Merge Network for a Non-von Neumann Accumulate Accelerator in a 3D Chip

Anirudh Jain and Sriseshan Srikanth
Georgia Institute of Technology
Atlanta, Georgia
Email: {anirudh.j, seshan}@gatech.edu

Erik P. DeBenedictis
Center for Computing Research
Sandia National Laboratories
Albuquerque, New Mexico
Email: epdeben@sandia.gov

Tushar Krishna
Georgia Institute of Technology
Atlanta, Georgia
Email: tushar@ece.gatech.edu

***Abstract*—Logic-memory integration helps mitigate the von Neumann bottleneck, and this has enabled a new class of architectures that helps accelerate graph analytics and operations on sparse data streams. These utilize merge networks as a key unit of computation. Such networks are highly parallel and their performance increases with tighter coupling between logic and memory when a bitonic algorithm is used. This paper presents energy-efficient on-chip network architectures for merging key-value pairs using both word-parallel and bit-serial paradigms. The proposed architectures are capable of merging two rows of high bandwidth memory (HBM) worth of data in a manner that is completely overlapped with the reading from and writing back to such a row. Furthermore, their energy consumption is about an order of magnitude lower when compared to a naive crossbar based design.**

1. Introduction

Moore's law projected that two-dimensional chips would hold exponentially more transistors over time, but computer performance is obviously flatlining [1]. Multiple hardware threads like those in multi-core, many-core, and general purpose graphics processing units (GPGPUs) can make use of additional transistors for many "general purpose" applications. However, adding parallelism into the traditional von Neumann architecture does not help applications that have a fundamentally low memory access locality, and only perform small amounts of computation per memory access. In hopes of resuming the trend of rising computer performance for such applications, research has begun on non-von Neumann accelerators that use logic and memory integrated in 2.5D and 3D modules – thus allowing the number of devices to increase without physically unrealistic reductions in transistor dimensions.

High Bandwidth Memory is an example of this integration, where 4-8 slightly modified DRAM chips are stacked using through-silicon vias (TSVs) on a logic-containing base layer (or, logic is possibly bonded via a silicon interposer for 2.5D), providing a larger number of shorter wires between memory and logic. This dense packaging is possible only because the stacked memory chips have low power individually, but overall modules dissipate over 100W at peak levels [2].

Integration of logic and memory on one chip is a key advance that mitigates the von Neumann bottleneck resulting from an off-chip bus between main memory and the cache hierarchy at the processing core [3], [4], [5]. The reduction in data movement through the memory hierarchy and the processing units results in both performance gain and energy savings. The reduction has widened the scope for research into various on-chip algorithms such as sorting and merging that were previously thought to be infeasible at reasonable scales.

In this work we present and evaluate several near-memory on-chip merge network architectures for key-value records stored in current High Bandwidth Memory (HBM) or successors according to the expected scaling path, including collision detection and compression (reduction) that is key to the associative array programming paradigm [6]. Our work focuses on merge networks that make use of the highly parallel bitonic sort algorithm [7].

The merge network has the desirable property that its performance will scale up with tighter coupling between the logic and the memory, which is equivalent to a hypothetical sequence of HBM chips where the current 128-bit data path gets wider. Tightening of coupling in 3D is the new scale-up path in current roadmaps [1].

First, this paper presents the design tradeoffs in a variety of word-parallel network-on-chip architectures for the merge network. When compared to a naive full-crossbar design, our approach achieves a latency reduction of $28\times$ and energy reduction of $4\times$ for merging (and compressing/reducing) a given set of records.

Second, by bit-transposing the in-memory representation of records, this paper presents a serial design that is able to leverage a bitonic bit-level sorting network that can be tightly integrated with HBM row reads at a burst (128-bit) granularity. We find that this approach offers a reduction of $2.5\times$ in energy consumption when compared with the word-parallel architecture.

The remainder of this paper is organized as follows: Section 2 presents a brief overview of a bitonic network and the Superstrider accelerator that utilizes the merge network as a fundamental unit of operation, although the network

architectures presented in this paper can be extended to independent sorting accelerators. Section 3 describes the fundamental design tradeoffs in marshalling data to and from a single unit of computation, Section 4 scales this to leverage several units of computation available and parallelism. Section 5 then describes our second approach of using a bit-level sorting network. Experimental results, related work and a conclusion are presented thereafter.

2. Background

2.1. Superstrider

Superstrider fundamentally reduces records stored in memory [4], [5]. A record is a key-value pair, with two registers of K records called Mem (memory) and Acc (accumulator) illustrated below in the terminology of the associative array computing paradigm [6].

TABLE 1. TWO GROUPS OF RECORDS, K=4

Mem				Acc			
[1]=2	[2]=2	[3]=1	[4]=2	[5]=3	[3]=2	[2]=1	[1]=1

Records can be any size needed by the application, but HBM rows are 16,384 bits long, transferred as registers of 128 128-bit records, so we will use 128-bit records as examples in this work, without loss of generality. The example records will be comprised of 64-bit key and value fields, both unsigned integers. Roadmaps project that bandwidth of 3D memory will increase over time, such as changing the HBM transfer to 64 256-bit words, 32 512-bit words, and so on [1].

The Superstrider architecture needs only one data operation for the accumulation function covered in this work although the second part of the operation has minor variants. As shown below, one register of K=4 records called Mem is concatenated with an equivalent amount of data in a register called the accumulator or Acc. Our standard definition of a register prohibits duplicate keys, but concatenating two registers into a double length "register" can yield duplicate keys initially – but not triplicates or higher. The first part of Superstrider's data operation concatenates the Mem and Acc registers, puts the records into sorted order, and merges records with the same key (which will be adjacent at that point). Merging involves deleting one record and replacing value field in the other record with the sum of the value fields, as shown below.

The higher-level algorithm also supplies a key called Pivot to the first part of the operation. The keys of all records surviving the merge, called real records, are compared with the pivot to produce two integer counts, called LT and GE, that reveal the number of records with keys less than and greater-than-or-equal to the pivot.

The Superstrider data operation puts the resulting 2K records back into Mem and Acc, but there are variants based on which records are moved to Mem versus Acc. One variant moves the *Rot* records with the smallest keys to Acc, another to Mem. Actually, the higher level algorithms uses LT and GE to compute *Rot* on the fly, yielding various different options.

