Analyzing Complex Memory Systems*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Neil Butcher
Dept. of Computer Science
University of Notre Dame
South Bend, IN, USA
nbutcher@nd.edu

***Index Terms*—component, formatting, style, styling, insert**

Several recent systems in the Top500 include manycore chips with complex memory systems, including multiple memory channels. Many manycore chips feature an intermediate layer of memory with higher bandwidth and lower capacity than main memory. The intermediate memory can be configured as either a cache or a separate address space. This paper uses Intel's Knights Landing (KNL) processor as a testbed, it includes both intermediate memory and multiple architectural knobs to adjust affinity. We present cache-oblivious and chunking algorithms for sort, matrix-multiply and Fast Fourier Transforms (FFT), and compare to state of the art codes. Experimenting with a wide range of problem types and algorithmic solutions gives insight into how affinity can affect performance. Chunking often achieves low utilization of the memory system as the cost of adding threads to move data outweighs the benefit of improved bandwidth The results achieved with straight-forward cache-oblivious codes are competitive with state of the art codes.

There are two major trends in processor architecture for highly parallel systems. First is the expansion from multicore to "manycore',' with up to hundreds of cores per chip. Second, many HPC systems are beginning to include additional layers of memory between the main memory and the top of the cache hierarchy. Such "multi-level memory" systems are referred to here as ***MLM***, with the additional levels of memory denoted as ***Intermediate Memory*** (IM). IM can operate as a separate address space or as a large cache.

Along with these changes has been the need to provide more memory bandwidth, especially because the number of cores on a single chip are increasing rapidly. This has resulted in rapid increases in the number of physical memory channels, both to the conventional DRAM main memory and to the IM, especially when the latter is implemented by technologies such as 3D memory stacks. This growth in both channels and cores means that the directory-like structures needed to translate between logical and physical addresses, and to handle cache coherency requests are getting more complicated, and must themselves be partitioned and distributed around the processing chip. Thus the central question raised by this paper is whether or not it matters how such structures are distributed, and what is their relationship to physical cores and physical memory channels.

The three benchmarks we look at in our work are sorting, dense matrix multiply, and Fast Fourier Transforms (FFT). These benchmarks provide a balance of different problem types. Sort is bandwidth bound. Dense matrix multiply is a compute-bound problem limited by the number of floating-point operations per second, but has some interesting characteristics when the matrices are non-square. 2D FFT transforms include non-uniformly strided memory accesses and floating-point operations.

We utilize two different algorithmic strategies to make use of IM. The first strategy we use is cache-oblivious algorithms [2]. A cache-oblivious algorithm aims to make efficient use of the cache without explicit knowledge of the cache size. Cache-oblivious algorithms often use a divide-and-conquer strategy that recursively creates subproblems in a depth-first manner. More recent work adapts Cache oblivious algorithms to consider multicore machines. The focus of these algorithms focused on performing efficiently on 8-16 core machines. The manycore cache-oblivious algorithms presented in this work also create independent subproblems with minimal synchronization and scale to a more significant number of cores. They are designed to perform well on real machines.

The other algorithmic strategy we employ is chunking. Buffering assumes two memories, a large, slower "main memory" and smaller IM memory that provides a bandwidth improvement. Conventional buffering techniques break the input into chunks approximately the size of the fast memory. A single chunk is moved into fast memory. Once in fast memory the subproblem is computed and then moved back to main memory. Once all the chunks have been computed, the chunked solutions are merged to produce a final result. A similar strategy has been implemented successfully on multicore processors in the work of Popovici et. al. [1]. Popovici acheives speedup of 1.2-3x speedup over FFTW and MKL by utilizing some of the threads to perform 'soft DMA' operations.

We perform create chunking and cache-oblivious algorithms for sort, matrix-multiply, and 2D FFT. For each of these problems, we vary the problem size to gain a complete picture of the impact the memory system has on these algorithms.

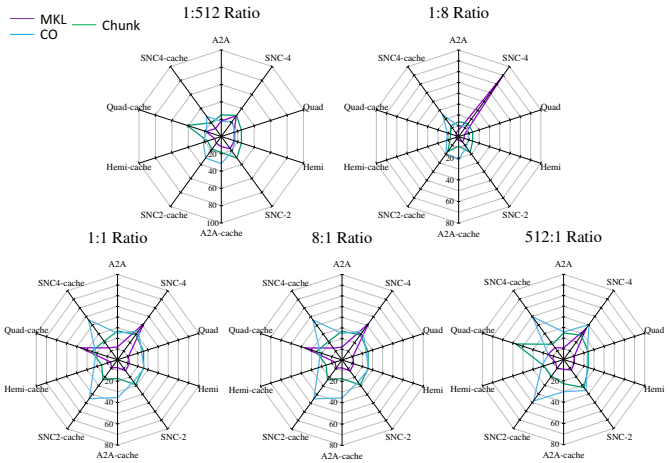

Fig. 1. Performance of FFT across each of the clustering modes, run with no MCDRAM and MCDRAM in cache mode

Matrix-multiply and FFT take a 2D mxn matrix as input, we adjust the shape of these matrices with a ratio r, m=n*r. By adjusting the size and ratio we get a more complete picture of the impact of the memory system.

