HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing

Mingyu Gao[†] and Christos Kozyrakis[†‡]
Stanford University[†] EPFL[‡]
{mgao12,kozyraki}@stanford.edu

ABSTRACT

The energy constraints due to the end of Dennard scaling, the popularity of in-memory analytics, and the advances in 3D integration technology have led to renewed interest in near-data processing (NDP) architectures that move processing closer to main memory. Due to the limited power and area budgets of the logic layer, the NDP compute units should be area and energy efficient while providing sufficient compute capability to match the high bandwidth of vertical memory channels. They should also be flexible to accommodate a wide range of applications. Towards this goal, NDP units based on fine-grained (FPGA) and coarse-grained (CGRA) reconfigurable logic have been proposed as a compromise between the efficiency of custom engines and the flexibility of programmable cores. Unfortunately, FPGAs incur significant area overheads for bit-level reconfiguration, while CGRAs consume significant power in the interconnect and are inefficient for irregular data layouts and control flows.

This paper presents *Heterogeneous Reconfigurable Logic* (HRL), a reconfigurable array for NDP systems that improves on both FPGA and CGRA arrays. HRL combines both coarse-grained and fine-grained logic blocks, separates routing networks for data and control signals, and uses specialized units to effectively support branch operations and irregular data layouts in analytics workloads. HRL has the power efficiency of FPGA and the area efficiency of CGRA. It improves performance per Watt by 2.2x over FPGA and 1.7x over CGRA. For NDP systems running MapReduce, graph processing, and deep neural networks, HRL achieves 92% of the peak performance of an NDP system based on custom accelerators for each application.

1. INTRODUCTION

The end of Dennard scaling has made all computing systems energy-bound [14]. This is particularly the case for the emerging class of in-memory analytics workloads, such as MapReduce and Spark, graph processing, and deep neural networks [8], where the lack of temporal locality leads to most energy spent on moving data to and from the memory system, dwarfing the energy cost of computation [24].

The obvious way to save energy for workloads with limited temporal locality is to avoid data movement altogether by executing computation closer to the data. Past efforts for processing-in-memory (PIM) ran into the high overheads of integrating processing units and main memory in a single chip [13, 23, 32, 33, 39, 44, 45]. The recent advances in the 3D integration technology [9, 42] provide a practical implementation approach, where compute units are introduced in the logic layer of a TSV-based memory stack. This has created renewed interest in near-data processing (NDP) [3], and several research efforts have recently demonstrated high potential in performance and energy improvements [1, 15, 16, 22, 46, 59].

A key decision for NDP systems is the type of processing elements used in the logic layer. Past PIM and NDP efforts have used a range of approaches, including general programmable cores with SIMD or multi-threading support [13, 16, 45, 46], throughput-oriented GPUs [59], reconfigurable arrays [15, 44], and custom accelerators (ASICs) [22, 60]. Each option represents a different tradeoff point between performance, area and power efficiency, and flexibility. Programmable cores and GPUs have the highest flexibility, but their area and power overheads limit the number of elements per chip, hence underutilizing the high memory bandwidth available in 3D stacks. Custom units are highly efficient but lack the flexibility to support a wide range of workloads. Reconfigurable units based on fine-grained (FPGA) or coarse-grained (CGRA) arrays have sufficient flexibility but introduce area and power overheads respectively. FPGA arrays incur high area overheads for the bit-level configurable blocks and interconnects. CGRA arrays incur high power overheads due to the powerful interconnect for complicated data flow support [21], and are typically inefficient for irregular data and control flow patterns.

This paper presents a reconfigurable logic architecture optimized for near-data processing, called *Heterogeneous Reconfigurable Logic* (HRL). The HRL array uses both coarse-grained functional units with a static bus-based routing fabric for data *and* fine-grained logic blocks with a bit-level network for control, in order to achieve the area efficiency of CGRA and the power efficiency of FPGA. It also features specialized units that support control flow branches, irregular data layouts, and the mismatch between compute pipeline depth and memory access width in analytics workloads. Using HRL arrays as the processing elements, we build near-data processing systems with support for efficient communication and synchronization across multiple stacks and memory channels [16]. Nevertheless, HRL arrays can also be used as efficient and flexible compute units in other scenarios as well.

We evaluate HRL for in-memory analytics frameworks including MapReduce, graph processing, and deep neural net-

978-1-4673-9211-2/16/$31.00

works. We show that HRL achieves the throughput and area efficiency of CGRA and the power efficiency of FPGA. Per unit of area, the HRL array achieves 1.7x higher power efficiency (performance/Watt) than the CGRA and 2.2x higher than the FPGA. For the overall NDP system with practical area and power constraints for the logic layer in the 3D stacks, HRL has a 1.2x and a 2.6x performance advantage over CGRA and FPGA respectively. Compared to NDP systems with custom accelerators for each workload we considered (the efficiency limit), the HRL-based NDP system is the only one that comes within 92% of its performance, while consuming 23% more power. Overall, HRL provides *high performance, high efficiency, and high flexibility* for a wide range of regular and irregular analytics workloads.

