RECONFIGURABLE FRAME WORK FOR CHIPMULTIPROCESSORS AND ITS APPLICATION IN MULTITHREADED ENVIRONMENT

Srivatsa S.K.[1], **Ravi Kumar Ch.**[2]

Prof. (rtd), Anna University, Chennai
Department of ECE, Prakasam Engineering College, Kandukur
Email: ravi_ece99@yahoo.com

ABSTRACT

Chip-multiprocessors are quickly gaining momentum and becoming popular in all areas of computational tasks . However the success of chip-multiprocessors depends on addressing the difficulty in the development of multithreaded applications . To overcome this difficulty it is necessary to develop new chip-multiprocessors with advanced programming models . Presently various architectures relies on software simulators which are too slow . This paper presents an alternative to simulation by designing platforms based on FPGAs for chip-multiprocessors and its application in multithreaded programs. The first prototype for chip- multiprocessors, ATLAS with hardware support for transactional memory, a technology aiming to simplify parallel programming is presented. ATLAS includes 8 embedded PowerPC cores that access coherent shared memory in a transactional manner .ATLAS uses the BEE2 multi-FPGA board to provide a system with 8 PowerPC cores that run at100MHz and runs Linux... ATLAS provides significant benefits for chip multiprocessors research such as 100x performance improvement over a software simulator and good visibility that help swith software tuning and architectural improvements. In addition to presenting and evaluating ATLAS, we share our observations about building a FPGA-based framework for chip-multiprocessors research. Overall, the ATLAS prototype provides an excellent framework for further research on the software and hardware techniques necessary to deliver on the potential of transactional memory.

I. INTRODUCTION

Processor vendors are turning en mass towards chipmultiprocessors (CMPs) as a practical way to turn increasing transistor budgets into scalable performance without the power and complexity challenges of aggressive uniprocessors. The trend towards chip-multiprocessors in the embedded domain is as strong as it is in the desktop and server domains. Several embedded vendors, such as ARM, ARC, Broadcom, Freescale, NEC, PMC-Sierra, and Raza Microelectronics, are selling chips or licensing designs for chip-multiprocessors for cache coherent shared memory configurations.. Moreover, there is significant research activity in porting embedded applications for chip-multiprocessors. While in some systems we can utilize chip-multiprocessor cores by running many programs concurrently, parallel programming is necessary in order to reduce the execution time of one program. Existing models for multithreaded programming using locks are challenging for most programmers as they introduce a trade-off between performance and correctness. Transactional Memory (TM)] has been proposed as a promising technology that can simplify concurrency management in multithreaded programs.TM allows programmers to define coarse-grain parallel tasks (transactions) that will be executed atomically and in isolation. TM executes these tasks in parallel on a chip-multiprocessor providing excellent performance. *Transactional memory*, a new multiprocessor architecture intended to make lock-free synchronization as efficient (and easy touse) as conventional techniques based on mutual exclusion. Transactional memory allows programmers to define customized read-modify-write operations that apply to multiple, independently-chosen words of memory.

This paper presents ATLAS, the first prototype of a chip-multiprocessors with hardware support for transactional memory. ATLAS includes 8 embedded PowerPC cores with a shared memory system. The data cache for each core is enhanced to buffer transactional state during optimistic execution and detect potential conflicts between concurrent transactions. To develop a technology like TM, collaborative research on both hardware and software aspects is necessary. Experimental chip prototypes can be used to speedup TM software research, but due to their fixed nature, they cannot help with the exploration of hardware alternatives. We mapped ATLAS to the BEE2 multi-FPGA board [6] to provide a full-featured prototype operating at 100MHz. The system boots the GNU/Linux operating system and provides support for transaction-based parallel programming and parallel application tuning. ATLAS provides a valuable tool for

validating the advantages of transactional memory. The FPGA mapping allows hardware researchers to tune hardware parameters as further insights are generated from application studies

II. TRANSACTIONAL MEMORY

Transactional memory (TM) provides an easy-to-use and high-performance parallel programming model for the upcoming chip-multiprocessor systems. Several researchers have proposed alternative hardware and software TM implementations. However, the lack of transaction- *Transactional Memory (TM)* [14] provides an alternative model for concurrency management. A TM system operates on shared data using sequences of instructions (transactions)which execute in an atomic and isolated manner [8].Transactional memory simplifies parallel programming by providing non-blocking synchronization with easy-to-write, coarse-grain transactions by virtue of optimistic concurrency. .

