Advanced Parallel Computing: Exercise #2

Due on Tuesday, May 5, 2015

Svend Dorkenwald, Günther Schindler

Inhaltsverzeichnis

Reading

The Future of Microprocessors

The authors of 'The Future of Microprocessors' analyse the parallelism in microprocessors which is needed to keep pace with the predigtion of Moore's Law. They show, that existing concurrency techniques like pipelining and superscalar processors are limited in their improvement due to limited istruction parallelism and cooling limitations. They claim that without more radical changes in processor design, microprocessor performance increases will slow dramatically in the future.

The authors solution is to improve in chip multiprocossors instead of seperate multicore systems with a focus on throughput performance. They argue that latency-critical software has to be parallelized into multiple parallel threads of execution to really take advantage of a chip multiprocessors. This parallelization needs to be more generic for unexperienced programmers so software can utilizes the short interprocessor communication latencies effectively.

Retrospectively, the authors were absolutly right about the 'future' of microprocessors. Their radical approach away from the traditional von Neumann architecture and seperated multiprocessors is todays common way. Therefor we give this paper a strong accept.

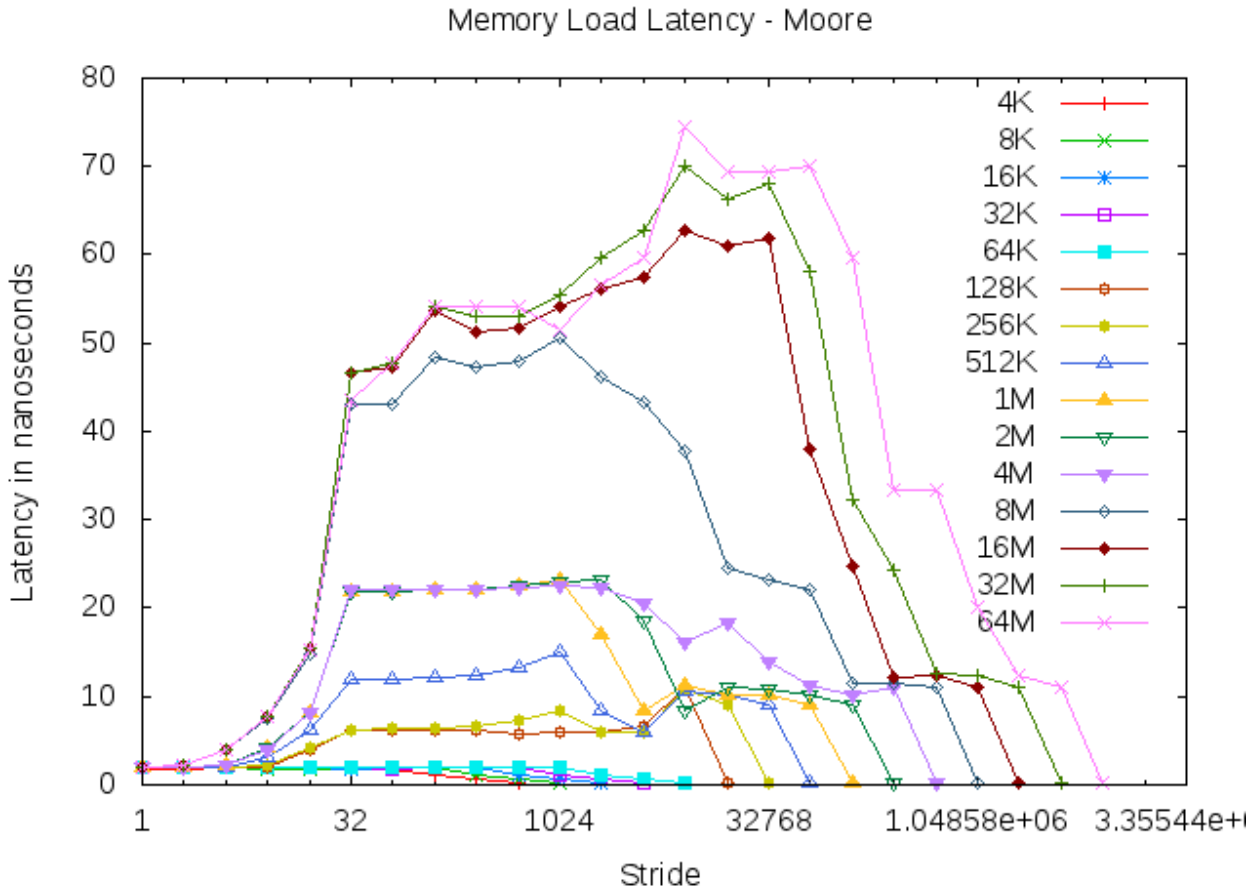
Software and the Concurrency Revolution

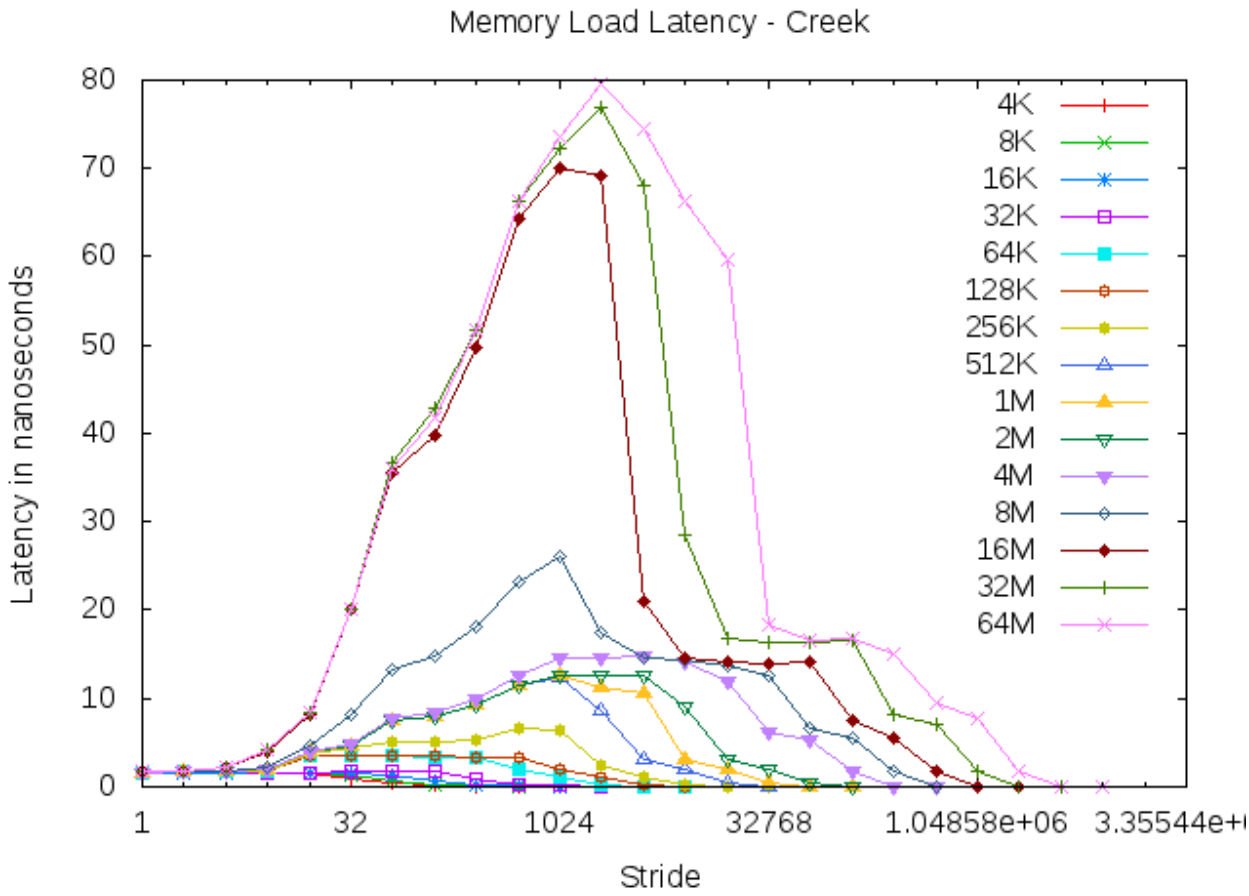
Sutter and Larus focus in this paper on the implications of concurrency for software. Concurrency was becoming more important since CPU manufactures were starting to produce multicore CPUs with often reduced clock speeds for each core. Besides performance they name and improvement of responsiveness by performing work asynchronously as second reason to program concurrently .
Their analyses leads to the conclusion that programmers are very likely to be overburdened by writing and debugging concurrent programs. Therefore, better tools need to be developed which itself is a difficult task. In terms of programming models they state that current models do not provide sufficient options to handle unstructered parallelism, specifically shared states. Especially locks introduce many new problems making them to a bad option.
The authors address important topics which even may have prevented the earlier introduction of multicore architectures. Although defining several features of better suited programming models and debugging tools they fail to give answers and ideas. Hence, this paper might only be helpful for outsiders and newbys to understand the predominant problem.
Nevertheless, we give this paper a week accept.

