What if DRAM is a Slow Peripheral?

Philip Machanick
School of IT and Electrical Engineering
University of Queensland
Brisbane, QLD 4072
Australia
Email: philip@itee.uq.edu.au

***Abstract*— The memory wall is approaching: the time when increases in processor speed will be masked by the high penalty of misses to DRAM. It would seem that the time when we must regard DRAM as a slow peripheral is some way off. But, given the possibility that the inherent latency problem of DRAM may not be solved, that possibility needs to be addressed. This paper presents some thoughts on how future DRAM may be organized, in the form of the proposed RAMnet architecture. RAMnet aims to provide alternative paths between processor and RAM, with a hierarchy of controllers with sufficient intelligence to optimize routing through the hierarchy. Key design goals include minimizing paths between components, and use of commodity components wherever possible.**

***Keywords*—memory wall, DRAM interconnect, DRAM latency, RAMnet memory architecture**

I. Introduction

The memory wall was first publicized in 1995 [15], and, since then, processor speed improvement has continued to follow Moore's Law (doubling the components available every year [12], resulting in a 50–100% speed improvement per year), while DRAM cycle time has continued to improve relatively slowly (7% per year).

In 2002, it was possible to purchase a commodity processor with a clock speed in excess of 2GHz, and the most aggressive commodity designs were 9-way superscalar. The average peak throughput rate of high-end commodity designs was better than 5 instructions per nanosecond, while commodity DRAM had a cycle time of around 30–40ns, resulting in miss penalties to DRAM in the hundreds of lost instructions.

Clearly, these numbers are not as significant as the cost of a disk access (5–10 milliseconds) or waiting for the network (also in the milliseconds), but DRAM is starting to display the attributes of a peripheral – enough latency to do other work while waiting for it – rather than a tightly integrated component of the CPU subsystem.

This paper reviews the problem briefly, proposes a solution in the form of the RAMnet architecture, and ends with some conclusions, including thoughts about future work.

II. The Problem

System designers continue to assume that DRAM can be fixed by small tweaks to the design. We have seen various models for adding cache to DRAM [7, 4], as well as various approaches at increasing throughput from DRAM [2, 4]. However, none of these approaches address the problem that a full memory access cycle has growing latency with respect to the processor.

One approach which does address latency, but only off-chip delays, is integrating a processor with DRAM on a chip, the IRAM approach [8]. IRAM does not eliminate the underlying latency trend, and has other drawbacks, including linking CPU and DRAM implementation, very different areas of design, which have not always been well done simultaneously by the same vendor.

Improvements to caches, too, will not solve the problem. Even a large (4MByte) fully-associative L2 cache only masks the effect of the memory wall to a limited extent, and does not scale significantly better in the face of the growing CPU-DRAM speed gap than a less aggressive L2 cache [10]. Approaches to improve associativity in conventional caches [5], therefore, will only stave off the problem, not make it go away.

The problem we have to face is that there will always be at least some references to DRAM which will incur the full latency of a DRAM operation. We can minimize this number, but, once minimized, it becomes a limit on CPU performance improvement.

III. A Proposal: RAMnet

Clearly, if there will always be some DRAM references whose latency can't be hidden, we need to look at strategies used elsewhere to deal with latencies too high to ignore. The obvious place to look is in the memory hierarchy – hence the RAMpage model [11], which proposes to move the whole hierarchy up a level, with the lowest-level SRAM functioning as main memory, and DRAM a paging device (backed up by disk as a 2nd-level paging device).

RAMpage gets its best results by taking context switches on misses to DRAM [10], and other approaches to having alternative work available for the processor, like simultaneous multithreading (SMT), have the potential to see similar gains [14, 9]. Hence, the only assumption made here is that the difficulties of bridging the CPU-DRAM speed gap will require that more than one instruction stream be active at a time and there could consequently be competing requests for DRAM.

While DRAM latencies remain orders of magnitude faster than disk latency, it is instructive to consider the way in which disk latencies have been minimized historically in high-end systems. Perhaps some of these ideas could be useful for future DRAM designs.

