

The producer, however, can save communication overhead by writing the data to the *Write Queue* and continuing with the next instruction without waiting for the data to reach the shared memory. The hardware associated with the *Write Queue* will automatically forward the data to the shared memory.

The consumer can save the overhead by using the *Distributed Shared Memory* in the following way. The large shared address space is partitioned into contiguous segments that can be read and written by any processor. Each segment is physically placed with its corresponding processor. This segment of shared memory, which is a *Distributed Shared Memory*, has dual ports, one of which is connected to the processor while the other is connected to the shared bus; the processor and the bus can access the memory at the same time. A space for the shared data can be allocated in the consumer's memory, since the producer and the consumer is known at the compile time, so that the producer is able to write the data to the consumer's *Distributed Shared Memory*. Then, the consumer can obtain the shared data, if the data has been transferred, whenever it needs without going through the bus, and without paying communication overhead. When there are more than one consumer, the producer can broadcast the data to all the consumers at once using only one bus cycle.

The producer, however, has to pay communication overhead when the *Write Queue* gets full due to bus saturation. We can reduce the bus saturation problem by localizing the communication and by configuring the bus to increase the total communication bandwidth. Another type of communication overhead occurs when the consumer is idling, waiting for the data, which is being transferred. For the medium/coarse grain concurrency, the overhead was very small, as presented in the *Architecture Benchmark* section.

5. SYNCHRONIZATION

In addition to efficient communication provided by the *Write Queues*, the *Distributed Shared Memory* and the *Configurable Bus*, the synchronization mechanism employed is also an important factor in the performance of a multiprocessor system. There are two frequently found synchronization types: mutual exclusive synchronization and barrier synchronization. When several processes try to access a shared resource (bus or memory), the mutual exclusive synchronization guarantees that only one process will get the right to access the shared resource while other processes wait until the resource is available. On the other hand, when processes must be synchronized at a certain point in the program, the barrier synchronization prevents any processes from advancing until every process in that group reaches the point and agrees on synchronization.

The barrier synchronization is useful to ensure the order of execution, thereby preserving the data dependency between processes. In a typical application, the producer produces a piece of data and then issues a barrier synchronization. To maintain data coherence, the corresponding consumer issues a barrier synchronization first and then read the data. When a block of data is produced and consumed, the barrier synchronization is issued only once for the whole block.

In most of the applications we have studied, the barrier synchronization seemed to be the most attractive technique. Therefore, the SMART architecture provides special hardware support for the barrier synchronization; it takes only two instruction cycles to check if all the processors are synchronized. It is also possible to synchronize selectively for a local group of processors. A primitive atomic operation for mutual exclusive synchronization is also provided which is similar to the one available in general purpose multi-processors.

6. ARCHITECTURE BENCHMARK

The intensive study of four DSP algorithms (256-points FFT, Echo Cancellor [2], Pitch Extractor [3], and 64x64 Matrix-Vector Multiplication) are shown to illustrate the key points of the SMART architecture (Table 1 and Table 2).

The four different algorithms were chosen to represent two major aspects of concurrent programming in the SMART architecture. The first aspect is the degree of communication. The algorithms were chosen to represent communication intensive algorithms (FFT), computation intensive algorithms (Echo Cancellation), and in between (Pitch Extraction and Matrix-Vector Multiplication) so that we can understand how the architecture performs for different types and degrees of communication. The second aspect is the type of concurrency exploited: pipelining (FFT), parallelism (Matrix-Vector Multiplication) or both (FFT, Echo Cancellation and Pitch Extraction). However, caution is required in analyzing the results using the above classification, since the degree of communication and the type of concurrency varies depending on the granularity, mapping, programming, problem size and other factors.

The benchmark programs simulated on the 16-processor system showed speedup from 13.29 to 14.58 over the single processor, as summarized in Table 1. Communication overhead, synchronization overhead, and processor idle time were measured to analyze the cause of performance degradation in our multiprocessor system. During the observation period, processors have paid less than 2% of the time for inter-processor communication overhead and much less than 1% of the time for synchronization overhead (the numbers are not included in the table). However, idle time of the processors was from 1.80% to 18.52%; it was especially high for the irregular programs, such as Echo Cancellor and Pitch Extractor, since the load balancing is extremely hard to achieve for those cases.

