Algorithmic Approach to Designing an Easy-To-Program System: Can It Lead to a HW-Enhanced Programmer's Workflow Add-On?

Uzi Vishkin

University of Maryland Institute for Advanced Computer Studies (UMIACS)
vishkin@umd.edu

Abstract—**Our earlier parallel algorithmics work on the parallel random-access-machine/model (PRAM) computation model led us to a PRAM-On-Chip vision: a comprehensive many-core system that can look to the programmer like the abstract PRAM model. We introduced the eXplicit Multi-Threaded (XMT) design and prototyped it in hardware and software. XMT comprises a programmer's workflow that advances from work-depth, a standard PRAM theory abstraction, to an XMT program, and, if desired, to its performance tuning. XMT provides strong performance for programs developed this way due to its hardware support of very fine-grained threads and the overhead of handling them. XMT has also shown unique promise when it comes to ease-of-programming, the biggest problem that has limited the impact of all parallel systems to date. For example, teachability of XMT programming has been demonstrated at various levels from rising 6th graders to graduate students, and students in a freshman class were able to program 3 parallel sorting algorithms.**

The main purpose of the current paper is to stimulate discussion on the following somewhat open-ended question. Now that we made significant progress on a system devoted to supporting PRAM-like programming, is it possible to incorporate our hardware support as an add-on into other current and future many-core systems? The paper considers a concrete proposal for doing that: recasting our work as a hardware-enhanced programmer's workflow "module" that can then be essentially imported into the other systems.

I. INTRODUCTION

Programming today's multi-core systems, as well as past and present multi-chip multiprocessors is not easy. In fact, relatively little has changed since the 2003 National Science Foundation Panel on Cyberinfrastructure reported that: *"to many users, programming existing parallel computers is as intimidating and time-consuming as programming in assembly language"*.

The parallel random-access machine/model (PRAM) theory of algorithms provides a well-established easy approach to parallel algorithmic thinking, JaJa'90 and Keller, Kessler&Traeff'01. A maxim that guided some of this early PRAM work was that parallel algorithmic thinking should be understood prior to attempting the design of a parallel system.

As the PRAM theory reached maturity around 1990, a debate developed on the role it can play for multi-chip multiprocessing, the only type of multiprocessing possible at the time. We note 3 positions on this subject that represent the range of the debate: (i) The well-cited LOGP paper Culler et al'93 stated that the PRAM theory is largely irrelevant for anything that can be ever built, mostly because of insufficient bandwidth among processors and among processors and memories. (ii) At the other end of the spectrum Wolfgang Paul originated the SB-PRAM multiprocessor multi-chip project. See, e.g., Formella, Keller &Walle'96. (iii) Culler, Singh&Gupta'99 opined that a breakthrough for the programmability of parallel machines may emerge if a machine that can look to the programmer like a PRAM can ever be built.

The opportunity for building an on-chip parallel machine with hundreds of processors or more using the amount of logic than can fit on a single chip emerged on our horizon in 1997. The key questions were whether this opportunity can address the concern of Culler et al'93 regarding bandwidth, and, if yes, become the game changer that will allow seeking a breakthrough, in line with Culler, Singh&Gupta'99.

Inspired by these questions, the PRAM has been our starting point in Vishkin et al'98 for designing from the ground-up a many-core on-chip computer system, called eXplicit Multi-Threading (XMT). Several insights from the Multi-threaded Architecture (MTA), a multi-chip system, Alverson et al'90, originated by Burton Smith, influenced the XMT system design. The MTA was renamed "Cray XMT" in 2006 (after Smith left Cray). Note that the Cray XMT is a different system than the XMT system discussed in this paper.

Capitalizing on the system design opportunities opened up by the new era of on-chip parallelism, XMT incorporates: (i) a prefix-sum functional unit that subject to some constraints executes a plurality of Fetch-and-Add (F&A) commands (XADD in X86) in the same time as a single F&A command, (ii) a higher-bandwidth, lower-latency interconnection network among processors and memory, (iii) control mechanisms that generalize the von-Neumann stored-program-plus-program-counter mechanism to spawn as many available threads (with their instructions already in place) as the hardware supports, within the same time that it takes to do that for spawning a single thread, (iv) reallocation of just-freed hardware to as many available threads as the freed hardware can support, and do that within the same time it takes to have one thread allocated, (iv) on-chip shared caches for a type of shared locality that was not possible in the multi-chip MTA; (v) up to thousands of light-weight processors (called thread control units, or TUCs), coupled with a powerful serial processor (master TCU, or MTCU);

the TCUs can be all activated at once by the MTCU without off-loading off-chip of data or instructions; the MTCU provides backwards compatibility on serial code as it does not fall behind a state-of-the-art uni-processor; XMT threads can be very fine-grained and involve irregular access to memory; and (vi) a programmer's workflow that relies on PRAM-like programming can guide XMT programming.

