

ERDC/ITL TR-19-8

Information Technology Laboratory



**US Army Corps
of Engineers®**
Engineer Research and
Development Center



Initial Benchmarking of the Intel 3D- Stacked MCDRAM

Benjamin S. Parsons

September 2019

The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdcenter.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at <http://acwc.sdp.sirsi.net/client/default>.

Initial Benchmarking of the Intel 3D-Stacked MCDRAM

Benjamin S. Parsons

*Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199*

Final Report

Approved for public release; distribution unlimited.

Prepared for Headquarters, U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Project number H29B80

Abstract

Modern, manycore processors are increasingly using high-bandwidth memory (HBM) to provide the necessary memory bandwidth to high core counts. The Knights Landing processor is one example, which is paired with a three-dimensional (3D) stacked memory, and a multi-channel direct random access memory (MCDRAM). The MCDRAM offers high-bandwidth memory, however, it is also important to understand the latency of the memory and how the memory will respond to noncontiguous accesses. This work provides information to programmers about how their applications will perform, and which of the many memory settings that will provide the best performance. These settings are non-trivial and can have a large impact on application performance. In addition to several micro-benchmarks, this work shows that using the MCDRAM can provide speedups up to 3.7x for a congregant gradient application, while giving slowdowns of 3x for certain hash table implementations. Given this range of performance, it is essential for programmers to understand this memory technology and how to use it appropriately.

DISCLAIMER: The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners. The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

Contents

Abstract.....	ii
Figures and Tables	iv
Preface.....	v
Acronyms and Abbreviations	vi
1 Introduction.....	2
1.1 Background.....	3
1.2 Objectives.....	3
1.3 Approach	4
1.4 Scope.....	4
2 System Specification and Configurations	5
3 Benchmarks.....	8
3.1 Stream.....	8
3.2 LMbench	8
3.2.1 Lat_mem_rd.....	8
3.2.2 Bw_mem	9
3.3 Google hash table.....	9
3.4 Intel benchmarks	9
4 Results.....	10
4.1 Stream.....	10
4.2 LMbench latency.....	12
4.3 LMbench bandwidth	16
4.4 Hash tables	17
4.5 Single node LINPACK.....	19
4.6 Conjugant gradient	20
5 Programmer/Administrator Recommendations	21
6 Future Work.....	23
7 Conclusion.....	24
References.....	25

Report Documentation Page

Figures and Tables

Figures

Figure 1. Knights landing tile.	5
Figure 2. Knights Landing architecture.	6
Figure 3. Flat mode DRAM - Stream bandwidth vs. number of threads, 400 MB buffer.....	11
Figure 4. Cache mode - Stream bandwidth vs. number of threads, 400 MB buffer.	11
Figure 5. Flat mode MCDRAM - Stream bandwidth vs. number of threads, 400 MB buffer.....	12
Figure 6. Flat mode DRAM, memory latency vs. buffer size, 4 MB buffer.	13
Figure 7. Cache mode, memory latency vs buffer size, 4 MB buffer.....	14
Figure 8. Flat mode MCDRAM, memory latency vs. buffer size, 4 MB buffer.	15
Figure 9. Memory latency vs buffer size, flat mode MCDRAM, 64 GB buffer.	15
Figure 10. Total memory bandwidth vs. number of processes, 4 MB buffer.	16
Figure 11. Single processor memory bandwidth vs. number of processes, 4 MB buffer.....	17

Tables

Table 1. Stream max bandwidth and speedup.	10
Table 2. Hash operation latency (ns) and speedup (DRAM/other), slowdowns shown in bold.	18
Table 3. Intel single node LINPACK.....	19
Table 4. Intel multi-processor LINPACK, GFLOP/s for various configurations.	19

Preface

This study was conducted for [the Name of Sponsoring Organization] under Project number H29B80, “Supercomputing Research Center” (SRC).

The work was performed by the Supercomputing Research Center (SRC), U.S. Army Engineer Research and Development Center, Information Laboratory (ERDC-ITL). At the time of publication, Mr. Robert M. Hunter was Chief, SRC; the Technical Director was Dr. Robert M. Wallace. The Deputy Director of ERDC-ITL was Ms. Patty Duett and the Director was Dr. Dr. David A. Horner.

COL Teresa A. Schlosser was Commander of ERDC, and Dr. David W. Pittman was the Director.

Acronyms and Abbreviations

Acronym	Meaning
CPU	Central Processing Unit
DoD	Department of Defense
DRAM	Dynamic Random Access Memory
ERDC	Engineer Research and Development Center
GB	Gigabyte
GB/s	gigabits/second
GFLOPS	gigaflops
GHz	gigahertz
IO	input/output
ITL	Information Technology Laboratory
KB	Kilobyte
MCDRAM	Multi-Channel Direct Random Access Memory
MB	Megabyte
MPI	message passing interface
ns	nanosecond
PCI	Peripheral Component Interconnect
SEID	Software Engineering and Informatics Division
SRC	Supercomputer Research Center
USACE	U.S. Army Corps of Engineers
2D	Two-dimensional
3D	Three-dimensional

1 Introduction

The Knights Landing architecture of the Intel Xeon Phi line consists of a manycore processor with unique architecture and features targeted at highly parallel workloads (Intel nd). This version of the Xeon Phi has several changes that make it fundamentally different from its predecessors. The move away from being a co-processor to the main bootable central processing unit (CPU) will change how programmers use the chip and possibly increase the types of problems that it is suited to solve. Additionally, the Knights Landing processor is paired with a three-dimensional (3D) stacked memory technology and a multi-channel direct random access memory (MCDRAM). This is a high-bandwidth memory that can be configured in a variety of ways, including as a cache to main memory. These types of high-bandwidth memory technologies are becoming increasingly common with manycore chips that require more memory bandwidth to feed the large number of cores.

The MCDRAM configuration options add a level of complexity for the programmers using this system, including the administrators that configure them. These settings are set at boot time, and affect both system performance and how the programmers interact with the system. Using the MCDRAM can be as simple as setting it to act as a transparent cache to main memory, or can be as involved as changing the program to be aware of the memory. These choices will need to be considered and evaluated in the context of the types of applications run at a given center.

While the theoretical bandwidth of the MCDRAM is very high, it is necessary to look at the memory performance using benchmarks to understand how it will behave when used by applications. Common linear algebra kernels read in multiple streams of memory, including write to others, and occurs, and therefore, it is important to understand the memory performance under these conditions. Understanding the latency of the MCDRAM, both in cached mode and when directly addressed, is another important consideration. While applications with good spatial locality should see performance improvements from the high-bandwidth memory, how will the latency effect applications with sparse access patterns? Additionally, programmers need to understand how the different MCDRAM modes affect these same performance characteristics

in order to completely understand the consequences of their configuration choices. While cached mode negates the need to change application code, it is unclear how its performance benefits will compare to directly addressing the MCDRAM.

1.1 Background

The Intel Xeon Phi is a line of computer processors that are highly parallel, utilizing many integrated cores (Intel nd). Initial versions of the Phi's were designed to be used as coprocessors, connecting to the Peripheral Component Interconnect (PCI) bus and performing calculations that were offloaded by the main processor. These coprocessors required special programming techniques to offload the calculations. Additionally, the data movement from the main processor to the co-processor could be costly if there were not enough computations to amortize the transfer time.

With the Knights Landing version of the Phi, launched in 2016, the processor can be used as the systems main and only processor, instead of being used strictly as a coprocessor. This greatly eases the burden placed on programmers when migrating code to this new architecture. It also eliminates the need for data transfer to the Phi making it applicable for more types of applications. Due to the parallel nature of the Phi, one of its main target markets is supercomputing, and Knights Landing has already been named as an offering by vendors (HPC Wire 2016).

