High Performance, Energy Efficiency, and Scalability With GALS Chip Multiprocessors

Zhiyi Yu and Bevan M. Baas

***Abstract*—Chip multiprocessors with globally asynchronous locally synchronous (GALS) clocking styles are promising candidates for processing computationally-intensive and energy-constrained workloads. The GALS methodology simplifies clock tree design, provides opportunities to use clock and voltage scaling jointly in system submodules to achieve high energy efficiencies, and can also result in easily scalable clocking systems. However, its use typically also introduces performance penalties due to additional communication latency between clock domains. We show that GALS chip multiprocessors (CMPs) with large inter-processor first-inputs–first-outputs (FIFOs) buffers can inherently hide much of the GALS performance penalty while executing applications that have been mapped with few communication loops. In fact, the penalty can be driven to *zero* with sufficiently large FIFOs and the removal of multiple-loop communication links. We present an example mesh-connected GALS chip multiprocessor and show it has a less than 1% performance (throughput) reduction on average compared to the corresponding synchronous system for many DSP workloads. Furthermore, adaptive clock and voltage scaling for each processor provides an approximately 40% power savings without any performance reduction. These results compare favorably with the GALS uniprocessor, which compared to the corresponding synchronous uniprocessor, has a reported greater than 10% performance (throughput) reduction and an energy savings of approximately 25% using dynamic clock and voltage scaling for many general purpose applications.**

***Index Terms*—Array processor, chip multiprocessor, energy efficient, globally asynchronous locally synchronous (GALS), low power, scalable.**

I. Introduction

MODERN deep submicrometer fabrication technologies enable very high levels of integration such as a recent dual-core 1.7 billion-transistor chip [1]. A highly promising approach to efficiently using these circuit resources is the integration of multiple processors onto a single chip (called a *chip multiprocessor* or *CMP*) to achieve higher performance through parallel processing. CMPs can potentially also provide increased energy efficiency by allowing the clock frequency and supply voltage to be reduced together to dramatically reduce power dissipation during periods when full rate computation is not needed and conditions permit.

Despite their promising benefits, complex systems built using deep submicrometer technologies encounter some unique challenges. One of the most critical is the design of the clocking system. Traditional globally synchronous clocking circuits have become increasingly difficult to design with growing chip sizes, clock rates, relative wire delays, and parameter variations [2]. Additionally, high speed global clocks consume a significant portion of system power budgets and lack the flexibility to independently control the clock frequencies of submodules to achieve high energy efficiency. The globally asynchronous locally synchronous (GALS) [3] clocking style separates processing blocks such that each block is clocked by an independent clock domain. This approach is a promising strategy to address global clock design challenges.

GALS systems are often highly energy efficient due to their simplified clock tree [4], and their enabling of joint clock and voltage scaling in system submodules [5], [6]. However, GALS clocking typically also introduces a performance penalty due to additional communication latency between asynchronous domains [5], [7].

We show that GALS CMPs under the right conditions can hide much of the GALS performance penalty and at the same time, take full advantage of its scalability and high energy efficiency. Along with a thorough investigation of GALS effects on system performance, we show that such GALS CMPs have small performance penalties compared to corresponding synchronous systems. Furthermore, the small performance penalty can be completely eliminated by using sufficiently large FIFOs for inter-processor communication and programming without multiple-loop communication links. Scalability is enhanced due to the lack of a need for a global clock tree. In addition, the potential energy savings from joint clock and supply voltage scaling is increased in the common situation when workloads have unbalanced computational loads for each processor, thereby increasing the probability processors can be tuned to save power. This work is distinguished from previous GALS multiprocessor evaluations [8] by not restricting the analysis to systems with global communication schemes.

This paper is organized as follows. Section II investigates several key design choices which impact the behavior of GALS systems. Section III introduces our GALS chip multiprocessor. Section IV investigates the effect of the asynchronous communication penalty to the performance of this GALS chip multiprocessor. Its scalability is shown in Section V compared to the corresponding synchronous system. Section VI investigates the power efficiency of this GALS chip multiprocessor when using clock frequency and supply voltage scaling. This paper concludes with a summary.

Manuscript received July 22, 2007; revised December 21, 2007. First published November 25, 2008; current version published December 17, 2008. This work was supported in part by Intel, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598 and CSR Grant 1659, IntellaSys, S Machines, Artisan, and a UCD Faculty Research Grant.

Z. Yu is with the Microelectronics Department, Fudan University, Shanghai, 201203 China (e-mail: zhyyu@ece.ucdavis.edu).

B. M. Baas is with the Electrical and Computer Engineering Department, University of California at Davis, CA 95616 USA (e-mail: bbaas@ucdavis.edu)

Digital Object Identifier 10.1109/TVLSI.2008.2001947

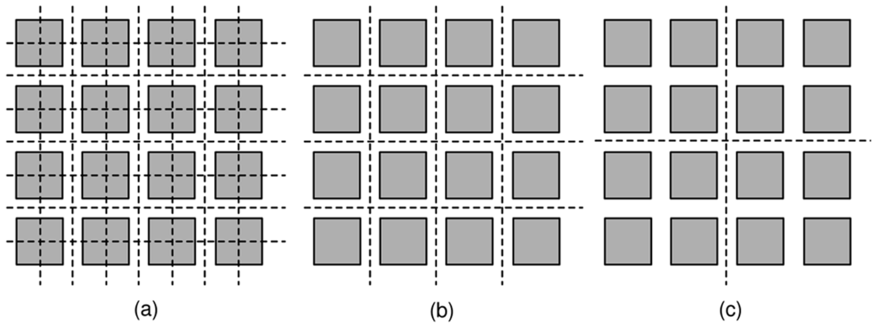


Fig. 1. Three example clock domain partitionings, from a sub-processor fine grain partitioning to a multi-processor coarse grain partitioning. The dotted lines show clock domain partition boundaries. Analyses and results are independent of whether modules are homogeneous or heterogeneous. (a) Each processor contains four clock domains. (b) Each processor contains one clock domain. (c) Four processors in each clock domain.

II. Exploring the Key GALS Chip Multiprocessor Design Options

Several design options impact the behavior of GALS CMPs. This section presents the three most fundamental parameters: clock domain partitioning, asynchronous boundary communication, and the inter-processor network.

A. Clock Domain Partition of GALS Chip Multiprocessors

The most basic issue in designing a GALS chip multiprocessor system is the granularity of the multiple clock domain partitions. The most fine grain method is to partition each processor into several clock domains, which we call a *GALS uniprocessor*. Another method is to partition each processor into its own clock domain. A more coarse grain method is to group several processors together into local shared clock domains. Fig. 1 illustrates these three methods.

Partitioning each processor into several clock domains is the most widely-studied partitioning [5], [6]. Its key advantage is providing opportunities to use clock and/or voltage scaling not only in individual processors, but also in modules inside each processor. Its primary disadvantage is a relatively significant performance penalty and design overhead.

Upadhyay *et al.* [9] investigated various levels of coarse-grain clock domain partitioning for a GALS chip multiprocessor. They compared power savings from simplified GALS clock trees versus additional power from the local clock generator plus asynchronous communication circuits for different partitioning granularities. They examined an array of 256 processors with each processor 2 mm $\times$ 2 mm in a 0.18-μm technology, and concluded that depending on the communication links, grouping 8–32 processors together is the most efficient. Unfortunately, the study did not consider power reduction effects of clock frequency and supply voltage scaling.

Since GALS uniprocessor and coarse-grained clock domain partitioning have been well studied, and because we find placing each processor in its own clock domain is a simple and efficient strategy, this approach is the focus of this work. Despite the fact power savings from simplified clock trees may not be able to compensate fully for overhead due to the local clock generator and asynchronous communication interface, it makes each processor highly uniform and simplifies the physical design. Furthermore, as shown in Section VI, it provides the flexibility to scale the clock frequency and supply voltage for each processor which can achieve significant power savings compared to a fully synchronous design.

B. Transferring Data Across Asynchronous Boundaries

Circuitry to reliably and efficiently move data across asynchronous clock boundaries is a key component in GALS systems. We can classify techniques for these domain crossings into one of two categories.

- *Single transaction handshaking* acknowledges each data word before a subsequent word can be transferred, and a corresponding latency exists for each data transfer and acknowledgment, which can significantly decrease the total data throughput.
- *Source synchronous multi-word flow control* routes the source clock along with the data to synchronize data writes. The source transmits data words without individual acknowledgments and it halts transfers when the destination indicates it can no longer accept data. Since the technique works well with high data rates (one word per clock cycle is a good operating point), provision must normally be made for the fact that the flow control signal may arrive multiple clock cycles after the destination module decides to halt the source module. The problem is easily addressed by reserving space in the destination buffer for these situations [10] and the control signals act on multiple data words, hence our name: *source synchronous multi-word flow control*.
  This technique generally requires a larger buffer memory but should normally sustain higher throughputs. The larger memory often does not present a significant area penalty when compared to the area of a large block such as a processor. Furthermore, as shown in Section IV-C2, the relatively larger memory has the benefit of hiding some communication latency.
  Although single-word latency may be larger compared to the single transaction handshaking technique, this is often not a significant concern because in the common case where more than a few data words exist in inter-processor first-input–first-output (FIFO) buffers, the inter-processor latency has no impact. Interestingly, if we consider the common case in digital signal processing (DSP) and embedded applications where data are transferred in blocks [e.g., the N words into and out of an N-point

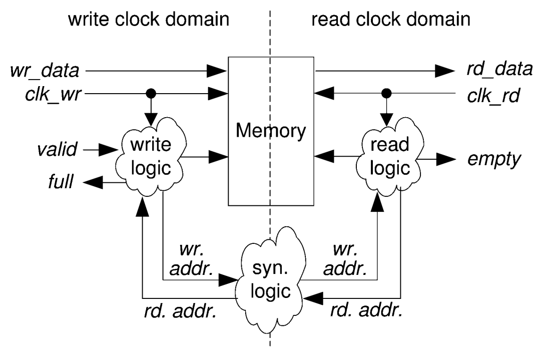


Fig. 2. Block diagram for a dual-clock FIFO utilizing a memory array as the FIFO buffer.

fast Fourier transform (FFT) processor], the *latency of a block of data* will likely be lower with the source synchronous multi-word flow control method compared to the single transaction handshaking method due to its higher throughput.