TABLE 2. BASIC SUPERSTRIDER OPERATION, K=4

Mem				Acc			
[1]=2	[2]=2	[3]=1	[4]=2	[5]=3	[4]=2	[2]=1	[1]=1
Merged							
[1]=2	[1]=1	[2]=2	[2]=1	[3]=1	[4]=2	[4]=2	[5]=3
Duplicate keys detected and merged							
[1]=3	del	[2]=3	del	[3]=1	[4]=4	del	[5]=3
Result							
[1]=3	[2]=3	[3]=1	[4]=4	[5]=3	del	del	del
An alternate result							
[0]=0	[0]=0	[1]=3	[2]=3	[3]=1	[4]=4	[5]=3	[F]=0

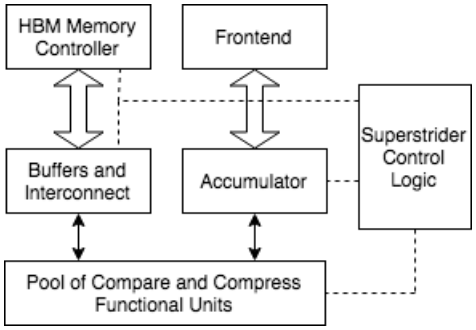

Figure 1. System Overview

Registers can hold less than K real records, with the remaining records called deleted or filler records depending on context. The most complete implementation of Superstrider uses the minimum and maximum values of the key field to indicate the record is filler, which, for the unsigned integer keys in this document, will be all zeros and all ones (0 and hexadecimal F in the illustrations). The higher level algorithm is not permitted to use these values of the key field for real records, but the underlying implementation sorts by key field irrespective of whether a record is real or filler. The sorting means a register in storage will always be ordered as all-zero filler, real records, and all-one filler.

Accumulation requires low-performance access to specific records for input, output and identifying an appropriate pivot. These, and specifics of the control logic implementing the above operations, are not covered in this paper.

Figure 1 presents a high level overview of a merge network. It is assumed that the front end, a traditional microprocessor, populates the Acc mentioned above with records that have to be merged.

2.2. Bitonic Merge

Bitonic merge is a fundamental building block of bitonic sorting networks [7]. A *bitonic* sequence is of the form $x_0 \leq x_1 ... \leq x_k \geq x_{k+1} ... \geq x_{n-1}$ or a rotation of such a sequence. In bitonic merge, comparisons are made across a stride, which starts out as half of the sum of lengths of the

sequences that are being merged. The smaller value element is swapped to the left side of the buffer. After *n* comparisons, where *n* is the length of one of the sequences, are made, the stride is halved and the compare and swap operation continues on both halves of the combined sequence. The steps described here continue until stride becomes one and pairs of neighbors are being compared. If the initial stride started out at a value of one and doubled until it reached *n* and then the steps described above were performed, the result would be a sorted sequence – hence the name bitonic sort. However, since Superstrider rows maintain the sorted invariant and the input data is a pre-sorted sequence, this paper deals with the merge algorithm starting with a stride of *n*. Without loss of generality, it is assumed that both input sequences are of identical length.

The total run time of bitonic sort is $\mathcal{O}(n \log_2^2(n))$. However, notice that each one of the individual comparisons can be made in parallel, and with *n* functional units, the complexity of the algorithm becomes $\mathcal{O}(\log_2^2(n))$. If we are concerned only with the merging of two pre-sorted bitonic sequences, the complexity is $\mathcal{O}(\log_2(n))$.

Finally, while the keys of a pair of records are being compared, the associated values can either be swapped or an associative collision/reduction function can be applied to them. Typical collision functions are trivial, such as addition, minimum etc., and can therefore be handled by the same computation unit that compares the keys.

3. Fundamental Network-on-Chip Tradeoffs

3.1. Baseline Design : Crossbar Based

Recall that the Mem buffer contains $K(= 128)$ sorted records from a SuperStrider HBM row and the Acc buffer contains *K* sorted records that are being inserted into the SuperStrider instance. However, since we wish to design our architecture such that it can be extended to sorting data that is completely unsorted to begin with, it should be possible to feed any record from either buffer to the processing unit.

The baseline single-comparator merge network (Figure 2) therefore consists of two SRAM buffers (Mem and Acc) of size K each, a comparator-collision function unit and a full crossbar that connects them. Recall that the bitonic merge operation involves comparing records across a level-dependent stride and applying the collision function on the values if the keys match. Based on the result of the comparison, either the values are reduced, and one of the records becomes the "real" record, while the other is marked for deletion, or the two records are swapped. The control unit decides which elements are compared and which buffer to write-back each record to.

Using the algorithm and architecture described above, for each buffer with K records, there are $\log_2 2K$ levels of bitonic merge, each one of them involving at most *K* reads from each buffer and at most *K* writes to each buffer. Since each buffer has two separate read and write ports, the reads and writes to the SRAM buffers can be performed in parallel for any stride of the bitonic merge operation.

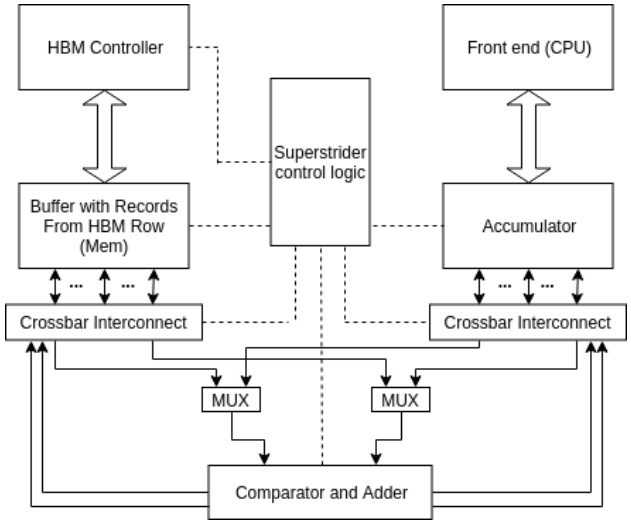

Figure 2. Crossbar Based Merge Network

This design does not require any additional buffers for storing intermittent results. However, the area, power and delay of a full crossbar scale quadratically with the number of entities it connects. This section now presents two alternative interconnects to a full crossbar to significantly improve its power-performance profile.

3.2. Rotating Buffer Design

Since a level of bitonic merge does not require access to all elements in the buffers at the *same* time, and, given that it has a structured dataflow path, a rotating buffer based design can be used, as shown in Figure 3. Mem and Acc are implemented as rotating/ring buffers with a single read port and a single write port, each. Two first-in-first-out (FIFO) buffers are added to stage swapped/compressed records back into Mem and Acc, to reduce the amount of rotation needed and to enable the reading of records separated by stride s in an efficient manner.