The Knights Landing version of the Phi, has several additional improvements over past versions, including being paired with MCDRAM, a high-bandwidth 3D stacked memory. 3D stacked memory stacks DRAM dies directly on top of each other, allowing for short wire distances and extra pins (Loh 2008). This technique offers 4x bandwidth improvements with Intels MCDRAM implementation (Colfax International 2016). With ever increasing numbers of cores on processors, innovate memory solutions are being proposed to feed the processors. While the MCDRAM is Intels implementation of this technology , similar versions can be expected to become increasingly common in high-end computer systems.

1.2 Objectives

The objectives of this study and technical report (TR) were to perform and document a series of micro and macro benchmarks performed on the MCDRAM. These results provide insight into the behavior of this memory

that will assist programmers with understanding what memory mode to use with their associated applications. Using the MCDRAM in cached mode can provide up to 4x speedup for streaming kernels, while directly addressing memory in flat mode provides up to 6x speedup, a non-trivial difference. Additionally, hash implementations with sparse memory access can see a 3x slowdown when run in MCDRAM, whereas a conjugant gradient application sees a 3.7x speedup. It is clear that no one configuration will be right for all applications, and that the cached mode setting, while seeming innocuous, can lead to slowdowns.

1.3 Approach

The approach of this study is to perform a series of micro and macro benchmarks on the MCDRAM. These benchmarks range from computationally intensive with dense memory access patterns to input/output (IO) bound applications with sparse accesses.

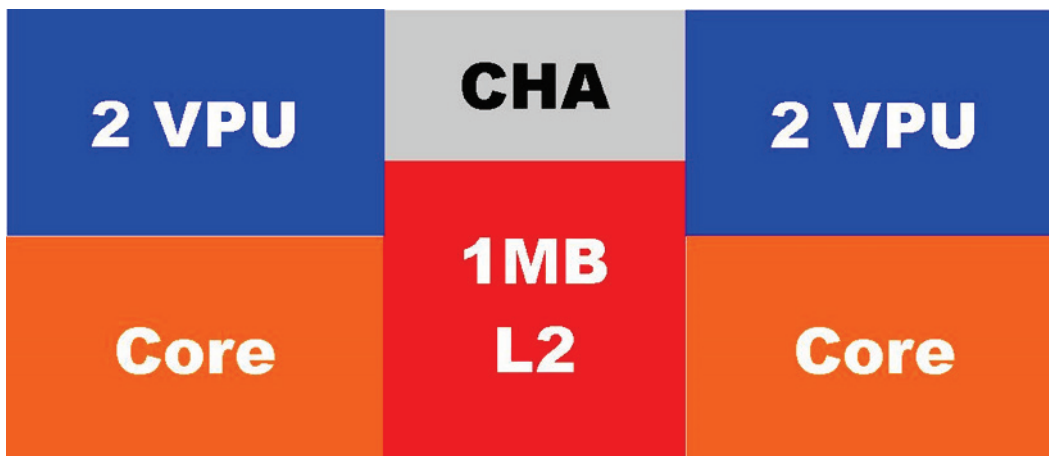
1.4 Scope

This study's scope is limited to a single compute unit with the Knights Landing version of the Phi and MCDRAM. It does not explore the effects of MCDRAM on distributed computations.

2 System Specification and Configurations

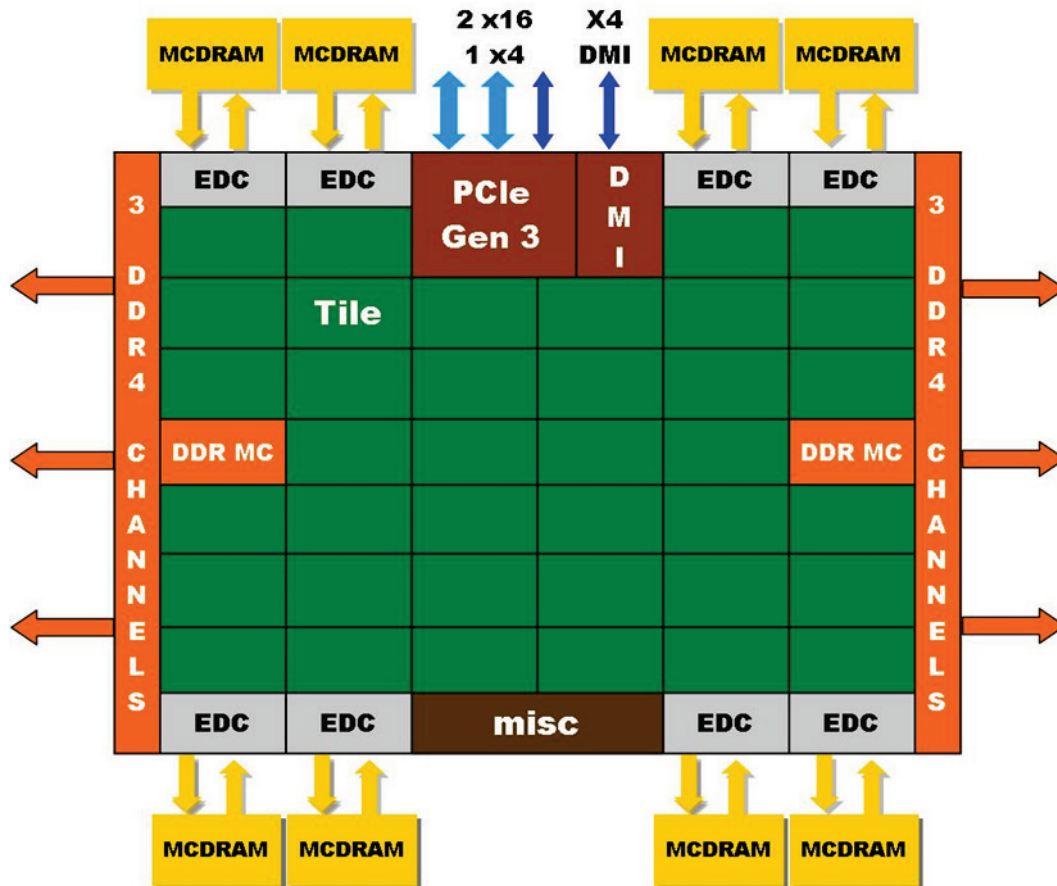
The Xeon Phi system used in this work was a single node developer's access platform obtained from Colfax research.* The system hosts a single Intel Xeon Phi 7210 processor (codenamed Knights Landing) as the only processor in the system (Intel 2018). The Xeon Phi 7210 has 64 cores with a base frequency of 1.3 gigahertz (GHz), and a max frequency of 1.5GHz. Each processing core has a 32 kilobyte (KB) data cache and shares a shared 1 megabyte (MB) L2 cache. Two cores are paired with one L2 cache, four vector processing units, and a cache tag directory to form a single tile, as shown in Figure 1. Each of these tiles are connected via a two dimensional (2D) mesh, and are arranged as shown in Figure 2. The system has 100 gigabytes (GB) of dynamic random access memory (DRAM) memory, in addition to 16 GB of MCDRAM, and a 3D DRAM technology stacked directly on the processor die. This MCDRAM is reported to have speeds over 400 gigabits/second (GB/s), compared to 90 GB/s for the DDR4 main memory. Version 16 of the Intel Parallel Studio was provided with the system and is used to compile all benchmarks. All programs are compiled with the new AVX 512 instruction flags (Colfax International 2016b). These new, longer vector instructions are also new to the Knights Landing version of the Phi, but will not be discussed in detail due to the focus on the MCDRAM memory.

Figure 1. Knights landing tile.



* Colfax. 2018. Developer access program (dap) for intel® xeon phi™ processor (formerly knights landing). This program is no longer supported by the manufacturer.

Figure 2. Knights Landing architecture.



The MCDRAM can be used in several ways, some of which are transparent to the programmer, and some of which involve explicit programmer interaction*. When run in *cache mode*, the MCDRAM is used as a last level cache to main memory, and is not accessible by the programmer directly. This mode requires no work by the programmer, but comes with an inherent risk. While the bandwidth in cache mode can be far better than using regular memory, the latency could be much worse. When the MCDRAM is used as a cache, it is added to the critical path for a miss, thus, increasing the miss-latency. Programs that have very large memory footprints with sparse access patterns could potentially see performance degradation.