Table 1. Benchmark Results for 16 Processors

	FFT 256-pts	Echo Cancellor	Pitch Extractor	Matrix-Vector Multiplication
Speedup	13.65	14.04	13.29	14.58
Idle Time %	8.91	18.52	16.12	1.80
Communication Overhead %	0.64	0.13	0.03	1.11
# of Buses	16	3	4	1
Average Bus Usage %	32.67	0.65	4.05	14.33
Average # of Bus Requests	1.00	1.59	1.50	8.62

functions both as a software development station and as a large auxiliary data storage unit. The workstation provides a UNIX environment for cross-compiling and developing application programs.

In the following sections, we will describe the four main features of the SMART architecture: the *Configurable Bus*, the *Write Queues*, the *Distributed Shared Memory* system, and the *Synchronization mechanism*. These features, implemented in special custom circuits, are included to significantly reduce the communication and synchronization overhead.

3. THE CONFIGURABLE BUS

When an algorithm with an irregular communication pattern is mapped onto a multiprocessor system with a regular and fixed processor interconnection, communications can be extremely inefficient. To solve this problem, a configurable shared bus architecture is proposed. The bus can be divided into several independent sections by switches which can be opened or closed dynamically under software control (Figure 2). Processors with local communications are grouped such that they can communicate among themselves independent of activities in other groups, thus increasing the overall bus bandwidth.

When the buses are disconnected to take advantage of local communication, the disconnected buses are linked by the *Bypass Unit* to support global communication. The address and data may be sent through several buses automatically to reach the memory in other bus groups through Bypass Units. The Bypass Unit let a processor read and write to any shared memory on any part of the bus even if they are not directly connected. Thus we can program independently of the bus configuration; the program results are logically the same, although the performance will be affected. From a programmer's point of view, the configurable bus is still a single shared bus supporting shared memory. (for the first prototype, only one direction of the *Bypass* is supported and a *Ring Connection* is not supported.) A *Ring Connection* is formed by connecting one end of the bus to the other end of the bus to

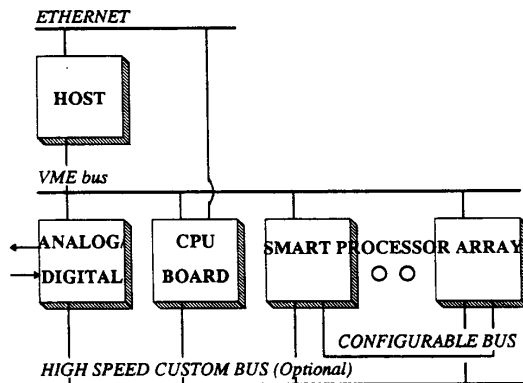
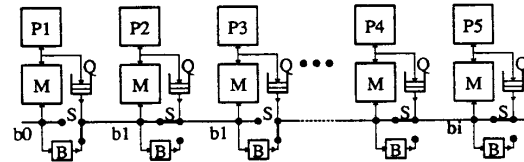


Figure 1. SMART System

reduce the maximal distance between any two processors in the array.

The configurable bus can simulate a single-shared bus (by closing all the switches except one), a one-dimensional systolic array interconnection (by opening all the switches) or an irregular communication pattern which is specific to the application algorithm, as shown in Figure 3.

When a group of processors are actively communicating with each other, the delay of communication between them can be reduced by closing the bus switches between the processors. On the other hand, when they are not frequently communicating with each other, the system throughput can be increased by opening those switches. Thus, the optimal bus configuration can be obtained by trading off between the delay of communication and the bus bandwidth.



Pi: Processor, M: Memory, S: Switch

bi: Bus, B: ByPass, Q: Write Queue

Figure 2. SMART Processor Array

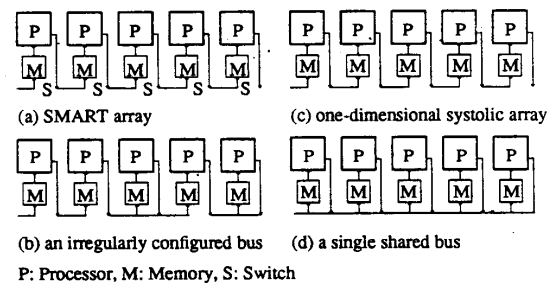


Figure 3. Configurable Bus

4. THE DISTRIBUTED SHARED MEMORY

Memory architecture is as important as the interconnection scheme in reducing the communication overhead. The shared memory system is favored for its ease of programming and its efficient support of interprocessor communication. For every piece of data used for inter-processor communication in the shared memory system, there is at least one processor (producer) which produces it, and one processor (consumer) which consumes it. Both of them pay inter-processor communication overhead by accessing the shared data through the shared bus.

