Brief Announcement: Performance Potential of an Easy-to-Program PRAM-On-Chip Prototype Versus State-of-the-Art Processor

George C. Caragea
University of Maryland
george@cs.umd.edu

A. Beliz Saybasili[*]
LCB Branch, NHLBI, NIH
saybasiliab@mail.nih.gov

Xingzhi Wen
NVDIA Corporation
xwen@nvidia.com

Uzi Vishkin
University of Maryland
vishkin@umiacs.umd.edu

ABSTRACT

We compare the Paraleap FPGA computer, a 64-processor hardware prototype of the PRAM-driven XMT architecture, with an Intel Core 2 Duo processor and show that Paraleap outperforms the Intel processor by up to 13.89x in terms of cycle counts. The comparison favors the Intel design, since the silicon area of an ASIC implementation of the 64-processor XMT design is the same as that of a *single* core.

Categories and Subject Descriptors: C.1.4 Parallel Architectures C.4 Performance of Systems

General Terms: Algorithms, Performance

1. INTRODUCTION

The eXplicit Multi-Threading (XMT) on-chip general purpose computer architecture is aimed at the classic goal of reducing single task completion time. It is a parallel *algorithmic* architecture in the sense that: (i) it seeks to provide good performance for parallel programs derived from Parallel Random Access Machine/Model (PRAM) algorithms, and (ii) a work flow for advancing from PRAM algorithms to XMT programs, along with a performance metric and its empirical validation are provided [4]. Scalability and ease of parallel programming are now widely recognized as the main stumbling blocks for extending commodity computer performance growth (e.g., using multi-cores) [3]. XMT provides a unique answer to both challenges: 1) To address the programmability issue, the following methodology is proposed. Given a problem, a PRAM style parallel algorithm is developed using the Shiloach-Vishkin 1982 Work-Depth (WD) Methodology. All the operations that can be concurrently performed in the first round are noted, followed by those that can be performed in the second round, and so on. Such synchronous description of a parallel algorithm makes it easy to reason about correctness and analyze for work (the total number of operations) and depth (number of rounds). The XMT programmer is then expected to use the XMTC language (basically C with two additional commands) for producing a multi-threaded program. Reasoning about correctness or performance can now be restricted to just comparison of the program with the WD algorithm, assuming that correctness and performance of the algorithm have been established, often a much easier task than directly analyzing the program. 2) By design, XMT also scales to one thousand cores and beyond, the direction in which Moore's law and the current technology trends point towards.

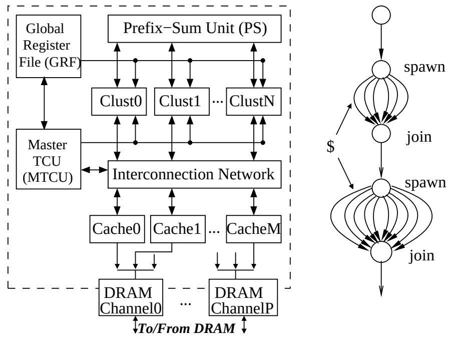
The main contribution of this brief announcement is the evaluation of an embodiment of the XMT Architecture, the FPGA-based Paraleap prototype, using three benchmarks and realistic input sizes, and contrasting our results with an Intel Core 2 Duo processor. In all our tests, the Paraleap architecture surpassed the Core 2 Duo processor. The more meaningful results show speed-ups ranging between 6.3x and 13.89x in terms of cycle counts. We consider this an extremely encouraging result for the further development of XMT. XMT is an academic project with a very modest budget, while Intel is an industry giant that invested many thousands of man-years in its processor.

2. PARALEAP: AN FPGA PRAM ON-CHIP SYSTEM

A **64-processor FPGA prototype** of the XMT chip was completed in early 2007 at the University of Maryland. This first commitment to silicon constitutes an important milestone for the project. It provided: (i) validation of the HDL description and implementability of the XMT architecture; (ii) a platform for running realistic size applications and benchmarks (the FPGA prototype runs about 11-12K times faster than our cycle-accurate simulator) and (iii) support for teaching parallel algorithmics to relatively big classes (30+ students) at all levels, from high-school to graduate level, while providing hands-on programming experience on actual hardware. The completion of the 64-core FPGA prototype was first announced in SPAA'07 with a more detailed description and evaluation appearing in [5].

The general **XMT architecture** (Figure 1(a)) includes a Master Thread Control Unit (MTCU), processing clusters ($Clust_0, \ldots, Clust_N$) each comprising of several TCUs, a high-bandwidth, low latency interconnection network, on-chip cache modules which share the DRAM Channels, a global register file (GRF) and a prefix-sum unit (PS). A cluster has functional units shared by several TCUs and one load/store port to the interconnection network, shared by

[*]Also Istanbul Technical University



(a) (b)

Figure 1: (a) Block diagram of XMT (b) The XMT execution model: switching between serial and parallel modes.

all its TCUs. Clusters also contain a compiler-controlled read-only buffer which can cache values likely to be re-used within the cluster. TCUs include an in-order pipeline, register unit and basic functional unit (ALU, BR, SFT), and also a small (e.g. 4-word) prefetch buffer unit used by the compiler to hide memory latencies through data prefetching.

