Virtualizing Physical Memory

The Impulse Group
University of Utah

Main memory is the performance bottleneck for a growing number of programs. The gap between CPU speed and memory latency has grown to the point where applications must have very good spatial and temporal locality to achieve anything close to the peak performance of a processor. Architects have increased the size of cache lines to amortize the cost of accessing main memory, but for some programs such a decision only makes matters worse. For example, it is common for "vectorizable" numeric applications, data mining, and multimedia codes with sparse access patterns to access only one word per cache line. On conventional memory architectures, the problem is unavoidable because cache lines are filled from contiguous physical addresses. If cache lines are 128 bytes and the application only uses a single doubleword from each line, under 7% of the system bus bandwidth is utilized effectively. For such applications, performance can be orders of magnitude below the processor's peak performance.

One solution to the problem of sparse data was employed by vector supercomputers – use vector load/store units and all-SRAM main memories. Obviously, such a solution is impractical for commodity systems. Another solution is to support non-coherent, uncached loads of sparse data streams, but such a solution is bandwidth inefficient and only succeeds if no reuse exists. The best solution would allow applications to control at a fine grain exactly what data is loaded into the cache and what data is evicted. Unfortunately, modern caches load and evict contiguous physical addresses, employ fixed caching policies, and accept few "hints" from applications about what to cache. One way to give programs fine-grained control over how the cache works would be to employ significant amounts of reconfigurable logic in the processor core and on-chip memory hierarchy. While intriguing, this solution would probably increase the base processor cycle time, which could decrease overall performance.

Our solution is to redefine what is meant by a physical address. When a processor loads a cache line, there is no inherent reason that the load *must* be satisfied from contiguous cells of physical memory (DRAM). It is only a convention, albeit one never challenged, that physical addresses correspond directly to cells of physical memory. In the Impulse project, the memory system can be configured to reinterpret memory requests from the processor — in particular, it can translate physical addresses before deciding what DRAM cells to read (or write) in response to the processor's request. This seemingly simple change to what is meant by a physical address can be exploited in a number of ways. For example, programs can download scatter/gather commands to the memory controller, similar to those supported by vector load/store units. Then, when the processor cache loads a contiguous line of "physical memory", the memory controller can form the line from discontigous physical memory locations. As another example, if a program accesses a column of a row-major array, the program could have the memory controller create a physical alias for the column. Once you reject the notion that there must be a direct mapping from physical address to DRAM cell, programs can have fine-grained control over what data is stored in each level of the processor cache.

We have performed some initial experiments into the performance benefits that this fine-grained control over physical addressing provides. In particular, conjugate gradient (CG) from the NAS benchmark suite performs quite poorly on conventional microprocessors because of the "random" nature of its memory accesses. In contrast, vector supercomputers have no problem getting very high performance out of CG, because they can gather-load CG's "random" vectors into vector registers without wasting memory bandwidth or cache space. When we simulated CG with an Impulse memory system, we found that it increased the performance of a fairly pedestrian microprocessor by two-thirds. If we added Impulse to a system with an aggressive superscalar processor, the relative improvement of using Impulse would be considerably higher. We are now looking at other applications that might benefit from Impulse's physical address retranslation (e.g., other vector-friendly applications from the NAS suite and a data mining application) to determine the breadth of its applicability.