A CONFIGURABLE MULTIPROCESSOR SYSTEM FOR DSP BEHAVIORAL SIMULATION

Wook Koh
Alfred Yeung
Phu Hoang
Jan Rabaez

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

SMART, a multi-processor architecture optimized for real-time behavioral simulation of Digital Signal Processing (DSP) systems, is presented. The first prototype, currently under development, contains 8 processors (peak 160 MFLOPS). The number of processors will be increased to 64 (peak 1.28 GFLOPS) in the future.

The SMART system features a *Configurable Bus* and *Bypass Units* to improve the bandwidth of a single shared bus by taking advantage of the local communication between processors. *Write Queues* and a *Distributed Shared Memory* system are provided to overlap the inter-processor communication with the processor computation, thereby making inter-processor communication delays almost invisible. Barrier synchronization, which is frequently used in DSP algorithms, is supported by hardware to yield low synchronization overhead.

1. INTRODUCTION

DSP applications have become a major component in the rapidly expanding field of Application Specific Integrated Circuits (ASIC). Examples of such applications in this field include digital audio, speech synthesis and recognition, telecommunication, image and video processing and robotics. Due to the high throughput and special input/output requirements, it is often necessary to design special purpose chips, which are optimized for only one single application. Recently, a great deal of attention has been focused on the development of computer aided design environments, which may help to shorten the design time of those dedicated devices.

It has been noted that a major part of the design effort for DSP systems is devoted to the algorithmic verification and specification process. This process, which is necessary to verify the functionality of the algorithm and to optimize the parameters, typically requires the processing of a large amount of data. The process also includes the simulation of the noise and distortion behavior, taking into account the effect of quantization, rounding and truncation. It is only after a careful checking of all those parasitic effects that Application Specific Integrated Circuits (ASIC) can be implemented.

The behavioral simulation of DSP algorithms, however, requires an enormous computational throughput. In this paper, we present a hardware accelerator SMART (an acronym for *Switchable Multi-processor Architecture supporting Real Time applications*), which attempts to increase the simulation speed by at least two orders of magnitude as compared to general purpose computer architectures.

The speedup is achieved by using the following two methods. First, in order to handle the number crunching bottleneck, a high performance DSP processor with both floating point and fixed point instructions is used as the core processing unit (DSP32C from AT&T Bell Labs). The processor executes up to 10 million floating point (single precision IEEE) multiplications and additions per second. In addition, it provides special instruction sets to support DSP applications. This results in a simulation speedup with at least an order of magnitude as compared to general purpose processors.

Second, an additional order of magnitude in simulation speedup can be obtained by exploiting the high degree of concurrency present in most signal processing algorithms. The SMART system should be able to simulate a large variety of DSP algorithms with various degrees and grains of concurrency. Pipelining and Parallelism are two methods used to achieve concurrency.

An initial study on this topic was published in [1]. There have been major changes and developments in the architecture since then. This paper will concentrate on only the multiprocessor architecture issues.

2. SMART SYSTEM OVERVIEW

The major components of the system are the SMART processor array, the CPU Board, the Analog/Digital Unit, the Interface Unit, and the *host system*, as depicted in Figure 1.

Computation is performed by the *SMART processor array* which consists of a set of programmable core processors (AT&T's DSP32C) with 32-bit floating-point capability (Peak 20 MFLOPS) connected to each other via a configurable shared bus (Figure 2). The array is controlled by the *CPU Board*, a micro-computer running a real-time Unix-like Operating System. Data is supplied to and received from the processor array in real-time through the *Analog/Digital Unit*. The *Interface Unit*, consisted of Ethernet, VME bus, and RS232, provides for efficient communication and cooperation between the Units. The *host* is a SUN workstation system that

The bus was configured for each algorithm, from 1 bus to 16 buses, to obtain the best results; the FFT program, when configured differently as a single-bus, showed 2.40 times of performance degradation due to the shortage of bus bandwidth. For each bus, the *bus usage %*, defined as the fraction of the observation period in which the bus was used, is measured to represent the degree of inter-processor communication through the bus. The *average bus usage %* (Table 1 and Table 2) is an average of the *bus use %* over all the buses.

