Introduction to the XMT ("PRAM-On-Chip") Chip Multiprocessor

Uzi Vishkin

A June 2006 update of the motivation for the explicit multi-threading (XMT) project follows:
1. The main commodity processor vendors decided that multi-core designs are going to dominate the commodity processor market.
2. The number of cores is expected to double every 18 months for the next decade and reach 256 in a decade.
3. Clock rates of commodity processors have stopped improving since mid-2003. This followed several decades in which clock rates have doubled every 18 months.
To understand how power considerations led vendors to shift their focus away from seeking performance improvements for single-core applications towards multi-core ones, see presentation by Intel Fellow Geoff Lowney
https://conferences.umiacs.umd.edu/paa/lowney.pdf.
Since this change of focus went unnoticed by most[1], we are effectively witnessing a quite revolution in computing. Performance improvement can now come from only one source: utilization of chip multi-processing. But what will be needed for effective utilization? We have a comprehensive answer to this question, as the discussion below suggests.

Generally, computer system performance comes in two flavors: (a) Capacity: given many computational tasks, performance is measured in terms of the number of tasks that can be completed in a given time. (b) Capability: given a single computational task, its completion time is measured. The term capacity captures the fact that one could use several computers operating concurrently if one computer cannot complete enough tasks in a given time. While getting more computers should cost more money, there is no issue of technical feasibility. Capability, on the other hand, is much more challenging. For example, if a single simulation for testing the effect of a new drug in some environment requires a certain number of operations and the operations have to be done one after the other, as most computer programs work, more computers (i.e., money alone) will not help finish the simulation faster.
The question then becomes how to utilize chip multi-processing for shortening single task completion time.

The UMD XMT project that started in 1997 anticipated this utilization challenge. Next, we review the state-of-the-art in parallel computing at that time the XMT project began.

As it turns out, much work has been done, especially since the 1970s, on multi-chip multi-processing, when many thought that multi-chip multi-processors will soon

[1] Recognition of this change of focus is likely to become common knowledge soon. The reason is that the upcoming 4th Edition of the Hennessy-Patterson popular computer architecture textbook is presented as follows: "The era of seemingly unlimited growth in processor performance is over: single chip architectures can no longer overcome the performance limitations imposed by the power they consume and the heat they generate. Today, Intel and other semiconductor firms are abandoning the single fast processor model in favor of multi-core microprocessors--chips that combine two or more processors in a single package. In the fourth edition of *Computer Architecture*, the authors focus on this historic shift …"

dominate high-end general-purpose computing. Funding came from government and industry and most of it went to building many interesting parallel computer machines. While several of these machine architectures were quite clever, their general trust was that parallel programming would evolve to fit these architectures. However, while these architectures were well suited for some important applications, their weakest point was that it was difficult to program them. As an example for the recognition of this weakness, note that earlier in the current decade DARPA chose to entitle its main program in the area as high-productivity (to mean both low development-time and high performance), replacing the high-performance computing title used in previous decades. Since starting doing research in parallel computing in 1979 Dr. Vishkin opined that parallel algorithmic thinking is an "alien culture", which, in particular, is drastically different from the prevailing serial paradigm. Such thinking must be first established in order to later provide specifications for a parallel machine. He further opined that a mathematical model, called PRAM for Parallel Random-Access Model (or Machine), would be a proper framework.

The PRAM is a simple extension to the standard RAM (for random access machine) model used to teach serial algorithms in every standard Computer Science curriculum. The PRAM assumes, for simplicity, that any number of concurrent requests to memory can be processed within the same time as one request. The fact that this assumption is unlikely to closely match any machine in the future was again not a new idea for the field: the standard serial RAM also assumes that any operation, including memory access, takes the same time. In his PhD thesis, Vishkin proposed a simple "work-depth" methodology for designing parallel algorithms: Formulate your algorithm in the form of rounds, where each round can include any number of operations that could all be performed concurrently had there been enough hardware to execute them. For performance, the design objective should be to minimize two parameters: (i) work - the total number of operations over all rounds, and (ii) depth – the number of rounds. A simple example for such a parallel algorithm follows.

Assume that we seek to sum *8* numbers. The algorithm will proceed in *3* rounds. In the first round, *4* addition operations are performed: (1) add the 1st and 2nd elements, (2) add the 3rd and 4th elements, (3) add the 5th and 6th elements, and (4) add the 7th and 8th elements. In the second round 2 addition operations are performed on the results of the first round: (1) add the 1st and 2nd results, and (2) add the 3rd and 4th results. In the third and last round, add the two results of the second round. The number of rounds, or depth, in the above example is *3*, and the total number of operations, or work, is *7 (4+2+1)*. Simple serial summation of *8* numbers needs *7* serial operations. In other words, the total number of operations remains the same in the parallel algorithm, but the number of rounds is *3* instead of *7*. This advantage becomes much more decisive when a larger number of elements need to be summed. For example, the serial algorithm will need *1023* rounds to sum *1024* numbers, but it can be readily verified that the parallel algorithm needs *10* rounds (and a total of *1023* operations) to obtain the same outcome. In other words, the parallel algorithm provides a potential for obtaining the same outcome *100* times faster.