The purpose of this note is to support suggestions to be made as part of the author's keynote presentation at ICCD'09. It is not meant to replace material already presented in published papers. For this reason, the presentation is not self-contained. Hopefully, the list of references, the level of the discussion, and the slides that we plan to make public through the XMT home page will make all our points accessible to interested readers. In fact, we expect that reading Wen&Vishkin'08 will be sufficient.

II. Some Proposed Challenges

The heart of the field needs to be reinvented for parallelism, which is quite a tall order. The main technical challenge is timely convergence to an easy-to-program highly scalable general-purpose platform for many-cores. The discussion below reviews 3 smaller challenges. Addressing them will contribute towards such convergence. Roles that XMT can play are emphasized.

1. Programmability by Every CS Major
Every person who majors in CS will have to be able to program the new many-core system. But, is it possible to build a many-core system that permits access by all CS majors? We have provided tentative evidence that this is doable: programming of the explicit multi-threaded (XMT) system we built has been taught at various levels, from rising 6th graders to graduate students.

Computer system research tends to limit benchmarking of new machines to performance. This is in spite of the fact that ease-of-programming of parallel systems for many-cores or otherwise is a known problem.

We suggest using *teachability* at various levels as a practical ease-of-programming benchmark for current and future many-core designs, in addition to performance. A necessary condition for programmability, it is relatively simple to make teachability at various grade levels a standard benchmark.

2. Getting the Business of Software Developers
Customers buying a computer interact with its software, but their link to the hardware is indirect, by nature. However, the cyclic process of hardware improvements leading to software improvements, which lead back to hardware improvements and so on, known as the *software spiral,* facilitated for many years a direct link between customers and hardware. Hardware designers could directly serve their customers: (i) A stable application-software base that could be reused and enhanced from one hardware generation to the next was available; and (ii) Better performance had been assured with each generation if only the hardware could run serial code faster, a reality popularized by the "Intel inside" advertizing campaign. Alas, the software spiral is now broken: (a)

nobody is building hardware that provides improved performance on the old serial software base; (b) there is no broad parallel computing application software base for which hardware vendors are committed to improve performance; and (c) no agreed-upon architecture currently allows application programmers to build such software base for the future.

Consequently, getting application software developers to switch to the emerging generation of many-core systems has become much more critical to serving the above customers. However, the incentive to develop software for the new machines has decreased considerably. Code development and maintenance is much more expensive, as initial development time is higher and code is more error prone. Not only that the investment is higher, the returns on it are much riskier: even if machines continue to support the development platform, some hard-to-predict future upgrades may offer new options for optimization of performance, allowing competitors to develop better software, at a lesser cost, by just adopting a wait-and-see approach. Thus, computer designers need to understand the legitimate concerns of software developers and do what they can to "woo" them.

XMT could affect the above discussion in two ways. First, it affirms concerns that hardware improvements that may significantly reduce investment in code development by just waiting till they are installed are indeed possible. The second way would be if there was a way for incorporating the needed hardware upgrades so as to support the broad family of PRAM algorithms. Since an XMT system provides such support, we suggest the following challenge.

3. Lower the Bar for Adoption of XMT by Vendors
So far, vendors either adopted established parallel architectures, generally known for the difficulty of programming them, or introduced new ones, whose programming is generally not much easier. For instance, programming for locality is often very difficult. Still, urgings by vendors to program for locality in order to match their hardware appear to be gaining new momentum.

To date vendors have limited their adoption of technologies addressing ease-of-programming to software technologies that do not require any hardware update. The appeal of these software technologies to vendors is clear, and some are quite elegant, but so far they have not "set the programmer free" in the same way that a hardware-based technology can. For example, none allows the programmer to start with WD design and proceed directly from there to a computer program in the same way that the XMT workflow allows. XMT, on the other hand, does require some hardware support. The question is whether we could come up with a list of hardware features that allows upgrading current or future many-core systems to harness the power of XMT; the shorter the list the better.

We must say that we were not very optimistic, until recently, about the prospects of making XMT an add-on option for many-core systems. The good news, however, is that to achieve better scalability a growing trend limits the role of cache coherence in many-core systems. As XMT

parallelism assumes a memory architecture that does not allow local (write) caches at the TCUs (thread control units), this trend will hopefully make it easier to augment many-core systems with some or all of the hardware features of XMT, and support the XMT programmer's workflow. The payoff can be substantial: (i) these systems could benefit from the ease-of-programming and performance of the XMT approach, (ii) it may be easier to convince software developers to bet on the new systems, and (iii) it should also be easier to convince instructors who will see an easy-to-program approach supported by vendors to start teaching it; the XMT project developed extensive teaching materials (and software release) that these teachers could use; these materials can also help them decide to do it.