With transactional memory, programmers define atomic blocks of code (transactions) that can include unstructured flow-control and any number of memory accesses.

To parallelize an application, a programmer must break the code up into multiple threads that can execute in parallel. The programmer must also synchronize the threads when they potentially operate on the same data in memory. A *transaction* is a finite sequence of machine instructions, executed by a single process, satisfying the following properties:

- *Serializability*: Transactions appear to execute serially ,meaning that the steps of one transaction never appear to be interleaved with the steps of another. Committed transactions are never observed by different processors to execute in different orders.
- *Atomicity*: Each transaction makes a sequence of tentative changes to shared memory. When the transaction completes, it either *commits*, making its changes visible to other processes (effectively) instantaneously ,or it *aborts*, causing its changes to be discarded. We assume here that a process executes only one transactional a time. Although the model could be extended to permit overlapping or logically nested transactions, we have seen no examples where they are needed. A TM system must implement the following mechanisms:

(1) isolation of stores until the transaction commits;

(2) conflict detection between concurrent transactions;

(3) atomic commit of stores to shared memory;

(4) rollback of stores when conflicts are detected.

Transactional memory provides the following primitive instructions for accessing memory:

- *Load-transactional* (LT) reads the value of a shared memory location into a private register.
- *Load-transactional-exclusive* (LTX) reads the value ofa shared memory location into a private register, “hinting” that the location is likely to be updated.
- *Store-transactional* (ST) tentatively writes a value from a private register to a shared memory location. This new value does not become visible to other processors until the transaction successfully commits.

III. IMPLEMENTATION

In this section, we give an overview of an architecture that supports transactional memory. An associated technical report gives detailed protocols for both bus based(snoopy cache) and network-based (directory) architectures.

Our design satisfies the following criteria:

- Non-transactional operations use the same caches ,cache controller logic, and coherence protocols they would have used in the absence of transactional memory.
- Custom hardware support is restricted to primary caches and the instructions needed to communicate with them.
- Committing or aborting a transaction is an operation local to the cache. It does not require communicating with other processes or writing data back to memory.

Transactional memory is implemented by modifying standard multiprocessor cache coherence

protocols. We exploit access rights, which are usually connected with cache residence. In general, access may be non-exclusive(shared) permitting reads, or exclusive, permitting writes. At any time a memory location is either (1) not immediately accessible by any processor (i.e., in memory only), (2)accessible non-exclusively by one or more processors, or(3) accessible exclusively by exactly one processor. Most cache coherence protocols incorporate some form of these access rights.

3.1 Multithreaded Applications:

Table 1 presents the 25 multithreaded applications we used in this study. The applications were parallelized using four parallel programming models: Java threads [4], Cand Pthreads [19], C and Open MP [11], and the ANL parallel processing macros. Java is increasingly popular and includes multithreading in the base language specification. Open MP is a widely adopted model based on high level compiler directives for semi-automatic parallelization. Pthreads is a widely available multithreading package forPOSIX systems. Finally, the ANL macros were designed to provide a simple, concise, and portable interface covering a variety of parallel applications. We use the Java, Pthreads, and ANL applications to study the use of transactions for non-blocking synchronization (29 applications).