The remainder of this section examines how IBM designed a disk system for scalability and low latency, goes on to introduce an analogous approach to designing a RAM subsystem, and ends with a discussion of the proposed new architecture.

A. IBM Mainframe Disk System

Let's look at the IBM mainframe disk system [6], as an example of a classic design of a low-latency peripheral architecture.

The dominant philosophy was to choose latency over throughput wherever a trade-off had to be made. The reason for this was the view (often found to be valid in practice) that it's much easier to improve throughput by adding hardware than to reduce latency. To put it in another way, you can buy bandwidth, but you have to design for latency.

The subsystem was divided into the following hierarchies:

- *control* – the hierarchy of controllers provided a range of alternative paths between memory and I/O devices, and controlled the timing of transfers
- *data* – the hierarchy of connections was the path over which data moved

The hierarchy was designed to be highly scalable; each section of the hierarchy could contain up to 64 drives, and the IBM 3090/600 CPU could have up to 6 such sections, for a total of 384 drives.

High expandability and low latency are hard to achieve simultaneously; IBM achieved this by use of hierarchical paths to connect many devices. This, plus all the parallelism inherent in the hierarchy, allowed simultaneous transfers, so high bandwidth was supported. At the same time, using many paths instead of large buffers to accommodate a high load minimized latency.

This architecture was very successful, judging both from the performance it delivered and the duration of its dominance of the industry.

B. Design for DRAM as a Peripheral

So, what can we learn from this in designing future DRAM subsystems?

First, we should try to minimize additional latencies introduced by competing for the interconnect, by providing alternative paths. Second, we should design more intelligent memory controllers, to optimise competing requests. Finally, we should attempt to exploit device characteristics to minimize latency where possible. In a change from the days of the IBM design, current market realities dictate that using commodity DRAM and SRAM components should be the aim, since these components are ciritical to cost, and lack of economy of scale and any other overheads for custom components may result in a design failing to achieve general acceptance.

Let's now consider how a hierarchical architecture can be built with the desired properties. First, as with the IBM disk architecture, memory controllers should have some intelligence, and attempt to route competing requests to minimize delays. In addition, the controllers should have some knowledge of relative speeds of components of the system. For example, relatively close to the CPU, a memory module could be made up of SRAM. The controller hierarchy would attempt to cache requests in this faster module. The difference from a conventional off-chip cache is that more intelligence could be applied to managing the contents, much as with software management of caches [1, 5] or the RAMpage SRAM main memory [11].

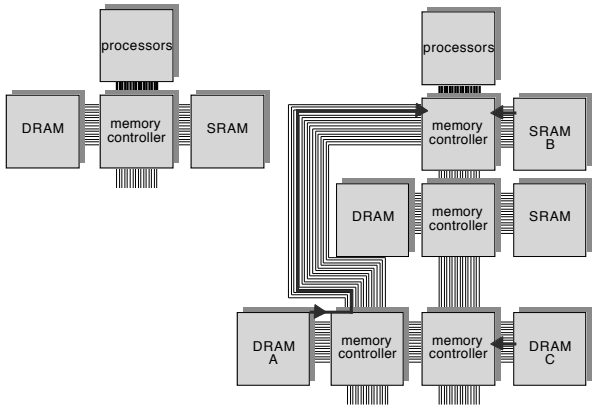

Fig. 1. A CPU and memory hierarchy arrangement showing variations on a hierarchy of controllers and memory modules. *The memory controller can interface to a memory module, a processor module containing one or more processors, or another memory controller (including a different design, e.g., to interface to an I/O bus).*

In Figure 1, two alternative arrangements of processors and memory are shown. The first has one module of SRAM and one module of DRAM attached to the processor module (which could contain multiple processors as well as cache). The second is a larger hierarchy with multiple levels of DRAM and SRAM. SRAM further from the processor would not be accessed as fast, but would still be faster than DRAM.