Initially, when $s = K$, the first records of both buffers are compared, and the buffers are rotated such that the second record is at the head, ready to be read subsequently, and so on. After the records are swapped/compressed by the comparator/adder, they are sent to the appropriate output FIFO buffer, as decided by the control unit.

After K such comparisons, $s = \frac{K}{2}$. s records from the first FIFO buffer are moved to Mem and the remaining are moved to Acc. This is then repeated for the second FIFO buffer. The result is that the records in the first locations of Mem and Acc are s records apart. Thereafter, the process of reading records out to the comparator/adder as described above can be repeated until $s = 1$.

One consequence of this interleaved movement of data for the various levels of bitonic merge is that it leaves the records shuffled once the $s = 1$ level is complete. In order to recover the final compressed and sorted record buffers, a number of reorder steps are required. $s = 1$ records have to

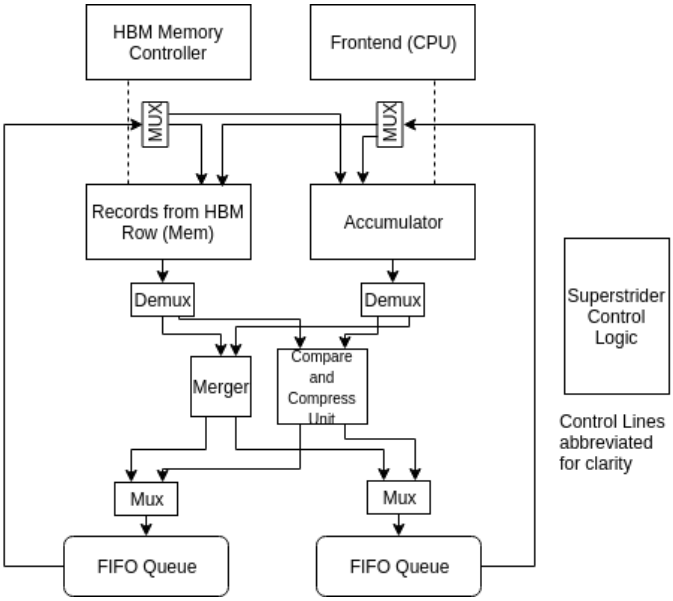

Figure 3. Rotating Buffer Design

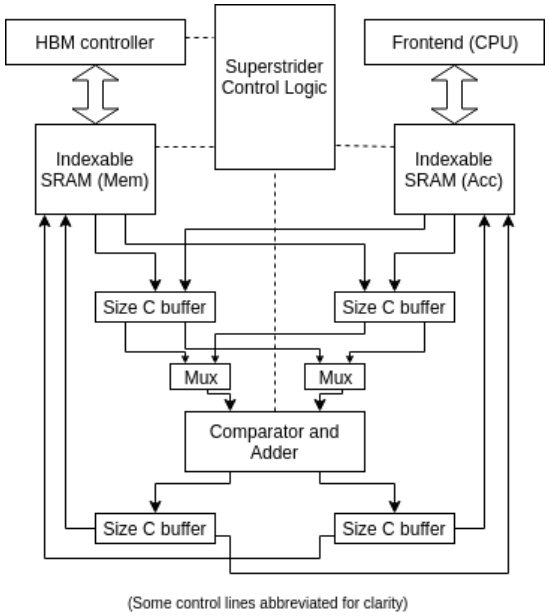

Figure 4. Indexing Buffer Design

be read in an alternating manner from both the buffers to first fill one of the rotating buffers, and then to fill the other one. s is then doubled and the process is repeated until $s = \frac{K}{2}$. This results in an additional latency and energy required that can be characterized by $Klog_2\frac{K}{2}$ read and writes from the input buffers to the FIFO buffers.

Although this design is less power-hungry than the crossbar-based design, it is slower because of the added complexity described above. As a result, the energy savings when compared to the baseline are minimal. Furthermore, when the initial sequences are not pre-sorted, the overhead of "rotation" becomes prohibitively expensive, thereby limiting the applicability of this design.

3.3. Indexing Buffer Design

To mitigate the inefficiencies above, an indexable SRAM-based design is proposed. Mem and Acc are implemented as dual-ported 2D SRAM structures where each SRAM block/line contains C records (Figure 4). At a high level, these structures can be viewed as direct mapped caches, without the overhead of tags, or, equivalently, as two-level crossbars if the cache decoder is viewed as a crossbar.

For each level of bitonic merge (stride s), a cache line is read into the first buffer, and the cache line corresponding to records that are s records away is read into the second cache line buffer. These pair of cache lines may be read from the same cache or from Mem and Acc respectively. For example, when $s = K$, the first cache line from both Mem and Acc are read, followed by the second lines from both and so on. When $s = \frac{K}{2}$, the first cache line and the $\frac{K}{2C}^{th}$ cache line from Mem are read into the cache line buffers. Note that two read ports per cache allows for the latter pair of cache lines to be read in parallel.

For each such pair of cache lines read, the records from each cache line buffer are then swapped/compressed using the comparator/adder, and the output cache line buffers are written back to Mem and Acc. Once all the cache lines have been processed, the stride is halved and the process repeats.

The indexable SRAM based design approach results in $\frac{K}{C}$ cache line reads and writes from both the Mem and Acc caches. The reads and writes can be performed in parallel during the "steady state" of operations. The total levels of the bitonic merge remain $\log_2 2K$ as discussed in the previous sections, and at most *K* comparisons are made during each level.

This design benefits from the use of cache line buffers that allow cheap (in terms of energy and latency) access to concurrent records once the cache line has been read in. Moreover, the hierarchical distribution into indexable SRAMs and smaller line size buffers retains the record access flexibility of the large crossbar design, thereby allowing the design to be extended to sorting and not just merging.