Table 1. Examples of Multithreaded applications

Prog. Model	Application	Problem Size	Source	Domain / Description
Java	MolDyn	2,048 Particles	JavaGrande	Scientific / Molelcular Dynamics
	MonleCarlo	10,000 Runs	JavaGrande	Scientific/Finance
	RayTracer	150×150Pixels	JavaGrande	Graphics / 3D Raytracere
	Crypt	200,000 Bytes	JavaGrande	Kernel / Encryption and Decryption
	LUFact	500×500Matrix	JavaGrande	Kernel / Solving $N \times N$Linear System
	Series	200 Coefficients	JavaGrande	Kernel / First N Fourier Coefficients
	SOR	$1{,}000 \times 1{,}000$Grid	JavaGrande	Kernel / Successive Over-Relaxation
	SparseMatmul	$250{,}000 \times 250{,}000$ Matrix	JavaGrande	Kernel / Matrix Multiplication
	SPECibb2000	8 Warehouses	SPECibb2000	Commercial / E-Commerce
	PMD	18 Java Files	DACapo	Commercial / Java Code Checking
	HSQLDB	10 Tellers, 1,000	DACapo	Commercial / Banking with hsql database
Pthreads	Apache	20 Worker Threads	Apache	Commercial / HTTP Web server
	Kingate	10,000 HTTP Requests	SourceForge	Commercial / Web proxy
	Bp-visioin	384×288 Image	Univ. of Chi.	Machine Learning / Loopy Belief Propagation
	Localize	477×177 Map	CARMEN	Robotics / Finding a Robot Position In a Map
	Ultra Tic Tac	5×5Board, 3 Step	SourceForge	Al/Tic Tac Toe Game
	MPE G2	640×480 Clip	MPEG S.S.G.	MultiMedia / MPE G2 Decoder
	AOL Server	20 Worker Threads	AOL Website	Commercial / HTTP web server
OpenMP	Equake	380K Nodes	SPE Comp	Scientific / Seismic Wave Propagation Simulation
	Art	640×90 Image	SPE Comp	Scientific / Neural Network Simulation
	CG	1400×1400 Matrix	NAS	Scientific / Conjugate Gradient Method
	BT	$12 \times 12 \times 12$ Matrix	NAS	Scientific / CFD
	IS	1M Keys	NAS	Scientific / Large-scale Integer Sort
	Swim	1.900×900 Matrix	SPE Comp	Scientific / Shallow Water Modelinq

IV. THE ATLAS CHIP-MULTIPROCESSOR SYSTEM

ATLAS is the first full-system framework for a Chip-multiprocessor with transactional memory support. This section presents its basic architecture, the hardware design, and its software environment. The prototype is currently operational at 100MHz, runs the GNU/Linux operating system, and runs multithreaded applications that use transactional memory.

ATLAS prototypes the Transactional Coherence and Consistency(TCC) architecture for hardware-based transactional memory . TCC assumes a set of processor cores with private first-level caches that are connected through a snooping bus to the shared memory (shared caches and DRAM).The cores execute transactions speculatively, while tracking the read- and write-sets in the data cache organization shown in Figure 1.

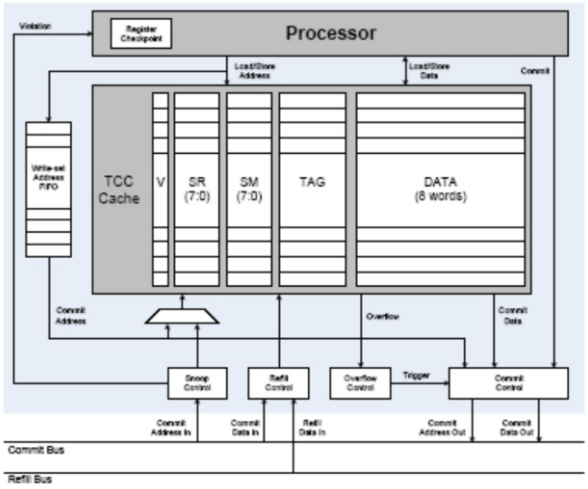


Fig. 1. The data cache organization for ATLAS

4.1 The ATLAS Hardware Design:

ATLAS includes 8 PowerPC405 cores that run multithreaded code for applications and a ninth core that handles the operating system and I/O devices. .

ATLAS is also part of the RAMP project that aims at developing FPGA-based technology for prototyping modern Chip-multiprocessor systems.

ATLAS prototypes the Transactional Coherence and Consistency(TCC) architecture for hardware-based transactional memory . TCC assumes a set of processor cores with private first-level caches that are connected through a snooping bus to the shared memory. As a core performs loads and stores within a transaction, it sets the speculatively-read (SR) and speculatively-modified (SM) bits to indicate that the corresponding word is now part of the transaction's read-set or write-set, respectively. Also, the first time a word is written n a specific cache line, the cache pushes a pointer to it into the write-set address FIFO. The cache operates as a write-buffer, isolating all writes from shared memory until the transaction completes .At the end of the transaction, the core arbitrates for permission to commit the write-set to the shared memory. Figure 3 shows the block diagram of ATLAS.