The somewhat random layout of the larger example is intended to illustrate a layout adantage of this model. Since there is no shared bus between all DRAM components, more choices as to board layout become possible. Further, since there is no shared bus, the path from the closest memory controller to the CPU can be made as short as possible, to minimize transmission delays. Taking this a step further, the interconnection layout in general can emphasize short paths between components especially those close to the CPU (those further may be permitted a longer path, with a corresponding increase in latency). The connection of the first memory controller to the CPU should be much faster than the rest of the interconnect since it would otherwise be a bottleneck. This faster interconnect would need to be designed to tighter tolerances than a conventional bus, but this could be achieved by putting this first controller and the processor module in a single package or multichip module (MCM) – as with some off-chip cache designs, like the Pentium II.

Figure 2 illustrates how an MCM could be packaged, with a processor module and a memory controller packaged together, and three interfaces to the memory controller going outside the package. It would also be possible to put one or two memory modules (e.g., an L3 cache) in the MCM package as a design option.

If there were alternative paths to the CPU, resulting in a higher aggregate bandwidth, RAM references could be routed

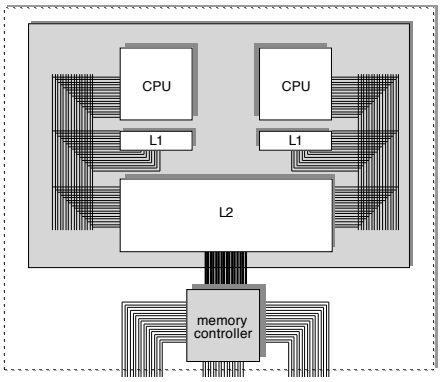

Fig. 2. A CPU and memory controller module in a single multichip module package. *For purpose of illustration, a 2-processor chip multiprocessor with a shared L2 cache is shown, but the detail of the processor packaging is largely irrelevant to the memory interconnect being proposed in this paper.*

through alternative paths, without requiring as high a bandwidth interconnect at all levels. For example, if the DRAM labeled A in Figure 1 has already started a transfer when the SRAM at B starts a transfer, B can go ahead, because the only part of the route it has in common is the fast interconnect linking the closest memory controller to the CPU. If in the meantime the DRAM at C needs to start a transfer, it can go ahead without conflict with the existing transfers.

Clearly, for all this to make sense, it should be possible to partition traffic between components of the hierarchy with alternative routes to the CPU. With a chip multiprocessor [13], it would, for example, be an option to allocate RAM used by each processor to a different RAM chip.

The next point to consider is what should go in the memory controllers. A processor core has become a relatively cheap component. For example, the latest generation of Xilinx FPGAs comes with the option of up to 4 microprocessor cores[1] integrated into an FPGA chip. It seems reasonable therefore that a memory controller should include a microprocessor core, which could manage placement of pages and optimal routing of requests through the memory system. However, managing page placement and routing of requests purely in software will add to latencies, especially when the memory subsystem is heavily loaded. To cover heavy load, hardware routing of requests would be preferable. Depending on available chip space, a hybrid solution with hardware routing and software which optimizes routing tables or other state information at times of low load should be possible.

It would be useful to separate control and data paths, to maximize opportunities for parallelism. Using a relatively narrow data bus would make this feasible, at the expense of requiring extra bus transactions. However, Rambus has shown that driving a narrow bus at high speed is a feasible option [2]. In this design, the short interconnects should make design of a high clock speed on a narrow bus easier than for Rambus.

Finally, the memory controllers should take advantage of the property of DRAM that an access takes time to set up, but sequential references can be fast, given burst modes of current DRAM implementations like SDRAM and Rambus [3]. References from different RAM modules should be pipelined, with transfers using the time when another module is waiting for its addressing to be set up.

Given the network-like features of the design, it is named *RAMnet*.

C. Advantages of the RAMnet Design

The design contains some elements of existing designs, particularly Rambus [2]. Like Rambus, it uses a relatively narrow interconnect to save cost. Also like Rambus, it supports multiple RAM transactions being pipelined, if they do not conflict. Unlike Rambus, it can use commodity RAM chips and packaging, and is designed to scale up without significant design changes.

Like IRAM [8], it proposes placing more intelligence close to DRAM, but that intelligence (in the memory controller) is intended to route responses to requests for DRAM more efficiently, rather than to move processing to DRAM. The advantage in the RAMnet approach is that it decouples CPU and DRAM implementation. If a processor is used in the memory controllers, it need not run the same instruction set as the main CPU module (and in fact would be transparent to the CPU module).