4. Scaling Word-Parallel Networks

4.1. Multiple Comparators

Recall that the primary motivation for choosing bitonic merge as the algorithm of choice as opposed to other merge techniques such as parallel merge-sort, quick sort, etc. was to leverage the structured nature of bitonic merge by extracting parallelism. Instead of utilizing a single comparator in a temporally repeated manner, the comparator can be spatially repeated. From the analysis presented in Section 3, it is clear

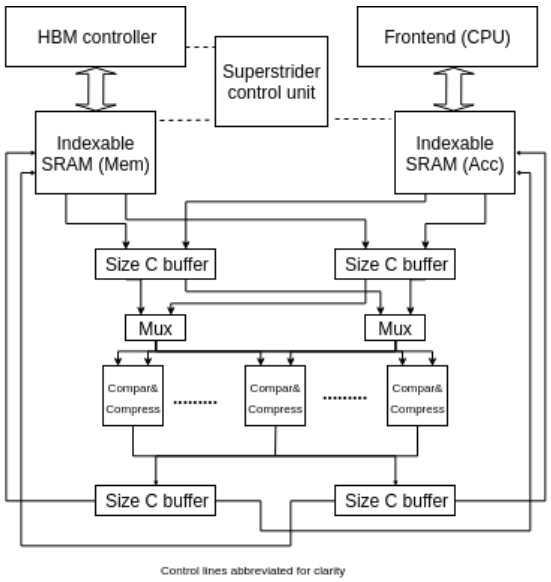

Figure 5. Multiple Comparator Design

that for a single comparator, the indexable SRAM based design presents the best trade-off between energy, performance and complexity. Therefore, for a multi-comparator design, this paper limits the analysis to the indexing buffer design.

Figure 5 shows the overall architecture of the indexable SRAM based design with multiple comparator/adder units. Similar to the architecture described in Section 3.3, this design has two dual ported indexable SRAMs (Mem and Acc), each connected to a network of comparators via C-sized buffers. To maximize performance and to minimize control logic complexity, it is easy to see that the number of comparators is C; a lower number of comparators would result in increased delay and a higher number would result in wastage.

The records from the pair of caches line read (similar to Section 3.3) are sent to the appropriate comparator serially. After records have been moved to the comparators, the C comparisons are made in parallel and records are sent to the corresponding output buffer after either a swap, a compression or a deletion to be written back to the Mem and Acc. As this write back is not on the critical path for a steady state of operations, it can be performed in parallel with the processing of the subsequent pair of cache lines. Moreover, the subsequent pair of cache lines can also be read in parallel with the swap/compression because the the delay incurred by the processing units is typically more than the latency of reading and writing from the Mem and Acc. In general, using multiple comparators allows the latency to be reduced by a factor of C.

When $s \geq C$, comparisons are made across cache lines resulting in $\frac{K}{C}$ reads per stride for a total of $\frac{K}{C}\log_2(\frac{K}{C})$ cache line reads. For the remaining $\log_2 C$ strides for which $s < C$, comparisons are made within a cache line, and every cache line in both Mem and Acc is read for every single stride. This results in a total of $\frac{K}{C}\log_2 C$ cache line reads,

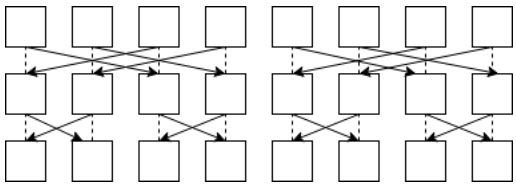

Figure 6. Compare and Compress Unit Array for C = 8

thus motivating the need to optimize pipelining for both these phases of the bitonic merge to improve utilization of read cache lines and reduce the energy and latency overhead the repeated cache accesses result in.

4.2. Pipeline Schedule

Arranging the comparators/adders as a $C \times C$ systolic array results in an elegant pipeline schedule [8].

For stride $s \geq C$, the (pipelined) cache line reads are statically routed to the systolic array, which is populated in a bottom-up manner, with the levels above the destination level acting as intermediate repeaters. That way, once the systolic array has been populated, all the comparisons/additions occur in parallel, and the resultant records can be streamed out in a lock-step fashion. Subsequent stream-ins to the systolic array from the caches can then occur in parallel with the stream-outs of the previously processed outputs in typical pipelined fashion.

For stride $s < C$ (there are $\log_2 C$ such strides), recall that the records of interest are within the same cache line. Therefore, the systolic array must make most use (process all possible strides) of data read from a cache line of Mem/Acc before it is written back into Mem/Acc. The results from one level in the systolic array to the one below it are passed in a staggered manner where the offset is equal to $\frac{s}{2}$ with $s = \frac{C}{2}$ for the first level, which is subsequently halved for every level.

Figure 6 shows the structure of such a pipeline for $C = 8$. The squares represent the compare and compress functional units, the dotted lines indicate links that are used in the first phase of the dataflow ($s \geq C$) and the solid lines indicate links that are used for the staggered connections for the second phase ($s < C$).

4.3. Pipeline Interconnect

Because of the dissimilarity in dataflows for the two phases as described above, an irregular network topology may seem like the obvious choice. However, to make it possible for this merge network to be compatible for various record sizes (applications) and across a variety of memory devices, it must be flexible for different C and K. Considering this requirement, a $C \times \log_2 C$ mesh topology has been chosen. Moreover, to mitigate the latency of multi-hop links, the interconnect network can utilize SMART routers [9] and leverage its single cycle long hop capability.

Finally, since the network traffic pattern is stride dependent, it can be determined statically and pre-programmed

into the control unit. Routers in the array (SMART or otherwise) can therefore utilize a bufferless design with no flow control overhead.

4.4. Write Buffers

Local write buffer. Recall that write out to the caches after processing is pipelined and can be overlapped with the read-in of subsequent cache lines. In order to minimize latency and reduce the number of read/write ports required in the SRAM caches, the processed records can be written into a local write buffer that can hold at least $2\log_2 C$ cache lines. Now, while the subsequent cache lines are being processed by the systolic array, the cache lines from the write buffer are written back to the SRAM caches.

Global write buffer. Along similar lines, but at a larger granularity, a write buffer of size K can overlap the write-back of the previously processed Mem/Acc contents into its HBM row with the processing of a newly read HBM row.

4.5. Interleaving HBM accesses

Data transfer from and to memory can be interleaved with the operations of the merge network. Specifically, only C records are required to feed one row of the systolic array. Once the first cache line is completely filled, the first comparisons are launched through the network. This can be done in parallel with the read of the next C records. This interleaved pattern is continued until all the records are read from memory. Therefore, the first stride of bitonic merge can be completely overlapped with an HBM row read. (The remaining strides require all the records in order to proceed.)

Once the first cache line of the Mem buffer has gone through the $s = 1$ level, it is streamed out to the global write buffer, while the subsequent cache lines are processed by the network. It should also be noticed that the next HBM row read can start in parallel with the other operations. This effectively takes the comparator network with the pipelined design off the critical path with respect to the memory read and write operations. Since the operations of the comparator network have very low latency as compared to the read and write latencies from HBM, row remap logic can be used to write from the global write buffer to the already open DRAM row in order to avoid the cost of row precharge and activation.