The MCDRAM can also be used in *Flat mode*, which allows the programmer to explicitly choose what goes into the MCDRAM. This can be done in two ways. The first is completed programmatically via an API, the

* Colfax. 2018. Developer access program (dap) for intel® xeon phi™ processor (formerly knights landing). This program is no longer supported by the manufacturer.

second does not involve changing the program, but places the entire memory space in MCDRAM. To specifically allocate MCDRAM within a program, the programmer can use the Memkind library to call a special malloc function. This requires modifying the program, but in only a minor way, because the Memkind API is semantically similar to malloc. This approach gives the distinct advantage that programmers can choose what memory blocks they would like to place in the MCDRAM.

The second way to use the high-bandwidth memory does not require any program modifications, but runs the entire program in MCDRAM. Using numactl, programs can be run using only the MCDRAM, so long as the memory footprint does not exceed the MCDRAM size (16 GB).

Hybrid modes can also be employed that divide a portion of the MCDRAM for cache and a portion for programmer use. This mode could be useful if a specific application was going to run that only needed a portion of the high-bandwidth memory for a critical array and the programmer still wanted to use the rest as a cache.

The Knights Landing can also be configured to act as multiple nodes on the same chip, with each node able to use cached or flat MCDRAM (Colfax International 2016a). The single node cluster settings divided the chip into two or four regions, with each region being assigned memory and MCDRAM located nearby on the 2D grid. Processors are also divided so that processors in the same node are located physically adjacent to one another. This prevents network congestion on the 2D grid by preventing communication outside of any quadrant on the grid. These modes can easily be used with hybrid MPI/OpenMP programs by adjusting the number of MPI processes per node and the number of OpenMP threads per process.

A key limitation to these modes is the need to reboot the system in order to change modes. In many supercomputing centers, users do not have permission to perform this operation, and special scheduler support is needed. Generally, single node reboots take 10–15 minutes to obtain the system image and run all configuration and tests, however, this time can increase as multiple nodes try to transfer the system image over the network. This reboot time would be quickly amortized by the long running jobs these systems utilize.

3 Benchmarks

3.1 Stream

Stream is a common benchmark used to measure the maximum bandwidth of a memory system (McCalpin 1995). Stream performs four different bandwidth tests. The first test is called *copy*, which performs the operation $a[i] = b[i]$ over an array. The second test is called *scale*, which adds in a scalar operation $a[i] = scalar * b[i]$. *Add* performs an add of two vectors, doubling the data read requirement, $a[i] = b[i] + c[i]$. Finally, *triad* performs the operation $a[i] = b[i] + scalar * c[i]$. Each of these tests are performed over an array that is large enough not to be cacheable. For multiprocessors, like the Phi, OpenMP is used to measure the total bandwidth from all processors, which is much higher than what a single processor can achieve.

3.2 LMBench

LMBench is a suite of performance tools used to measure basic system metrics (McVoy and Staelin 1996). While there are a large number of tests in the suite, this work will focus on one that measures memory latency and one that measures memory bandwidth.

3.2.1 Lat_mem_rd

Lat_mem_rd measures the latency of a memory system. In order to capture the latency of every level of the memory hierarchy Lat_mem_rd performs multiple read operations within an array with a given stride. A range of strides and array sizes are used to completely understand the memory system latency. Small array sizes fit in a given cache, so when testing those sizes, you are measuring the latency of that one particular cache. As array size increases, the computer cannot cache all the data any longer, so the next level cache latency is the effective latency. Eventually, the array becomes large enough that the benchmark is measuring the main memory latency. Different stride sizes are used to avoid the effects of large cache line sizes and prefetchers. Generally, if a given stride is producing a lower latency for a certain size than other stride sizes, that stride is getting multiple hits in the same cache line or getting hits from a prefetcher.

3.2.2 Bw_mem

Bw_mem measures the bandwidth of the memory system. Different options are available including a *read* test, a *write* test, and a *read/write* test. The size of the buffer to be used by the test is also configurable. In tests performed for this study, the *read* option with a 400MB buffer was used. This test was used to measure individual process performance on the various memory systems as the number of threads increase.

3.3 Google hash table

These hash table benchmarks were created by Google and provided several hash table implementations, including a performance test to compare them (Google nd). There are two hash table implementations included, a sparse hash map, and a dense hash map. The sparse hash map uses quadratic probing, and generally keeps about half of the buckets empty. The dense hash map also uses quadratic probing and keeps about two-thirds of the buckets empty. This offers better performance at the expense of memory. The performance test also times the hash function provided in the c++ library, which in this case is from Intels icpc 16.0.3.

The performance tests measure several functions of the API including map_grow, map_predict/grow, map_replace, map_fetch_random, map_fetch_sequential, map_fetch_empty, map_remove, map_toggle, and map_iterate. In order to save space when necessary, a geometric mean on the speedups of these measures will be used, with any noteworthy metrics highlighted in the text.

3.4 Intel benchmarks

Three benchmarks were included with Intel Parallel Studio version 16 including, a single node LINPACK (Intel 2018a), multi-node LINPACK (Intel 2018b), and conjugant gradient (Intel 2018c). All benchmarks were compiled following Intels gui delines that included the use of the AVX 512 instructions. These benchmarks were not run to determine the maximum numbers for this particular system, to do that, more benchmark tuning would presumably be required. Instead, these benchmarks serve as examples of reasonably tuned programs and are meant to be used to compare performance of the various memory subsystem modes.

4 Results

4.1 Stream

Table 1 shows the bandwidth measurements from the stream benchmark for each memory configuration and each test within the benchmark. For each test, multiple sizes and numbers of threads were used, and the values in Table 1 represent the maximum measured bandwidth of these settings. The add benchmark performs the best, utilizing two input arrays, one output array, and minimal arithmetic. The maximum measured bandwidth is 446.6 GB/s which surpasses the claimed bandwidth of 400GB/s. In all cases, using the MCDRAM in cached mode outperformed the regular DRAM only memory, seeing speedups of 3.3–4x. Using the MCDRAM as the only memory provided the best results, seeing speedups of 4.6–6.3x over traditional DRAM memory.

Table 1. Stream max bandwidth and speedup.

	DRAM	Cached	Speedup	MCDRAM	Speedup
Add	70.5 GB/s	279.7 GB/s	4.0x	446.6 GB/s	6.3x
Copy	66.4 GB/s	225.2 GB/s	3.4x	380.0 GB/s	5.7x
Scale	66.5 GB/s	218.5 GB/s	3.3x	353.0 GB/s	5.3x
Triad	70.4 GB/s	241.2 GB/s	3.4x	326.1 GB/s	4.6x

Figure 3 shows the performance of the stream benchmark as the number of threads used increases from 1 to 256. The number 256 was chosen as the maximum number of threads because each processing core on the Knights Landing processor is designed to simultaneously run four threads. Figure 3 shows the results for traditional DRAM memory. As the number of threads increases, the total bandwidth increases up to about 32 threads. At this point, the main memory is saturated and little improvement is seen, with the bandwidth staying near 70 GB/s.

Figure 4 shows the stream results on the MCDRAM in cached mode. Cached mode increases the overall bandwidth, including the number of threads needed to realize the maximum bandwidth. In this case, the memory does not saturate until 64 threads are used, and it reached speeds of 175 GB/s, over double that of just DRAM. Figure 5 shows the stream results for the MCDRAM in flat mode. In this case, all 256 cores are needed to achieve the maximum bandwidth, which is near 350GB/s.

Figure 3. Flat mode DRAM - Stream bandwidth vs. number of threads, 400 MB buffer.