Dual-clock FIFOs [11], [12] are well-suited to provide asynchronous boundary communication using the source synchronous multi-word flow control method. They operate by partitioning the FIFO such that the write portion is in one clock domain and the read portion is in another clock domain. Fig. 2 is a high level diagram of a dual-clock FIFO [10]. In the synchronization logic block, the read and write addresses pass across the asynchronous interface and are used to calculate whether the FIFO is full or empty. Dual-clock FIFOs introduce extra communication delay compared to a corresponding synchronous design, and exact values vary depending on many circuit, fabrication technology, and operating condition factors.

C. Inter-Processor Network

The networking strategy between processors also strongly impacts the behavior of GALS CMPs.

Smith [8] proposes and analyzes a GALS chip multiprocessor with inter-processor and processor-memory communication through a shared global bus. This scheme provides very flexible communication, but places heavy demands on the global bus; thus, the system performance is highly dependent on the level and intensity of the bus traffic. Furthermore, this architecture lacks scalability since increased global bus traffic will likely significantly reduce system performance under high traffic conditions or with a large number of processors.

A distributed network strategy such as a "nearest neighbor" mesh distributes the interconnections across the entire chip and the communication load for each inter-processor link can be highly reduced. This architecture also provides near-perfect physical scalability due to its tile-based architecture without global wires.

III. GALS CMP

To illustrate principles presented in this paper, we draw upon a GALS CMP that was recently fabricated [13] which to the best of our knowledge, is the first tile-based chip multiprocessor with GALS clocking. A special feature of the design is that it provides a mode where all processors operate fully synchronously, thereby providing an excellent testbed for a GALS versus non-GALS comparison. This section provides a brief overview of the chip's design and application mapping.

Pleiades [14] and FAUST [15] use GALS clocking, but they both use the single transaction handshake scheme for their asynchronous boundary communication which performs quite differently from the source synchronous multi-word flow control used in this work. The processor array by Ambric [16] uses GALS clocking but no details have been reported regarding its operation. Other clocking styles such as rationally-related clocking used by Synchroscalar [17] and mesochronous clocking used by an 80-core chip [18] are clearly different from GALS-clocked chips.

This work is the first to present an analysis of the performance and power effects of the GALS style when applied at the CMP level.

A. GALS and Non-GALS CMPs

The aforementioned CMP contains multiple uniform simple processors, as shown in Fig. 3(a). Each processor contains a small instruction memory that is not a cache and is designed to be written only by signals external to the chip; therefore, its contents are intended to be constant during normal operation. Processors also contain a small data memory that is not a cache and is fully under the control of software running on each processor. These memory size and organization choices are not a necessary limitation of the architecture, but merely choices made to reduce memory system complexity in this test chip, and because they are sufficient for our targeted applications. Processors contain two 32-word input FIFOs which can be configured to connect to any two of the four neighboring processors, thereby enabling inter-processor communication via the 2-D mesh network. Connections between neighboring processor tiles therefore consist of two opposing-direction unidirectional links. Each processor contains a 16-bit datapath and executes 32-bit instructions. The 6×6 GALS chip multiprocessor is implemented in 0.18-μm CMOS technology [13].

Fig. 3(b) shows a single processor in the GALS system. Processors utilize individual programmable ring oscillators that are configurable over a wide range of frequencies. Each processor also contains two dual-clock FIFOs, which write and read in independent clock domains and reliably transfer data across the asynchronous boundaries. For increased characterization capability, a configurable number of synchronization registers are inserted at the clock domain interface to alleviate metastability. Two synchronization registers are used in the experiments shown in Section IV.

The synchronous processor is shown in Fig. 3(c), and local clock oscillators are unnecessary since a global clock is provided to all processors. Processors' FIFOs are fully synchronous for this system. The synchronous chip multiprocessor is emulated using special configurations in the chip, and uses a global clock signal without synchronization registers between asynchronous boundaries inside the inter-processor FIFOs.

The extra circuitry for supporting the GALS clocking style—the local oscillator and logic in the FIFOs related to dual-clock operation—occupies only approximately 1% of the

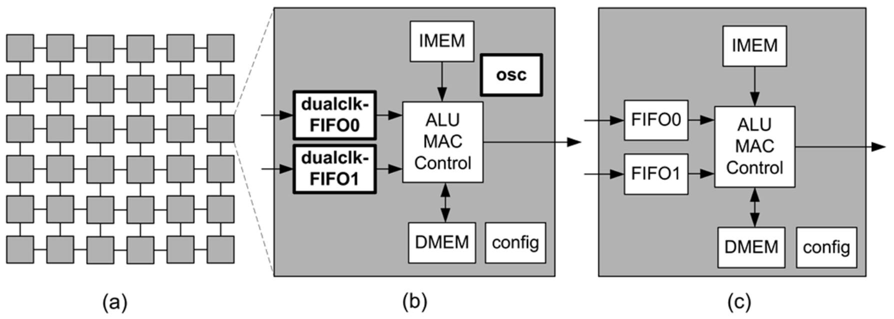


Fig. 3. Two CMPs: one using a fully synchronous style and the other using a GALS style with per-processor clock domains. (a) 6×6 chip multiprocessor, (b) single processor in the GALS system, (c) single processor in the synchronous system.

processor's area. Considering the fact the GALS system has a vastly simplified clock tree, the area difference between a GALS system and a synchronous-only system is negligible.

B. Applications and Mapping on the GALS Multiprocessor

Programming the described GALS chip multiprocessor is accomplished by dividing applications into several tasks, coding each task independently, and mapping each task onto independent or shared processors. The mapped applications we consider include: an 8-point discrete cosine transform (DCT) using 2 processors, an 8×8 DCT using 4 processors, a zig-zag transform using 2 processors, a merge sort using 8 processors, a bubble sort using 8 processors, a matrix multiplier using 6 processors, a 64-point complex FFT using 8 processors, a JPEG encoder core using 9 processors, and a fully-compliant IEEE 802.11a/g (Wi-Fi) wireless LAN transmitter using 22 processors [19].

We have chosen many of the selected DSP and embedded tasks because they are especially well suited for our processor array for two main reasons. First, these tasks require relatively small amounts of data and instruction memory, and all instructions and "static data" (e.g., data coefficients) reside on the chip. In some cases (e.g., the FFT application), one or more processors are programmed to serve only as data storage and perform no computation on the data [19]. Second, the presented applications are also relatively easy to partition into many smaller sub-tasks.

All tasks and applications were parallelized and coded by hand, and their programs are unscheduled and lightly optimized—with the exception of the DCT transforms and the JPEG encoder which have been moderately optimized. Clearly, many implementations are possible; these examples should be taken only as reasonable representative implementations.

As an example, the 4-processor 8×8 DCT application is shown in Fig. 4. The 8×8 DCT is performed with two 1-D 8-point DCTs in the first and third processors using an efficient algorithm [20], and with transposes of the rows and columns of the data block in the second and fourth processors.

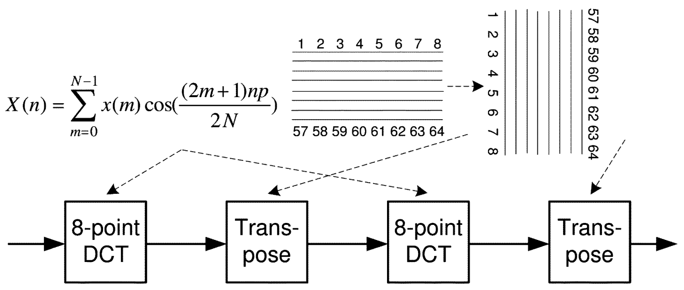


Fig. 4. 8×8 DCT implementation using 4 processors.

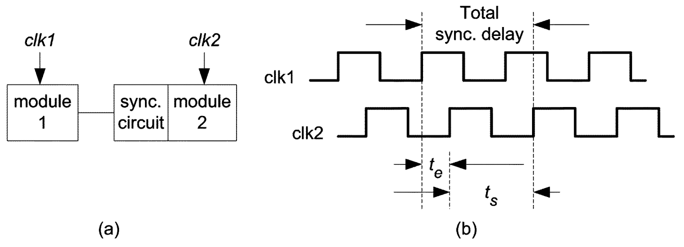


Fig. 5. GALS system module boundary and timing of the synchronization delay across the boundary. (a) A simple GALS system. (b) Sync. delay = clk edge alignment $(t_e)+$ sync. circuit (t_s).

IV. Reducing and Eliminating Performance Penalties in GALS Chip Multiprocessors

GALS systems require synchronization circuits between clock domains to reliably transfer data. Clock phase edge alignment time for unmatched clocks and synchronization circuitry introduces a synchronization delay as illustrated in Fig. 5. This delay normally results in a reduction of performance (throughout).

In this section, we discuss in depth principles behind how GALS clocking affects system throughput and find several key architectural features which can hide the GALS effects. Fully avoiding any GALS performance penalties is possible for the described GALS chip multiprocessor. To simplify the discussion, in this section both GALS and synchronous systems use the same clock frequencies.

