AIM: Accelerating Computational Genomics through Scalable and Noninvasive Accelerator-Interposed Memory

Jason Cong, Zhenman Fang, Michael Gill, Farnoosh Javadi, Glenn Reinman*
Center for Domain-Specific Computing, University of California, Los Angeles

ABSTRACT

Computational genomics plays an important role in health care, but is computationally challenging as most genomics applications use large data sets and are both computation-intensive and memory-intensive. Recent approaches with on-chip hardware accelerators can boost computing capability and energy efficiency, but are limited by the memory requirements of accelerators when processing workloads like computational genomics. In this paper we propose the accelerator-interposed memory (AIM) as a means of scalable and noninvasive near-memory acceleration. To avoid the high memory access latency and bandwidth limitation of CPU-side acceleration, we design accelerators as a separate package, called AIM module, and physically place an AIM module between each DRAM DIMM module and conventional memory bus network. Experimental results for genomics applications confirm the benefits of AIM. Due to the much lower memory access latency and scalable memory bandwidth, our noninvasive AIM achieves much better performance scalability than the CPU-side acceleration when the memory system scales up. When there are 16 instances of accelerators and DIMMs in the system, AIM achieves up to 3.7x better performance than the CPU-side acceleration.

CCS CONCEPTS

• **Computer systems organization** → **Special purpose systems**; • **Hardware** → **Memory and dense storage**;

1 INTRODUCTION

Computational genomics has tremendous potential as a means of providing customized medical care [18]. For example, rather than using an array of chemotherapy drugs to treat cancer, genetic analysis can allow a physician to select a particular drug or treatment that is appropriate for the particular pathology of the cancer. It can also help with preventative treatment such that doctors can better identify those diseases to which a patient is genetically susceptible.

In general, genomics applications can be classified into two main categories: genome reconstruction and genomics diagnosis/analysis, as illustrated in Figure 1. First, a patient has her DNA sequenced

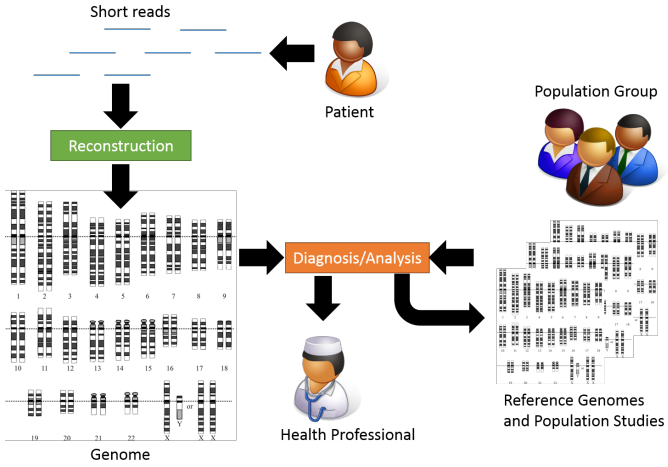

Figure 1: Genomics application work flow: genome reconstruction and genomics diagnosis/analysis.

to produce a collection of short *reads*, i.e., small pieces of 50-150 base pairs, since current DNA sequencing technologies are not able to read the entire genome [36]. These short reads are then reconstructed (e.g., alignment, mark duplication, insertion, and deletion) to a whole genome and aligned to the canonical reference genome. Second, the genome can then be analyzed in a variety of ways [9]. On one hand, this genomics data from the patient can be compared against the canonical reference genome to help a health professional, in a clinical diagnosis, customize the patient's medical care. On the other hand, this genome can be compared against other individuals' genomes as part of a population study to help clinicians and biomedical researchers better understand diseases.

Most genomics applications use extremely large data sets. Storage for a single human genome occupies around 6 GB of space—and that would first need to be reconstructed from a much larger pool of short reads that take hundreds of GB space. For example, the Burrows-Wheeler transform (BWT) [28], one core component of genome reconstruction backward search, requires around 8 GB of memory footprint. Furthermore, population analysis based on genomics data often requires the comparison of multiple genomes [9]. This makes genomics applications not only computation-intensive, but also memory-intensive, and therefore it usually takes a couple of weeks to complete the computation [48].

To make computational genomics useful in a clinical setting, we need to reduce the computational overhead of obtaining and using genetic data to the point where it can become a common medical practice. Among various acceleration candidates such as multicore, GPU, and customized hardware acceleration, the customized accelerator-rich architecture proves to be one of the most

*Zhenman Fang and Farnoosh Javadi made equal contribution and co-led this paper. Email: {zhenman, farnoosh}@cs.ucla.edu

MEMSYS 2017, October 2–5, 2017, Alexandria, VA, USA

ACM ISBN 978-1-4503-5335-9/17/10…$15.00
https://doi.org/10.1145/3132402.3132406

Table 1: Last-level cache (LLC) miss rate for genomics applications running on CPU-side acceleration.

Applications	BWT	DynProgram	MergeSort	FastEpistasis
LLC miss	61%	68%	51%	72%

energy-efficient, high-performance alternatives [10, 11]. However, even though the computation aspects can be satisfied through customized accelerators, the memory requirements of genomics applications are such that the CPU-side acceleration would easily starve the accelerators due to long memory access latency and insufficient memory bandwidth [16].

To better understand the memory behavior of genomics applications, we profile the latest genomics sequencing pipeline running on CPU-side accelerators with a single DIMM (detailed experimental setup will be presented in Section 4). As shown in Table 1, the four genomics applications studied in this paper have high cache miss rates, where caches are unable to help feed accelerators the data in time due to a lack of exploitable locality. As a result, a near-memory acceleration can help reduce the memory access latency. In addition, these genomics applications usually have very high memory bandwidth requirements (around 200GB/s). Therefore, the acceleration approach has to achieve scalable memory bandwidth by increasing the size of the memory systems (i.e., introducing more DRAM DIMMs). More quantitative evaluation of the memory requirements of genomics applications will be presented in Section 2.2.

To achieve lower memory latency and scalable memory bandwidth, we investigate near-memory acceleration and aim to achieve minimum invasiveness to existing systems. Figure 2 shows the conventional CPU and memory architecture connected by the memory network. There are a variety of ways one could integrate accelerators closer to the memory DIMMs. For example, prior work integrated accelerators at the memory controller [44], at the DIMM level [40], or even at the DIMM bank level [26]. Integrating accelerators into a processor or DIMM die is complex, and getting a commercial-grade, highly efficient accelerator packaged into the same die as a commercial-grade, highly efficient conventional core or DIMM requires both a large engineering effort and cooperation between industry players. These factors together make an argument for moving accelerators that perform streaming style computation over large volumes of data into a separate die package entirely, and moving toward a more modular design.

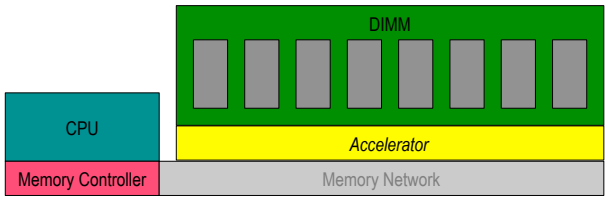

Figure 2: Conventional CPU (and memory controller) and memory architecture connected by the memory network.

In this paper we propose an accelerator-interposed memory (AIM), where the accelerator is located on an AIM module that is physically placed between the memory DIMM and the memory network. In addition, to achieve better communication between different AIM modules, we further propose a multi-drop AIM bus that connects them together. In summary, AIM offers the following advantages.

1. **Noninvasive AIM design.** The AIM design is noninvasive to the existing CPU, memory controller, memory bus, and DIMMs. It simplifies the authorship of customized accelerators as a separate package. A set of AIM modules can be introduced to a machine that consists of off-the-shelf everything and runs off-the-shelf software. The implemented accelerators also communicate with other devices using the well-established protocols that the CPU uses to communicate with the memory, which limits the need for testing communication protocols.
2. **Lower memory access latency.** Since the AIM module is much closer to its associated local DIMM, the access latency to the local DIMM is much lower (around 17 nanoseconds instead of the original 112 nanoseconds, or even worse when it exceeds the memory network bandwidth). In addition, inter-DIMM access latency is also much lower through the efficient AIMBus (an additional 10 nanoseconds if no congestion in AIMBus).
3. **Scalable acceleration and memory bandwidth.** Instead of integrating many instances of each accelerator into the CPU, AIM distributes each accelerator instance between each DIMM and the memory network. Not only the accelerators are easier to scale but also it can avoid any bottleneck at the memory network and the artificial CPU pin-out limit. Moreover, the memory capacity and bandwidth can scale well with the aggregation of all DIMM interfaces when the application memory layout is optimally partitioned.
4. **Shared memory between AIM modules and CPU.** The AIM modules literally use the same memory as the CPU; thus, shared memory is automatic, without the need for additional costly hardware abstraction.