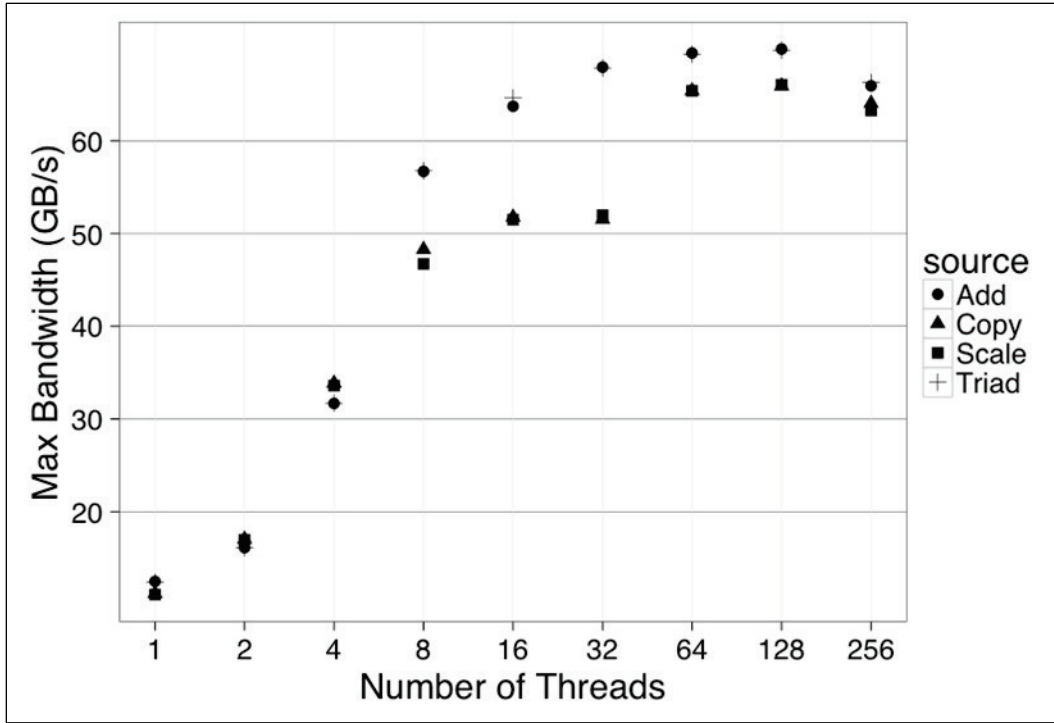
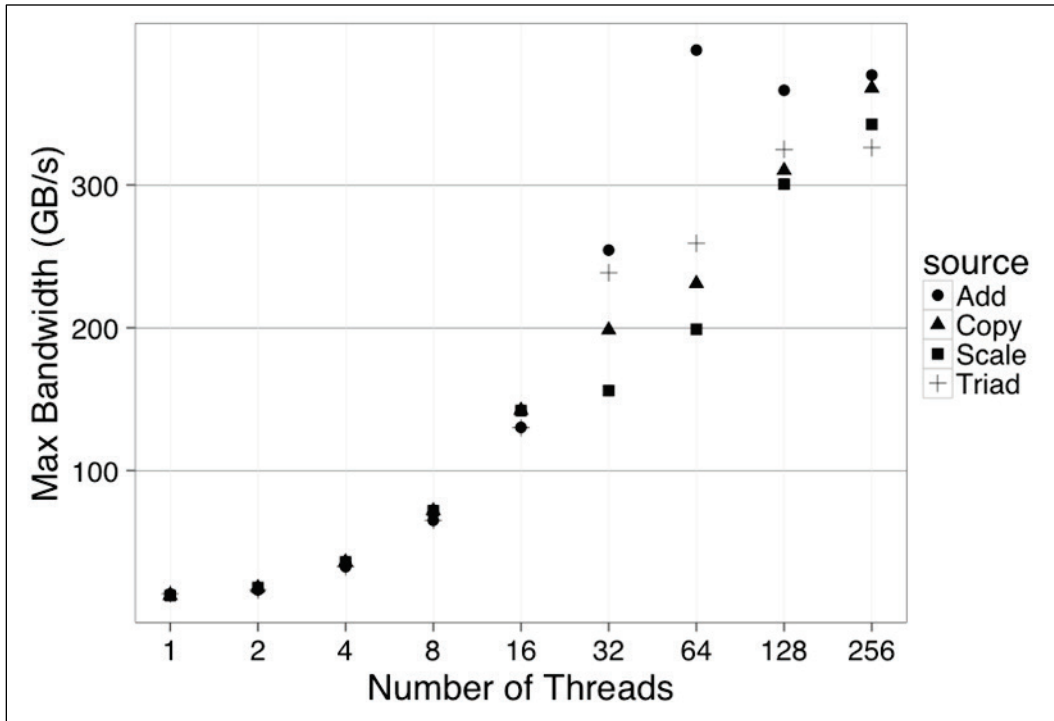
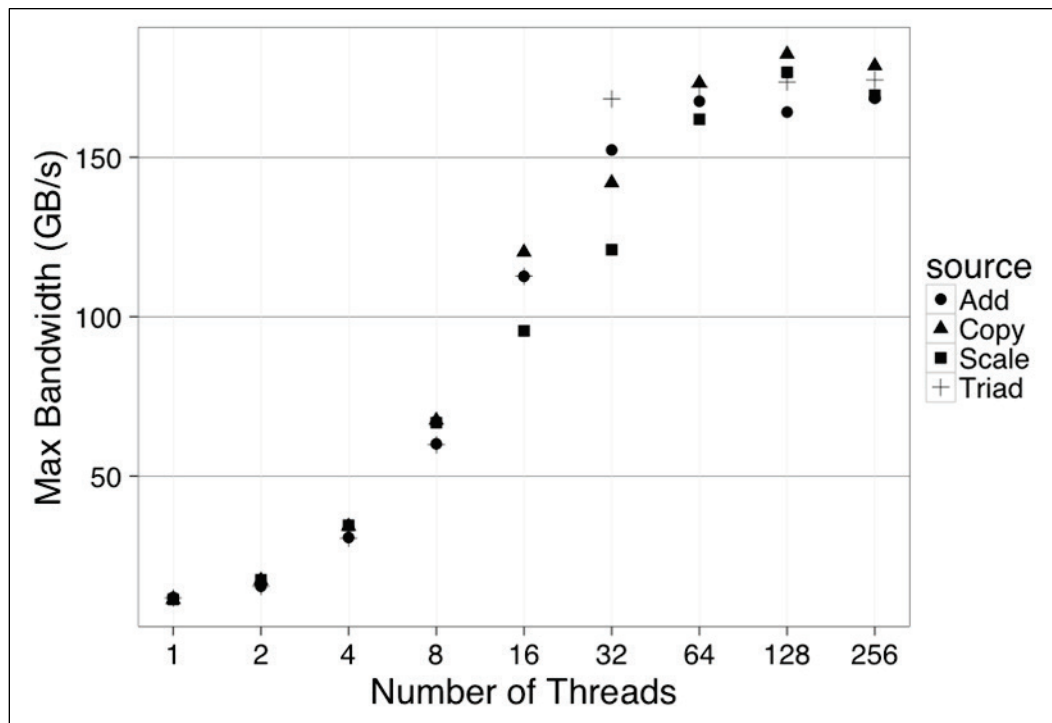


Figure 4. Cache mode - Stream bandwidth vs. number of threads, 400 MB buffer.



These results clearly show the MCDRAM is capable of the advertised bandwidth, however, it needs to be used in flat mode to achieve it. While the MCDRAM in cached mode gives only about half of the performance of the MCDRAM in flat mode, it is still nearly double that of the traditional DRAM memory. This means that even programs with working sets greater than the 16GB MCDRAM can potentially see some performance improvements.

