Four cycles are needed to travel through the crossbar.

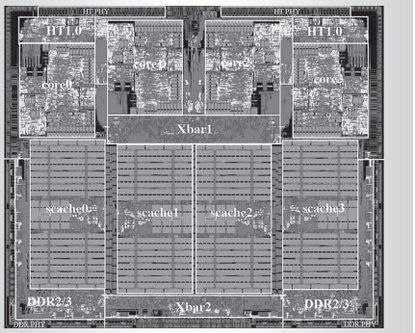

Figure 10. Layout of the four-core Godson-3. The HyperTransport links are at the top, the CPU cores and L2 caches are in the center, and the DDR2/DDR3 controllers and physical layer are at the bottom. Two crossbars connect these components.

some custom-designed macros and manual placement and routing. Full-custom macros include a four write port and four read port (4w4r) 64 × 64 register file, a 64 × 64 CAM for TLB, a phase-locked loop (PLL), and a HyperTransport link. To reduce clock cycle time, we manually mapped specific datapath modules or modules with replicated structure to the cell library and placed them in a bit-sliced structure. The clock tree takes an H-tree structure and is mainly manually placed and routed. We use a clock skew technique for the critical-path pipeline stage to borrow time from adjacent pipeline stages.

Figure 10 shows the layout of the four-core Godson-3. The chip includes 425 million transistors and the die size measures 14,240 micrometers by 12,205 micrometers. The chip's highest frequency is 1.0 GHz, and its power dissipation ranges from 5 to 10 W depending on the application.

Our recent work includes an eight-core Godson-3 chip design with improved frequency. Moreover, research on teraflops-scale many-core chips is also in progress. MICRO

Acknowledgments

The work presented here is supported by the National High-Tech Research and Development 863 Program of China under Grant No. 2008AA110901, the National Basic Research 973 Program of China under Grant No. 2005CB321600, and the National Natural Science Foundation of China under Grant No. 60736012. We thank Lingyi Huang, Huandong Wang, Dan Tang, Menghao Su, Xiaoyu Li, Songsong Cai, Pengyu Wang, Ge Zhang, Haihua Shen, Dandan Huan, Baoxia Fan, Liang Yang, Yanping Gao, Jiangmei Wang, Ru Wang, Ying Xu, Bing Xiao, Xu Yang, Shiqiang Zhong, Feng Zhang, Ying Zhao, Zongren Yang, Liqiong Yang, Lu Zhang, Hao Cui, Hang Yu, Zhuo Gao, Xiangku Li, Zichu Qi, Cheng Qian, Weidan Wu, Jinzhao He, and Jin Zhang for their contributions to the Godson-3 project.

References

1. R. Kessler, ''The Alpha 21264 Microprocessor,'' *IEEE Micro,* vol. 19, no. 2, Mar./Apr. 1999, pp. 24-36.
2. K.C. Yeager, ''The MIPS R10000 Superscalar Microprocessor,'' *IEEE Micro,* vol. 16, no. 2, Mar./Apr. 1996, pp. 28-41.
3. T. Horel and G. Lauterbach, ''UntraSparc-III: Designing Third-Generation 64-bit Performance,'' *IEEE Micro,* vol. 19, no. 3, May/June 1999, pp. 73-85.
4. A. Kumar, ''The HP PA-8000 RISC CPU,'' *IEEE Micro,* vol. 17, no. 2, Mar./Apr. 1997, pp. 27-32.
5. R. Kalla et al., ''IBM POWER5 Chip: A Dual-Core Multithreaded Processor,'' *IEEE Micro,* vol. 24, no. 2, Mar./Apr. 2004, pp. 40-47.
6. P. Kongetira et al., ''Niagara: A 32-Way Multithreaded Sparc Processor,'' *IEEE Micro,* vol. 25, no. 2, Mar./Apr. 2005, pp. 21-29.
7. K. Sankaralingam et al., ''Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture,'' *Proc. 30th Int'l Symp. Computer Architecture* (ISCA 03), ACM Press, 2003, pp. 422-433.
8. W. Hu et al., ''Microarchitecture of the Godson-2 Processor,'' *J. Computer Science and Technology,* vol. 20, no. 2, Mar. 2005, pp. 243-249.
9. A. Klaiber, ''The Technology behind Crusoe Processors,'' white paper, Transmeta Corp., Jan. 2000.
10. A. Chernoff et al., ''FX!32: A Profile-Directed Binary Translator,'' *IEEE Micro,* vol. 18, no. 2, Mar./Apr. 1998, pp. 56-64.
11. K. Ebcioglu et al., ''Dynamic Binary Translation and Optimization,'' *IEEE Trans. Computers,* vol. 50, no. 6, June 2001, pp. 529-548.
12. F. Bellard, ''QEMU, a Fast and Portable Dynamic Translator,'' *Proc. Usenix Ann. Technical Conf.,* Usenix Assoc., 2005, pp. 41-46.
13. Incisive Xtreme Series Datasheet, http://www.cadence.com/rl/Resources/datasheets/Cadence_6569_DS_R2.pdf.

Weiwu Hu is a professor of computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high-performance computer architecture, parallel processing, and VLSI design. Hu has a PhD in computer science from the Institute of Computing Technology.

Jian Wang is an associate professor of computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high-performance computer architecture and operating systems. Wang has an MS in computer science from the Institute of Computing Technology.

Xiang Gao is an assistant professor of computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high-performance computer architecture, parallel processing, and operating systems. Gao has a PhD in computer science from the University of Science and Technology of China.

Yunji Chen is an assistant professor of computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include hardware verification and high-performance computer architecture. Chen has a PhD in computer science from the Institute of Computing Technology.

Qi Liu is a PhD candidate in computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer architecture, virtualization, and binary translation. Liu has a BS in computer science from the University of Electronic Science and Technology of China.

Guojie Li is the director of the Institute of Computing Technology, Chinese Academy of Sciences, and an academician at the Chinese Academy of Engineering. His research interests include high-performance computer architecture, parallel algorithms, and artificial intelligence. Li received his PhD in computer science from Purdue University.

Direct questions and comments about this article to Weiwu Hu, Institute of Computing Technology, Chinese Academy of Sciences, P.O. Box 2704-25, Beijing, China, 10019; hww@ict.ac.cn.