While conceptually simple, there are several challenges to achieving such a design. This paper discusses how we met these challenges, and shows experimentally (based on simulation) that a system featuring AIM can help bring the promise of computational genomics to the clinical setting.

The remainder of this paper is organized as follows. We first give an overview of the genomics applications we chose and present their memory requirements in Section 2. Then we propose the AIM architecture in Section 3, and discuss how AIM provides scalable memory capacity and bandwidth, minimizes system impact, and optimizes memory layout, and how software will be able to make use of AIM. We describe the evaluation methodology of AIM in Section 4. We present the acceleration results using AIM for genomics application in Section 5. Section 6 discusses relevant prior art. Finally, we conclude in Section 7.

2 GENOMICS APPLICATIONS AND THEIR MEMORY REQUIREMENTS

Computational genomics has promoted applications in many fields, such as medicine, biotechnology and genome reconstruction. The field also includes studies of intra-genomics phenomena such as epistasis and other interactions between loci within the

genome [36]. We choose four representative computational genomics applications: the Burrows-Wheeler transform (BWT) and dynamic programming (dynProgramming) are used for genome reconstruction; merge sort and fast espistasis are used for genome diagnosis/analysis. Then we measure their memory requirements to motivate our AIM design.

2.1 Genomics Applications

In this subsection we briefly describe the four representative genomics applications.

Burrows-Wheeler Transform. One of the main algorithms for aligning short reads back to a human genome is the Burrows-Wheeleer transform (BWT) [28, 30]. It matches billions of short strings (about 50-150 characters) to a reference genome which is about 3 billion characters long. We divide the reference genome and those short reads among all the DIMMs. Each AIM module will align its portion of short reads to the applicable portion of the reference genome. Whenever an AIM module needs data in the reference genome that is not in its own local DIMM, it will use the AIMBus to access the remote DIMM that has the data.

Dynamic Programming. One of the common problems in biology is to estimate the similarity between short reads and the reference genome, or between DNA and protein sequences [20]. One of the main algorithms for string comparison and finding similarity is dynamic programming [2]. It relies on solving larger problems starting with a set of small sub-problems. We divide the total DNA and protein sequences into different DIMMs. Each DIMM performs the comparison of a subset of sequences using the dynamic programming algorithm for each pair. AIMBus is used to access a remote DIMM when the sequence is not in the local DIMM.

Merge Sort. Merge sort, a comparison-based sorting algorithm, is another commonly used algorithm in genomics. For example, it is used in SAM (Sequence Alignment/ MAP) format for storing large nucleotide sequence alignments [29]. Merge sort is a divide-and-conquer algorithm. It divides the input array in two halves, sorts for the two halves, and then merges the two sorted halves. It is composed of multiple phases, where the first phase is completely parallel, and each AIM module sorts the data of its attached local DIMM. In the following phases, data from every two DIMMs are merged together into one DIMM through the AIMBus, and only one AIM module will perform the sort. As a result, the parallelism factor is cut down by two. This will continue until all data are merged and sorted by one single AIM module.

Fast Epistasis Detection. Genetic interactions called epistasis have been widely studied. It can have an influence in cancer [4], hypertension [45], obesity [38] and other complex diseases [32]. However, the detection of gene-gene interactions in genome-wide association studies (GWAS) needs a massive amount of computation and data. The number of single-nucleotide polymorphisms (SNPs) is mostly on the order of 10^6-10^7 in humans and the total number of pairs of SNPs that need to be studied can be on the order of 10^{12}-10^{14} [24]. In addition, data will be gathered on thousands of individuals. We leverage the fast epistasis code from [24]. In this algorithm, we store the SNPs through all the DIMMs. Each AIM module will find the correlation of the SNP pairs inside its local DIMM, and for the pairs that are not in one DIMM, the AIMbus will be used to get the data from a remote DIMM.

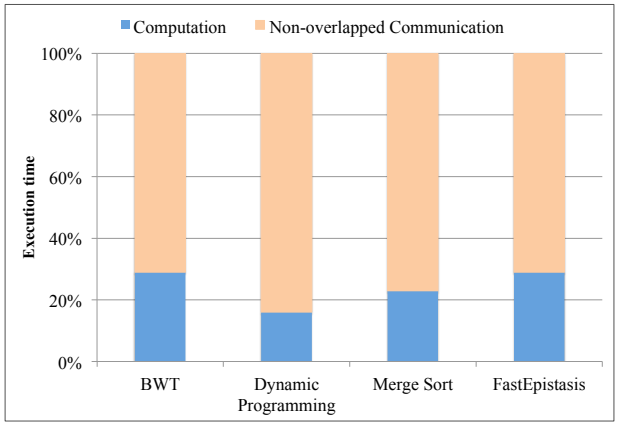

Figure 3: Percentage of computation and non-overlapped communication in the total execution time for genomics applications using CPU-side acceleration.

2.2 Memory Requirements of Genomics

In this section we characterize the memory requirements for the genomics applications that we chose.

Table 2: Memory footprint for genomics applications.

Applications	BWT	DynProgram	MergeSort	FastEpistasis
Memory size	8 GB	8 GB	6 GB	4 GB

1. Table 2 summarizes the total amount of data each application uses. To make the simulation time-affordable, we limit the size within 8GB for each application. In real applications, the memory footprint could be much larger.
2. To motivate the design of AIM, in addition to the last-level cache miss rate in Section 1, we further profile the percentage of pure computation and non-overlapped memory access in the total execution time. The data is collected for the CPU-side acceleration baseline with 16 instances of accelerators and DIMMs, detailed in Section 4. Shown in Figure 3, more than 70% of the total execution for these genomics applications is spent on the memory system after the computation is accelerated using customized accelerators. This implies that the near-memory acceleration can bring much performance benefit since it achieves a much lower memory access latency.
3. We also calculate the targeted bandwidth for these genomics applications using the same experimental setup assuming the memory access overhead is removed. As shown in Figure 4, the targeted bandwidth is from 168GB/s to 202GB/s.[1] However, the actual achieved bandwidth is less than 50GB/s due to the limited memory network bandwidth.

[1]Note that we limit the size of our applications for simulation purposes. For real applications with larger memory access sizes, the targeted bandwidth could be even higher.

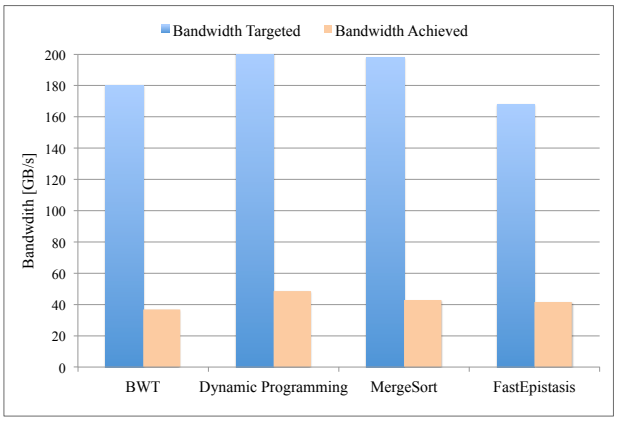

Figure 4: Targeted bandwidth and achieved bandwidth for genomics applications using CPU-side acceleration.

3 ACCELERATOR-INTERPOSED MEMORY

To achieve scalable near-memory acceleration, we propose a non-invasive accelerator-interposed memory (AIM), as illustrated in Figure 5. Because the design of accelerators has been widely studied [10], we will primarily focus on how to interpose the accelerators near the existing memory system. First, we discuss the design philosophy of AIM, including how to provide scalable memory capacity and bandwidth, minimize system impact, and optimize the memory layout. Then we present the detailed implementation of the AIM modules and AIMBus. Finally, we illustrate how to program the applications for the AIM system.