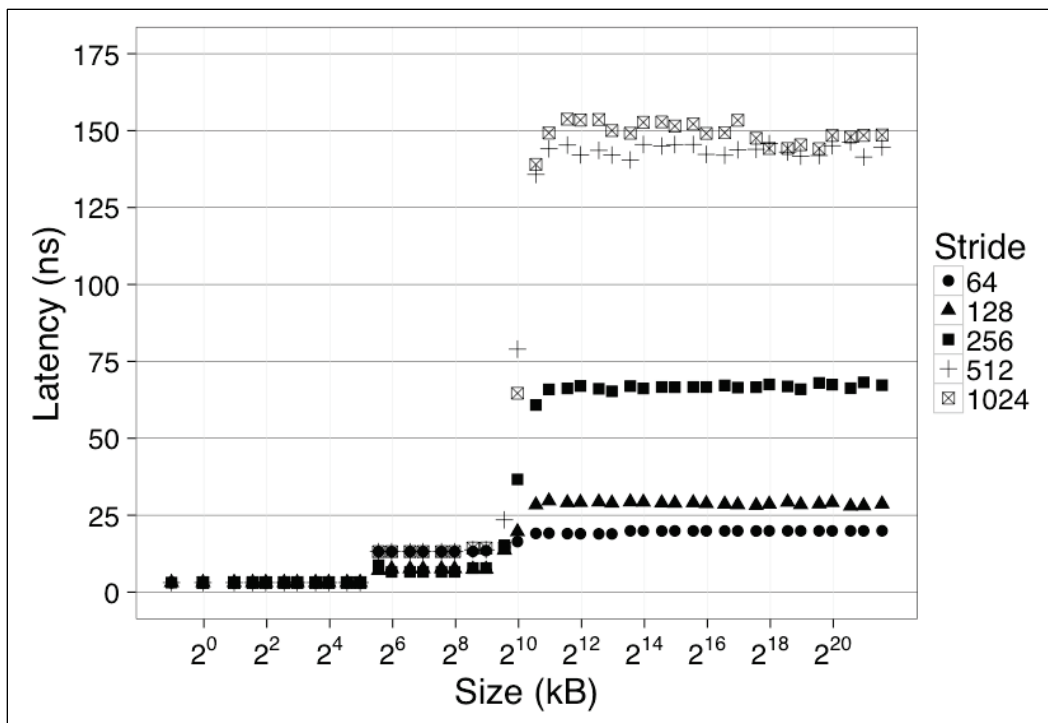
Figure 5. Flat mode MCDRAM - Stream bandwidth vs. number of threads, 400 MB buffer.



4.2 LMbench latency

The memory latency as measured by the `lat_mem_rd` benchmark is given in Figure 6. For each of the three memory configurations, the lowest buffer sizes result in a high number of hits in the L1 cache, with a latency of about 3 nanoseconds (ns). This can be seen as the lowest and leftmost tier on each of the three graphs.

Figure 6. Flat mode DRAM, memory latency vs. buffer size, 4 MB buffer.



Once the buffer size becomes greater than the 32 KB L1 D-cache, the 1 MB L2 cache is utilized, resulting in latency measurements of 7 or 14 ns. In this tier, two distinct lines can be seen, the lowest being at 7 ns and the higher being at 14 ns. In cases like this when certain stride values result in a lower latency (7 ns) than others (14 ns), the lower values are generally attributed to the effects of prefetching. These effects can be difficult to predict without knowing the prefetching algorithms that are being used. This same effect can be seen in the final tier, which shows the latency to main memory.

The rightmost tier for buffer sizes greater than 1 MB represents the latency to main memory. This means something different in each of the three modes, and is the point where the graphs start to diverge. As with the L2 cache, the smaller stride values are affected by the prefetching of memory, and have a lower latency. The higher latency should be interpreted as the memory latency. For the traditional DRAM this is about 150 ns, the MCDRAM is about 175 ns, and the cached mode MCDRAM is about 175 ns. While the stream benchmarks showed a substantial bandwidth increase when using the MCDRAM, it comes at a cost of higher latency, about 1.15x. This means that using the MCDRAM can hurt program performance if the memory access pattern is sparse.

The buffer size used in the above tests is 4 MB, which is smaller than the 16 GB MCDRAM. This means that in cached mode, the entire buffer can fit into the MCDRAM cache. In order to examine the latency on MCDRAM cache misses, a very large buffer (64 GB) was used. These results are shown in Figures 6, 7, and 8. The lower tiers for the L1, L2, and main memory remain the same, but for very large sizes the latency increases dramatically. There is a plateau at 190–200 ns that is presumably the latency for a miss in the MCDRAM, but there is also a number of latencies dramatically higher. It is presumed by the author that these high latencies are a results of compounding MCDRAM misses that become increasingly expensive as the bandwidth between the MCDRAM and the main DRAM saturates. These repeated misses seem to dramatically degrade performance as data is swapped from DRAM to MCDRAM. Further studies are needed to confirm this hypothesis, but programmers should be weary that applications with sparse memory access patterns could create this condition.

Figure 7. Cache mode, memory latency vs buffer size, 4 MB buffer.

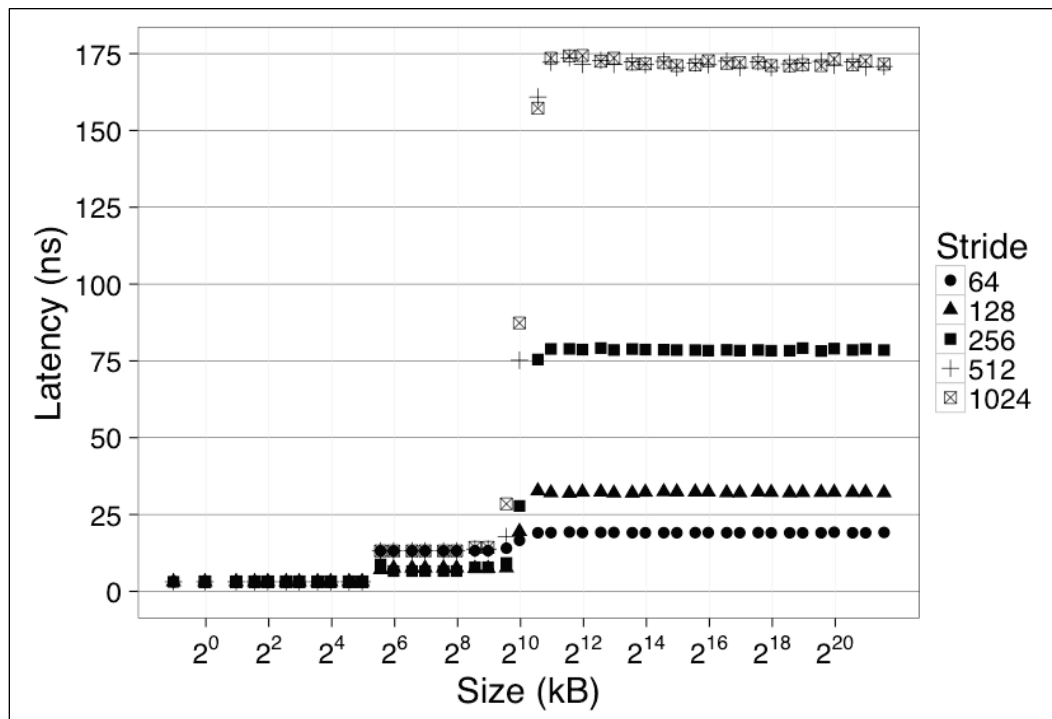


Figure 8. Flat mode MCDRAM, memory latency vs. buffer size, 4 MB buffer.

