Notepad memory: NUMA all the way up

Gabriela Barrantes, Hajime Inoue, Josh Karlin
Department of Computer Science
University of New Mexico
{gbarrant, hinoue, karlinjf}@cs.unm.edu

July 30, 2004

1 Problem: The Memory Wall

Most papers about the memory wall begin with a sentence very similar to this:

> "We all know that the rate of improvement in microprocessor speed exceeds the rate of improvement in DRAM memory speed – each is improving exponentially, but the exponent for microprocessors is substantially larger than that for DRAMs" [3].

The papers then describe their solution. This is most often a modified cache structure or a new prefetching scheme. These are simply variants of the agreed-upon solution to the memory wall: a large L2 cache.

The current state-of-the-art L2 cache is a beast indeed.[1] It occupies some 60% of the processor, [2] and contains a variety of kludges to decrease its miss rate and miss penalty: victim caches, hardware prefetching, sub-block placement, and critical-word first. This solution brings to mind the Ptolomean model of the solar system: it gave accurate predictions, but it was overly complex (aside from being wrong).

We believe the memory wall is a problem with a simple and obvious solution: make memory as fast as the processor. Making all of memory as fast as processor is too expensive, of course, but that is not what we propose. Simply make some of memory fast, and that will be enough.

2 Solution: Notepad memory

We call this memory Notepad memory (NPM), because it is like scratchpad memory (SPM) but for general-purpose systems. The last general computer that used SPM was the PDP-10 which was introduced in 1968 [1]. Notepad memory, together with a few current trends in operating systems and languages form a synergistic solution to the memory wall problem.

Our proposal is to eliminate all on-chip cache logic and use it instead as directly addressable memory. Notepad memory is this memory, addressed, for example, as the first megabyte of memory. This is simply a memory address range that runs at processor speeds. *It is up to the software to manage it so it runs at optimal performance.*

We further simplify it by eliminating the virtual memory infrastructure. The processor doesn't support it. Instead, we place the burden of memory management squarely on the entity most able to predict its resource needs (no, not the programmer) – a higher-level virtual machine.[3] Moving applications from statically-compiled images to emulators and interpreters is a clear current trend, as exemplified by the increasing popularity of Java, Microsoft's Common Language Infrastructure, Cross Architecture emulators such as Transmeta's Crusoe, and even native-to-native emulators such as Valgrind.

We believe our design is better than an L2 cache because it is simpler. It is the RISC philosophy brought to the memory hierarchy. We do away with context switches, and with the overhead and die infastructure required for virtual memory. Our design removes the need for associativity, eviction policies, buffers, and prefetching strategies. Our processor is both easier to design, cheaper to produce, and less power-hungry.

The tradeoff is in the runtime, of course. The responsibilities of the L2 cache are now handled by the virtual machine. The virtual machine does the profiling required to determine the proper residency of working sets and data in the NPM. The allocation/garbage collection system optimizes locality.

Our solution seems to be a radical departure from modern architectural designs, yet it is not. It is merely a simplification and extension of current trends. We admit there is work to be done in making such a design feasible. Perhaps the Notepad memory could emulate cache for those programs unable to run at acceptable speeds. Much work is also required for compiler design and profile-directed optimizations to benefit from the Notepad design. Perhaps NPM can borrow from current NUMA designs. Perhaps novel architectural features such as LOAD history buffers could be used to assist. These features, instead of hiding the non-uniformity of memory, would aid in the exploitation of its heterogeneity. The world will always look non-uniform. We need to get used to it.

References

[1] *DECSystem-10/DECSystem-20 Processor Reference Manual AD-H391A-T1*, updated 5th edition, June 1982.

[2] John Hennessy and David Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2 edition, 1996.

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

[1] Yet, remarkably effective. Less than 1% of loads are misses for many applications [2].

[2] L2 size based on visual observation of the Pentium M die.

[3] The programmer could also control the NPM directly.