A. Related Work

Significant previous research has studied the GALS uniprocessor—in which portions of each processor are located in separate clock domains. Results have shown GALS uniprocessors experience a non-negligible performance reduction compared to a corresponding synchronous uniprocessor. Fig. 6 shows branch control hazards for synchronous and GALS uniprocessor versions of a simple DLX RISC processor [21] and gives an intuitive explanation for the performance reduction of GALS uniprocessors. The example GALS uniprocessor's pipeline has stages in their own clock domains. During taken

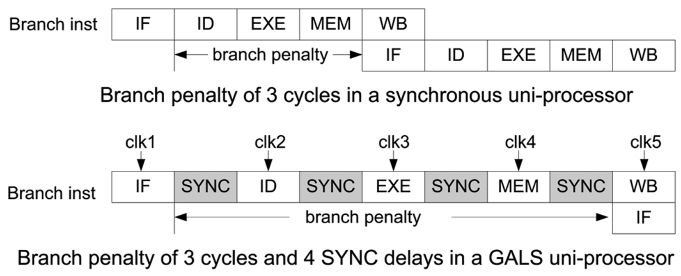

Branch inst
IF
ID
EXE
MEM
WB
branch penalty
IF
ID
EXE
MEM
WB
Branch penalty of 3 cycles in a synchronous uni-processor
clk1
clk2
clk3
clk4
clk5
Branch inst
IF
SYNC
ID
SYNC
EXE
SYNC
MEM
SYNC
WB
branch penalty
IF
Branch penalty of 3 cycles and 4 SYNC delays in a GALS uni-processor

Fig. 6. Pipeline control hazard penalties of a five stage synchronous uniprocessor and a five-stage GALS uniprocessor.

branch instructions, the synchronous processor has a three-cycle control hazard, while the GALS system has a $3 + 4 \times \mathrm{SYNC}$ cycle penalty, significantly reducing system performance. Other pipeline hazards generate similar performance impacts. Studies of GALS uniprocessor performance have reported reductions of 10% [22] and 7%–11% [7] compared to fully synchronous designs. Semeraro *et al.* found that the performance penalty of a GALS uniprocessor can be as low as 4% [5] by dynamically adjusting the synchronization logic to minimize the synchronization delay, and this performance degradation can be further reduced to 2% [23] by adding out-of-order superscalar execution features. However, minimizing the synchronization delay is generally not easy to implement, and fully avoiding the GALS performance penalty is still not achievable using this approach.

Other related work includes the performance analysis reported by Smith [8] for a GALS chip multiprocessor with a shared memory and a global shared bus. With a shared global bus, processors experience additional latency and therefore a performance penalty compared to the equivalent synchronous design when making transactions across the bus to the shared memory. The reported performance penalty was between 1% and 33% with an average of 6.5% for simulated cache block read and write traffic loads between 20% and 80% across the global bus.

B. Comparison of Application Performance of CMPs: GALS Versus Synchronous

A cycle-accurate verilog register transfer level (RTL) description of the previously described GALS multiprocessor chip is used to measure application performance. The RTL model exactly matches the fabricated chip and therefore exactly models its performance. However, although the fabricated chips and the RTL model operate exactly the same *logically*, different processors on the same or different dies in general have different performance and power characteristics due to process, voltage, and temperature variations. Since the focus of this paper is on the differences between GALS and synchronous systems, the results presented in this work are based on the RTL model unless specified otherwise. In addition, this choice removes data skew inherent to our particular collection of fabricated chips, and enables precise and repeatable measurements.

In our comparisons, the synchronous system uses a single global clock and has no synchronization registers in its communication clock boundaries. The GALS system uses a local

TABLE I
AVERAGE STEADY-STATE NUMBER OF CLOCK CYCLES TO COMPLETE THE INDICATED APPLICATIONS (APPLICATION'S FIRST IN → LAST IN, OR APPLICATION'S FIRST OUT → LAST OUT = 1/*THROUGHPUT*) MAPPED ONTO A SYNCHRONOUS ARRAY PROCESSOR AND A GALS ARRAY PROCESSOR, USING 32-WORD INTER-PROCESSOR FIFOS

	Synch. array (cycles)	GALS array (cycles)	GALS perf. reduction (cycles)	GALS perf. reduction (%)
8-point DCT	41	41	0	0%
8×8 DCT	498	505	7	1.4%
Zig-zag	168	168	0	0%
Merge sort	254	254	0	0%
Bubble sort	444	444	0	0%
Matrix multiplication	817.5	819	1.5	0.1%
64-point complex FFT	11,439	11,710	271	2.3%
JPEG encoder	1439	1443	4	0.3%
802.11a/g tx packet	69,700	69,971	271	0.3%

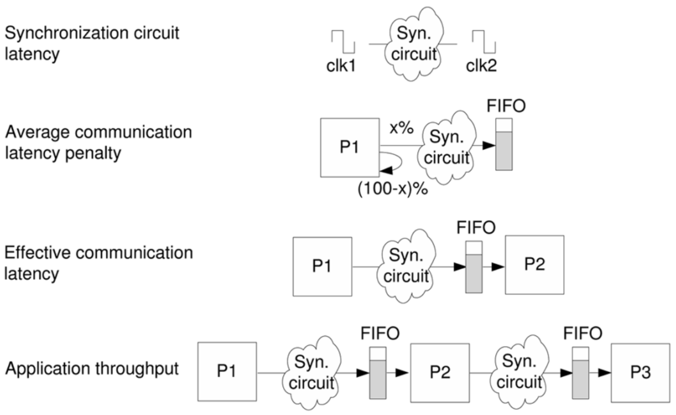


Fig. 7. Illustrations of three key latencies and application throughput in GALS multiprocessor systems.

oscillator and two synchronization registers across each clock boundary.

Data in Table I report throughput (actually 1/*throughput* in terms of clock cycles to complete the given application) for the applications mentioned in Section III-B. The first two columns show the computation time for these applications on both synchronous and GALS CMPs. The third and fourth columns list the absolute and relative performance penalty of the GALS chip multiprocessor. The performance of the GALS system is nearly the same as the synchronous system with an average of less than 1% performance reduction, which is much smaller than the 10% performance reduction of the GALS uniprocessor [22], [7], or the 5% performance reduction of the GALS chip multiprocessor with a shared memory and global bus [8].

C. Analysis of the Performance Effects of GALS

The very small performance reduction of the GALS chip multiprocessor motivates us to understand the factors that affect performance in GALS style processors. Fig. 7 shows the chain of events that allow synchronization circuit latency to finally affect application throughput. It is a complex relationship and several methods are available to hide the GALS penalties. We now take a closer look at the four timing metrics illustrated in Fig. 7.

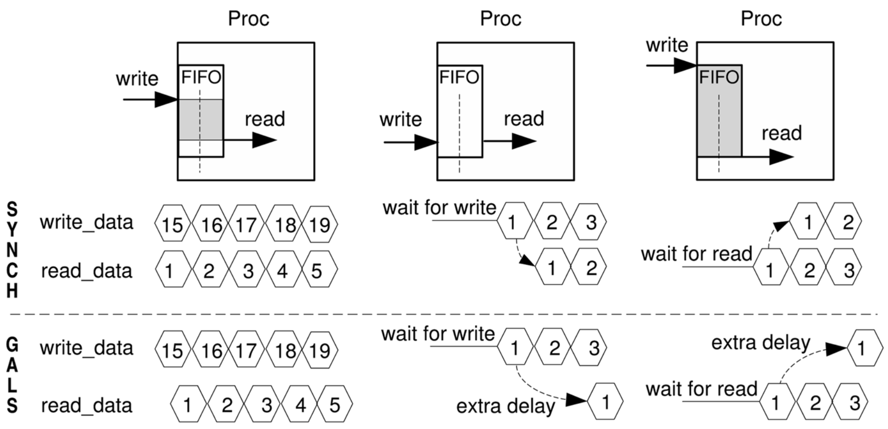


Fig. 8. FIFO operation when the FIFO is: (a) neither full nor empty; (b) empty; and (c) full. Correct operation requires delaying reads when the FIFO is empty, and delaying writes when the FIFO is full. Full speed operation is permitted when the FIFO is partially full.

TABLE II
FRACTION OF THE TIME THE INTER-PROCESSOR COMMUNICATION IS ACTIVE FOR EACH PROCESSOR EXECUTING SEVERAL APPLICATIONS; P1–P9 REPRESENT PROCESSORS 1–9, RESPECTIVELY

	P1	P2	P3	P4	P5	P6	P7	P8	P9	Ave.
8-pt DCT	0.19	0.19	–	–	–	–	–	–	–	0.19
8×8 DCT	0.12	0.12	0.12	0.12	–	–	–	–	–	0.12
Zig-zag	0.57	0.38	–	–	–	–	–	–	–	0.47
Merge sort	0.06	0.02	0.05	0.03	0.03	0.06	0.03	0.03	–	0.04
Bubble sort	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	–	0.03
Matrix mult.	0.11	0.18	0.26	0.31	0.33	0.03	–	–	–	0.20
64-pt FFT	0.01	0.02	0.03	0.02	0.01	0.02	0.03	0.02	–	0.02
JPEG encod.	0.04	0.04	0.04	0.07	0.04	0.05	0.003	0.18	0.06	0.06

- *Synchronization circuit latency* is inherent in every asynchronous boundary crossing due to mismatch in clock phases and overhead of synchronization circuits.
- *Average communication latency penalty* takes into account the fact that the synchronization circuit path is normally not active every cycle. Thus, the synchronization latency should be weighted by the fraction of the time the path is active ($x\%$ in Fig. 7), which is the average communication latency.
- *Effective communication latency* takes into account cases where the downstream processor does not consume data immediately after it is received. In other words, the average communication latency has its impact only when a program is waiting for these delayed data, and the impact is zero otherwise.
- *Application throughput* is the metric of highest importance. The effective communication latency impacts application throughput only when there is communication feedback. As more fully shown in Section IV-C3, one-way communication does not result in throughput penalties under certain conditions.

The following three subsections discuss in further detail each step from the synchronization circuit latency to the application throughput, and show methods to hide the GALS performance penalties.