The four outer FPGAs, labeled as User FPGAs, are connected in a star topology through the 5th FPGA, designated as the Control FPGA. Figure 3(a)shows the block diagram of User FPGA.3(b)Control FPGA

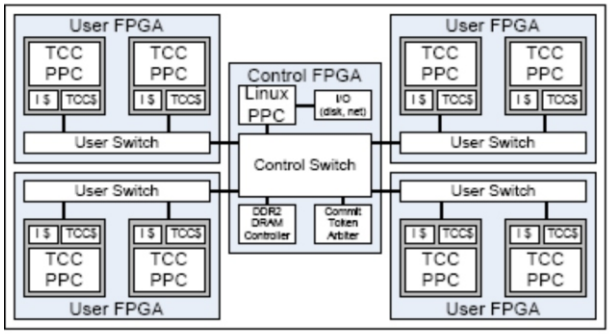


Fig. 2. Block diagram of ATLAS system

Each FPGA includes two PowerPC405 cores enhanced with a TCC data cache (TCC PPCs).We use the hardcore PowerPC cores in the Virtex II-ProFPGAs. The data cache design, written in synthesizable Verilog, is attached to the PowerPC cores through the IBM Processor Local Bus (PLB). The cache has 32 byte lines and can be 1, 2, or 4-way set associative (each way is 8 KB, resulting in cache sizes of 8, 16, or 32 KB). The write-set address FIFO can be configured to be 0.5, 1, 2, 4 or 8 KB. The internal data cache in the PowerPC cores is disabled. The TCC cache is in turn connected to a network switch(shared by two cores) that forwards cache misses and commit requests to the control FPGA through a central switch..

The Control FPGA is a hub that connects all processors to the shared memory and I/O devices. The current design does not use secondary caches, though this is not fundamental. As depicted in Figure 3(b), the Control Switch interfaces switch the commit token arbiter, the DDR controller and the PowerPC 405 core that runs Linux (Linux PPC). The Linux PPC uses its built-in caches but its memory writes are broadcast to all other processors for coherence purposes. All traffic sent through switches is packetized with a single packet format to simplify routing around the system.

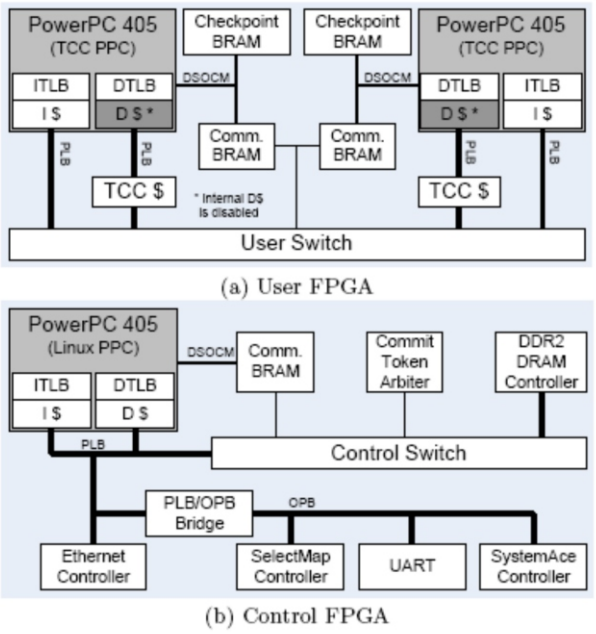


Fig. 3. Block diagram of (a)User FPGA,(b)Control FPGA.

4.2 The ATLAS Software Design.

The ATLAS software stack consists of an API for programming with transactions and the system software For application programming, the user partitions work into parallel threads and defines atomic transactions within each thread. The ATLAS system software is summarized in Figure 4.ATLAS runs Linux only on the Linux PPC. The eight TCCPPCs in the user FPGAs run a simple runtime kernel that coordinates with Linux. This approach is similar to the intel MISP concept for efficient Chip-multiprocessors. During the program execution, the Linux PPC core handles all interrupts due to external devices. It also handles any OS functions needed to support the execution on the TCC PPCs, like system calls or exceptions such as a TLB miss. For example, on a TLB exception, the TCC PPC sends the faulting address to the Linux PPC. The Linux PPC regenerates the exception, runs the corresponding OS code to resolve it (e.g. access the page table for the proper translation entry), and sends the information back to there questing TCC PPC. An important part of the ATLAS system is the support or performance tuning of user applications.

At the current scale of the ATLAS system, a single core running the OS is sufficient to serve eight cores running application code. Using a single core for the OS allows us to run conventional Linux without special consideration for concurrency in the OS code. In larger scale configurations of ATLAS. Transactional memory makes it easy to write a correct parallel program. Nevertheless, a program may still include performance bottlenecks such as frequent transaction violations or expensive overflows.