When many processors request the bus at the same time, the communication can be temporarily congested, which can cause a bus saturation for a short period of time. The degree of congestion is measured by the number of processors requesting (arbitrating) the bus when at least one is requesting. The *Matrix-Vector Multiplication* benchmark shows that an average of 8.62 processors, among 16 processors, requested the bus at the same time, although the bus is used only 14.33 % of the time.

The same set of benchmarks, simulated on 64 processors, showed speedup from 25.94 to 53.89. The configured bus, in many cases, has performed a factor of 2 to 3 times better than fixed bus interconnections (a single-shared-bus and a pipelined bus). In general, the configurable bus becomes more and more important as more processors are used and as the programs become more irregular or more communication bounded.

Although the SMART architecture has successfully reduced the overhead of communication and synchronization to a low value, the idle time accounts for most of the performance degradation. More speedup improvements can be expected (by 10% - 20%) with more careful load balancing of application programs.

7. SOFTWARE SUPPORT ENVIRONMENT

A multi-processor simulation environment is only useful when high level support for the algorithm specification and the processor partitioning is available. The designer should be able to describe the algorithms using a high level language. We have selected SILAGE [4] and C as our target languages. SILAGE is a data-flow type language developed for the specification of DSP algorithms. Being an applicative language, SILAGE is ideally suited for the description of algorithms with inherent concurrency. It does not require the explicit expression of parallelism in the program. On the other hand, the C programming language is supported because of its popularity in the programming world. In the current state, a programmer writes one program for each processor in either C or DSP32C assembly language.

A high level software system is under development to map the algorithmic description onto the multi-processor architecture with a balanced loading and an optimal usage of the communication system. Since the communication mechanisms of the SMART architecture are configurable, the partitioning and mapping of processes to processors can be achieved more easily than in the case of a general purpose architecture.

Table 2. Benchmark Results for 64 Processors

	FFT 256-pt	Echo Canceller	Pitch Extractor	Matrix-Vector Multiplication
Speedup	25.94	41.80	53.89	34.95
Idle Time %	23.10	28.81	13.68	17.19
Communication Overhead %	33.39	0.73	1.11	15.45
# of Buses	64	10	12	1
Average Bus Usage %	60.07	2.96	30.90	34.17
Average # of Bus Requests	1.00	2.20	2.57	32.74

8. CONCLUSION

A major part of the design effort for DSP systems is devoted to the algorithmic specification and verification process. The execution of those simulations on a general purpose computer tends to consume an enormous amount of CPU-time. To overcome the limit in the high throughput requirement, the SMART multi-processor architecture features a software configurable communication pattern, low communication and synchronization overhead and fast floating point DSP core processors to speedup the program-compile-simulate process for a large range of DSP applications.

The benchmark analysis shows that the SMART system gets speedup close to the number of processors (16 and 64 processors) even when problems are communication bounded, irregular and in different classes of concurrency and communication pattern.

A prototype of the system with 8 processors (160 MFLOPS) has been designed and is being implemented; the 64 processor system (1.28 GFLOPS) will be built in the future. Research on the software side to automate the partitioning and mapping of the program to the multiprocessors will ease the job of programming.

REFERENCES

- [1] W. Koh, A. Yeung, P. Hoang, J. Rabaey, "A MULTI-PROCESSOR SYSTEM FOR DSP BEHAVIORAL SIMULATION," *IEEE Workshop on VLSI Signal Processing, III*, 1988.
- [2] B. MARC, et. al., "Ontwikkeling Van Een Digitale Echo-canceller Chip Voor Toepassing In Een Handenvrij Telefoonstelsel," *Katholieke Universiteit Leuven*, 1986/1987.
- [3] R. J. Sluyter et. al, "A Novel Method for Pitch Extraction from Speech and a Hardware Model Applicable to Vocoder Systems," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1980, pp. 45-48.
- [4] P. Hilfinger, "SILAGE, A High Level Language and Silicon Compiler for Digital Signal Processing," *Proceedings IEEE CICC Conference 1985*, Portland, May 1985.