1) Hiding Synchronization Circuit Latency by Localizing Computation: An obvious but nonetheless noteworthy point is that the asynchronous boundary circuit affects performance only when signals cross it, so the effect from this circuit latency can be dramatically decreased if data communication across the asynchronous boundary is reduced. This may initially sound like a difficult goal, but in fact key parameters such as clock domain granularity and application partitioning can easily affect this coefficient by orders of magnitude.

In many GALS systems the asynchronous boundaries have frequent traffic. For example, in a GALS uniprocessor each instruction crosses several asynchronous boundaries while flowing through the pipeline. But in a GALS chip multiprocessor, computation is much more localized in each processor's clock domain and communication through the asynchronous boundaries is less frequent and thus the GALS effect is much lower.

Table II shows the fraction of the time the inter-processor communication path is active for several applications. Inter-processor communication is often infrequent, especially for complex applications such as the 64-point complex FFT and JPEG encoder that show average inter-processor communication probabilities of only 2% and 6%, respectively. Cross-clock domain communication is equivalent to inter-processor communication for the architecture shown in Fig. 1(b) and is even lower for the architecture shown in Fig. 1(c). In comparison, the example GALS uniprocessor shown in Fig. 6 has a cross-clock domain communication probability of 100% since every instruction requires the crossing of (multiple) clock domain boundaries.

2) Hiding Average Communication Latency by FIFO Buffering: Not every data word transferred across an asynchronous boundary results in effective latency. As Fig. 8(a) shows, the GALS system has the same data flow as the synchronous system when the FIFO is neither empty nor full. The arbitrary phase of the read domain clock is shown skewed with respect to the write domain clock for illustration purposes only and does not result in any performance reductions.

On the other hand, FIFO stalls caused by full and empty conditions do introduce extra latency in GALS systems, although to different extents. A *FIFO empty stall* occurs when a processor reads an empty FIFO and must wait (stall) until data is available, as illustrated in Fig. 8(b). When this type of stall occurs, GALS

TABLE III
AVERAGE STEADY-STATE NUMBER OF CLOCK CYCLES TO COMPLETE THE INDICATED APPLICATIONS (APPLICATION'S FIRST IN → FIRST OUT, OR APPLICATION'S LAST IN → LAST OUT = Latency) MAPPED ONTO A SYNCHRONOUS ARRAY PROCESSOR AND A GALS ARRAY PROCESSOR, USING 32-WORD INTER-PROCESSOR FIFOS

	Synchronous array latency (cycles)	GALS array latency (cycles)	GALS latency increment (cycles)	GALS latency increment (%)
8-point DCT	78	82	4	5.0%
8×8 DCT	996	1003	7	0.7%
Zig-zag	221	224	3	1.3%
Merge sort	542	552	10	1.8%
Bubble sort	1966	1984	18	0.9%
Matrix multiplication	1705	1720	15	0.8%
64-point complex FFT	26,741	27,319	578	2.1%
JPEG encoder	2726	2741	15	0.5%
802.11a/g tx packet	115,477	117,275	1798	1.5%

systems will incur extra delay compared to synchronous systems. If the stalled processor is on the application's critical path, this additional delay penalizes system throughput and latency.

A *FIFO full stall* occurs when a processor writes a full FIFO and must wait until there is writable space. This can be viewed as a blocking write operation. As shown in Fig. 8(c), when this type of stall occurs, GALS systems incur extra delay compared to synchronous systems, however, in many cases this delay will not cause a reduction in system throughput and latency, depending on the specific nature of the application. Specifically, a FIFO full stall will not degrade performance if the stalling processor is never on the application's critical path or does not affect a processor that is on the critical path—this is a common situation since by its very definition, the stalling processor has filled up its output buffer and is waiting.

Table III shows the effective latency difference between the GALS chip multiprocessor and the corresponding synchronous system over several applications. This processing time latency is measured as the time difference between when data enters the array and when the corresponding results exit the array. Due to the sufficiently large FIFO buffer of 32 words, there are relatively few resulting FIFO stalls, and the effective latency penalty of the GALS chip multiprocessor is small with an average of less than 2%. Interestingly, this latency penalty is larger than the throughput penalty—which is less than 1% as shown in Table I.

A key point is thus: *latency penalties do not always result in throughput penalties*. This result is further explored in the following subsection.

3) Throughput Penalty Caused by Latency Penalty From Communication Loops: Reading an empty FIFO or writing a full FIFO results in extra computational latency on a word level, but does not always reduce application throughput. Generally speaking, simple *one-way communication* does not affect system throughput, what really matters is *communication loops*—in which two units wait for information from each other.

In a one-way communication path as shown in Fig. 9(a), the system throughput is dependent on the slowest unit and is not related to the communication—assuming communication is not the slowest unit, which is true in many cases. However, throughput is significantly impacted when the communication

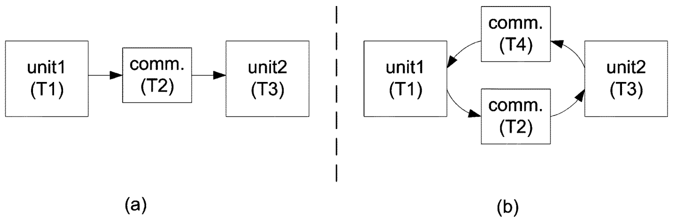


Fig. 9. System throughput in: (a) a one-way communication path and (b) a communication loop path. For a GALS system, we assume *unit1* and *unit2* are in different clock domains and therefore the *comm.* communication delays are significant. For both synchronous and GALS systems, throughput is not reduced with one-way communication (assuming communication time is less than computation time), but is reduced in the loop case.

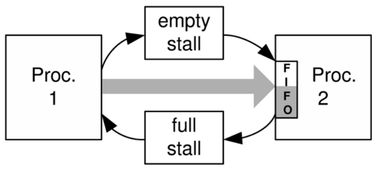


Fig. 10. Common stall communication loop exists when the data producer *proc. 1* and data consumer *proc. 2* are alternatively busy and idle, the FIFO alternates between being empty and full, and processors stall appropriately. The wide arrow is the direction of data transfer, and thin arrows show how FIFO stalls are generated.

has feedback and generates a loop, as shown in Fig. 9(b). If *unit 1* and *unit 2* both need to wait for information from each other, the throughout will be dependent on the sum of unit execution time and communication time. Then the communication time affects the performance of both synchronous and GALS systems, but the GALS system has a larger performance penalty due to its larger communication time.

A similar conclusion can be drawn for GALS uniprocessors. The GALS overhead increases the communication latency between pipeline stages. In instructions without pipeline hazards, the GALS uniprocessor maintains the same throughput as the synchronous uniprocessor although with larger latency, since it has only one-way communication. However, during instructions such as taken branches (where the new PC needs feedback from the execution result), a communication loop is formed, and the GALS uniprocessor therefore incurs a throughput penalty.

In our GALS chip multiprocessor, pure FIFO-full stalls or FIFO-empty stalls alone as in Fig. 8(b) and (c) generate one-way communication and have no effect on system throughput. Fig. 10 shows a not-uncommon FIFO stall communication loop. Sometimes processor 1 is too slow and results in FIFO-empty stalls. Sometimes processor 2 is too slow and results in FIFO-full stalls. Having both FIFO-full stalls and FIFO-empty stalls (obviously at different times) in a link produces a communication loop and this reduces system performance for both synchronous and GALS systems, albeit with less of a penalty for a synchronous system.

Results in Table I show that the GALS chip multiprocessor has nearly the same performance as the synchronous chip multiprocessor. The GALS multiprocessor performance reduction is much less than the GALS uniprocessor's reduction. This implies that the performance penalty sources—communication across asynchronous boundaries, FIFO stalls, and FIFO stall

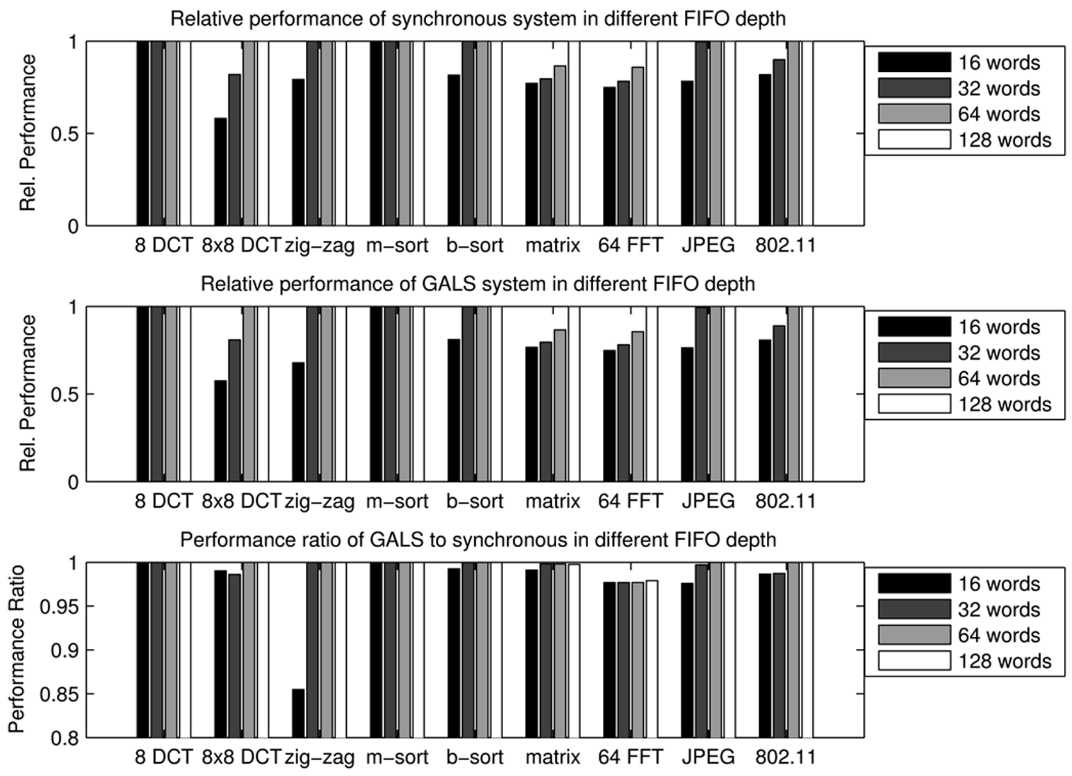


Fig. 11. Performance of synchronous and GALS array processors with different FIFO sizes.

loops—are much less likely to occur in our applications than in the operation of a uniprocessor through pipeline hazards. These results match well with intuition since pipeline hazards are common in most computer programs.