4.6. Record Compression

Once two records with the same key are compared, one of the records is marked with a deletion flag. The other copy of the record, referred to as the "real" record thus far in this paper, gets the result from the result of the reduction operation. Since there are no duplicate keys in Mem or Acc individually, there will be at most one pair of records with the same key for every key, and it is guaranteed that the real copy will contain the collision function output of the values. The bitmask of deleted records is stored at a cache line granularity. It is crucial that the record that is marked for deletion is not explicitly deleted (in any manner, just yet) so that the data flow of the remaining bitonic merge steps does not get affected. Once all the stages of bitonic merge have executed, all the records which don't have the delete flag set are streamed from the cache lines from Mem buffer to a global write buffer and subsequently from the Acc.

5. Sorting Network Approach

5.1. Key Observations

1) The cache-based word-parallel architecture can be further improved by tweaking the representation of the records in memory. Suppose an HBM row with K records is organized such that K keys are first laid in a contiguous manner, followed by the K values. In the first phase, reading out the keys from the Mem cache would result in fewer cache lines being accessed, thereby improving energy efficiency. In the second phase, since the keys have already been compared and swapped in the first phase, the final positions of the records are known, and the value fields can therefore be accessed and swapped/reduced as needed.
2) A bit-level sorting network works by comparing the MSBs of both inputs first. If they do not match, then one input is clearly greater than the other, and the subsequent bits of the input do not require processing. If a swap is performed for the MSBs, then the swap is performed for the subsequent bits as well. If the MSBs do not match, then the process just outlined has to be repeated by treating the next bit as the new MSB instead, and so on. If it turns out that all the bits of the inputs match, then it means that the inputs are identical. In this paper, this is the scenario where keys are equal, thereby warranting reduction on values in the second phase.
3) When the collision function is addition, a single full adder can serve as a bit-comparator in the first phase for the keys, and as a reducer for the second phase involving values. However, although the data flow is from MSB to LSB for a comparison (as described above), for addition, the data flow is from LSB to MSB because of carry.
4) In this paper, recall that the record size is 128-bits (64-bit key, 64-bit value), such that $K = 128$. Furthermore, commodity HBM has 128-bit wide channels, and the time taken to read such a quantum of data is in the order of a few nanoseconds, which is higher than several full-adder operations' worth of delay.

5.2. High level algorithm

A highly energy-efficient, bit-serial design based on the above insights can be formed, and is described in this section.

However, this requires tweaking the representation of records in memory a little further. Since the bit-level sorting network expects the MSBs first, the first K bits of an HBM row would be the MSB bits of each of the keys of the K records. As each key is 64-bit, the K keys are strided

TABLE 3. BIT SERIAL NETWORK OPERATION EXAMPLE, *K*=4

Mem				Acc			
[1]=2	[2]=2	[3]=1	[4]=2	[5]=3	[4]=2	[2]=1	[1]=1
Sort							
[1]=2	[1]=1	[2]=2	[2]=1	[3]=1	[4]=2	[4]=2	[5]=3
Delete flags							
0	1	0	1	0	0	1	0
Prefix sum							
0	1	1	2	2	2	3	3
Record index – Prefix sum							
0*	0	1*	1	2*	3*	3	4*
2*K* – Prefix sum							
8	7*	7	6*	6	6	5*	5

across 64 such "chunks", from MSB to LSB. Since additions require data in LSB-first format, the values in that HBM row are stored in chunks in a manner similar to keys, but from LSB to MSB instead.

Before the architecture is described, a high level functional example is first presented.

Table 3 summarizes the mathematical behavior of the sorting network. The first row of the table repeats the same example from section 2, followed by the sorted and merged records into a single list. The next row comprises "delete flags", which are 1 for records that must be deleted (locations 1, 3 and 6 in this example) to yield a list without duplicates.

To squeeze out the records that need to be deleted, parallel prefix operations transform the delete flags into addresses in the range $0...2K-1$, with the address on each record identifying its appropriate position in the output. Prefix sum is a specific form of parallel prefix where an input list x_i is transformed to an output list y_i such that $y_i = \sum_{j=0}^{i} x_j$, or, x plus the running sum of all the elements earlier in the input list. We now subtract the prefix sum from its position in the list, labeled as "record index–prefix sum". For real records, or records with the delete flag in the 0 state, this subtraction yields the desired position in the output list. These values have been labeled with an asterisk and form the list 0* 1* 2* 3* 4*, which is the desired packed sequence.

5.3. Bit serial network

Figure 7 shows serial data formats on the left and the hardware structure on the right. The hardware comprises a continuously operating network with features of both bitonic sorting and parallel prefix networks. Each record comprises a bit stream, 128 bits for current HBM technology, with $2K$ of these records flowing upward through the logic structure in Figure 7b. While the records start at reference point (0) and end at reference point (5) in the same format, Figure 7a shows how the processing inserts additional fields, rearranges fields, and eventually puts the result back into the original form.

At reference point (0), Figure 7, the keys are read from both Mem and Acc. For a single chunk, the K pairs of bits are compared through a bitonic network of 8 ($= \log_2 2K$) levels, where each row contains 128 full adders. At each level, an additional K length bit-vector is used to indicate if swapping is required or not. Once all the keys have been streamed-in in their entirety, the bit-vector contains information regarding which keys are duplicate and which among them are the "real" ones. As a result, when the value bits are subsequently streamed in LSB first, they can consult this bit-vector and accordingly, swap, addition or noop can be performed as they pass through this same bitonic network of 8 levels.

Note that this bit-vector is now analogous in functionality to the "delete flags" from Table 3, and is denoted as D in Figure 7. Recall that these flags also eventually flow through the 8-level network. Therefore, the bit-serial data flow is augmented with a gap before the value bits are streamed in (see reference points (1) and (2)). As these flags go on to contain the address in the final sorted list, these require $\log_2 2K$ bits per record. Again, because these flags will later influence the reordering of records (keys and values), reference point (3) performs temporal reorganization to ensure that they appear at the head of the stream.

Parallel prefix on $2K$ elements can now be performed with a $\log_2 2K$ stage network. Over the course of the calculation, the D flags get transformed into the address in the final sorted list in reference point (4). The parallel prefix calculation involves two full adders for each bit but just one wire – i. e. the same wiring as a bitonic merger but different logic in the comparison module. A final $\log_2 2K$ stage bitonic network yields the desired output.