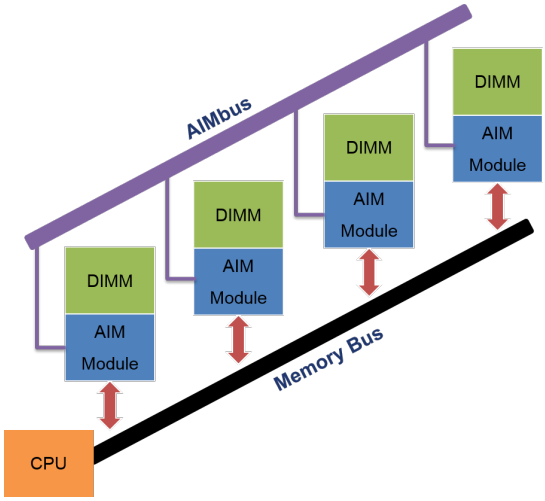

Figure 5: An overview of the AIM architecture.

3.1 AIM Design Philosophy

3.1.1 Provide Scalable Memory Capacity and Bandwidth. As characterized in Section 2, most genomics applications have a large memory footprint and high bandwidth requirement, which is exacerbated by accelerators that push the already-large amount of memory accesses into a much shorter time period. Therefore, our AIM design has to place accelerators closer to memory in such a way that we can scale both memory capacity and bandwidth by increasing the number of DIMMs in the system.

To achieve high scalability, we propose to tie an AIM module to each DIMM, as shown in Figure 5. Each AIM module consists of a separate package that attaches to a conventional DIMM interface on the motherboard. It is placed between the DIMM and the memory network to achieve lower memory access latency and avoid the bandwidth limit of the memory network. Each AIM module also acts as a pass-through for communication between the memory controller of the CPU and the DIMM to which it is attached.

To enable efficient communication between different AIM modules, we further connect AIM modules via a separate sideband interconnect much like the SLI connection used between some graphic cards [3]. We call this sideband interconnect the *AIMBus*: it is a multi-drop bus in which several AIM modules can simultaneously connect to a single channel. AIMBus arbitrates among the AIM modules to determine which one may broadcast to other AIM modules.

3.1.2 Minimize System Impact. To achieve a noninvasive design, we want to make use of off-the-shelf hardware for CPU, DIMMs, and the system motherboard to minimize design impact. There are already a number of constraints regarding the way in which the CPU and main memory interact over the memory network, and we want to be sure that these constraints are not broken by our interposing of accelerators between DIMMs and the motherboard.[2]

First, we need to ensure that memory requests from the CPU are handled correctly, even when the DRAM is busy handling an access request issued from an AIM accelerator. The current DRAM transaction cannot be interrupted, and yet the CPU memory controller is expecting a response at a specific time. To achieve this, we make use of the well-established error correcting codes (ECC) to return a dummy data value that will appear as an error to the memory controller. This will cause the memory controller to retry the request again. And the accelerator will stall subsequent requests for a period of time to ensure that the DRAM is not busy during the retry. To handle the additional latency from passing CPU requests through the AIM module, we intentionally advertise the DRAM access latency to the memory controller as being slower than it actually is. Since we only fake the ECC when there is contention between CPU and AIM accelerators, the fake ECC approach has little impact on the ECC coverage in our experiments.

Second, AIM should work within the confines of the existing memory model. To achieve this, AIM maintains a particular address range for the CPU to control AIM modules. We call this region the *accelerator control memory region (ACMR)*. This ACMR is reserved such that the operating system is unable to allocate it for other purposes. There are two types of ACMR: configuration ACMR and polling ACMR. More details will be presented in Section 3.4.

3.1.3 Optimize Memory Layout. At a high level, a system using AIM is structured as a hierarchy, with the CPU as the master and AIM modules serving as workers. Each AIM module and its attached local DIMM acts as a small independent system capable of independent computation, while the CPU views all of the memory

[2]We are currently exploring an implementation of this technology through a collaboration with Micron.

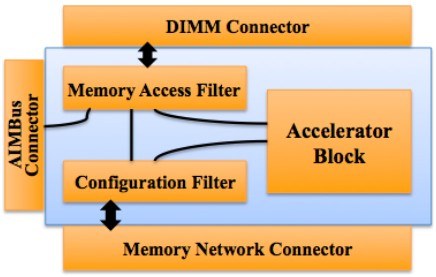

Figure 6: Internal design of a single AIM module.

in the entire system as shared memory. When one AIM module wants to access data in another remote DIMM, it uses the AIMBus.

Since the memory access latency to the local DIMM is faster than the remote DIMM and is not limited by the AIMBus bandwidth, our goal is to optimize the memory layout so that most of the data accessed by each AIM module is in the local DIMM, and there is minimum communication between different AIM modules. The process of mapping data to DIMMs is conceptually similar to data tiling [37]. We currently partition data onto AIM for each application manually as explained in Section 2.1. In our future work, we will investigate compiler support for automatic partitioning.

3.2 AIM Module Implementation

To make the AIM module more flexible for different accelerators and applications, we use an FPGA-based package that can be reprogrammed for different accelerators. Figure 6 presents the modular design of an AIM module that can be easily plugged into existing systems. The key internal components of an AIM module are described as follows. In addition, each AIM module also has a memory network connector and DIMM connector to connect with the conventional memory network and DIMM, respectively.

1. **Configuration filter.** The configuration filter interfaces with the CPU (via the memory network) and the accelerator block inside the AIM module. When a memory request arrives from the memory network (i.e., CPU), the configuration filter checks whether the address is in the configuration ACMR. If so, the request will be redirected to the configuration buffer of the associated accelerator block. Otherwise, the request is considered a conventional memory request from the CPU, and it will be forwarded to the memory access filter. Since the configuration filter simply examines whether the address is within the ACMR range, it can be done using two comparators in parallel. This can be accomplished in a single cycle and achieves a 500MHz clock frequency on an FPGA [46]— i.e., 2 nanoseconds.
2. **Accelerator block.** This consists of both the accelerator for computation and a configuration buffer used to process the memory requests for the configuration ACMR. We omit the details of the accelerator design because it has been widely studied [10, 11, 15, 17].
3. **Memory access filter.** The memory access filter gets memory requests from the CPU via the configuration filter, the accelerator block within the local AIM module, and the AIMBus for remote AIM modules. Whenever a memory request reaches the memory access filter, it routes and tracks the request source (i.e., CPU, AIM accelerator) in a memory access table. If an accelerator requests a memory access that is not in the range of its attached DIMM, the memory access filter routes it to the AIMBus connector. The AIMBus connector then broadcasts the request to the remote DIMMs through the AIMBus. Whenever the memory response comes back from the attached memory DIMM, the memory access filter checks its memory access table to find the associated outstanding memory request and forwards the response to the request source. The memory access filter prioritizes requests in order of most important to least important: CPU request, local accelerator request, remote request from the AIMBus. We did not find any starvation in any of our single-application experiments by using this simple arbitration mechanism. In future work, we will investigate more advanced mechanisms to avoid starvation for the multi-program environment, especially when both the CPU and the AIM modules are running some applications simultaneously.
   The memory access table is organized similar to miss status holding registers (MSHRs), where an incoming request allocates an entry and a satisfied request invalidates the table entry. Each entry in the table has a tag to indicate the source for the particular in-flight request. The memory access table is sized as the maximum number of concurrent pending access requests that the attached DIMM module can support. Since the memory access filter uses a single flag indicating that the pending request is from the CPU or the AIM module, it can also be accomplished in a single cycle and achieves a 500MHz clock frequency—i.e., 2 nanoseconds.
4. **AIMBus connector.** The AIMBus connector is a port to connect the AIM modules to the AIMBus. This AIMBus connector broadcasts any memory requests of the AIM accelerator that fall outside of the attached DIMM. All AIM modules on the AIMBus will listen for the request and determine whether or not the requested address is on their local DIMM. If so, the AIM module will service the remote request and broadcast the result back on the AIMBus for recipient by the originally requesting AIM module. Only one of the AIM modules can be the bus master at any time, and we make use of a polling-based arbitration policy. Quantitative evaluation of the average memory access latency will be presented in Section 5.2.