In Table I, the 8-point DCT, zig-zag, mergesort, and bubblesort have no GALS performance penalties since they have only one-way FIFO stalls. The other applications have about 1% performance penalty on average due to FIFO stall loops.

D. Eliminating Performance Penalties

1) Increasing FIFO Sizes: Increasing the FIFO size will reduce FIFO stalls as well as FIFO stall loops, and hence increase system performance and reduce the GALS performance penalty [24]. With a sufficiently large FIFO, there will be no FIFO-full stalls and the number of FIFO-empty stalls can also be greatly reduced; then the communication loop in Fig. 10 will be broken and no GALS performance penalties will result.

The top and middle subplots of Fig. 11 show performance with different FIFO sizes for the synchronous and GALS systems, respectively. We assume all FIFOs in the system are the same size and thus, their sizes are scaled together in this analysis. Whether using a synchronous or GALS style, in general, increasing the FIFO size increases system performance. Also, a threshold FIFO size exists above which the performance does not change. The threshold is the point when the FIFO-full stalls no longer occur due to having a large enough FIFO, and increasing the FIFO size further gives no additional benefit. The threshold is dependent on the application as well as the mapping placement. In our case, the thresholds for the 8×8 DCT and 802.11a/g are 64 words, for JPEG and bubble sort are 32 words, and for the 8-point DCT and merge sort they are less than or equal to 16 words.

The bottom subplot of Fig. 11 shows the performance ratio of the GALS system to the synchronous system. The ratio normally stays at a high level larger than 95%. When increasing the FIFO size, the ratio generally increases due to fewer FIFO stalls and fewer FIFO stall loops. The ratio normally reaches 1.0 at the threshold, which means the FIFO stall loops are all broken and the GALS system has the same performance as the synchronous system. The exception in the examples is the FFT in which the GALS system always has a noticeable performance penalty of approximately 2%. This comes from the multiple-loop communication links and will be explained in the following subsection.

2) Breaking Multiple-Loop Communication Links: Fig. 12 shows a multiple-loop communication link example. In this case, processor 1 and processor 2 send data to each other, and each processor has both FIFO full stalls and FIFO empty stalls. When the FIFO is large enough, there will be no FIFO-full stalls, but FIFO-empty stalls can still occur. So, a communication loop is still possible in the case illustrated in Fig. 12 since FIFO-empty stalls alone can generate a stall loop. For example, several processors in the FFT application shown in Fig. 13 are used as data storage (Memory) coprocessors and they send and receive data to computational (Butterfly) processors [25] thus generating multiple-loop links.

In order to avoid any performance penalties, programmers must avoid these types of multiple-loop implementations. For example, in the FFT case, the *Memory* processor and *Butterfly* processor can be combined into one processor.

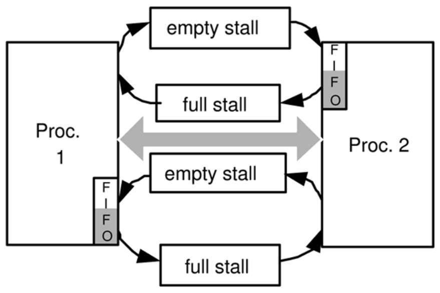


Fig. 12. Example of multiple loop communication links between two processors.

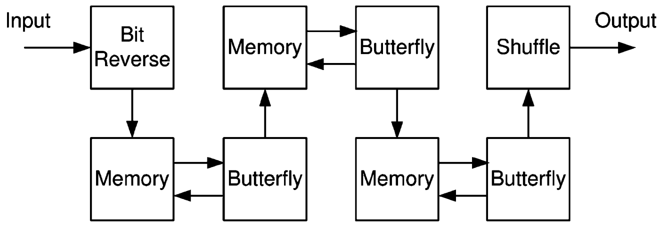


Fig. 13. 64-point complex FFT implementation containing multiple-loop links [25].

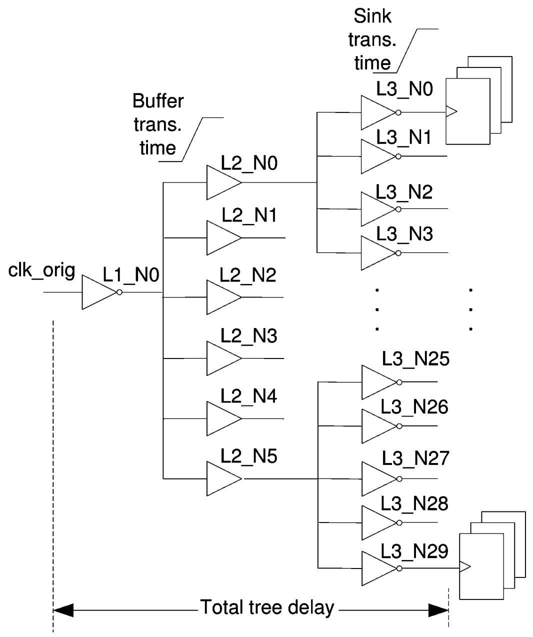


Fig. 14. Example clock tree for a single processor.

TABLE IV
DATA FOR GLOBALLY SYNCHRONOUS CLOCK TREES WITH DIFFERENT NUMBERS OF PROCESSORS IN THE ARRAY

Processor array size	Num of buffer levels	Buffer trans. time (ps)	Sink trans. time (ps)	Total num of buffers	Max. skew (ps)	Total tree delay (ps)
1 × 1	3	120	97	37	47	555
2 × 2	6	120	115	157	49	1081
3 × 3	8	120	117	320	49	1150
4 × 4	10	121	133	633	59	1400
5 × 5	9	177	123	787	70	1700
6 × 6	10	174	143	1097	72	1800
7 × 7	12	170	171	1569	85	1900
8 × 8	14	251	119	1992	92	2100
9 × 9	13	204	119	3012	103	2200
10 × 10	13	228	141	3388	97	2300
11 × 11	15	169	120	3762	116	2450

V. SCALABILITY ANALYSIS OF GALS CMPS

One of the key benefits of a GALS CMP with distributed communication is its scalability: it allows a simple insertion of processors onto a chip to expand a processor array. For a synchronous system, the clock tree must be redesigned for different processor arrays and the design difficulty increases quickly along with the chip size. In addition, the clock skew normally increases due to the more complex clock tree and larger circuit parameter variation effects.

A. *Auto Generated Clock Trees for Different Sizes of Chip Multiprocessors*

Multiple clock trees are generated using Cadence Encounter with an Artisan 0.18-μm standard cell library. An example clock tree for a single processor is shown in Fig. 14 and the first row of Table IV. It uses 37 buffers arranged in 3 levels, has 47 ps worst-case clock skew, approximately 555 ps delay, 120 ps buffer transition time, and 97 ps sink transition time. The *buffer transition time* is the signal rise time at the inserted buffers, the *sink transition time* is the signal rise time at the clocked registers, and the *total tree delay* is the time from clock root to registers. The target constraint parameters for the clock tree include: 50 ps clock skew, 2000 ps delay, and 120 ps buffer and sink transition times.

As the number of processors increases, the synchronous global clock tree becomes more complex and therefore more difficult to design. Table IV lists several key parameters of clock trees for arrays made up of different numbers of processors. In general, all listed parameters tend to increase as the number of processors increases. However, in some cases, the parameters do not increase monotonically. This is due to the complex nature of the clock tree optimization problem which is affected by many discrete parameters (e.g., there can be only an integer number of buffer levels), and also by the time-limited non-optimal Cadence Encounter optimization runs. For example, the 10 × 10 array has lower skew than the 9 × 9 array, but it also has a significantly higher *buffer transition time* and *sink transition time*—which likely come from the increased load of tree buffers since they both have 13 levels of buffers.

Different clock tree design methods would likely have different results from our experiments. Techniques such as full-custom layout and deskewing technologies [26] can certainly generate lower skew results but they dramatically increase design effort, may increase area and power consumption, and are not commonly used. For example, active deskewing and scan-chain adjustments used for Itanium [27] enable a very large clock tree with skew less than 10 ps, but the clock generation and distribution circuits dissipate approximately 25 W of power. Whichever methods are used, the trend that larger chip sizes result in more complex clock trees and larger clock skew is true in general; and furthermore, impacts from circuit parameter variations are increasing with more advanced CMOS fabrication technologies.