The reader should now be able to reason about the pipelined, bit-serial sorting network that functions at a granularity of 128 bits, with the following additional information:

- Prefix sum can be used to compute addresses for deleted records by counting down from $2K-1$, as shown in the final line in Table 3. Numbering real records from 0 upwards and deleted records from $2K-1$ downward will obviously use each of the $2K-1$ addresses exactly once.
- The transformation required to sort records when numbered as described is the reverse of a bitonic sort, i. e. bitonic on the right and sorted on the left.
- The required network is actually a reversed bitonic merge. Recall that a bitonic merge has the long wires on the left; the needed reverse bitonic merge has the short wires on the left. However, the logic in each comparison module is the same as a bitonic merge.

6. Experimental Methodology

The architectures proposed in this paper are evaluated on a cycle accurate simulator that models latency and power based on published literature, as follows. CACTI is used for buffers and caches, with 22nm technology transistors operating at 0.9V [10]. The comparator models are derived from a recent work that utilizes 22nm [11]. For the interconnect, the energy and latency for dedicated links is negligible [9]. However, this paper uses a mesh topology instead, as explained in Section 4.3, for flexibility, and the

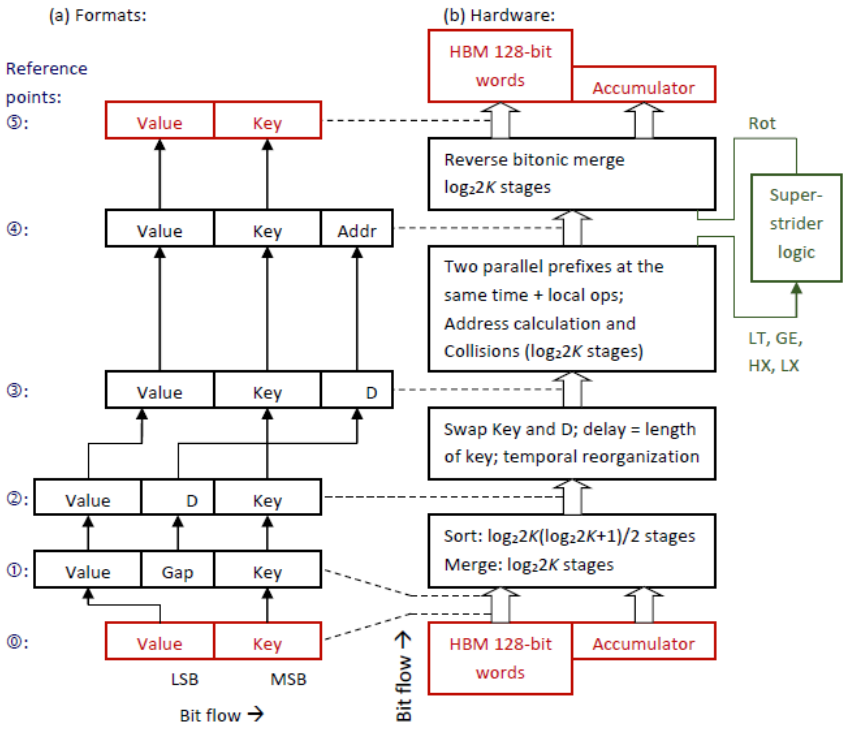

Figure 7. Bit Sorting network approach

models are derived from [12]. Finally, the HBM model is based on the JEDEC standard [13].

Since the bit sorting network described in section 5 is the more energy efficient approach according to the above methodology, we implemented an RTL model for Superstrider using this merge logic approach on Intel/Altera Quartus II to evaluate the performance and design complexity, and also to test for functional correctness. The gate count and area of the merge logic were computed using the Adaptable Logic Module Approach described in [14]. Further reductions from the total utilization to get energy and latency for just the merge logic can be made by removing the area taken by the "AHMES" microprocessor that acts like the test bench driver. Note that we've assumed that the AHMES front end accounts for about 800 ALMs, and that each ALM is equivalent to 6 NAND gates.

7. Results

7.1. Word-parallel architectures

Single comparator. Figure 8 shows the latency and energy consumed for each of the three word-parallel single comparator designs. As discussed in section 3, when compared to the crossbar based baseline, the rotating buffer based design, although consumes lower power, has limited energy savings because of its increased latency of operation. The indexing buffer based design overcomes the limitations of the other designs and significantly improves over them in terms of both energy and latency. As such, for word-parallel architectures, the cache based design is chosen.

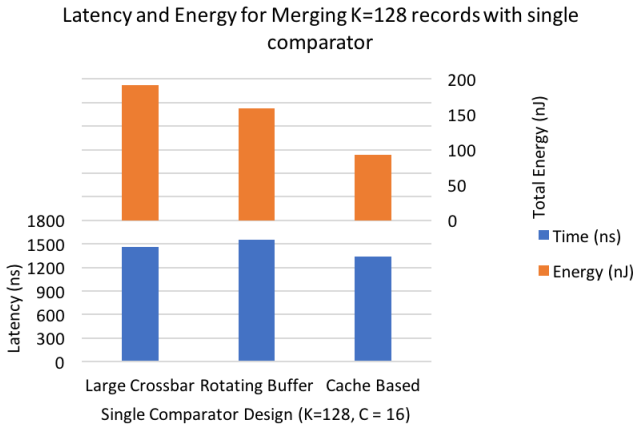

Figure 8. Single Comparator Latency and Energy

Multiple comparators. Figure 9 shows a comparison between a K comparator architecture using the large crossbar, the C comparator cache based version, and the pipelined design described in section 4 for merging $K = 128$ records with caches that can hold $C = 16$ records per cache line where applicable. As expected, the latency and energy required for reading K records from the buffer with large crossbars is significantly higher than those for reading records from the indexable SRAMs. The pipelined design leverages the parallelism inherent in bitonic merge and is able to fully utilize cache lines for all the intra cache line