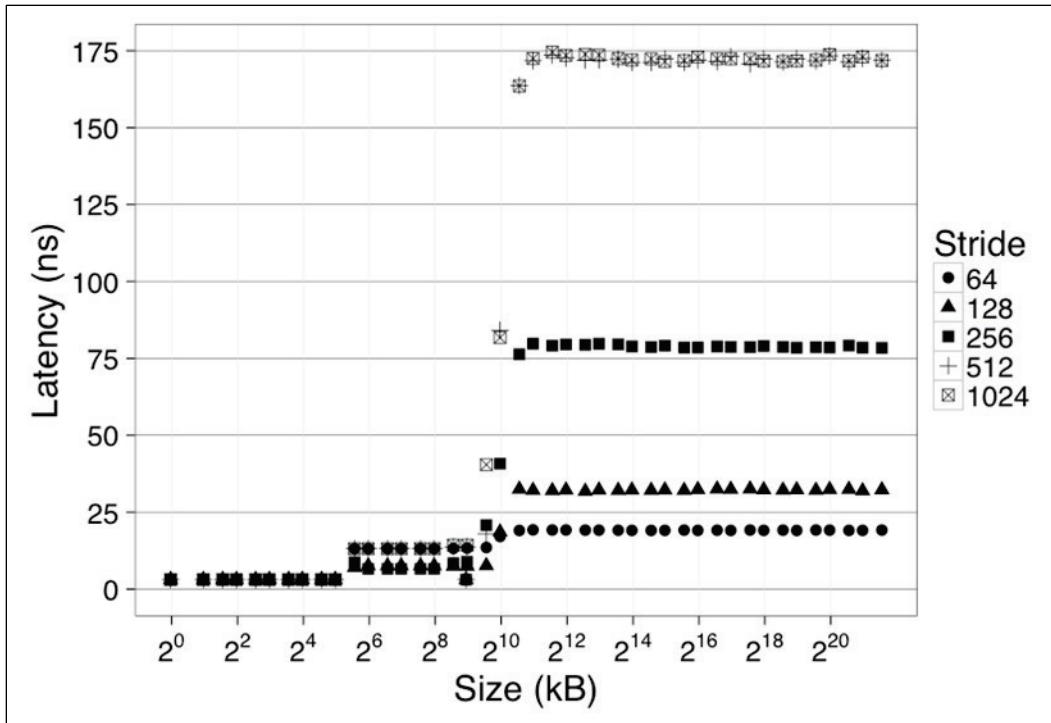
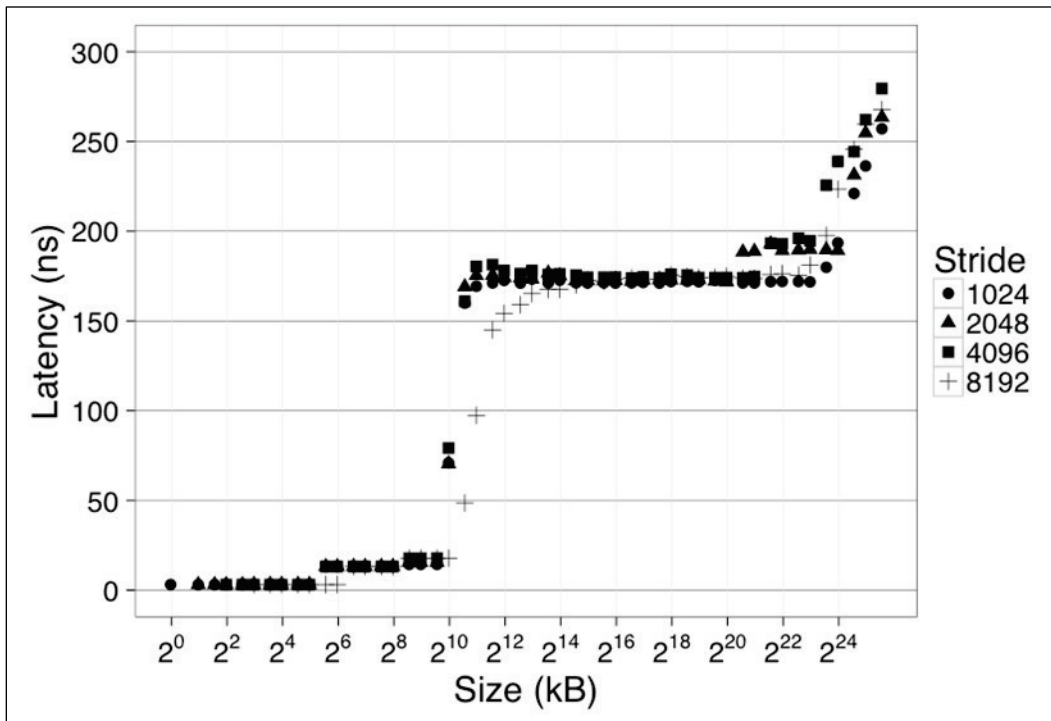


Figure 9. Memory latency vs buffer size, flat mode MCDRAM, 64 GB buffer.



4.3 LMBench bandwidth

The bandwidth benchmark from LMBench was used to measure the read bandwidth of varying numbers of processes. Figure 10 shows the total read bandwidth of all threads as the number of threads is increased. The bandwidths are identical until more than eight threads are used, at which point, the MCDRAM provides superior bandwidth. Figure 11 shows the same test, but reports the average bandwidth as seen by a single thread. Again, the bandwidths are similar until over eight threads are used, at which point the per thread bandwidth from the traditional DRAM drops dramatically as the memory bandwidth is saturated. The MCDRAM sees some per-thread bandwidth drop off, however, it is minimal.

These results demonstrate that it is necessary to utilize a large number of threads in order to realize the benefits of the MCDRAM. Certain memory constrained applications run with fewer possible processes per node than the system can allow, because each process needs a very large memory footprint. These applications would not likely see any performance improvement from the MCDRAM, if too few processes are used to realize its maximum bandwidth.

Figure 10. Total memory bandwidth vs. number of processes, 4 MB buffer.

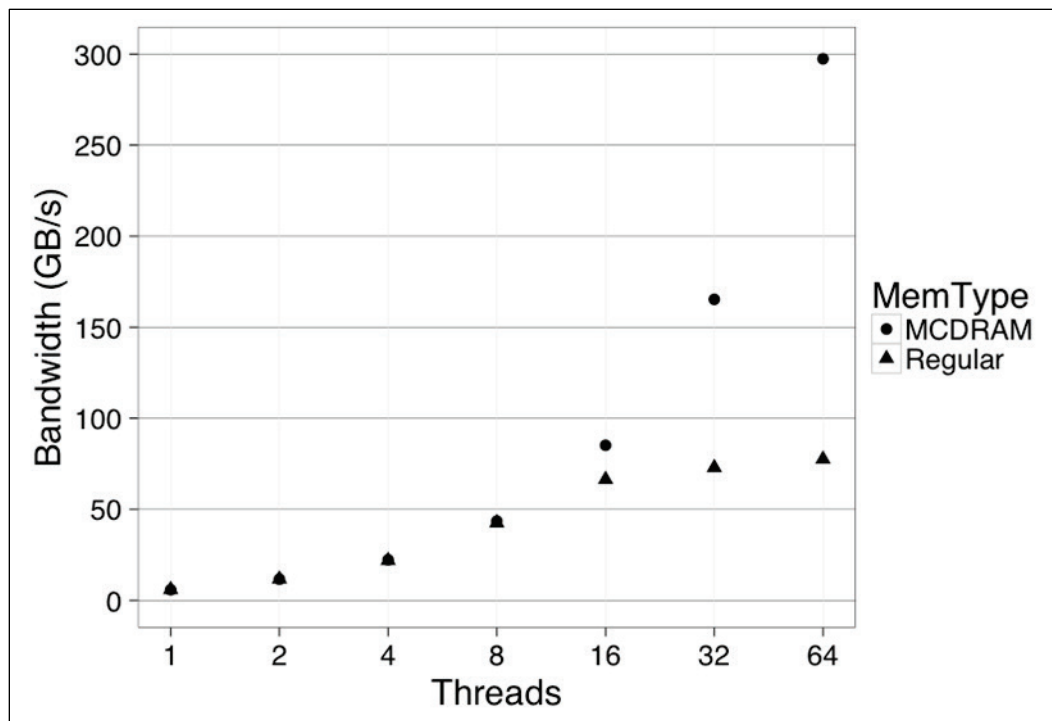
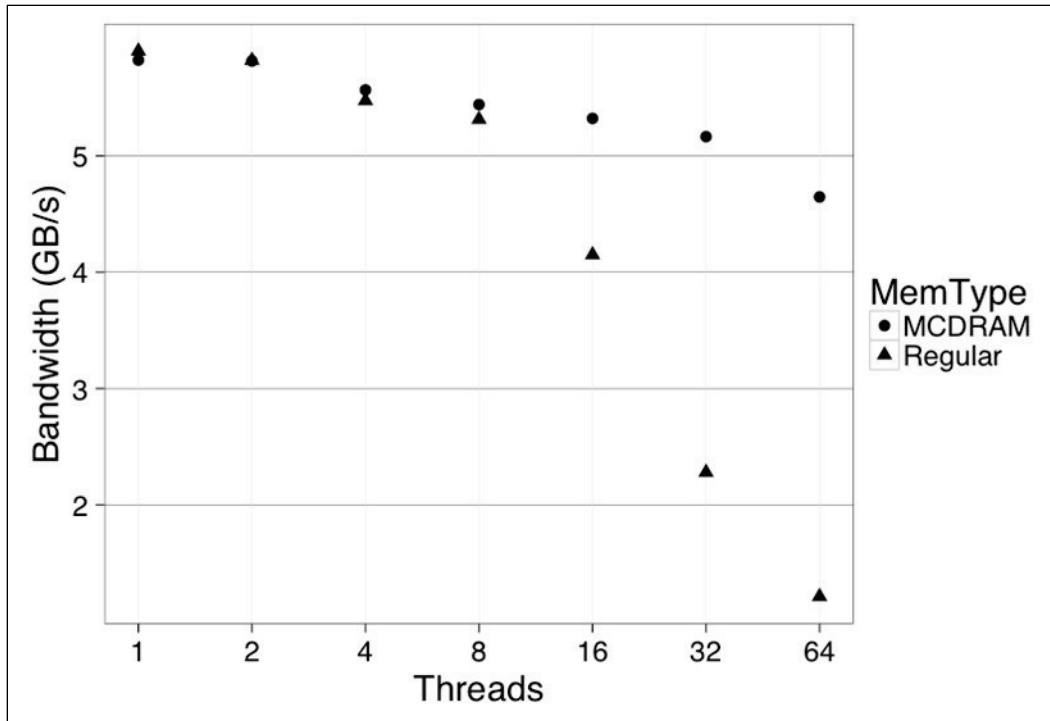