3.3 AIMBus Implementation

We propose adding an AIMBus shared among all the AIM modules in order to enable efficient inter-DIMM communication, not to limit the accelerators to only using their attached DIMM module, and to increase data capacity in data transmission. We use a multi-drop AIMBus in which several AIM modules can be simultaneously connected to a single channel. Multi-drop buses have the advantage of simplicity and extensibility. Based on [5, 41, 43], we can have multiple devices (i.e., AIM modules) connecting to the multi-drop AIMBus. The master will send the command and if slaves do not respond within a certain time-out (silence), the master may retry the same command or send a different command until it receives a response. An arbitration process will determine which device should be the recipient. Every message sent over the multi-drop AIMBus is received by all AIM modules and ignored by all except

Table 3: Detailed parameters of the evaluated CPU-side acceleration and memory-side acceleration architecture.

Copy of accelerators	1, 2, 4, 8, 16
Shared L2 (LLC) cache	32 banks, 8MB, 8-way set associative, 10 cycles latency
Coherence protocol	MOSI protocol for CPU-side acceleration
FPGA region	FPGA from Xilinx Zynq 702 board. 52K LUTS, 106K FF, 140 BRAM, 220 DSPs
DRAM memory	DDR3-1600 DIMMs, 12.8 GB/s peak bandwidth, 11.25 ns CAS latency 1, 2, 4, 8, 16 DIMMs, total capacity of 8 GB for all cases except MergeSort
Memory network	a DIMM tree topology, the root has 51.2 GB/s peak bandwidth for CPU, four channels in the sub-tree, each sub-tree channel has 12.8 GB/s peak bandwidth and supports up to 4 DIMMs
AIMBus	3.2 GB/s, 6.4 GB/s, 12.8 GB/s (default), 25.6 GB/s peak bandwidth, multi-drop bus

the single recipient. We used the polling method in which each AIM module should be polled around every 10 nanoseconds. It can be done by the POLL command or any other appropriate command. Though current technology may limit the bandwidth of a single-channel multi-drop bus, we believe we will have higher bandwidth in the future. For experimental purposes, we use an AIMBus with a bandwidth of 3.2 GB/s, 6.4 GB/s, 12.8 GB/s, and 25.6 GB/s to demonstrate the benefits of AIMBus with an increasing bandwidth trend. Note that in our current settings, each DIMM has different memory address regions, so there is no need for the coherency requirement of AIMBus.

3.4 AIM Programming

In a system using AIM, the CPU serves as the master and AIM modules serve as workers. Next we present how to launch and poll/end the AIM modules.

Launching AIM modules. To configure the AIM modules (workers) before performing any task, we maintain a particular memory address range called configuration ACMR (accelerator control memory region), which is primarily used for the configuration of accelerators and for communication of control signals. In order to protect against the OS allocating regions of the ACMR as regular program memory, a system featuring AIMs indicates that the ACMR is memory mapped I/O for a dummy device. Memory requests to the ACMR are routed independently from the normal memory requests; this is done by the memory controller that overrides the DIMM select bit.

Polling/ending AIM modules. Because the memory controller does not support unsolicited responses from memory, an AIM module cannot send information to the CPU without having a CPU request (read) to its polling ACMR. In theory, requiring the CPU to poll the AIM modules for progress updates (e.g., to see whether a task has been completed) may seem highly inefficient. In practice, however, the inefficiency of polling is offset by the long-running nature of the data parallel computations over large volumes of data, combined with a quite predictable accelerator execution time [10]. As a result, the AIM module can generate an estimated time of the task execution. The CPU polls current estimates, then waits until the estimated time has elapsed before polling again. This process repeats until a poll results in a notification that the task is complete. As a result, the polling of AIM modules is very effective.

Software running on AIM. As presented above, software running on the CPU communicates with AIM modules by reading and writing memory within the ACMR region. To make it easier to program software running on AIM, we use virtual address for the accelerators. One issue is the address translation for the accelerators. To minimize the address translation overhead, we use the large page size (2MB) and leverage the CPU TLB and OS to translate virtual address to physical address. As a result, this address translation overhead is negligible. Another issue we have to handle is the data coherency for the ACMR. To make the data coherent between the CPU and AIM modules, a cache block mapping to the ACMR must be flushed from the cache into the memory before any change is made visible to the AIM module. This cache flush can be done through an explicit invalidation by the software running on the CPU. Once flushed, the cache block is written to memory and visible to the relevant AIM module(s). The exact protocol for interacting between the CPU software and the AIM accelerators is application-specific, and thus beyond the scope of this section. Instead, our methodology focuses on extending a mechanism for communication, rather than making demands on a particular communication protocol.

4 EVALUATION METHODOLOGY

In our experiments we simulate the AIM module implementations according to the reconfigurable low-end FPGAs; the retail price of a low-end Zynq FPGA board (including an ARM processor, DRAM, and other components) is around $100. Note that the Zynq FPGA chip itself can be less than $10 at volume purchase, which is relatively cheaper than DRAMs. We evaluate both the recent CPU-side acceleration [10, 11] and our memory-side acceleration (AIM). In both cases, we make use of the same set of accelerators—the only difference is that the accelerators are within a CPU chip or an AIM module. We chose this point of comparison, instead of a comparison to a conventional CPU without accelerators, since there has been extensive prior work illustrating the performance and energy benefits of CPU-side acceleration over conventional CPU [10, 11, 15, 17]. We thus felt that it would be a more meaningful study if the comparison was made to a more recent CPU-side acceleration.

Table 3 lists the detailed architecture parameters. To illustrate the better scalability of AIM, we demonstrate scaling from 1 to 16 DDR3 DIMMs. Though we can scale the memory capacity when increasing the number of DIMMs, we keep the total memory capacity at 8GB for a fair comparison. [3] Each AIM module is implemented using a low-end FPGA such as the one used in the Xilinx Zynq 720 board. For the CPU-side acceleration, we also scale the copy of accelerators

[3] MergeSort is an exception where we use 8 GB per DIMM because it keeps merging data from two DIMMs to one DIMM.