Scalability is a key consideration in the design. It is important that "scalable" designs scale to small configurations as well as large, so that they potentially have a mass market, to underwrite development costs. The minimal version of this design is one with a single memory controller, and one to three SRAM or DRAM modules. Such a design should in principle be comparable in cost to standard RAM designs, provided that the controller cost could be kept low.

The biggest win in the design for the implementor is that it avoids the need for a long high-speed bus. All interconnects are short, and only the connection between the memory controller closest to the CPU and the processor needs to be relatively fast. As discussed before, that interconnect could removed from the board design by packaging it in an MCM with the processor module.

IV. Conclusion

This paper has presented some ideas for implementing a scalable RAM system, RAMnet, based on ideas from a highly scalable disk system. The major design considerations are minimizing board-level interconnect distance, providing a variety of options for minimizing latency including alternative paths through the hierarchy and pipelining operations where possible, and providing flexible support for a range of different price-performance trade-offs.

Clearly, if market acceptance is to be facilitated, the design should look as close as possible, from the CPU's perspective, to an ordinary DRAM. Organizing which memory contents goes where, therefore, should ideally be the task of the controllers, without any special information from the CPU or CPUs. Investigation of how to achieve this goal would be an important

[1] See `http://www.xilinx.com/ipcenter/` for some options.

research goal in validating the practicality of the design.

Further, the design of a suitable switching strategy which would minimize latency, yet support software intervention, would be another key issue to be investigated.

While significant details remain to be worked out, the RAMnet idea appears to be feasible, and the potential benefits make it worth working through the missing details.

References

[1] D.R. Cheriton, G. Slavenburg, and P. Boyle. Software-controlled caches in the VMP multiprocessor. In *Proc. 13th Int. Symp. on Computer Architecture (ISCA '86)*, pages 366–374, Tokyo, June 1986.

[2] R. Crisp. Direct Rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November/December 1997.

[3] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. Performance comparison of contemporary DRAM architectures. In *Proc. 26th Annual Int. Symp. on Computer Architecture*, pages 222–233, Atlanta, Georgia, May 1999.

[4] B. Davis, T. Mudge, B. Jacob, and V. Cuppu. DDR2 and low latency variants. In *Solving the Memory Wall Problem Workshop*, Vancouver, Canada, June 2000. In conjunction with 26th Annual lnt. Symp. on Computer Architecture.

[5] Erik G. Hallnor and Steven K. Reinhardt. A fully associative software-managed cache design. In *Proc. 27th Annual Int. Symp. on Computer Architecture*, pages 107–116, Vancouver, BC, 2000.

[6] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffmann, San Francisco, CA, 1st edition, 1990.

[7] Hideto Hidaka, Yoshio Matsuda, Mikio Asakura, and Kazuyasu Fujishima. The Cache DRAM architecture: A DRAM with an on-chip cache memory. *IEEE Micro*, 10(2):14–25, March/April 1990.

[8] C.E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanović, N. Cardwell, R. Fromm, J Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick. Scalable processors in the billion-transistor era: IRAM. *Computer*, 30(9):75–78, September 1997.

[9] J.L. Lo, J.S. Emer, H.M. Levy, R.L. Stamm, and D.M. Tullsen. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Trans. on Computer Systems*, 15(3):322–354, August 1997.

[10] P. Machanick. Scalability of the RAMpage memory hierarchy. *South African Computer Journal*, (25):68–73, August 2000.

[11] P. Machanick, P. Salverda, and L. Pompe. Hardware-software trade-offs in a Direct Rambus implementation of the RAMpage memory hierarchy. In *Proc. 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 105–114, San Jose, CA, October 1998.

[12] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 19 April 1965.

[13] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-7)*, pages 2–11, Cambridge, MA, October 1996.

[14] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *Proc. 22nd Annual Int. Symp. on Computer Architecture (ISCA '95)*, pages 392–403, S. Margherita Ligure, Italy, June 1995.

[15] W.A. Wulf and S.A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.