The global memory address space is evenly partitioned into the cache modules using a form of hashing. The hash function operates at cache line granularity, mapping contiguous address spaces uniformly amongst the cache modules. A particular memory location can be found in at most one cache module at any one time. Within each module, the order of operations to the same location is preserved, and a store operation can be acknowledged as soon as the module accepts the request, regardless if it is a hit or a miss.

There are no local caches at the TCUs, effectively *sidestepping the cache-coherence problem.* This is essential to the scalability of the design to an envisioned 1024-TCU XMT computer and beyond, as cache coherence mechanisms use non-trivial amounts of chip area and power, as well as adding execution overheads.

Paraleap is an embodiment of the XMT architecture, with the system specifications as listed in Table 1.

Clock rate	75 MHz	No. TCUs	64
DRAM size	1GB	Clusters	8
DRAM chann.	1	Cache modules	8
Mem. data rate	0.6GB/s	Shared cache	256KB

Table 1: The Paraleap FPGA Prototype

The XMT Paraleap prototype is built on an FPGA development board using three FPGA chips: two Xilinx Virtex-4 LX200 and one Virtex-4 FX100. The utilization rate of the two highest loaded FPGA chips was 99% and 94%. The Xilinx ISE Design Suite was able to meet timing requirements for up to 100MHz for individual sub-modules, but the clock rate had to be reduced to 75MHz for the system, since the high utilization rate significantly limited the flexibility of the Place-and-Route process. Future embodiments of the XMT architecture will not have this limitation, allowing for higher clock speeds to be achieved.

The **XMTC programming language** is an extension of standard C used to program the XMT processor. Execution switches between serial and parallel, as shown in Figure 1(b). Threads are usually short, and the execution model is Single-Program, Multiple-Data (SPMD). XMTC includes constructs to explicitly manage parallelism, namely a `spawn` instruction to start a parallel section, a special identifier `$` holding the unique thread ID while in parallel mode and a prefix-sum instruction `ps` which is executed efficiently by a dedicated hardware mechanism and can be used for low-overhead inter-thread coordination.

3. EXPERIMENTAL EVALUATION

As duly acknowledged later, a senior engineer at Intel suggested that we augment prior benchmarking efforts by evaluating the performance of the Paraleap system and compare it with a state-of-the-art existing processor. A set of benchmarks and input sizes was recommended to us as constituting a fair comparison. For each benchmark, we wrote two implementations: (i) a **parallel** program, written in XMTC to be executed on Paraleap and (ii) a **serial** C program for the Intel Core 2 Duo architecture. We used the following three benchmarks for evaluation:

SpMV: A sparse matrix, stored in Compact Sparse Row (CSR) format, is multiplied by a dense vector. The implementation is straightforward: the serial version simply multiplies each row with the vector one at a time, while the parallel implementation processes all rows in parallel, using exactly one thread per row.

FFT: 1-D Fast Fourier Transformation. We used the Radix-2 Cooley-Tukey algorithm as the basis of our implementations. The algorithm runs in stages; first, a “twiddle” table, used to combine the input and output of stages, is computed. Next, a binary bit reversal pass prepares the input data. The main part of the algorithm consists of $\log N$ “butterfly” computation stages. The serial implementation follows this algorithm. For the parallel version, we parallelized each of the stages using the XMT parallel programming model. Note that at the present time, Paraleap has only integer arithmetic support; to allow for a fair comparison, we implemented both the serial and parallel versions using fixed-point arithmetic with the same precision.

Quicksort: Sorting an array of integer values. For the serial implementation of this benchmark, we implemented the standard Quicksort algorithm found in any serial algorithms textbook. The parallel version of quicksort follows the algorithm introduced in [2]: in the first phase, the array is iteratively partitioned using a fetch-and-add based parallel scheme, taking advantage of the XMT prefix-sum primitive. When the number of partitions exceeds a threshold, the execution switches to its second phase, where each partition is sorted by exactly one thread.

Table 2 describes the **input data** used in our experiments. For each benchmark, we created two datasets: a small one, that is comparable in size to the on-chip cache, and a larger one that exceeds the cache size for both Paraleap and the Intel Core 2 Duo processor.

Experimental Setup.

We collected cycle counts for the parallel XMTC programs running on Paraleap and for the serial C implementations on the Intel Core 2 Duo system as follows:

Parallel execution We compiled each benchmark using the XMTC compiler, which is a port of GCC 4.0.2 to the XMT platform. We used the maximum level of optimization

	Small		Large	
Program	N	Footprint	N	Footprint
SpMV	22K	200KB	4M	33MB
FFT	8K	192KB	4M	96MB
Quicksort	100K	781KB	20M	153MB

Table 2: Datasets used. Footprints represent the total amount of memory used at runtime.

supported (-O3), and activated data prefetching optimizations tuned for XMT. Instructions to use the cluster read-only buffers were introduced manually in the code, but we expect this process to be automated in a future version of the compiler. We ran the compiled benchmarks on Paraleap and collected the cycle counts reported by the system.