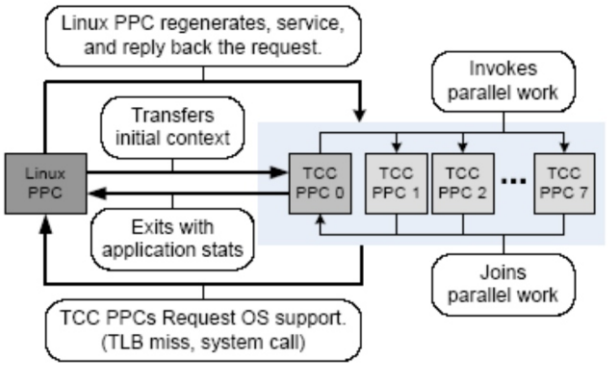


Fig. 4. Overview of ATLAS system software.

Table 2. Atlas Design Characteristics

CPUs	8 PowqerPC 405 cores (TCC) at 100 Hz 1 PowerPC 405 core (Linux) at 300 MHz 16 KB 2-way I-cache (9 cores) 32 KB 4-way TCC D-cache (8 cores) 16 KB 2-way PowerPC internal D-cache (1 core)

Main Memory	512 MB DDR2 at 200 MHz
I/O	10/100 Mbps Ethernet, RS232 UART, 512 MB Compact Flash
OS	Montavista 3.1 (Linux kernel ver. 2.4.30)
EDA Tools	Xilinx EDK 7.11
User FPGA	Xilinx XC2VP70, 17,641 LUTs (26%), 212 KB BRAMs (32%)
Ctrl FPGA	Xilinx XC2VP70, 16,284 LUTs (24%), 66 KB BRAMs (10%)

V. EVALUATION

Table 2 presents the default configuration of the ATLAS design and its resource utilization. The design is currently operational at 100MHz and runs Linux.

5.1 Methodology

To evaluate ATLAS, we ran five applications on both ATLAS and TASSEL, an established software simulator for the TCC architecture. The applications include three scientific benchmarks (radix, mp3d, ocean); a hash table micro benchmark that performs random insert, remove, and lookup operations on a hash table; and vacation, a benchmark that emulates a travel reservation management and database system. All applications include multiple threads that operate on data in shared memory in an irregular manner. We usethe following evaluation metrics:

- Wall Clock Time: We observe how much faster ATLAS executes each application compared to TASSEL by wall clock time (as seen by the user).
- Speedup: We examine the estimated architectural speedup of each application normalized to the uniprocessor execution time in each system. We compare the TASSEL and ATLAS speedup to verify that the FPGA design constraints do not affect the accuracy of execution results.
- Execution Time Breakdown: To fully understand the speedup trends, we measure and compare the execution time breakdown for each system. The execution time is divided into the following components: “Useful” time, stall cycles due to cache misses, time spent committing transactions, idle cycles due to thread imbalance(synchronization), and cycles lost due to violations. Such breakdowns are important for both verification]purposes and to provide programmers and architects with further profiling information

VI. OBSERVATIONS

The design experience from ATLAS allows us to make some observations and identify challenges for constructing FPGA-based frameworks for CMP research. This section summarizes the most important issues.

Despite the lower clock frequency as compared to modern workstations, FPGA-based CMP environments can outperform software simulators by two orders of magnitude.

The major challenges when mapping ASIC-style RTL for a CMP system on an FPGA are highly associative The first challenge is mapping highly-associative structures that are commonly used in CMP designs to implement content-addressable memories (CAM), hardware schedulers ,victim caches, and state bits with gang set/reset capabilities. FPGA vendors now embed CAMs in their chips. Although CAMs have many useful applications for searching, they do not help with structures that use gang set or reset operations .memory structures, large lower level caches, and interconnect fabrics that span across FPGAs. FPGA-based frameworks have been considered to require more effort than software simulators. This belief is mainly due to the time required for RTL development of the various smaller system components that are necessary for full system modeling. The availability of pre-designed IP libraries has significantly reduced the design efforts for FPGAs. ATLAS makes heavy use of components available through Xilinx and RAMP such as the DRAM and ethernet controllers. The PowerPC 405 core and the accompanying Xilinx software(Xilinx Microprocessor Debugger or XMD) provide good examples of hardware and software support for debugging and profiling. The capabilities of these systems provided a link through a JTAG chain from the PowerPC coresto the well-known GNU debugger.