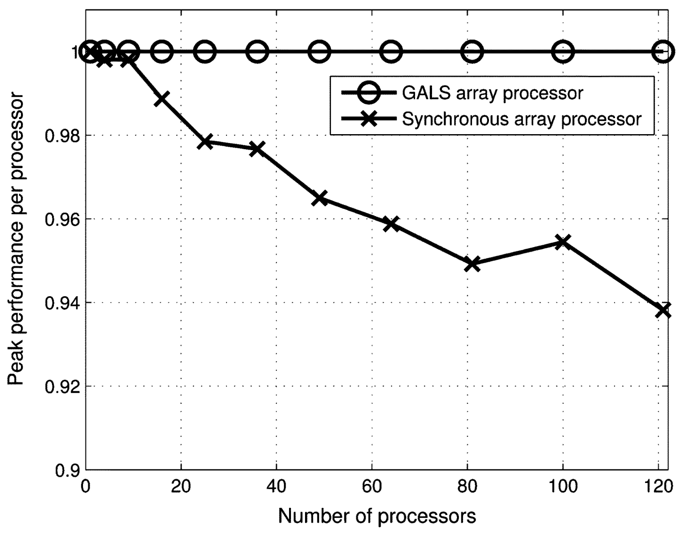


Fig. 15. Considering only static clock skew, the peak performance per processor of GALS processor arrays is constant with the number of processors, and it falls with larger synchronous processor arrays.

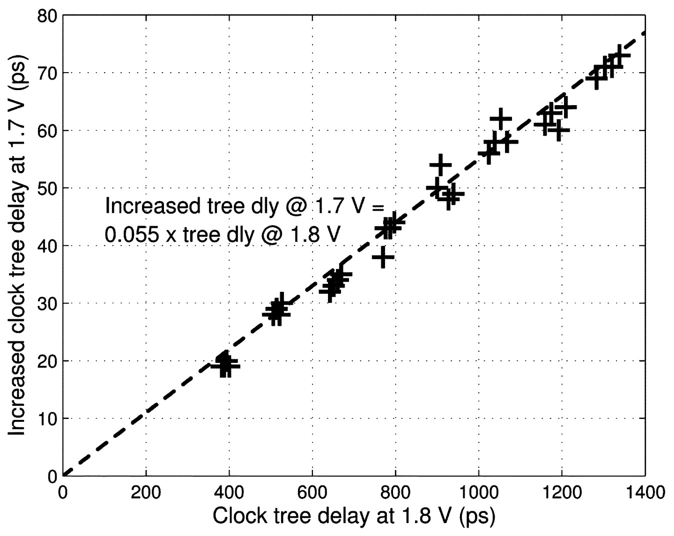


Fig. 16. Increased clock tree delay from reducing supply voltage from 1.8 to 1.7 V for different clock trees with widely varying total delays. Data is from transistor-level spice circuit simulations. Absolute clock skew (in picoseconds) becomes significant for larger clock trees.

B. Effect of Clock Tree on System Performance

The clock period of modern processors expressed in fanout-of-4 (FO4) delays normally ranges between 10 to 20 [28], and is determined by the pipelining of the processor as well as clock distribution factors such as clock skew and jitter. Each FO4 delay in 0.18 μm is about 65 ps. For our analysis, we investigate the effect of clock tree design on system performance assuming the maximum logic delay (including register clock-to-Q time and setup time) within one clock period is 1000 ps, which is 15.4 FO4 delays and is in the typical range for modern high performance processor designs [29]. Since clock skew adds directly to the minimum permissible cycle time, higher levels of clock skew clearly result in lower performance machines. For example, a single hypothetical 1000-ps processor with 47 ps of clock skew could operate with a clock cycle time no faster than 1047 ps.

The relative system peak performance for different processor arrays is shown in Fig. 15. The peak performance of the GALS chip multiprocessor increases exactly linearly with the number of processors since its clock skew is completely independent of the array size due to its lack of a global clock tree—it is 47 ps in this analysis. The synchronous chip multiprocessor scales well when the number of processors is small but by the time it grows to 49 processors (approximately 33 mm^2 in 0.18-μm technology), its performance is 96.5% of the GALS array processor's. Performance continues to degrade with more processors since its clock skew increases along with the chip size. With 121 processors (approximately 80 mm^2 in 0.18-μm technology), the performance of the synchronous array processor is 93.8% of the GALS array processor's.

The globally synchronous clock skew becomes worse when system parameter variations which are not included in the Encounter tool simulation are included. Parameter variations have increasingly impacted system performance along with advancing technologies. The main parameter variations include process variation, voltage variation and temperature variation [2]. These variations affect circuit delay and hence affect clock tree uncertainty. Different clock tree architectures are affected differently by parameter variations. For example, clock trees with fewer inserted buffers [30] and lower fanout loads [31] are less affected by parameter variations.

Supply voltage variation is one of the most important parameter variations. The Encounter CAD tool estimates the peak voltage drop is 0.041 V for our single processor tile due only to the internal power grid. This voltage drop increases with larger chip sizes and a reasonable conservative estimate is 0.1 V of voltage drop, especially if dynamic workload changes are included. Therefore, we investigate operation when the supply voltage is reduced from 1.8 to 1.7 V in a 0.18-μm technology with a nominal supply voltage of 1.8 V. Fig. 16 shows the increased clock tree delay at the reduced supply voltage for different clock trees by varying the number of clock tree stages from 3 to 10, and fanouts of each stage from 4 to 7. These values are typical for the experiments detailed in Table IV. As shown in Fig. 16, the increased clock delay is around 5.5% of the original clock tree delay over a wide range of designs. It is then reasonable to estimate that voltage variation (really only voltage drop in this analysis) increases the clock skew by 5.5% of the clock delay.

However, as mentioned previously, process and temperature variations also contribute significantly to performance degradation. Hashimoto *et al.* [31] show that three primary variation sources: voltage variation, transistor length variation, and temperature variation, contribute approximately 40%, 40%, and 20%, respectively to the overall timing variations in an oscillator's clock period. Therefore, using the voltage variation estimate of 5.5%, we estimate all parameter variations cause a clock skew of $5.5\%/0.40 = 13.75\% \approx 13\%$ of the clock delay. Fig. 17 shows the performance reductions of GALS and synchronous CMPs due to these three parameter variations. With 49 processors, the performance of the synchronous multiprocessor is 0.786/0.95 = 82.7% of the GALS multiprocessor's

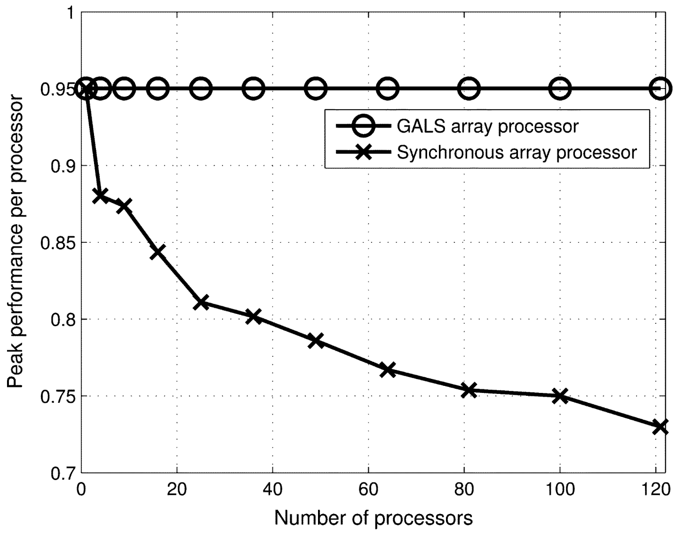


Fig. 17. Peak performance per processor of GALS and synchronous array processors where the clock period calculation includes: (1) a 15 FO4 logic delay; (2) static clock skew; and (3) clock skew equal to 13% of the total clock tree delay due to variations as shown in Fig. 16.

performance. The gap becomes larger as the number of processors increases. With 121 processors, the performance of the synchronous multiprocessor is 0.730/0.95 = 76.8% of the GALS multiprocessor's performance.

VI. Energy Efficiency Analysis Of Clock Frequency Scaling

The GALS clocking style provides opportunities for using clock and voltage scaling to significantly increase energy efficiencies. The computational load in a GALS chip multiprocessor can be heavily unbalanced which increases the potential reduction in energy dissipation when using clock and voltage scaling for each processor. In fact, it is possible to scale down the clock frequencies for some processors without reducing system performance whatsoever.

Clock frequency and supply voltage may be fixed during execution, and this approach is called *static clock/voltage scaling*. This is the method addressed in this paper for the GALS chip multiprocessor. In order to obtain greater power savings, *dynamic clock/voltage scaling* can be used where the clock frequency and supply voltage are changed during runtime to potentially achieve further reduced power dissipation and performance degradation.

A. Related Work—Clock Scaling in the GALS Uniprocessor

Fig. 18 shows an example GALS uniprocessor implementation which increases energy efficiency by reducing the clock frequency of modules that are less heavily used. In the figure, the frequency of the MEM module clock, *clk4*, is reduced when executing the first code block since it has few MEM instructions. The frequency of the WB module's clock, *clk5*, can be reduced when executing the second code block since it has few WB instructions. Unfortunately, reducing the clock of some modules in GALS uniprocessors reduces system performance. The static scaling method reduces energy dissipation by approximately

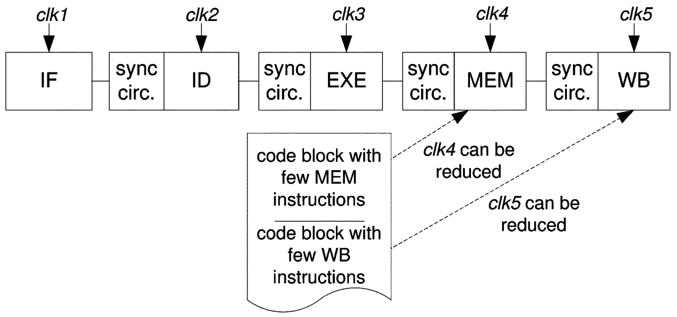


Fig. 18. Clock scaling in a GALS uniprocessor.

16% with an approximately 18% reduction in performance [22]. The dynamic scaling method achieves 20%–25% energy savings along with a 10%–15% performance reduction [5], [7], [6].