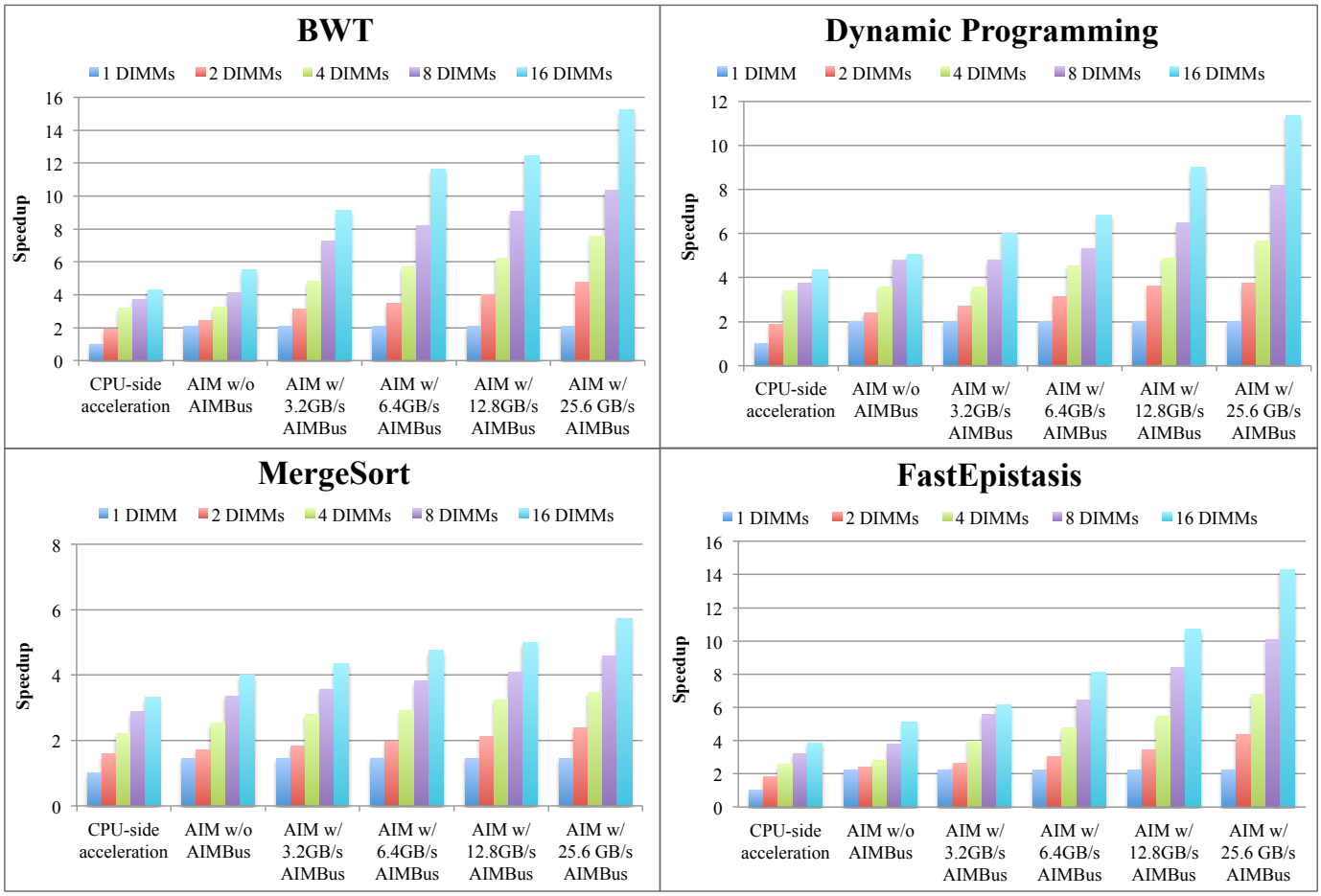

Figure 7: Performance scalability of CPU-side acceleration and AIM with different AIMBus settings for genomics applications, as the number of accelerators and DIMMs increases from 1 to 16.

from 1 to 16 within a single socket. All the CPU accelerators share a coherent 32-banked 8MB last-level cache (L2). For the conventional memory network, we use a DIMM tree topology: the root has 51.2 GB/s peak bandwidth for CPU; there are four channels in the sub-tree where each sub-tree channel has 12.8 GB/s peak bandwidth and supports up to four DIMMs. For our proposed AIMBus, we vary the AIMBus peak bandwidth with 3.2 GB/s, 6.4 GB/s, 12.8 GB/s, and 25.6 GB/s to demonstrate the impact of AIMBus [41]. Note that the idea of AIM is not limited to DDR3, it can be applied to DDR4 or GDDR5 as well. And the AIMBus can also be improved with the application of future technologies. We use this setting for experimental purposes only.

We model the above architecture by extending the widely used Simics [31] and GEMS [33] simulator. We used the Xilinx Vivado HLS [14] to calculate the latency and power of the integrated FPGA accelerators. We use the characteristics of a PCIe bus to estimate the power of the AIMBus [21] and DRAMPower [6] to model DIMM power consumption. A set of four representative genomics applications described in Section 2 are used to illustrate the benefits of our proposed AIM system.

5 AIM RESULTS

In this section we evaluate the performance gains for the genomics applications on AIM compared to the CPU-side acceleration. We first demonstrate the overall performance scalability of AIM when there is an increasing number of DIMMs. We also vary the peak bandwidth of the AIMBus. To gain deep insights into the performance gains, we compare the average memory access latency, aggregate memory bandwidth, and memory network utilization/congestion between AIM and CPU-side acceleration. Finally we present the overall energy savings of AIM.

5.1 Overall Performance Scalability

We demonstrate the overall performance scalability of AIM compared to the CPU-side acceleration when there is an increasing number of DIMMs (and associated CPU accelerators or AIM modules) from 1 to 16. To illustrate the impact of the AIMBus design, we include the following settings with different AIMBus bandwidths: 1) AIM without AIMBus, 2) AIM with 3.2 GB/s AIMBus, 3) AIM with 6.4 GB/s AIMBus, 4) AIM with 12.8 GB/s AIMBus (this is the default version we will use in the rest of this paper), 5) AIM with

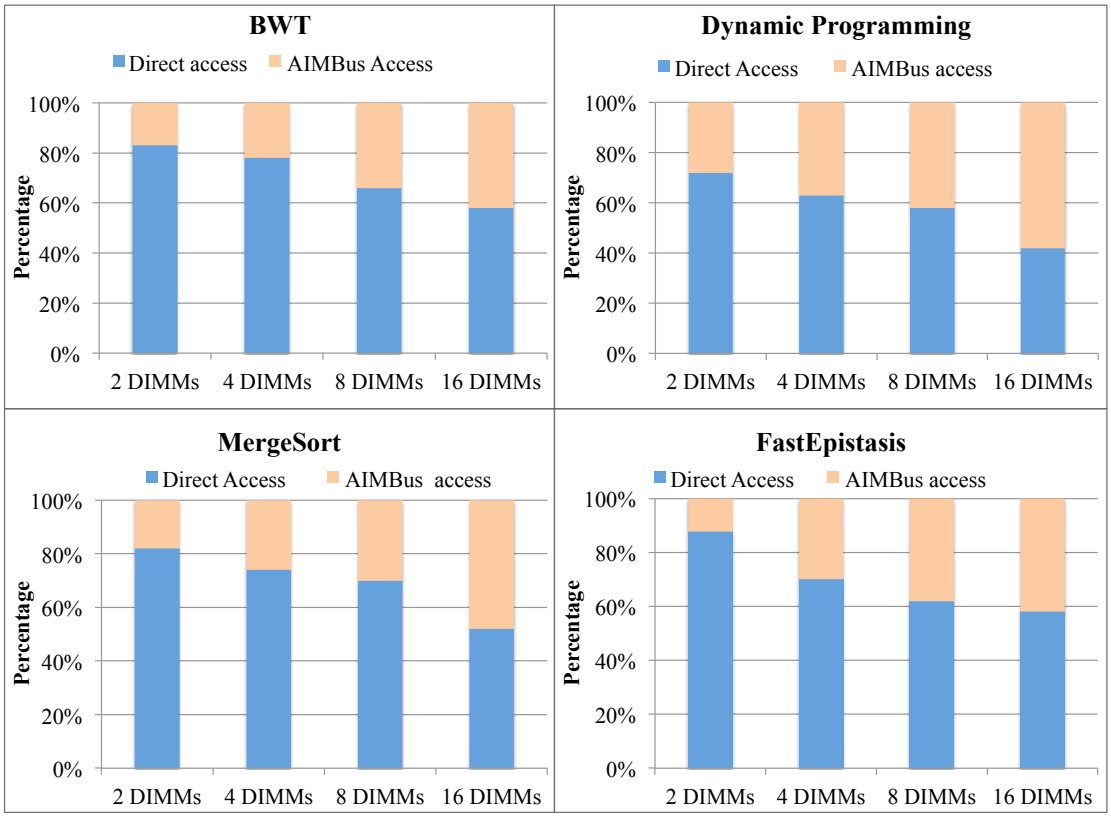

Figure 8: Percentage of memory accesses that use the local DIMM and remote DIMM, as the number of DIMMs increases.

25.6 GB/s AIMBus. All performance is normalized to the one CPU-side accelerator with one DIMM baseline. Figure 7 presents the detailed performance speedup for the genomics applications.

There are a number of observations that can be extracted from Figure 7. For ease of description, we take BWT as an example since all benchmarks (except merge sort) have results similar to that of BWT. Merge sort has limited parallelism in later merge steps, and thus has marginal performance speedup for AIM over CPU-side acceleration.