Our baseline system uses Intel's PHI 7250 (a.k.a. "Knight's Landing" (KNL)) as its processor. This chip has 34 "tiles" containing two cores, a 1MB shared L2, and a "Caching Home Agent" (CHA) that serves as a directory for coherency traffic for some part of the address space. Two memory controllers and six conventional DDR4 memory ports (three per controller) provide roughly 96GB of storage with 90GB/s of bandwidth.

The KNL has a configurable IM called MCDRAM implemented with eight 3D stacks of DRAM chips, each with 2GB capacity and a separate port into the processor chip. Two such ports are physically close to each quadrant of the chip. The behavior of the MCDRAM is adjusted in the BIOS at boot time and can act as either an extension to the main memory ("flat mode"), a large L3 cache ("cache mode"), or both ("hybrid"). The aggregate MCDRAM has roughly 16GB of storage and provides approximately 400GB/s bandwidth [3].

We compare the execution time of the different FFT algorithms in Fig. 1. The "thread-buffering IM" strategy runs in flat mode with the buffers allocated to MCDRAM. The rest of the points on the radar plot use the MCDRAM in cache mode. The best performance for the square case is MKL in quadrant mode, giving a relative speedup of roughly 4x over the cache-oblivious algorithm. These speedups vary in magnitude with smaller problem sizes of the same ratio. The general trend is that the CO algorithm benefits most from larger problem sizes. The cache-oblivious strategy appears to perform slightly better than the thread buffering strategy, with speedups ranging from 1-1.5x.

We see significantly less variation in performance in hemisphere than in quadrant mode. The square ratio, MKL, is a speedup of roughly 4x compared to the CO algorithm. For 512 and 1/512 ratios, MKL is roughly 2x faster than cache-oblivious and 3x for the 8 and 1/8 ratios. The square problem is half the size of the other ratios, resulting in a lower execution time.

The thread-level buffering algorithm consistently gains performance by managing the IM memory explicitly, particularly in A2A IM, Quad IM, and Hemi IM. Using dedicated copy threads to move data into IM buffers increases the overall bandwidth utilization and improves performance. The effectiveness of copy threads demonstrates that FFT is a bandwidth-limited problem. The thread-level buffering strategy does not appear effective in SNC-2 or SNC-4 mode, even with the buffers allocated to IM. We ensure the buffers are in the same quadrant as the threads utilizing them, but the incoming data to be copied could reside in any quadrant. The thread-level buffing algorithm in hemisphere mode gains performance of roughly 1.2x-1.4x. Since most of the increased bandwidth utilization comes from the copy threads, thread-level buffering does not exhibit the same performance gains as the other strategies.

The cache-oblivious algorithm gains consistent speedup when running in cache mode. The exception is that the 512 and 1/512 ratios in quad cache and the square problem in SNC-2 cache do not improve. The cache-oblivious algorithm performs consistently across all the modes. This consistency across modes suggests that the code may be portable to other MLM architectures. The benefit from configuring the IM as a cache in hemisphere mode ranged from 1.4x-1.6x.

Our work provides a few key observations about the design of memory systems—the benefit of IM changes with clustering mode and algorithm (equivalent non-cache mode). We list the speedup of the algorithm in flat mode versus cache mode. CO Sort has a speedup range of 1.55 - 1.78x and MKL Matrix-multiply ranges from 0.64 - 1.68x, MKL FFT ranges from 0.05 - 16x. In some cases, matrix multiply and FFT lose performance by adding an IM cache. We demonstrate by comparing the fastest clustering mode with the slowest clustering mode, MKL Square FFT – 1 – 16.4x. Cache Oblivious algorithms port effectively across clustering modes, but chunking algorithms appear ineffective on the KNL architecture.

REFERENCES

[1] Popovici, Doru Thom and Low, Tze Meng and Franchetti, Franz "Large bandwidth-efficient FFTs on multicore and multi-socket systems". *22018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 1:379–388, 2018.

[2] Frigo, Matteo and Leiserson, Charles E and Prokop, Harald and Ramachandran, Sridhar "Cache-oblivious algorithms". *2020 IEEE International Conference on Cluster Computing*, 1:523–530, 2020.

[3] Odajima, Tetsuya and Kodama, Yuetsu and Tsuji, Miwako and Matsuda, Motohiko and Maruyama, Yutaka and Sato, Mitsuhisa "Preliminary Performance Evaluation of the Fujitsu A64FX Using HPC Applications". *Foundations of Computer Science*, 1:285–297, 1999.