Figure 11. Single processor memory bandwidth vs. number of processes, 4 MB buffer.



4.4 Hash tables

The results for the hash table performance test are given in Table 2. Because of the sparse memory access pattern of hash table operations, this benchmark was of particular interest after observing the latency measurements of the MCDRAM. The table shows results for three different hash map implementations. The dense and sparse implementations are provided by Google, and the standard is from the system `c++` library. For each test, four operations of interest are reported, and three different object sizes are tested. For many test configurations, the high latency of the MCDRAM did not hurt the hash table performance. The Google hash tables received little slowdown, less than 10%. The standard hash implementation fared much worse, running as much as 3x as long on MCDRAM as in regular DRAM. This demonstrates that for applications with sparse memory access patterns, using the MCDRAM offers little benefit in any cases and slowdowns in certain cases.

Table 2. Hash operation latency (ns) and speedup (DRAM/other), slowdowns shown in bold.

		4 byte object				8 byte object				16 byte object						
		DRAM	Cache	MCDRAM		DRAM	Cache	MCDRAM		DRAM	Cache	MCDRAM				
Sparse	grow	757.9	758.8	1.0	758.3	1.0	1028.7	1024.9	1.0	1028.8	1.0	1210.65	1213.4	1.0	1216	1.0
	predict/grow	311.3	311.7	1.0	312.8	1.0	507.9	508.3	1.0	507.5	1.0	595.65	592.75	1.0	594.7	1.0
	replace	123.4	123.5	1.0	123.3	1.0	149.9	150.1	1.0	150.2	1.0	160.2	161.2	1.0	161.3	1.0
	fetch random	537.1	476.7	1.1	539.8	1.0	459.2	518.7	0.9	515.4	0.9	442.45	493.35	0.9	486.7	0.9
Dense	grow	136.0	138.6	1.0	138.4	1.0	175.0	175.4	1.0	174.8	1.0	218.85	222.35	1.0	221.95	1.0
	predict/grow	56.9	57.0	1.0	57.2	1.0	77.6	77.6	1.0	77.7	1.0	93.7	94.1	1.0	94.3	1.0
	replace	18.9	18.9	1.0	19.0	1.0	30.2	30.3	1.0	30.3	1.0	44.95	45.85	1.0	46.05	1.0
	fetch random	90.1	102.9	0.9	103.1	0.9	180.7	208.3	0.9	208.6	0.9	186.15	216.6	0.9	214.95	0.9
Standard	grow	998.7	997.4	1.0	2315.1	0.4	1127.2	1151.6	1.0	3298.0	0.3	981.35	972.3	1.0	1146.3	0.9
	predict/grow	954.9	953.4	1.0	1767.9	0.5	1112.4	1108.9	1.0	3243.2	0.3	888.25	923.5	1.0	1048.35	0.8
	replace	479.4	479.8	1.0	481.0	1.0	525.6	541.8	1.0	540.9	1.0	416.7	421.9	1.0	429.55	1.0
	fetch random	3464.3	4065.3	0.9	4410.3	0.8	3493.4	3898.1	0.9	4280.8	0.8	3000	3428.15	0.9	3777.05	0.8

4.5 Single node LINPACK

The results for the single node LINPACK provided by Intel, with the Intel compiler suite, are provided in Table 3. This benchmark was run with 256 OpenMP threads for each of the three memory configurations. The performance of the Knights Landing processor remained constant in each of the three modes. While the MCDRAM performed worse than the traditional DRAM on the hash table benchmark, no such slowdown was present in this benchmark, which does not have the sparse access pattern of the hash table benchmark.

The results for the multi-node LINPACK implementation provided by Intel are given in Table 4. For the three memory modes, this benchmark was run with four MPI process, 64 threads per process. The performance is measured in gigaflops (GFLOPS), which remains constant across the three memory modes. Because this benchmark is designed for multiple-nodes, the cluster-mode, and the single-node cluster modes with two and four nodes were tested. In these modes, the Knights Landing chip acts as if it is either two or four separate nodes. The DRAM memory is divided so that nodes use memory that is physically closer to their processing cores, and the MCDRAM is split among the nodes and is used as a cache for each memory bank. Each of these cluster modes were tested with one message passing interface (MPI) process per cluster. Both cluster modes performed better than the non-cluster modes, with the two-cluster modes receiving a 1.35x speedup, and the four-cluster mode receiving a 1.12x speedup.

Table 3. Intel single node LINPACK.

	GFLOP/s
Flat Mode DRAM	1691.7
Cache Mode	1682.2
Flat Mode MCDRAM	1691.5

Table 4. Intel multi-processor LINPACK, GFLOP/s for various configurations.

	GFLOP/s
Flat Mode DRAM	287.7
Cache Mode	282.1
Flat Mode MCDRAM	286.8
Cluster Mode 2 <i>2 MPI Proc.</i>	385.5
Cluster Mode 4 <i>4 MPI Proc.</i>	321.1

4.6 Conjugant gradient

The high-performance conjugant gradient benchmark provided by Intel received the largest speedup from the MCDRAM, with the results given in Table 5. This benchmark saw speedups reaching 3.7x when using the MCDRAM in cache mode. The benchmark was run with a varying number of MPI processes and threads per process, and in all cases using the MCDRAM in flat or cached mode provided large performance improvements. The flat mode MCDRAM was not tested with four MPI processes, because at this point, the memory footprint exceeded 16 GB. The cluster modes with both two and four clusters were also tested, and like the multi-processor LINPACK, benchmark showed improvements over the non-cluster mode, reaching an extra 10% speedup.

Table 5. Intel high perf. conjugant gradient, GFLOP/s for various configurations.