1. For the one-DIMM case, AIM achieves around 2x performance speedup compared to the CPU-side acceleration baseline, mainly because the lower latency (around 17 nanoseconds) is achieved by moving the accelerator closer to the DIMM. Detailed memory access latency numbers with increasing number of DIMMs are presented in Section 5.2.
2. When the number of DIMMs increases, the performance of CPU-side acceleration does not scale well when exceeding four DIMMs. It is mainly limited by the memory network bandwidth and pin-out that connects the CPU to the memory modules. Section 5.3 demonstrates a more quantitative evaluation of the memory network utilization.
3. The performance of AIM without AIMBus usually shows slightly better performance compared to the CPU-side acceleration baseline, because for most of the memory accesses, the AIM module accesses its associated local DIMM and does not need to go through the memory network. But we find that the performance is still not satisfactory. The main reason is that for any remote DIMM access between DIMM 1 and DIMM 2, the request must go from DIMM 1 to the CPU and then go to DIMM 2. Then similarly, the response goes from DIMM 2 to the CPU and finally goes to DIMM 1. This incurs a very long inter-DIMM access latency. Figure 8 presents the local and remote DIMM access percentage for each genomics application. With the number of DIMMs increasing, the percentage of remote memory accesses increases significantly. This drives the need for AIMBus which can achieve a much lower remote memory access latency (with an additional 10 nanoseconds if no AIMBus congestion).
4. The performance for AIM with AIMBus scales linearly with the increasing number of DIMMs, and can achieve a much better speedup compared to AIM without AIMBus. When the bandwidth of AIMBus increases from 3.2 GB/s to 25.6 GB/s, the absolute performance speedup keeps increasing. For the default 12.8 GB/s AIMBus bandwidth, it can achieve up to 12.4x performance speedup with 16 DIMMs compared to the CPU-side acceleration with one DIMM as baseline. Even compared to the CPU-side acceleration with 16 DIMMs, AIM with 16 DIMMs can achieve up to 3.7x speedup with a 25.6 GB/s AIMBus.

5.2 Average Memory Access Latency

The average memory access latency for BWT backward search is shown in Table 4. In the CPU-side acceleration baseline, a sharp

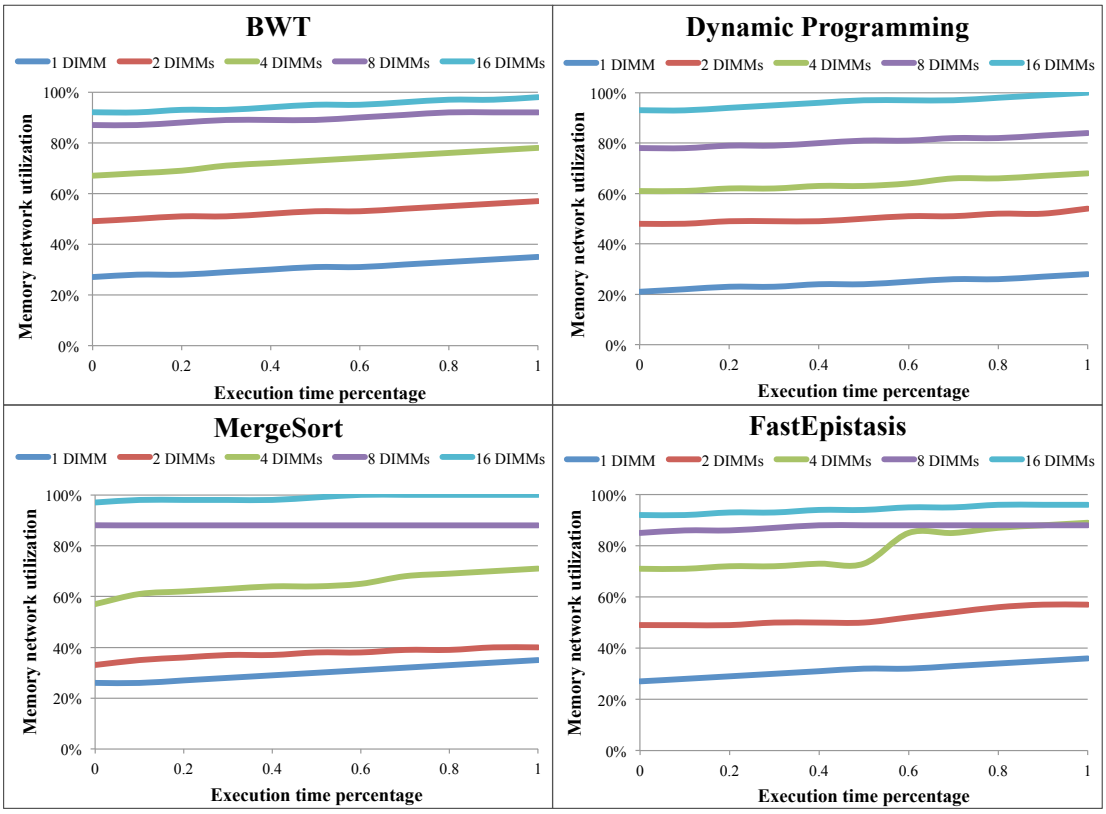

Figure 9: Memory network bandwidth utilization for the CPU-side acceleration baseline: congestion after 4 DIMMs.

Table 4: Average memory access latency in nanoseconds for the BWT application (similar for others).

Number of DIMMs	1	2	4	8	16
CPU-side acceleration (ns)	112	113.5	130.5	237.5	487
AIM w/ 12.8 GB/s AIMBus (ns)	17	19	20.1	23	27.5

increase in memory latency is observed once the memory wall is hit. As more accelerators and DIMMs are added to the CPU-side acceleration, the pressure on the memory system increases. When it exceeds the available memory network bandwidth, all additional accesses result in a sharp increase in memory access latency, as accesses begin to build up behind the memory controller. This results in a reduction in per-accelerator performance, as an increased number of accelerators share the limited bandwidth. In contrast, in the system with AIM (and 12.8 GB/s AIMBus by default), many of the accesses are local to the associated DIMM which only exploits around 17 nanoseconds. Only remote DIMM accesses have to compete for the bandwidth of the AIMBus, which still incurs a much shorter latency because different DIMMs are directly connected by the AIMBus and do not go through the conventional memory network. The average memory access latency of AIM with 12.8 GB/s AIMBus is also shown in Table 4. It is much shorter than the memory access latency of CPU-side acceleration with the number of DIMMs increasing. While not shown, other benchmarks exhibited similar memory access latency.

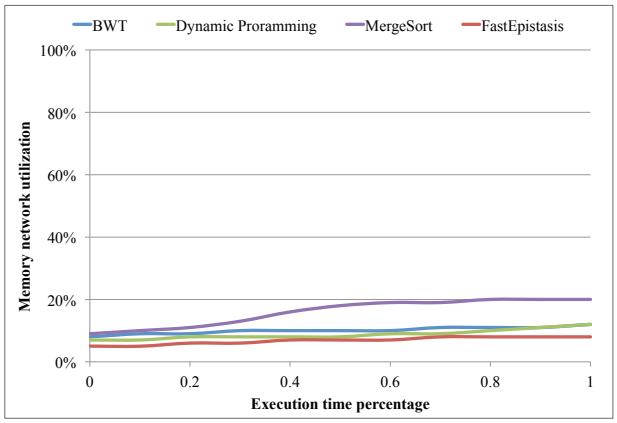

Figure 10: Low memory network bandwidth utilization for AIM with 12.8 GB/s AIMBus using 16 DIMMs.

5.3 Memory Network Bandwidth Utilization

In most cases, our CPU-side acceleration baseline system encounters the memory wall after adding more than four DIMMs. To illustrate this, we profile the utilization of the memory network bandwidth of the CPU-side acceleration in Figure 9. Even though the accelerator resources are scaled up in the CPU side as the number of DIMMs are scaled up, no performance improvement is observed once the memory network is fully saturated. Figure 10 illustrates

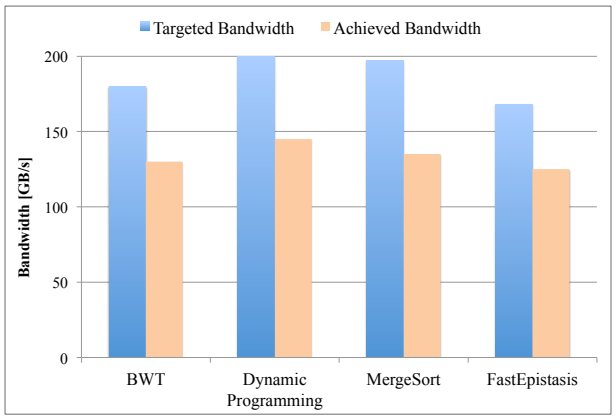

Figure 11: Achieved aggregate memory bandwidth for AIM with 12.8 GB/s AIMBus using 16 DIMMs: around 70% of targeted bandwidth.