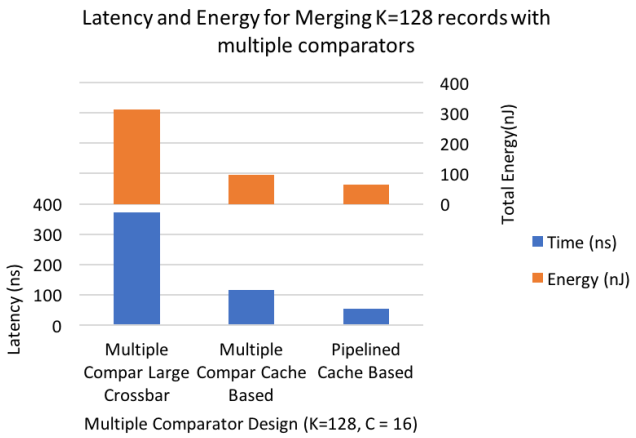

Figure 9. Multiple Comparator Performance

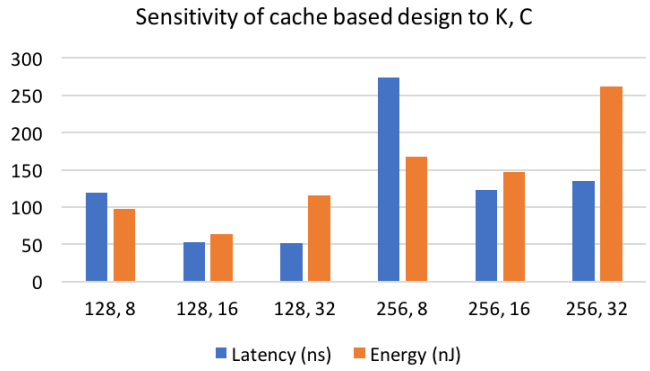

Figure 10. Sensitivity to record size, input size and cache line size for word-parallel architectures.

strides, thereby significantly reducing overheads incurred in reading and writing back records to the buffers for various intermittent levels of bitonic merge. When compared with the single-comparator baseline, the pipelined cache based design is $28\times$ faster and uses $4\times$ less energy. When compared with the multiple-comparator baseline, it is $6\times$ faster and uses $5\times$ less energy.

Sensitivity to K and C. The values of K and C are affected by the cache line sizes, record sizes and HBM row sizes, and changing any one of them can impact the latency and energy utilization of the merge operation. In order to gain insights into this change, a sensitivity analysis is presented in Figure 10. As expected, increasing the value of K with a constant C causes the energy and latency to grow by a factor of $\mathcal{O}(n \log_2 n)$. On the other hand, holding K constant and increasing C results in a reduction in latency and energy due to the added parallelism from using more compare and compress units. The latter trend is not perfectly linear since an additional overhead is incurred when reading from and writing to the bigger cache line. Empirically, our results indicated that for merging records up to $K = 256$, $C = 16$ provided the best energy-time performance, since higher values of C, such as $C = 32$ result in more expensive (both time and energy) intermediate buffer reads and writes.

Further improvements. The energy consumption for the multiple comparator pipelined merge network can be broken down into the caches, the compare and compress units, and the network links. The caches and buffers used 86% of the overall energy consumed, while the network interconnects use about 12% of the total energy, and the compare and compress units only consume 2% of the total energy. This motivates the need for a smarter organization of records within the HBM rows so that the size and number of buffers used can be reduced.

One potential solution is to populate all the keys in the first half of the HBM row followed by values in the second half. This can reduce the required buffer sizes by half since, first, the keys can be streamed in and sorted, and then collisions/swaps can be detected and recorded. This can be followed by the read of the values for rearrangement and compression. The tradeoff is the potentially additional work the front end (processor) may have to do in order to ensure that the accumulator is arranged in the manner described here, as well as additional overhead in lookup of a record. We have not evaluated the performance of this optimization. However, the sorting network, whose results are presented next, relies on such rearrangement of data to further reduce energy consumption.

Another approach is described in [15]; the idea is to skip ahead and avoid comparing certain pairs when performing the comparisons by using a clever layout of records. However this complicates the control logic significantly, and evaluation for this has been left for future work.

7.2. Sorting network approach

The bit-serial sorting network was evaluated for various configurations of K to demonstrate its scalability. The gate count results from RTL have been summarized in table 4. The largest configuration size for which RTL results are presented, is limited by the capability of the demo FPGA board that was available to us.

TABLE 4. BASIC SUPERSTRIDER OPERATION.

Network width $2K$				
2^3	2^4	2^5	2^6	2^7
ALM count, including AHMES				
884	1185	1900	3784	7644

Table 5 is a summary of the resources required for the highly optimized bit-serial sorting network, and attempts to demonstrate the scalability of this merge architecture with expected increasing memory row sizes [1].

7.3. Comparison of word-parallel vs bit-serial approaches

The model described in section 6 was applied to the sorting network approach as well, and the total energy for

TABLE 5. SORTING NETWORK RESOURCES

Attribute	HBM	General
Key size	$L_{\text{key}} = 64$	L_{key}
Value size	$L_{\text{value}} = 64$	L_{value}
Record size	$L_{\text{record}} = 128$	$L_{\text{record}} = L_{\text{key}} + L_{\text{value}}$
Width	$K = 128$	$K = 2^n$
Network stages	24	$3 \log_2 2K$
First bit emerges	clock cycle 96	$4 \log_2 2K + L_{\text{key}}$
Last bit clear	128 cycles after first bit	L_{record} cycles after first bit
Gate count	92,160 gates per channel	$2K$ x $3 \log_2 2K$ cells
Energy per row	24.6 nJ per channel	

merging $K = 128$ records was found to be about 24.6nJ. For a frame of reference, note that this is similar to the energy consumed in reading an entire HBM row (including row activation) [13], and is about $2.5\times$ lower than the energy consumed by the word-parallel architecture. As expected, the energy efficiency of the sorting network approach stems from using single bit comparator units and its reduced usage of SRAM structures.

In terms of latency, both, the word-parallel and sorting network approaches operate such that they are completely interleaved/overlapped with HBM reads and writes, as discussed in Sections 4 and 5.

Finally, recall that, in order to achieve superior energy efficiency, the sorting network requires data to be stored and input in a re-arranged fashion. Furthermore, a value lookup requires accessing several non-contiguous columns of an HBM row, which, for commodity devices, would require a read of redundant columns.

7.4. Comparison with other hardware sorting networks

When compared with other state-of-the-art hardware merge sorter [16], for merging two sets of 128 records, using $C = 16$ for our word-parallel pipelined approach, and $E = 8$ for the Mashimo, et al approach, we find that our approach presented in this work achieves a speedup of about $5\times$ at an energy cost of about $1.2\times$. The primary reason for the speedup is the highly parallel nature of bitonic merge in our design. However the use of various intermediate SRAM buffers slightly increases the energy utilization.