III. THE XMT APPROACH

The wealth of the parallel random-access machine/model (PRAM) theory of algorithms is well documented. The explicit Multi-Threading (XMT) project has been driven by a PRAM-On-Chip vision, seeking to build an easy-to-program parallel computer comprising thousands of processors on a single chip using a PRAM-like programming model. XMT has gone through significant hardware (e.g., 64-processor machine) and software prototyping. A software release allows experimentation with the XMT environment on any standard computer platform.

Interestingly, starting with the PRAM might not have been an obvious choice. Technology constraints guide us away from tightly coupled concurrency in programs; e.g., away from the PRAM and towards multi-threading. On the other hand, multi-threaded programs are notoriously difficult to design or analyze for correctness or performance.

1. The XMT Programmer's Workflow
The XMT programming approach incorporates an elegant workaround overviewed below. Based on a programmer's model that comprises multiple levels of abstractions, XMT provides a "*workflow*" *from a PRAM algorithm to an XMT program*, and, if desired to its performance-tuning. Given a problem, a PRAM-style parallel algorithm is developed for it using the Shiloach&Vishkin'82 Work-Depth (WD) methodology, very much in line with JaJa'90 and Keller, Kessler&Traeff'01. All the operations that can be concurrently performed in a 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: spawn and prefix-sum) for translating this basic concurrency to a multi-threaded program. The workaround is that 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.

This workflow workaround, with its multiple levels of abstraction, made a difference with respect to ease-of-programming. It allowed college freshmen (and even high-school students) to solve the same problems they get in typical freshmen serial programming course assignments using (XMT) parallel programming. On the other hand, even graduate students at the top of the class made embarrassing mistakes when they tried to shortcut the parallel algorithm stage.

For examples for advancing from WD to performance tuning, see Vishkin, Caragea&Lee'08.

2. The XMT Hardware Enhancements
Below we briefly review the XMT hardware features.
- A **prefix-sum functional unit**. See the appendix of Vishkin'97 and US Patent 6,542,918. Such a functional unit provides enhanced hardware implementation of a plurality of concurrent (atomic) Fetch-and-Add instructions, where the Add is limited to very small integers. It should be of independent interest to other parallel designs as it allows for fast coordination among parallel processes.
- **Extension of the von-Neumann program-counter + stored-program apparatus** plus **broadcast of SPMD (single-program multiple-data) code**, plus **independence of order semantics (IOS)**. The prefix-sum unit is used for several purposes including enhancing automatic allocation of thread IDs. These (mostly) control mechanisms are the heart of the XMT approach. See Naishlos et al'03, Wen&Vishkin'08 and US patents 6,463,527 and 7,523,293.
- **Prefix-sum to memory and reduced synchrony on-chip interconnection network**. US patent 6,768,336 and follow-up work in Balkan et al'07&'08.
- Most of the above features would benefit from a **uniform memory architecture (UMA), where during parallel execution no local (write) caches are used**.
- How to **build the memory architecture so that the many-core computer will not fall behind on either serial code or parallel code**. US patent application 20090119481 shows how to gracefully upgrade from a uniprocessor to up to thousands of processors on-chip without losing on backwards compatibility on serial code, and dynamically moving back and forth between serial and parallel execution. Note that GPU approaches tend to look at the parallel GPU unit as a co-processor; thus, off-loading the execution to the co-processor (and back) needs to be planned and its costs can be significant, which must limit their effective use of the parallel GPU unit.
- How to enhance the above using **nesting** of threads with hardware and software methods. For a limited input on this see US patent application 20090125907

Finally, a stable **compiler** (for programs written in XMTC) that builds on GCC Tzannes at al'06 is now part of our software release.

IV. SOME EVIDENCE
XMT is easy to build. A single graduate student, with no prior design experience, completed the XMT hardware description (in Verilog) of a 64-processor FPGA prototype in just over 2 years. XMT is also silicon-efficient. Our ASIC design indicates that a 64-processor XMT needs the same

silicon area as a (single) current commodity core. The approach goes after any type of application parallelism regardless of its amount, regularity, or grain size and is amenable to standard multiprogramming (i.e., where the hardware supports several concurrent OS threads).