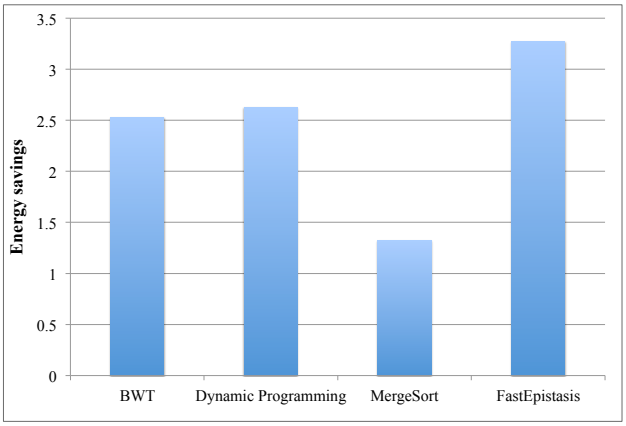

Figure 12: Energy savings for genomics applications of AIM over CPU-side accelerators for 16 accelerators/DIMMs.

a similar measurement of the memory network utilization for systems featuring AIM with 16 DIMMs. As clearly shown, nearly all of the memory network traffic is eliminated while executing the benchmark.

5.4 Aggregate Memory Bandwidth

To further demonstrate the benefits of AIM, we present the actual aggregate memory bandwidth of AIM with 12.8 GB/s AIMBus using 16 DIMMs. As shown in Figure 11, it achieves around 70% of the targeted bandwidth, which is much better than the CPU-side acceleration shown in Figure 4. Note that the peak bandwidth we provide for AIM with 16 DIMMs is 16*12.8GB/s = 204.8 GB/s, but usually it is impossible to achieve this peak bandwidth.

5.5 Energy Savings

Finally, we also present the overall energy savings of AIM with 12.8 GB/s AIMBus over CPU-side acceleration using 16 accelerators and DIMMs in total. As shown in Figure 12, AIM achieves 1.3x to 3.3x energy savings over CPU-side acceleration for different applications. Most of the energy savings come from the performance improvement as presented before. There is little power saving in AIM although it almost eliminates the conventional memory network utilization. The reason is that it introduces an AIMBus which consumes comparable power to that of the memory network.

6 RELATED WORK

Early work such as CRAM [13], IRAM [35] and PIM [40] integrate compute engines directly into DRAM modules. These architectures are useful for simple operations, such as vector addition, but not for more complex applications. Although these architectures have very high bandwidth, their cost and design complexity are high due to the integration of accelerators directly into the internal structure of DRAM. Such designs are less able to leverage economy of scale to keep DRAM prices low. In contrast, our design is modular and does not require any modification to an existing system.

More recently, some work like Copacobana [39] integrates FPGA accelerators directly into DIMMs. However these designs require a change to the memory controller to allow it to interact with the modules. These modules are mostly useful for embedded and specialized systems. In contrast, our system does not need any modification to the existing system. There are some commercial approaches that add FPGA as a separate socket into the Intel Xeon server for acceleration and share the same memory. Examples include the Convey machines [12] and the Intel-Altera Heterogeneous Architecture Research Platform (HARP) [19]. However, these FPGA accelerators still incur regular long memory access latency.

Most recent studies focus on 3D stacked memory, such as the Hybrid Memory Cube (HMC) [22] and High-Bandwidth Memory (HBM) [25], which present an entirely new category of high-performance memory with large bandwidth and lower latency. There has been some work that places hardware accelerators in such 3D stackable memories. For example, [47] suggested placing hardware accelerators at the bottom layer of a 3D stacked memory and accelerating multiple big-data workloads like sorting, string matching and memcopy. [1] suggested combining a DRAM-aware reshape accelerator integrated with 3D-stacked DRAM and accelerating data reorganization routines selected from the Intel Math Kernel Library package. Our approach is orthogonal to 3D stacked memory and could also be integrated into a system like the 3D stacked HMC—in fact, it would allow a unification of the memory network and AIMBus.

There is also work on tackling the limitations of processor packaging pin-out by introducing new interconnects such as radio frequency [7] and optical networks [42]. Although these technologies are effective for overcoming the memory wall to some degree, they are ultimately limited by the bandwidth achievable with their aggregation of frequencies or wavelengths, as well as the aggregate bandwidth of all DIMMs in the system. Our design is only limited by the aggregate bandwidth of all DIMMs in the system, irrespective of the bandwidth of the memory network or pin-out.

In addition to the above near memory computing architectures, a distributed flash store has been proposed to accelerate big data analytics (Blue Database Machine) [23] by putting accelerators closer to disk. Our design is similar in that we integrate computation

with memory, but we do so at a different level where we may also have closer communication with the CPU.

Finally, some prior work has been done in accelerating genomics applications using FPGA-based hardware acceleration [8, 34]. However, these studies are limited in their scalability when adding new memory modules. There has also been some work in using GPUs for computational genomics like BWT [27] and the detection of epistasis [24], which are much more power-hungry than our AIM accelerators.

7 CONCLUSION

In this paper we present a novel, scalable and non-invasive near-memory acceleration platform called AIM that interposes accelerators near the DRAM. AIM envelopes the FPGA accelerators as a separate package and places it between the DIMM and memory network. It uses off-the-shelf everything including OS, CPU, DRAM, and memory controller. A much lower memory access latency and scalable memory capacity and bandwidth are achieved when adding more DIMMs into the system. Our design is beneficial for highly data-parallel computations with poor locality and bandwidth-intensive workloads—like computational genomics. These tasks are heavily impacted by the overwhelming memory wall bottleneck in conventional CPU-side acceleration platforms. We demonstrate an up to 3.7x performance speedup of AIM compared to the CPU-side acceleration.

8 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This work is partially supported by the Center for Domain-Specific Computing under the NSF InTrans Award CCF-1436827; funding from CDSC industrial partners including Baidu, Fujitsu Labs, Google, Huawei, Intel, IBM Research Almaden and Mentor Graphics; C-FAR, one of the six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA; and UCLA Institute for Digital Research and Education Postdoc Fellowship.

REFERENCES

[1] Berkin Akin, Franz Franchetti, and James C Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 131–143.

[2] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. 1990. Basic local alignment search tool. *Journal of molecular biology* 215, 3 (1990), 403–410.

[3] Vijay Anand. 2004. NVIDIA's Scalable Link Interface (SLI). *HardwareZone. com, Jun* 30 (2004).

[4] Alan Ashworth, Christopher J Lord, and Jorge S Reis-Filho. 2011. Genetic interactions in cancer progression and treatment. *Cell* 145, 1 (2011), 30–38.

[5] Multi-Drop Bus. 2011. Multi-Drop Bus / Internal Communication Protocol. (2011). http://www.vending.org/images/pdfs/technology/mdb_version_4-2.pdf

[6] Karthik Chandrasekar, Christian Weis, Yonghui Li, Benny Akesson, Norbert Wehn, and Kees Goossens. 2012. DRAMPower: Open-source DRAM power & energy estimation tool. *URL: http://www. drampower. info* (2012).

[7] M. F. Chang, I. Verbauwhede, C. Chien, Z. Xu, J. Kim, J. Ko, Q. Gu, and B. Lai. 2005. Advanced RF/Baseband Interconnect Schemes for Inter- and Intra-ULSI communications. In *IEEE Transactions on Electron Devices*.

[8] Jason Chiang, Michael Studniberg, Jack Shaw, Stephen Seto, and Kevin Truong. 2006. Hardware accelerator for genomic sequence alignment. In *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*. IEEE, 5787–5789.

[9] Francis S Collins, Eric D Green, Alan E Guttmacher, and Mark S Guyer. 2003. A vision for the future of genomics research. *Nature* 422, 6934 (2003), 835–847.

[10] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, and Glenn Reinman. 2012. Architecture support for accelerator-rich cmps. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, 843–849.

[11] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, and Glenn Reinman. 2012. CHARM: a composable heterogeneous accelerator-rich microprocessor. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 379–384.

[12] Convey. 2015. [Online]. *Available:* (2015). http://www.conveycomputer.com/products/hcseries/

[13] Duncan G Elliott, Michael Stumm, W Martin Snelgrove, Christian Cojocaru, and Robert McKenzie. 1999. Computational RAM: Implementing processors in memory. *Design & Test of Computers, IEEE* 16, 1 (1999), 32–41.