VII. CONCLUSION

The primary goal of transactional memory is to make it easier to perform general atomic updates of multiple independent memory words, avoiding the problems of locks (priority inversion, convoying, and deadlock). We studied a set of existing multithreaded applications in order to characterize their common case behavior with transactional memory systems.ATLAS is the first full-system prototype of a CMP with hardware support for transactional memory. Mapped on the BEE2 multi-FPGA board, ATLAS operates at 100 MHz,runs Linux, exhibits good performance on parallel applications and out performs our software simulator by two orders of magnitude. The ATLAS design indicates that FPGA based frameworks can be an extremely useful tool for Chip - multiprocessors.

Nevertheless, the ATLAS experience also indicates the challenges that researchers must face when mapping CMP designs to FPGAs. The issues include challenges in mapping ASIC-style CMP RTL on to FPGAs, the selection criteria for the base processor, and debugging and profiling support in pre-designed IP libraries. The insights from application studies will be very useful in terms of improving TM implementations and programming models. ATLAS will also allow us to study the use to transactions in the operating system code. From a hardware point of view, we are interested in studying further hardware support for parallel application development (debugging and tuning).

REFERENCES

[1] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," Dr. Dobb's Journal, vol. 30, March2005.

[2] B. Lewis and D. J. Berg, Multithreaded Programming withPthreads. Prentice Hall, 1998.

[3] M. Herlihy and J. E. B. Moss, "Transactional memory:Architectural support for lock-free data structures," in Proc. ofthe 20th Intl. Symp. on Computer Architecture, pp. 289–300,1993.

[4] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D.Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis,and K. Olukotun, "Transactional memory coherence and consistency," in Proc. of the 31st Intl. Symp. on ComputerArchitecture, pp. 102–113, June 2004.

[5] C. S. Ananian, K. Asanovic, B. C. Kuszmaul, C. E. Leiserson,and S. Lie, "Unbounded Transactional Memory," in Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture (HPCA'05), (San Franscisco, California),pp. 316–327, February 2005

[6] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing TransactionalMemory," in ISCA '05: Proc. of the 32nd Annual Intl. Symp.on Computer Architecture, pp. 494–505, June 2005.

[7] K. E. Moore, J. Bobba, M. J. Moravan, M. D.Hill, and D. A.Wood, "LogTM: Log-Based Transactional Memory," in 12thIntl. Conf. on High-Performance Computer Architecture,February 2006.

[8] N. Shavit and D. Touitou, "Software transactional memory," inProc. of the 14th Annual ACM Symp. on Principles ofDistributed Computing, (Ottawa, Canada), pp. 204–213,August 1995.

[9] ARM11MP Core.http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

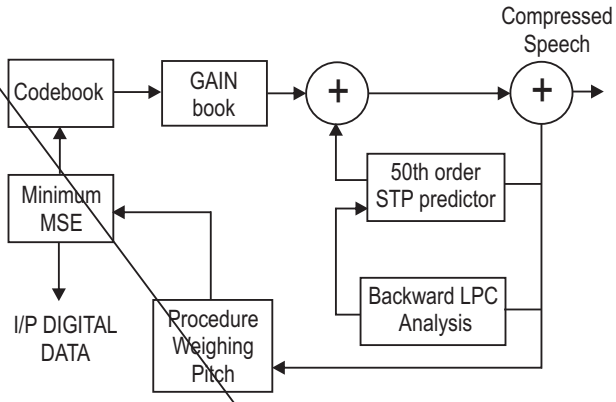
[10] Fall processor forum: The road to multicore. http://www.instat.com/fpf/05/index05.htm.

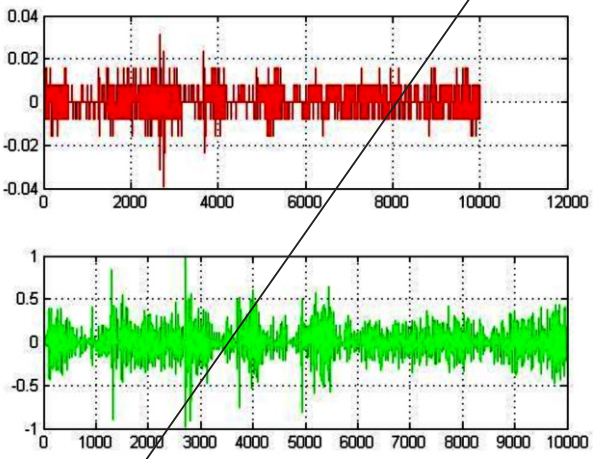
[11] C. S. Ananian, K. Asanovic, B. C. Kuszmaul, C. E. Leiserson,and S. Lie. Unbounded Transactional Memory. In Proc.of the 11th Intl. Symp. on High-Performance Computer Architecture(HPCA'05), pages 316–327, San Franscisco, California,February 2005.