B. Unbalanced Processor Computational Loads in GALS Chip Multiprocessors Increase Their Power Savings Potential

Traditional parallel programming methods normally seek to balance computational loads in different processors. On the other hand, when using processors able to change their clock frequencies, unbalanced computational loads are less of a problem, and in fact give more opportunities to reduce the clock frequency and supply voltage of some processors to achieve further power savings without degrading system performance. Releasing the constraint of a balanced computational load enables the designer to explore wider variations in other parameters such as program size, local data memory size, and communication methods.

Fig. 19 shows the unbalanced computational load among processors when mapping our applications onto a chip multiprocessor. The computational load difference for different processors in complex applications such as the JPEG encoder and the 802.11a/g transmitter can be more than 10 times.

C. Finding the Optimal Clock Frequency—Computational Load and Position

When using clock/voltage scaling, the system performance will normally be reduced. We seek to scale the clock frequencies of some processors in the GALS multiprocessor while still maintaining the same performance. The optimal clock frequency for each processor depends strongly on its computational load, and also depends on its position and relationship with respect to other processors.

Fig. 20 shows the system throughput versus the clock frequencies of four processors in the 8×8 DCT, whose implementation is shown in Fig. 4. The computational loads of the four processors are 408, 204, 408, and 204 clock cycles per 8×8 DCT, respectively. The throughput changes with the scaling of the second and fourth processor much more slowly than the scaling of the first and third processors. This illustrates the intuitive point that a processor with a light computational load is more likely to maintain its performance with a reduced clock frequency. Somewhat counter-intuitively however, the second and fourth processors have the same light computational load, but the throughput changes with the fourth processor scaling

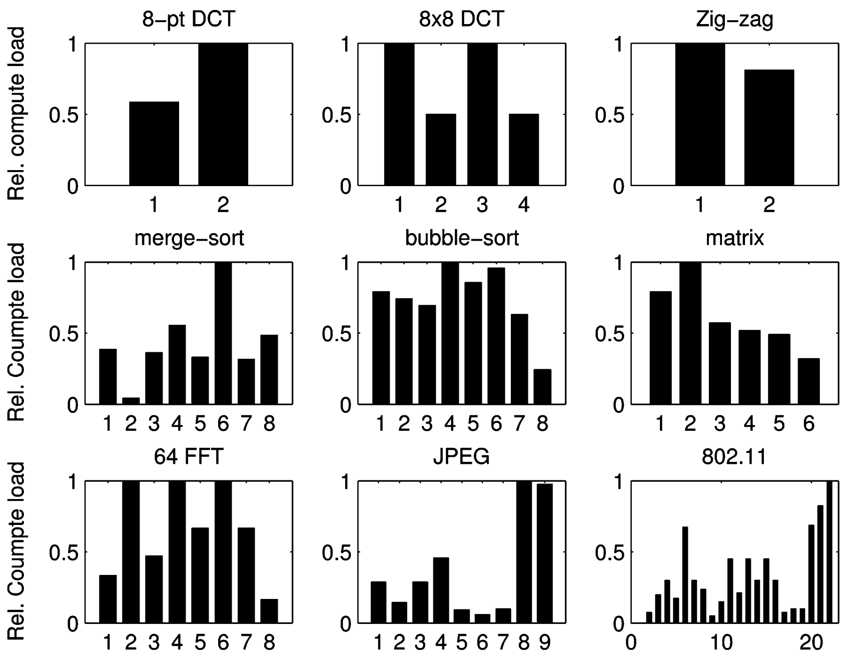


Fig. 19. Relative computational loads of different processors in nine applications illustrating unbalanced loads.

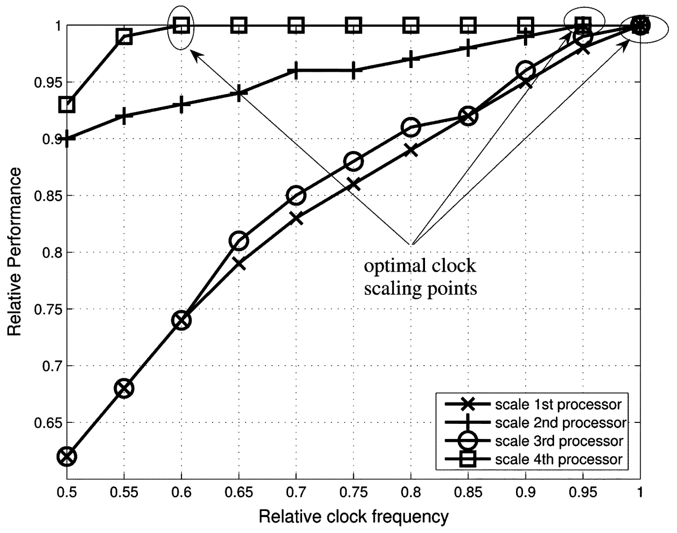


Fig. 20. Throughput changes with statically configured processor clocks for the 4-processor 8×8 DCT application.

much more slowly than the second processor's scaling. Minimal power consumption is achieved with full throughput when the relative clock frequencies are 100%, 95%, 100%, and 57% of full speed, respectively.

The reason for the different behavior of the second and fourth processors comes from their different positions and FIFO stall characteristics as shown in Fig. 21. The second processor has FIFO-empty stalls when it fetches data too quickly from the first processor, and it has FIFO-full stalls when it sends data too quickly to the third processor. The fourth processor has only FIFO-empty stalls.

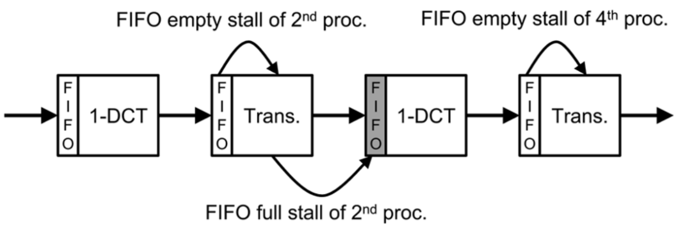

FIFO empty stall of 2nd proc.
FIFO empty stall of 4th proc.
FIFO
1-DCT
Trans.
FIFO full stall of 2nd proc.

Fig. 21. Relationship of processors in the 4-processor 8 × 8 DCT application illustrating the differing stalling possibilities for the second and fourth processors.

D. Power Reduction Due To Clock and Voltage Scaling

Reducing the clock frequency allows for a reduction in voltage to obtain further power savings. The relationship between clock frequency, voltage and power has become much more complex in the deep submicrometer regime because of other parameters such as leakage power. Therefore, we use frequency-voltage-power data measured from a 0.18-μm processor [32] to estimate power consumption from a given clock frequency. Fig. 22 shows the relationship between clock frequency and its corresponding power consumption due to clock/voltage scaling for the processor.

To find an estimate of the optimal clock frequency for each processor, clock frequencies in processors were reduced in 1% increments starting from full rate until the overall system performance was reduced. We used these lowest frequency values which did not impact system performance in our subsequent power reduction estimates. These clock reductions are "free" in the sense that they do not come with any performance drawbacks, and further reductions in frequency would clearly bring greater power reductions, albeit with some performance loss.

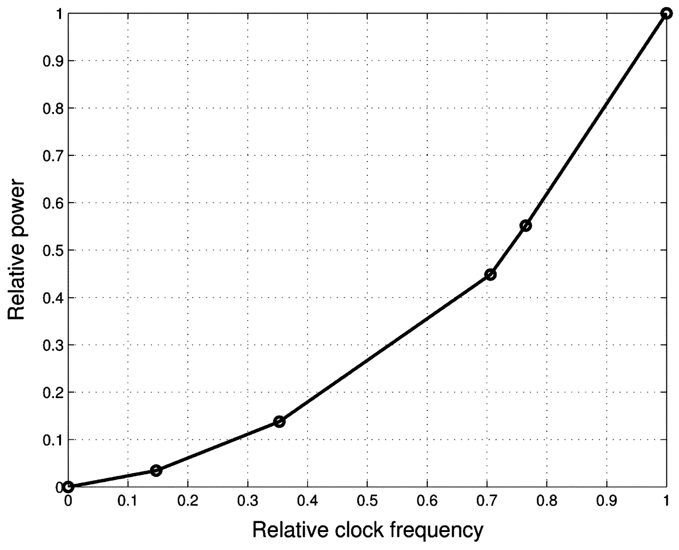


Fig. 22. Relationship between a processor's power consumption with a varying clock frequency when the supply voltage is the minimum possible voltage for that clock speed [32].

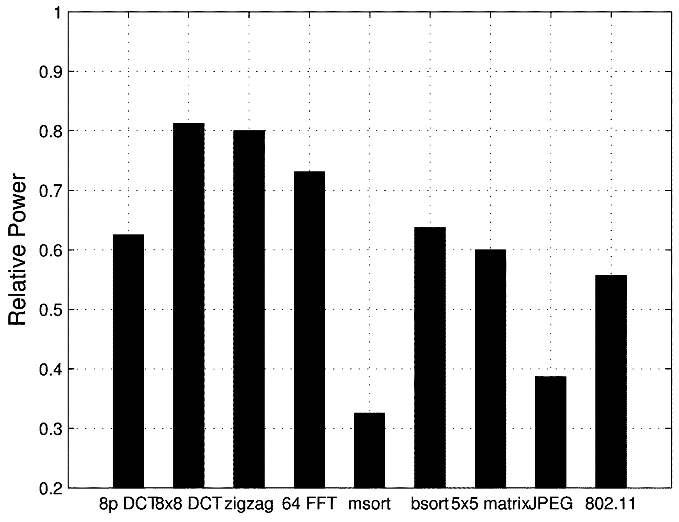


Fig. 23. Relative power over several applications for the GALS array processor with static clock and supply voltage scaling compared to a synchronous array processor. The clock and supply voltage scaling is done without any reduction in performance.