During the 1980s and early 1990s, the computer science algorithms research community developed the PRAM algorithmic theory leading to an algorithmic knowledge-base second in size only to serial algorithms.
During 1988-90 three main (standard) algorithms textbooks chose to include big PRAM algorithms chapters. The PRAM has become the model of choice for parallel algorithms in all major algorithms and computer science theory communities and was taught everywhere.
There was one problem. The bandwidth, namely the amount of data per time unit (e.g., number of bits per second) that can be moved between multi-chip multi-processors was limited and PRAM algorithms did not lead to fast programs on such multi-chip machines. The conclusion of many was that the PRAM theory is of no use. XMT is turning this conclusion upside down.

In 1997 it became clear to us that on-chip multi-processing is becoming possible.
The observation that on-chip bandwidth among multi-processors can be orders of magnitude higher than among multi-chip multi-processors led to developing a "PRAM-On-Chip" Vision.
Some of the main issues in translating this vision into a computer system architecture that can be realized were as follows:
- How to allocate operations that can be performed concurrently to the executing hardware? The above work-depth methodology provides algorithms that operate in a sequence of rounds; however, the number of operations in a round is independent of the amount of hardware available and the number of operations can vary greatly from one round to the next.
- How to build high throughput memories and interconnection networks? While we do not elaborate on this topic here, we note that technology leads toward memory architecture whose response time to a memory request is often hard, or even impossible, to predict. The reason is that cache memories are used and response time depends on whether the requested item happened to be cached. Interconnection network congestion brings about hard to predict dependences on other concurrent memory requests.
- PRAM algorithms are described one step at a time. The PRAM model prescribes how simultaneous accesses to the same memory location for read or write purposes are resolved, but does not state how such resolutions are implemented. In particular, PRAM algorithms often allow an "arbitrary concurrent write" resolution, where several concurrent attempts to write into the same memory location result in one of these writes, but we don't know in advance which one. Note also that such "semantics" cannot even be expressed in any of the common serial programming languages.
One idea was to get good performance in view of the memory response time predictability problem by providing a language which does not enforce a tighter order among concurrent activities than mandated by the algorithm and then efficiently implementing it.
- More generally, modern computer hardware operates as an assemblage of "finite state machines", or FSMs (a standard term in the technical literature). To get concurrently operating FSMs do as much work as possible, one would like to prevent a situation where any of them needs to wait on another. But this is impossible if they are to work toward a shared objective. How to get several FSMs to work in concert toward an objective if they

never need to wait for one another? The principled new idea behind the overall XMT design was to have the FSMs operate without waiting for one another to the extent possible. Specifically, for any point in time during the execution of any possible computer program, the design seeks to extract from the program the maximum freedom that the FSMs can get to operate without waiting for one another. We called this design principle “no-busy-wait finite-state-machine ideal”.
The following issues also needed to be addresses.
- How to provide competitive performance (with any serial processor) on serial code, as well as good speed-ups even when the amount of parallelism provided by the code is relatively small?
Historically, parallel architecture designers tended to worry about scaling up to larger amounts of parallelism, but not about scaling down to small amounts or even to offer competitive performance on serial code. The latter is often referred to as “backwards compatibility”. Backwards compatibility is very important: it appears to be the first issue on the mind of industry executives.
- All successful general-purpose computers since the 1940s rely on the so-called Von-Neumann apparatus. Is there a way to upgrade, rather than completely replace, this successful apparatus to handle parallelism? We allowed ourselves to mention here the von-Neumann apparatus, which is based on a stored-program accessed by a program-counter, since it is widely known and is often even studied in high-schools.
- Many application programmers tend to use application programming interfaces (APIs) such as MATLAB, Verilog/VHDL, OpenGL, and game development engines. Can code written for such APIs lead to good performance on a PRAM-On-Chip platform? Note that success in automatic extraction of parallelism from serial code written for performance on serial machines has been rather limited. But extracting parallelism from code written for such APIs could be easier. We already made significant strides in demonstrating that.

The UMD XMT project has developed a comprehensive answer to all these issues.

Performance gains exceeding a factor of 100 have been demonstrated in simulations for quite a few typical applications. Such gains should be possible already in 65nm technology.