Pointer Chasing Benchmark

As part of this exercise, we should analyse the results of the Pointer Chasing Benchmark on the Moore system, as well as on another system. We chose therefor one node of the Creek system.

Following plots expose the results of both experiments.


Memory Load Latency - Moore
Latency in nanoseconds
Stride
0
10
20
30
40
50
60
70
80
1
32
1024
32768
1.04858e+06
3.35544e+
4K
8K
16K
32K
64K
128K
256K
512K
1M
2M
4M
8M
16M
32M
64M



Memory Levels, Cache Size and Associated Latency

The graph from the Moore system shows five distinct sets of curves, corresponding to the different memory levels. The lowest curve shows the behaviour of the l1-Cache. Since the biggest fitting array is 64kB, we expect the size of the L1-Data cache is 6kB. The inherent latency is about 1.9nsec. The next two curves belong to the L2-Cache which size is, according to the fitting arrays, 512kB. The gap between 256kB and 512kB is due to a cache miss. Corresponding latency is about 6-7ns for the L2-Cache. According the the fourth curve, the L3-Cache is 4MB in size. Associated latency is approximatly 22ns. The last curve defines the access time of about 53ns for the main memory.

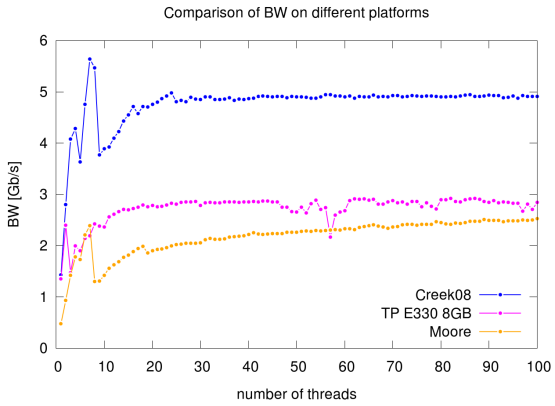
Similiar to the description above, we guess the size of the L1-Cache on Creek to 32kB by a latency of 1.6ns. The size of the L2-Cache is 256kB by a latency of 3.5ns. The size of the L3-Cache is 8MB by a latency of 14ns.

Associativity of the caches and the TLB

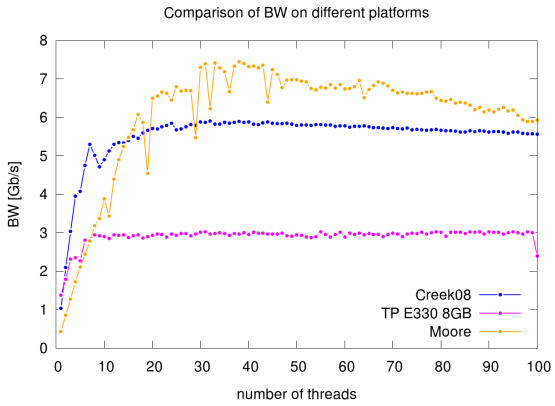
We can tell that the cache on Moore has an associativity of 2 because of the drop when the stride was half of the array size, since the two addresses being accessed would fit in a single set. If the cache would be direct mapped, there would be no drop of the access time at this point.
Since the same drop occurs on Creek at a stride of a quarter of the array size, we guess a 4-way associativity.

Multi-threaded load bandwidth

In this section we tested the performance of three devices: Creek08, Moore, ThinkPad E330 (8GB RAM). Therefore, we loaded an amount of data with different numbers of threads from RAM and measured the time until the last thread finished. To increase the quality of the measurement we repeated each of them for five seconds and averaged the times.
For the first measurement we loaded 4MB per thread from RAM (Abbildung 1a) and for the second 1GB in total (Abbildung 1b).



(a) Bandwidth comaprsion for loading 4MB per thread from RAM.



(b) Bandwidth comparison for loading 1GB from RAM.

Surprisingly, Moore performs worse for the first scenario. It seems as if the bandwidth on Moore is highly dependable on the data size whereas this is barely not the case for the two other platforms. This is also supported by the decline of the BW-curve for Moore in Abbildung 1b. In general the maximum bandwidths reported by Intel for the different platforms were not reached. Probably, the speed of the RAM slots is the main bottleneck. These theoretical numbers may also only be reached for very large data sizes.