MPI Processes Threads per process	1 256	2 128	4 64	4 32
Flat Mode DRAM	9.1	9.7	9.5	10.6
Cache Mode	26.3	35.1	35.6	36.7
Flat Mode MCDRAM	26.2	35.7	X	X
Cluster Mode 2	NA	36.5	39.7	28.3
Cluster Mode 4	NA	34.2	39.4	33.2

5 Programmer/Administrator Recommendations

These results show that there is no right answer as to how to use the MCDRAM on the new Intel Phi Knights Landing. While the MCDRAM can provide very high-bandwidth to memory hungry applications with multiple threads, it comes with an inherent risk that applications with sparse memory patterns can see slowdowns. While it would be nice if system administrators could simply pass the problem to programmers, the different memory and cluster modes are set at boot time, and therefore, cannot be changed. This forces system administrators to decide what users will receive. A careful survey of your service center representative applications is recommended to understand the types of memory patterns that are used.

Flat mode seems to be a safe choice, it provides power users the ability to manually program the MCDRAM from their applications, other users will carry on oblivious to its existence. This method will do-no-harm, but depending on your service center user base, this new memory technology could go virtually unused. Setting the system to cached mode will allow all users to take advantage of the MCDRAM, and this comes with two distinct disadvantages. First, there are potential slowdowns with applications that have sparse memory access patterns. This could lead to behavior that users do not anticipate and is hard for them to debug. Second, in cached mode, power users lose the ability to manually program the MCDRAM from inside their applications, a step that is necessary to achieve maximum performance.

The trade-off between cluster modes seems to be much more straightforward, with well scaling MPI applications performing better in cluster mode. In this mode, the user loses the ability to run one OpenMP or otherwise threaded program on a single node, so an understanding of your centers applications is necessary. Additionally, poorly scaling hybrid OpenMP/MPI programs might be better off in single mode, if the OpenMP parallelism is more scalable than the MPI parallelism. Overall however, the typical types of applications that are run on supercomputers should run better with the single node cluster modes.

In an ideal world, programmers would be able to set the memory configuration options of the Knights Landing at job time. In order for this to be possible, either Intel would need to modify the Knights Landing to be adjustable at runtime, or batch schedulers would need to accept arguments about memory options and reboot the nodes into the right mode before jobs run. Rebooting compute nodes in large clusters takes on the order of minutes, but the speedup and slowdowns shown in this TR make such a cost worthwhile for long running jobs, which are typical on these types of systems.

6 Future Work

The MCDRAM memory shows promise in accelerating applications that require high memory bandwidth, as shown in this work. While these results show the capabilities of the MCDRAM, they do not explore using the programming API to specifically allocate arrays into the high bandwidth memory. Greater performance gains could be obtained by programmers with knowledge of their applications by placing critical memory segments into MCDRAM. This approach would provide high bandwidth to the data in the MCDRAM, and would also alleviate pressure to the main memory, thus, increasing its performance. Case studies involving specific applications being programmed to utilize the MCDRAM would be an obvious next step in research, as it would be enlightening to see the performance of these memory tuned applications compared to normal DRAM and cached mode.

7 Conclusion

The high bandwidth MCDRAM on the Knights Landing processor provides programmers access to memory with up to 6x performance of traditional DRAM. This huge performance increase comes with many additional considerations that programs must examine in order to ensure their applications behave as expected. One such consideration is the increase in latency, which could lead to 3x slowdowns for certain sparse memory patterns. Additionally, high numbers of threads need to be performing memory accesses simultaneously in order to realize the maximum bandwidth. Programmers are also faced with options regarding the single node cluster configurations, and all of these options must be set when the processor is booted. Programmers need to carefully consider these options, as well as their applications characteristics, in order to properly configure their systems to see performance gains. This work aims to serve as a guide to helping programmers understand how to use the MCDRAM and how it will affect their application performance. The MCDRAM is a technology with a huge potential benefit, but it is not effortless to transition to this technology.

References

- Colfax International. 2016a. Clustering modes in Knights Landing processors. In *Technology Exploration*. Accessed on 16 January 2019. <http://colfaxresearch.com/knl-numa/>
- Colfax International. 2016b. MCDRAM as high-bandwidth memory (HBM) in knights landing processors: Developer's guide. Accessed on 16 January 2019. <http://colfaxresearch.com/knl-mcdram/>
- Colfax International. 2016c. Guide to automatic vectorization with intel AVX-512 instructions in knights landing processors. In *Technology Exploration*. Accessed on 16 January 2019. <http://colfaxresearch.com/knl-avx512/>
- HPC Wire. 2016. Getting ready for KNL? Take a lesson from NERSC on optimizing. Accessed on 16 January 2019. <https://www.hpcwire.com/2016/02/10/nersc-getting-ready-knl/>
- Google Sparsehash/Sparsehash. nd. [Google sparsehash](https://github.com/sparsehash/sparsehash) <https://github.com/sparsehash/sparsehash>.
- Intel® Software Development Zone. 2018a. Developer Guide for Intel® Math Kernel Library for Linux. Intel® optimized LINPACK benchmark for Linux. Accessed on 16 January 2019. <https://software.intel.com/en-us/node/528615>
- Intel® Software Development Zone. 2018b. Developer Guide for Intel® Math Kernel Library for Linux. Intel optimized MP LINPACK benchmark. Accessed on 16 January 2019. <https://software.intel.com/en-us/node/528619>
- Intel® Software Development Zone. 2018c. Developer Guide for Intel® Math Kernel Library for Linux. Getting started with intel optimized HPCG Accessed on 16 January 2019. <https://software.intel.com/en-us/node/599526>
- Intel®. nd. Intel®. xeon phi product family x200. Accessed on 16 January 2019. <http://ark.intel.com/products/family/92650/Intel-Xeon-Phi-Product-Family-x200>
- Loh, G. H. 2008. 3D-stacked memory architectures for multi-core processors. In *International Symposium on Computer Architecture* 36(3): 453–464. doi:10.1145/1394608.1382159.
- McCalpin, J. D. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* 19–25.
- McVoy, L., and C. Staelin. 1996. Lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, San Diego, CA. <http://dl.acm.org/citation.cfm?id=1268299.1268322>
- Merritt, Rick. 2012. Cray will use intel MIC branded xeon phi. In *EETimes designlines*. Accessed on 16 January 2019. http://www.eetimes.com/document.asp?doc_id=1261951

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) September 2019		2. REPORT TYPE Final report		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Initial Benchmarking of the Intel 3D-Stacked MCDRAM				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Benjamin S. Parsons				5d. PROJECT NUMBER H29B80	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Information Technology Laboratory U.S. Army Engineer Research and Development Center 3909 Halls Ferry Road Vicksburg, MS 39180-6199				8. PERFORMING ORGANIZATION REPORT NUMBER ERDC/ITL TR-19-8	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Headquarters, U.S. Army Corps of Engineers Washington, DC 20314-1000				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Modern, manycore processors are increasingly using high-bandwidth memory (HBM) to provide the necessary memory bandwidth to high core counts. The Knights Landing processor is one example, which is paired with a three-dimensional (3D) stacked memory, and a multi-channel direct random access memory (MCDRAM). The MCDRAM offers high-bandwidth memory, however, it is also important to understand the latency of the memory and how the memory will respond to noncontiguous accesses. This work provides information to programmers about how their applications will perform, and which of the many memory settings that will provide the best performance. These settings are non-trivial and can have a large impact on application performance. In addition to several micro-benchmarks, this work shows that using the MCDRAM can provide speedups up to 3.7x for a congregant gradient application, while giving slowdowns of 3x for certain hash table implementations. Given this range of performance, it is essential for programmers to understand this memory technology and how to use it appropriately.					
15. SUBJECT TERMS MCDRAM High Bandwidth Memory (HBM)			Knights Landing (KNL) Benchmarking		
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT UNCLASSIFIED	b. ABSTRACT UNCLASSIFIED	c. THIS PAGE UNCLASSIFIED			SAR