We also demonstrated good performance, programmability and teachability. Highlights include: evidence of 100X speedups on general-purpose applications on a simulator of 1000 on-chip processors in Gu&Vishkin'06, and speedups ranging between 15X to 22X for irregular problems such as Quick-sort, breadth-first search (BFS) on graphs, finding the longest path in a directed acyclic graph (DAG), and speedups in the range of 35X -45X for regular programs such as matrix multiplication and convolution on the 64-processor XMT prototype versus the best serial code on XMT, in Wen&Vishkin'08. Caragea et al'09 demonstrates nearly 10X average performance improvement potential relative to Intel Core 2 Duo for a 64-processor XMT chip that uses the same silicon area as a single core.

The teachability of our approach has been extensively demonstrated. Over 100 students in grades K-12 have already programmed XMT and it even entered the regular syllabus of the year-long parallel computing course at Thomas Jefferson High-School for Science and Technology, Alexandria, VA. See Vishkin et al'09 for a recent presentation at the Computer Science for High School Workshop.

V. CONCLUSION

The algorithmic approach to designing a parallel system gave XMT some desired capabilities. The open question that this note starts to address is the understanding of the extent to which these capabilities can also be imported into other systems. We hope that ICCD system designers will be able to advance this opportunity further.

REFERENCES

[1] Explicit Multi-Threading (XMT): A PRAM-On-Chip Vision. Homepage: www.umiacs.umd.edu/~vishkin/XMT/

[2] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In Proceedings of the 4th international conference on Supercomputing, 1990.

[3] A. O. Balkan, G.C. Caragea, A. Tzannes, and U. Vishkin. Programmer's manual for XMTC language, XMTC compiler and XMT simulator. University of Maryland Institute for Advanced Compuer Studies, 2005-2009.

[4] A. O. Balkan, M. Horak, G. Qu, and U. Vishkin. Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing. In Hot Interconnects, Stanford, CA, 2007.

[5] A. O. Balkan, G. Qu, and U. Vishkin. An Area-Efficient High-Throughput Hybrid Interconnection Network for Single-Chip Parallel Processing. In 45th Design Automation Conference, Anaheim, CA, June 8-13, 2008.

[6] G. Caragea, B. Saybasili, X. Wen, and U. Vishkin. Performance potential of an easy-to-program PRAM-On-Chip prototype versus state-of-the-art processor. In Proc. ACM SPAA, Calgary, Canada, 2009.

[7] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In Principles Practice of Parallel Programming, pages 1-12, 1993.

[8] D. E. Culler, J. P. Singh, and A. Gupta. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, Inc, 1999.

[9] A. Formella, J. Keller, and T. Walle. Hpp: A high performance PRAM. In Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II, pages 425-434, London, UK, 1996. Springer-Verlag.

[10] P. Gu and U. Vishkin. Case study of gate-level logic simulation on an extremely fine-grained chip multiprocesor. J. Embedded Comp., 2:181–190, 2006.

[11] J. JaJa. An Introduction to Parallel Algorithms. Addison-Wesley, 1992.

[12] J. Keller, C.W. Kessler and J.L. Traeff. Practical PRAM Programming. Wiley-Interscience, 2001.

[13] D. Naishlos, J. Nuzman, C.-W. Tseng, and U. Vishkin. Towards a first vertical prototyping of an extremely fine-grained parallel programming approach. Theory of Computing Systems, Special Issue for SPAA 2001. Springer, 36:521-552, 2003.

[14] Y. Shiloach and U. Vishkin. An $O((n^{**}2)\log n)$ parallel max-flow algorithm. J. Algorithms, 3(2):128-146, 1982.

[15] A. Tzannes, R. Barua, G. Caragea, and U. Vishkin. Issues in writing a parallel compiler starting from a serial compiler, draft. Technical report, University of Maryland Institute for Advanced Computer Studies, 2006.

[16] U. Vishkin. From algorithm parallelism to instruction-level parallelism: an encode-decode chain using prefix-sum. In *Proc. 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[17] U. Vishkin, G. Caragea, and B. Lee. Models for advancing PRAM and other algorithms into parallel programs for a PRAM-On-Chip platform. In Handbook of Parallel Computing: Models, Algorithms and Applications. Editors: S. Rajasekaran and J. Reif. CRC press, 2008.

[18] U. Vishkin, S. Dascal, E. Berkovich, and J. Nuzman. Explicit multi-threading (XMT) bridging models for instruction parallelism. In Proc. 10th ACM symposium on Parallel algorithms and architectures SPAA, 1998.

[19] U. Vishkin, R. Tzur, D. Ellison and G.C. Caragea. Parallel programming for High Schools. Keynote presentation, CS4HS Workshop, Carnegie-Mellon University, July 2009, power point presentation available though the XMT home page.

[20] X. Wen and U. Vishkin. FPGA-based prototype of a PRAM-on-chip processor, ACM Computing Frontiers, Ischia, Italy, May 5-7, 2008.