7.5. "Full" logic memory integration

Both the word-parallel and serial implementations can process an HBM row in well under 100ns with under 0.1uJ energy consumption. Speed is not a critical issue because this is slightly faster than the memory can read and write the data. Dividing these numbers yields about 1W power dissipation, or about 8W for all 8 independent channels in an HBM module. The original HBM I/O drivers consume about 4W, but HBM2 and other variants consume up to 20W [2]. With the exception of the original version of HBM, Superstrider consumes less energy than the I/O drivers that could be disabled during operation.

However, there is room for a lot of additional work. A DRAM chip during refresh reads rows out of the transistor array and writes them back with no change for about 46 fJ per bit [17] or 0.75 nJ per 16Kb HBM row. What this means is that if the energy efficiency of the implementations in this work could be further reduced by around 50x, they could be added directly to all the DRAM bitline buffers without significantly increasing chip dissipation. The semiconductor roadmap shows reduction in gate energy from around 5 fJ to 0.81 fJ by 2033, which is a factor of 7 [1]. Further tuning of physical place and route, and perhaps improved memory technology could help achieve a degree of logic-memory integration that is truly scalable.

8. Related Work

Sorting and merging find applications in a wide variety of problems [7]. They have been extensively studied with various sequential and parallel solutions being formulated over the years, and consequently hardware merge networks have been explored both as standalone accelerators as well as fundamental components of larger off-chip accelerators in both academic and industry research papers [4], [18]. Some of the past work has ranged from using systolic arrays [8] to more recently using FPGAs for accelerating large scale sorting, such as data center workloads [16], [19], [20]. However this is the first work to our knowledge that tackles the problem at a finer (HBM row) granularity with a closely integrated logic-memory approach maintaining energy minimization as one of its primary goals. It should be noted that in order to directly compete with the FPGA sorting solutions presented above, we will need to extend our approach to a hierarchical, potentially tree based multi-merger design. However, that work is beyond the scope of this paper.

Moreover, the work presented in this paper can be easily extended to in-network collectives such as the Mellanox [21] routers and their Scalable Hierarchical Aggregation Protocol (SHArP) [22] where reduction operations from various MPI processes are performed within the network layer itself. The word-parallel architecture presented in this work can be integrated within the routers to provide an energy efficient associative-operation reducer.

9. Conclusion

Recent advancements in 2.5D/3D technologies as well as an increased prevalence of applications that have low memory access locality with very little computation per memory access have given rise to a renewed interest in logic-memory integration. Superstrider is a recently proposed accelerator that leverages this paradigm to help tackle what is known as the von Neumann bottleneck. Merge networks are at the heart of Superstrider as well as of traditional sorting networks. This paper presents the design and analysis of scalable and flexible architectures for such networks and improves upon naive approaches by an order of magnitude in terms of performance and energy consumption.

10. Acknowledgements

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U. S. Department of Energy or the United States Government. Approved for unlimited unclassified release, Sandia National Laboratories SAND2018-9939 C.

The authors would like to thank Dr. Thomas M Conte, and the anonymous reviewers of ICRC for providing valuable feedback.

References

[1] "International roadmap for devices and systems – 2017 executive summary," https://irds.ieee.org/images/files/pdf/2017/2017IRDS_ES.pdf, 2017.

[2] J. Hruska, "Advantages of hbm over gddr5," https://www.extremetech.com/extreme/226240-sk-hynix-highlights-the-huge-size-advantage-of-hbm-over-gddr5-memory, 2016.

[3] J. Zhao, Q. Zou, and Y. Xie, "Overview of 3-d architecture design opportunities and techniques," *IEEE Design Test*, vol. 34, no. 4, pp. 60–68, Aug 2017.

[4] S. Srikanth, T. M. Conte, E. P. DeBenedictis, and J. Cook, "The superstrider architecture: Integrating logic and memory towards non-von neumann computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, Nov 2017, pp. 1–8.

[5] E. P. DeBenedictis, J. Cook, S. Srikanth, and T. M. Conte, "Superstrider associative array architecture: Approved for unlimited unclassified release: Sand2017-7089 c," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, pp. 1–7.

[6] J. Kepner and J. Gilbert, *Graph Algorithms in the Language of Linear Algebra*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2011.

[7] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 307–314. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468121

[8] U. Schwiegelshohn, "A shortperiodic two-dimensional systolic sorting algorithm," in *Systolic Arrays, 1988., Proceedings of the International Conference on*. IEEE, 1988, pp. 257–264.

[9] C. H. O. Chen, S. Park, T. Krishna, S. Subramanian, A. P. Chandrakasan, and L. S. Peh, "Smart: A single-cycle reconfigurable noc for soc applications," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 338–343.

[10] S. J. E. Wilton and N. P. Jouppi, "Cacti: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[11] J. V. M. R. D. V. Manikandan, V.P.Muralikrishna, "Static carry skip adder designed using 22-nm strained silicon cmos technology operating under wide range of temperatures," in *International Journal of Engineering and Technical Research (IJETR) ISSN: 2321-0869*, vol. 8, no. 2, Feb 2018.

[12] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jiménez, "Reducing network-on-chip energy consumption through spatial locality speculation," in *Proceedings of the Fifth ACM/IEEE International Symposium*, May 2011, pp. 233–240.

[13] "High bandwidth memory (hbm) dram," https://www.jedec.org/standards-documents/results/HBM.

[14] "Altera fpga white paper," https://www.altera.com/en_US/pdfs/literature/wp/wp-01003.pdf, 2006.

[15] M. F. Ionescu and K. E. Schauser, "Optimizing parallel bitonic sort," in *Proceedings 11th International Parallel Processing Symposium*, Apr 1997, pp. 303–309.

[16] S. Mashimo, T. V. Chu, and K. Kise, "High-performance hardware merge sorter," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 1–8.

[17] E. P. DeBenedictis, "3d software: A new research imperative," *Computer*, vol. 50, no. 10, pp. 74–77, 2017.

[18] S. Jun, S. Xu, and Arvind, "Terabyte sort on fpga-accelerated flash storage," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 17–24.

[19] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel hardware merge sorter," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 95–102.

[20] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on fpgas," *The VLDB Journal*, vol. 21, no. 1, pp. 1–23, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00778-011-0232-z

[21] "Mellanox in network collective switch," http://www.mellanox.com/related-docs/prod_silicon/PB_SwitchIB2_EDR_Switch_Silicon.pdf.

[22] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. Zahavi, "Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction," in *Proceedings of the First Workshop on Optimization of Communication in HPC*, ser. COM-HPC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–10. [Online]. Available: https://doi.org/10.1109/COM-HPC.2016.6