[14] Tom Feist. 2012. Vivado design suite. *Xilinx, White Paper Version* 1 (2012).

[15] Hubertus Franke, Jimi Xenidis, Claude Basso, Brian M Bass, Sandra S Woodward, Jeffrey D Brown, and Charles L Johnson. 2010. Introduction to the wire-speed processor and architecture. *IBM Journal of Research and Development* 54, 1 (2010), 3–1.

[16] Cristina Y González, Marta Bleda, Francisco Salavert, Rubén Sánchez, Joaquín Dopazo, and Ignacio Medina. 2013. Multicore and cloud-based solutions for genomic variant analysis. In *Euro-Par 2012: Parallel Processing Workshops*. Springer, 273–284.

[17] Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. 2011. Dynamically specialized datapaths for energy efficient computing. In *HPCA'11*. IEEE, 503–514.

[18] Alan E Guttmacher, Amy L McGuire, Bruce Ponder, and Kári Stefánsson. 2010. Personalized genomic information: preparing for the future of genetic medicine. *Nature Reviews Genetics* 11, 2 (2010), 161–165.

[19] Intel HARP. 2015. [Online]. *Available:* (2015). http://www.sigarch.org/2015/01/17/call-for-proposals-intel-altera-heterogeneous-architecture-research-platform-program/

[20] Laiq Hasan, Zaid Al-Ars, and Stamatis Vassiliadis. 2007. Hardware acceleration of sequence alignment algorithms-an overview. In *Design & Technology of Integrated Systems in Nanoscale Era, 2007. DTIS. International Conference on*. IEEE, 92–97.

[21] Analog ICs. 2012. NXP SEMICONDUCTOR Analog ICs. (2012).

[22] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 87–88.

[23] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2016. BlueDBM: Distributed Flash Storage for Big Data Analytics. *ACM Trans. Comput. Syst.* 34, 3, Article 7, 31 pages. https://doi.org/10.1145/2898996

[24] Tony Kam-Thong, C-A Azencott, Lawrence Cayton, Benno Pütz, André Altmann, Nazanin Karbalai, Philipp G Sämann, Bernhard Schölkopf, Bertram Müller-Myhsok, and Karsten M Borgwardt. 2012. GLIDE: GPU-based linear regression for detection of epistasis. *Human heredity* 73, 4 (2012), 220–236.

[25] Joonyoung Kim and Younsu Kim. 2014. HBM: Memory Solution for Bandwidth-Hungry Processors. In *HotChips*.

[26] Christoforos Kozyrakis, Joseph Gebis, David Martin, Samuel Williams, Ioannis Mavroidis, Steven Pope, Darren Jones, David Patterson, and Katherine Yelick. 2000. Vector IRAM: A media-oriented vector processor with embedded DRAM. In *Proc. Hot Chips XII*.

[27] Tak-Wah Lam, Kunihiko Sadakane, Wing-Kin Sung, and Siu-Ming Yiu. 2002. A space and time efficient algorithm for constructing compressed suffix arrays. In *Computing and Combinatorics*. Springer, 401–410.

[28] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (2010), 589–595.

[29] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. 2009. The sequence alignment/map format and SAMtools. *Bioinformatics* 25, 16 (2009), 2078–2079.

[30] Yongchao Liu and Bertil Schmidt. 2012. Evaluation of GPU-based seed generation for computational genomics using Burrows-Wheeler transform. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 684–690.

[31] Peter S. Magnusson et al. 2002. Simics: A Full System Simulation Platform. *Computer* 35 (2002), 50–58.

[32] Teri A Manolio, Francis S Collins, Nancy J Cox, David B Goldstein, Lucia A Hindorff, David J Hunter, Mark I McCarthy, Erin M Ramos, Lon R Cardon, Aravinda Chakravarti, et al. 2009. Finding the missing heritability of complex diseases. *Nature* 461, 7265 (2009), 747–753.

[33] M. Martin et al. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. In *Computer Architecture New, Sep 2005*.

[34] Agathoklis Papadopoulos, Ioannis Kirmitzoglou, Vasilis J Promponas, and Theocharis Theocharides. 2013. FPGA-based hardware acceleration for local complexity analysis of massive genomic data. *Integration, the VLSI Journal* 46, 3 (2013), 230–239.

[35] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *MICRO'97* 17, 2 (1997), 34–44.

[36] Jonathan Pevsner. 2009. *Bioinformatics and functional genomics*. John Wiley & Sons.

[37] Louis-Noël Pouchet, Peng Zhang, P. Sadayappan, and Jason Cong. 2013. Polyhedral-Based Data Reuse Optimization for Configurable Computing. In *21st ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'13)*. ACM Press, Monterey, California.

[38] Tuomo Rankinen, Aamir Zuberi, Yvon C Chagnon, S John Weisnagel, George Argyropoulos, Brandon Walts, Louis Pérusse, and Claude Bouchard. 2006. The human obesity gene map: the 2005 update. *Obesity* 14, 4 (2006), 529–644.

[39] S. Kumar C. Paar J. Pelzl G. Pfeiffer M. Schimmler. 2009. *Breaking Ciphers with COPACOBANA .A Cost-Optimized Parallel Code Break.*

[40] Toshia Sunaga, Peter M Kogge, et al. 1996. A processor in memory chip for massively parallel embedded applications. *IEEE J. of Solid State Circuits* (1996), 1556–1559.

[41] Michael Tan, Paul Rosenberg, Jong Souk Yeo, Moray McLaren, Sagi Mathai, Terry Morris, Huei Pei Kuo, Joseph Straznicky, Norman P Jouppi, and Shih-Yuan Wang. 2009. A high-speed optical multi-drop bus for computer interconnections. *Applied Physics A* 95, 4 (2009), 945–953.

[42] Michael Tan, Paul Rosenberg, Jong Souk Yeo, Moray McLaren, Sagi Mathai, Terry Morris, Huei Pei Kuo, Joseph Straznicky, Norman P Jouppi, and Shih-Yuan Wang. 2009. A high-speed optical multi-drop bus for computer interconnections. *Applied Physics A* 95, 4 (2009), 945–953.

[43] M. Tan, P. Rosenberg, Jong Souk Yeo, M. McLaren, S. Mathai, T. Morris, J. Straznicky, N.P. Jouppi, Huei Pei Kuo, Shih-Yuan Wang, S. Lerner, P. Kornilovich, N. Meyer, R. Bicknell, C. Otis, and L. Seals. 2008. A High-Speed Optical Multi-Drop Bus for Computer Interconnections. In *High Performance Interconnects, 2008. HOTI '08. 16th IEEE Symposium on*. 3–10.

[44] John Watkins, Raymond Roth, Michael Hsieh, William Radke, Donald Hejna, Byung Kim, and Richard Tom. 1993. A memory controller with an integrated graphics processor. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD'93. Proceedings., 1993 IEEE International Conference on*. IEEE, 324–338.

[45] Scott M Williams, MD Ritchie, JA Phillips Iii, E Dawson, M Prince, E Dzhura, A Willis, A Semenya, M Summar, BC White, et al. 2004. Multilocus analysis of hypertension: a hierarchical approach. *Human heredity* 57, 1 (2004), 28–38.

[46] Xilinx. 2014. Enabling High-Speed Radio Designs with Xilinx All Programmable FPGAs and SoCs. (2014). Retrieved May 20, 2016 from http://www.xilinx.com/support/documentation/white_papers/wp445_hi-speed-radio-design.pdf

[47] Salessawi Ferede Yitbarek, Tao Yang, Reetuparna Das, and Todd Austin. 2016. Exploring specialized near-memory processing for data intensive operations. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1449–1452.

[48] Eleftheria Zeggini, Michael N Weedon, Cecilia M Lindgren, Timothy M Frayling, Katherine S Elliott, Hana Lango, Nicholas J Timpson, JR Perry, Nigel W Rayner, Rachel M Freathy, et al. 2007. Wellcome Trust Case Control Consortium (WTCCC), McCarthy MI, Hattersley AT: Replication of genome-wide association signals in UK samples reveals risk loci for type 2 diabetes. *Science* 316, 5829 (2007), 1336–1341.