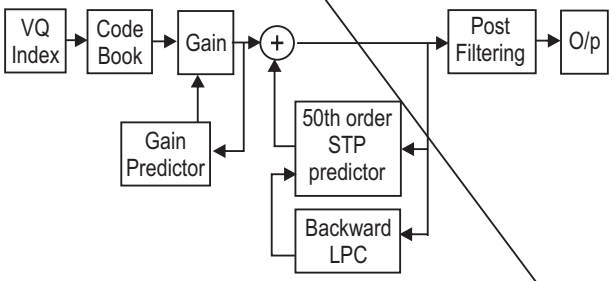
[12] Arvind, K. Asanovic, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J.Wawrzynek.RAMP: Research accelerator for multiple processors -a community vision for a shared experimental parallelHW/SW platform. Technical report, 2005.

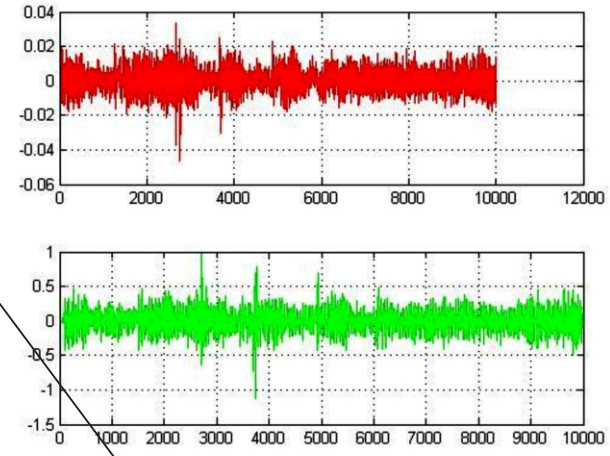
[13] H. Chafi, C. Cao Minh, A. McDonald, B. D. Carlstrom,J. Chung, L. Hammond, C. Kozyrakis, and K. Olukotun.TAPE: A Transactional Application Profiling Environment.In ICS '05: Proc. of the 19th Ann. Intl. Conf. on Supercomputing,pages 199–208. June 2005.

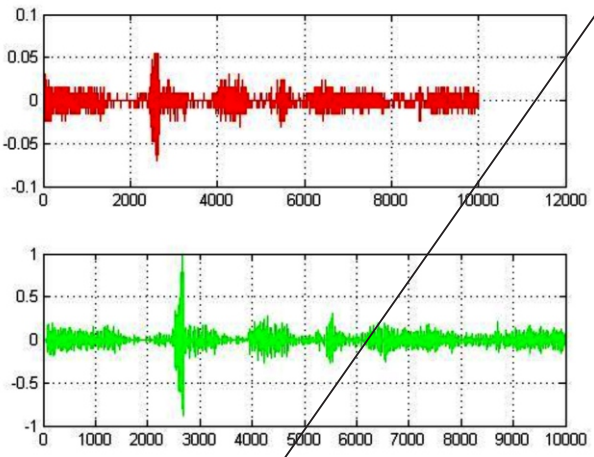
[14] C. Chang, J. Wawrzynek, and R. W. Brodersen. BEE2: Ahigh-end reconfigurable computing system. IEEE Designand Test of Computers, 22(2):114–125, Mar/Apr 2005


Compressed
Speech
Codebook
GAIN
book
+
+
50th order
STP predictor
Minimum
MSE
I/P DIGITAL
DATA
Procedure
Weighing
Pitch
Backward LPC
Analysis


0.04
0.02
0
-0.02
-0.04
0
2000
4000
6000
8000
10000
12000
1
0.5
0
-0.5
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


VQ
Index
Code
Book
Gain
+
Post
Filtering
O/p
Gain
Predictor
50th order
STP
predictor
Backward
LPC


0.04
0.02
0
-0.02
-0.04
-0.06
0
2000
4000
6000
8000
10000
12000
1
0.5
0
-0.5
-1
-1.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000


0.1
0.05
0
-0.05
-0.1
0
2000
4000
6000
8000
10000
12000
1
0.5
0
-0.5
-1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000