Using the previously mentioned frequency-power model, we estimate the relative power consumption of the GALS multiprocessor compared to the synchronous multiprocessor using static clock frequency and supply voltage scaling for several applications. Results are shown in Fig. 23. The GALS system achieves an average power savings of approximately 40% without affecting performance. Since it is difficult to obtain exact optimal clock frequencies and supply voltages in real implementations, these results should be viewed as best case estimates. Nevertheless, these power savings are significantly higher than GALS uniprocessor results which are reported to save approximately 25% energy when operating with a performance reduction of more than 10% [22], [5], [7].

VII. Conclusion

We show that the application throughput reduction of the GALS style comes from asynchronous boundary communication and communication loops, and that it is possible to design GALS multiprocessors without this performance penalty. A key advantage of the GALS chip multiprocessor with distributed interconnect compared to the other GALS systems is that asynchronous boundary communication and communication loops occur far less frequently and therefore the performance penalty is significantly lower. The proposed GALS array processor has a throughput penalty of less than 1% over a variety of DSP workloads, and this small penalty can be further reduced by sufficiently large FIFOs (dependent on the application) and programming without multiple-loop communication links.

Local clock oscillators in GALS multiprocessors simplify the clock tree design and enable nearly perfect system scalability. More processors can be placed onto a chip without any clock tree redesign. As the number of processors increases, clock skew in the synchronous multiprocessor system increases quickly due to more complex clock trees and process, voltage, and temperature variations. With 121 processors (approximately 80 mm^2 in 0.18-μm technology), the peak performance of the GALS multiprocessor can be more than 20% greater than the corresponding synchronous multiprocessor.

Unbalanced computational loads in CMPs increase the opportunity for independent clock frequency and voltage scaling to achieve significant power savings. The GALS chip multiprocessor can achieve approximately 40% power savings *without any reduction in performance* over a variety of DSP and embedded workloads using *static* clock and voltage scaling for each processor. These results compare well with a reported 25% energy reduction and 10% performance reduction of GALS uniprocessors for a variety of general purpose applications.

Data presented in this paper are based on simulations of a fully-functional fabricated GALS chip multiprocessor [13] and physical designs based on the chip. Results from this work apply to systems with three key features as discussed in Section II, namely: 1) multi-core processors (homogeneous and heterogeneous) operating in independent clock domains; 2) source synchronous multi-word flow control for asynchronous boundary communication; and 3) distributed interconnect, such as a mesh. While results certainly vary over different applications and specific architectures, systems with these features should still exhibit the following benefits over many workloads: good scalability, small performance reductions due to asynchronous communication overhead, and large potential power reductions from clock frequency and supply voltage scaling.

Acknowledgment

The authors would like to thank E. Work, T. Mohsenin, and other VCL colleagues; R. Krishnamurthy, M. Anders, S. Mathew, and Y. P. Cheng.

References

[1] S. Naffziger, B. Stackhouse, and T. Grutkowski, "The implementation of a 2-core multi-threaded Itanium family processor," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2005, pp. 182–183.

[2] S. Borkar, T. Kainik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. IEEE Int. Conf. Des. Autom.*, Jun. 2003, pp. 338–342.

[3] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, Oct. 1984.

[4] T. Meincke, A. Hemani, S. Kumar, P. Ellervee, J. Oberg, T. Olsson, and P. Nilsson, "Globally asynchronous locally synchronous architecture for large high-performance ASICs," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1999, pp. 512–515.

[5] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proc. IEEE Int. Symp. High-Perform. Comput. Arch.*, Feb. 2002, pp. 29–40.

[6] E. Talpes and D. Marculescu, "Toward a multiple clock/voltage island design style for power-aware processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 5, pp. 591–603, May 2005.

[7] E. Talpes and D. Marculescu, "A critical analysis of application-adaptive multiple clock processor," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2003, pp. 278–281.

[8] S. F. Smith, "Performance of a GALS single-chip multiprocessor," in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl. (PDPTA)*, Jun. 2004, pp. 449–454.

[9] A. Upadhyay, S. R. Hasan, and M. Nekili, "Optimal partitioning of globally asynchronous locally synchronous processor arrays," in *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, Apr. 2004, pp. 26–28.

[10] R. W. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, and B. Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 10, pp. 1125–1134, Oct. 2007.

[11] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[12] T. Chelcea and S. M. Nowick, "A low-latency FIFO for mixed-clock systems," in *Proc. IEE Comput. Soc. Ann. Workshop VLSI (WVLSI)*, Apr. 2000, pp. 119–126.

[13] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas, "An asynchronous array of simple processors for DSP applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2006, pp. 428–429.

[14] H. Zhang, , V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1697–1704, Nov. 2000.

[15] D. Lattard *et al.*, "A telecom baseband circuit based on an asynchronous network-on-chip," in *Proc. ISSCC*, Feb. 2007, pp. 258–259.

[16] A. M. Jones and M. Butts, "TeraOPS hardware: A new massively-parallel MIMD computing fabric IC," in *Proc. Hotchips*, Aug. 2006, Session 5.

[17] J. Oliver, R. Rao, P. Sultatna, J. Crandall, E. Czernikowski, L. W. Jones, D. Franklin, V. Akella, and F. T. Chong, "Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor," in *Proc. Int. Symp. Comput. Arch.*, Jun. 2004, pp. 150–161.

[18] S. Vangal *et al.*, "An 80-tile 1.28 TFLOPS network-on-chip in 65 nm CMOS," in *Proc. ISSCC*, Feb. 2007, pp. 98–99.

[19] M. Meeuwsen, O. Sattari, and B. Baas, "A full-rate software implementation of an IEEE 802.1 la compliant digital baseband transmitter," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 2004, pp. 297–301.

[20] K. K. Parhi, *VLSI Digital Signal Processing Systems*. New York: Wiley, 1999.

[21] D. A. Patterson and J. L. Hennessy, *Computer Architecture—A Quantitative Approach*, 2nd ed. San Mateo, CA: Morgan Kaufmann, 1999.

[22] A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors," in *Proc. Int. Symp. Comput. Arch.*, May 2002, pp. 158–168.

[23] G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, S. G. Dropsho, and S. Dwarkadas, "Hiding synchronization delays in a GALS processor microarchitecture," in *Proc. Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Apr. 2004, pp. 159–169.

[24] Z. Yu and B. Baas, "Performance and power analysis of globally asynchronous locally synchronous multi-processor systems," in *Proc. IEEE Comput. Soc. Ann. Symp. VLSI*, Mar. 2006, pp. 378–384.

[25] O. Sattari, "Fast Fourier transform on a distributed digital signal processor," M.S. thesis, Elect. Comput. Eng. Dept., UC Davis, Davis, CA, 2004.

[26] C. E. Dike, N. A. Kurd, P. Patra, and J. Barkatullah, "A design for digital, dynamic clock deskew," in *Proc. Symp. VLSI Circuits*, Jun. 2003, pp. 21–24.

[27] P. Mahoney, E. Fetzer, B. Doyle, and S. Naffziger, "Clock distribution on a dual-core multi-threaded Itanium-family processor," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2005, pp. 292–293.

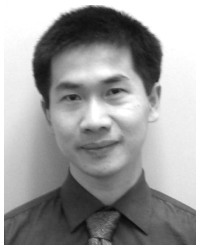
[28] M. Horowitz and W. Dally, "How scaling will change processor architecture," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2004, pp. 132–133.

[29] B. Flachs, S. Asano, S. H. Dhong, P. Hotstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano, "A streaming processing unit for a cell processor," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2005, pp. 134–135.

[30] D. C. Sekar, "Clock trees: Differential or single ended?," in *Proc. Int. Symp. Quality Electron. Des.*, Mar. 2005, pp. 545–553.

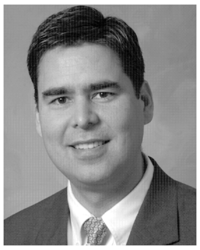
[31] M. Hashimoto, T. Yamamoto, and H. Onodera, "Statistical analysis of clock skew variation in H-tree structure," in *Proc. Int. Symp. Quality Electron. Des.*, Mar. 2005, pp. 402–407.

[32] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.



Zhiyi Yu received the B.S. and M.S. degrees in electrical engineering from Fudan University Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, Davis, in 2007.

He is currently an Associate Professor with the Microelectronics Department, Fudan University, Shanghai, China. He was a Hardware Engineer with IntellaSys Corporation, Cupertino, CA. His research interests include high-performance and energy-efficient digital VLSI design, architectures, and processor interconnects, with an emphasis on many-core processors. He was a key designer of the 36-core Asynchronous Array of simple Processors (AsAP) chip, and one of the designers of the 167-processor second generation computational array chip.



Bevan M. Baas received the B.S. degree in electronic engineering from California Polytechnic State University, San Luis Obispo, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1990 and 1999, respectively.

In 2003, he became an Assistant Professor and in 2008 an Associate Professor with the Department of Electrical and Computer Engineering, University of California, Davis. He leads projects in architecture, hardware, software tools, and applications for VLSI computation with an emphasis on DSP workloads. Recent projects include the asynchronous array of simple processors (AsAP) chip, applications, and tools; low density parity check (LDPC) decoders; FFT processors; viterbi decoders; and H.264 video codecs. From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa Clara, CA, as an early employee and served as a core member of the team which developed the first IEEE 802.11a (54 Mbps, 5 GHz) Wi-Fi wireless LAN solution. During the summer of 2006, he was a Visiting Professor in Intel's Circuit Research Lab.

Dr. Baas was a National Science Foundation Fellow from 1990 to 1993 and a NASA Graduate Student Researcher Fellow from 1993 to 1996. He was a recipient of the National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by AISES in 2006. He is an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS and has served as a member of the Technical Program Committee of the IEEE International Conference on Computer Design (ICCD) in 2004, 2005, 2007, and 2008. He also serves as a member of the Technical Advisory Board of an early stage technology company.