Serial execution For the second part of our experiment, we used a desktop system with an Intel Core 2 Duo E6300 CPU rated at 1.86GHz, with 64KB L1 and 2MB L2 cache per processor and 2GB DDR2-667 DRAM. Choosing a compiler for the Intel x86 architecture can have a significant influence on execution performance, since different compilers utilize the large array of features of the architecture in different ways. We used two compilers in our experiments: (i) the widely used open-source GNU C Compiler (GCC) and (ii) the Intel C++ Professional Compiler for Linux (ICC v11.0), the most recent version at the time of writing. Just as for the Paraleap compiler, we used the highest level of optimization for both compilers. In addition, we enabled the advanced optimizations available for the ICC compiler on the Core 2 Duo architecture: SIMD vectorization using SSE3, software data prefetching and auto-parallelization. We ran each benchmark 5 times and collected cycle counts using the *Time Stamp Counter* 64-bit register instead of the more coarse-grained system timers. This ensured better precision and factored out OS interferences.

An FPGA prototyping/cycle-accurate emulation methodology for projecting the cycle counts for an 800MHz ASIC implementation, using DDR2-800 SDRAM was introduced in [6]. We are using the same configuration for our current experiments.

Results.

The speed-ups for the three benchmarks between the Paraleap and the Intel Core 2 Duo systems are presented in Table 3. The speed-up figures represent the ratio between the **number of clock cycles** needed for the execution of each of the benchmarks on the Paraleap and the Intel Core 2 Duo computers.

	Core2-ICC		**Core2-GCC**	
Program	*small*	*large*	*small*	*large*
SpMV	6.7	3.3	6.3	3.26
FFT	9.51	2.51	8.76	2.71
Quicksort	13.07	7.75	13.89	8.18

Table 3: Clock cycle speed-ups of Paraleap vs. Intel Core 2

The lower speed-up numbers for the *large* datasets can be explained by the large difference in cache size between Paraleap and Core 2 Duo (256KB compared to 2x2MB). However, there is no reason that the cache size of a future XMT system will be smaller than its contemporaries, addressing this discrepancy.

Discussion.

In terms of silicon area, the comparison above is tilted in favor of the Intel design. The silicon area of an ASIC implementation of the 64-processor XMT design is roughly the same as that of a *single* core of the Intel Core 2 Duo. The same holds with regards to the compiler, since both compilers used for the Intel platform are mature, established products, while the XMTC compiler performs only basic optimizations at this stage.

When it comes to clock speed, the comparison becomes much more involved. An important reason for the comparison is getting a feel for the scalability potential of XMT relative to possible upgrades of the Intel. We felt that the latter is too speculative for the current paper, but we do not see why the clock speed of a 1024-TCU XMT should be lower than a same-generation many-core processor that incorporates cache-coherence and uses the same silicon area.

4. CONCLUSION

The XMT project provides a viable answer to the two biggest challenges currently facing the architecture community: ease of programing and scalability. The results of the evaluation we performed show that the XMT Architecture, even at this early stage of development, can outperform one of the most widely used current processors.

Finally note that a complete programming environment including a configurable cycle-accurate simulator and basic compiler can be downloaded to any standard computer platform (Windows, Linux, Mac) through the XMT software release website [1], allowing anyone to evaluate the architecture and programming model without requiring access to the hardware.

Acknowledgments

Pradeep Dubey, Intel suggested the benchmarks used in the current paper as well as the comparison methodology. We would also like to gratefully acknowledge the work of Alexandros Tzannes and James Edwards on the compiler infrastructure and experimental platform, as well as other members of the XMT team. The name Paraleap was chosen out of 6000 submissions in an open naming context conducted by UMD.

5. REFERENCES

[1] Software release of the explicit multi-threading (XMT) programming environment. www.umiacs.umd.edu/users /vishkin/XMT/sw-release.html, August 2008.

[2] P. Heidelberger, A. Norton, and J. Robinson. Parallel Quicksort Using Fetch-and-add. *IEEE Transactions on Computers*, 39(1):133–138, Jan 1990.

[3] M. Snir. Multi-core and parallel programming: Is the sky falling? *The Computing Community Consortium Blog*, www.cccblog.org/2008/11/17/multi-core-and-parallel-programming-is-the-sky-falling/, November 2008.

[4] U. Vishkin, G. C. Caragea, and B. C. Lee. *Handbook of Parallel Computing: Models, Algorithms and Applications*, chapter Models for Advancing PRAM and Other Algorithms into Parallel Programs for a PRAM-On-Chip Platform. CRC Press, 2007.

[5] X. Wen and U. Vishkin. FPGA-based Prototype of a PRAM-On-Chip Processor. In *ACM-CF'08: Proceedings of the 2008 conference on Computing frontiers*, pages 55–66, New York, NY, USA.

[6] X. Wen and U. Vishkin. The XMT FPGA Prototype/Cycle-accurate-simulator Hybrid. In *WARP08: The 3rd Workshop on Architectural Research Prototyping*, Beijing, China, June 2008. In conjunction with ISCA 2008.