2. BACKGROUND AND MOTIVATION

2.1 Related Work on PIM/NDP

Research on computing close to data started in the 1990s with *Processing-in-Memory* (PIM) projects. EXECUBE integrated multiple SIMD/MIMD cores with DRAM on a single chip and scaled to distributed configurations [33]. IRAM combined a vector processor with DRAM to achieve high bandwidth at low power for media tasks [45]. Also based on SIMD, Computational RAM added processing elements to the DRAM sense amplifiers [13]. Smart Memories [39] and DIVA [23] were tiled architectures that combined processor and DRAM arrays. FlexRAM integrated arrays of small SPMD engines interleaved with DRAM blocks and a RISC processor on a single chip [32]. Active Pages used reconfigurable logic for the computation elements to achieve both high flexibility and high efficiency [44].

Close coupling of processing and memory is now practical using 3D stacking with through-silicon vias (TSVs) [9, 42]. 3D stacking is already used to introduce large caches on top of processor chips [30, 36, 37]. For main memory, there are two prominent efforts, Micron's Hybrid Memory Cube (HMC) [27] and JEDEC High Bandwidth Memory (HBM) [29], that provide a logic die at the bottom of multiple stacked DRAM chips. The common use of the logic layer is to implement peripheral circuits, testing logic, interface to processor chips, and/or memory controllers. The logic layer can also accommodate different types of processing units to enable *near-data processing* (NDP) [3].

There are several recent proposals for NDP architectures focusing primarily on in-memory analytics. NDC connected the host processor with multiple daisy chains of HMC stacks augmented with simple, programmable cores [46]. The embarrassingly parallel phases of the application ran on the NDP cores, while the host processor handled the communication. NDA used coarse-grained reconfigurable arrays (CGRAs) [6, 21, 40, 50] as its processing elements on top of commodity DRAM devices with minimal change to the data bus [15]. The host processor was used for data shuffling and accelerator control. Both NDC and NDA showed significant performance and energy benefits. Tesseract was a near-data accelerator for large-scale graph processing [1]. It used simple programmable cores for processing and introduced a low-cost message-passing mechanism with specialized prefetchers to support efficient communication between

```
struct vertex_t {
    const uint32_t in_deg;
    const uint32_t out_deg;
    double rank;
    double sum;
    uint32_t collected;
};
struct edge_t {
    vid_t src;
    vid_t dst;
};
struct update_t {
    vid_t dst;
    double contribution;
};

update_t pagerank_scatter(vertex_t sv, edge_t e) {
    return { e.dst, sv.rank / sv.out_deg };
}

void pagerank_gather(
        vertex_t& dv, update_t u, double beta) {
    dv.sum += u.contribution;
    dv.collected++;
    if (dv.collected == dv.in_deg) {
        dv.rank = beta * dv.sum + (1 - beta);
        dv.sum = 0;
        dv.collected = 0;
    }
}
```

Figure 1: The PageRank scatter and gather kernels.

memory partitions. Practical near-data processing proposed a more general solution for shared-memory NDP systems running various in-memory analytics workloads [16]. It also used general-purpose multi-threaded cores, and presented an efficient coherence and communication model for balanced and scalable NDP systems. TOP-PIM stacked a throughput-oriented GPU to exploit the increased memory bandwidth [59], while other systems have used various custom hardware accelerators [22, 60].

2.2 Requirements for NDP Logic

3D stacking allows NDP systems to eliminate the energy overheads of moving data over long board traces. It also provides an order of magnitude higher bandwidth between the stacked memory banks and the compute units in the logic layer. However, the actual performance depends on how much processing capability one can fit within the area and power constraints of the logic layer. The ideal compute technology for NDP should (1) be *area-efficient* in order to provide sufficient compute throughput to match the high bandwidth available through 3D stacking; (2) be *power-efficient* in order to reduce total energy consumption, and to avoid causing thermal issues in the DRAM stacks; (3) provide *high flexibility* to allow for reuse across multiple applications and application domains.

NDP systems are a natural fit for data-intensive analytics applications with large memory footprint, such as in-memory MapReduce [11, 58], graph processing [17, 18], and deep neural networks [5, 10]. While the computations in these applications are mostly simple arithmetic operations, the data flows, data layouts, and control flows can be quite different, irregular, and non-trivial. As an example, Figure 1 shows the PageRank scatter and gather kernels, following the edge-centric gather-scatter programming

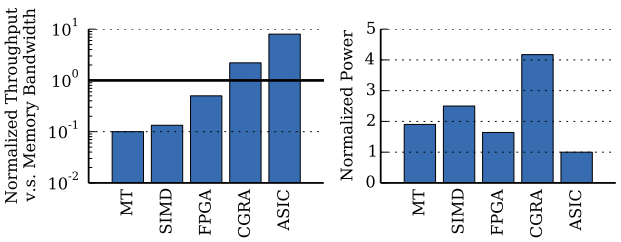


Figure 2: Computational throughput and power consumption for different logic types implementing the PageRank scatter kernel: multi-threaded (MT) cores, SIMD cores, FPGAs, CGRAs, and custom units (ASICs).

model [48]. First, data flow patterns in different kernels can vary a lot. Some may be as simple as one operation (`pagerank_scatter`), while others can be more complicated and irregular (`pagerank_gather`). Second, wide complex data types (such as `vertex_t` and `update_t`) are commonly used as input/output for the processing kernels. They often include multiple basic data types with various lengths and irregular alignment, and the data layout can be significantly reorganized with little computation in the kernel (see `pagerank_scatter`). Simple IO schemes will not work well with such complicated data layouts and imbalanced compute-to-IO ratios. Third, conditional branches are frequently used to determine the final output values, requiring efficient dynamic control flow support.

2.3 Comparison between Logic Alternatives

The previous NDP projects in Section 2.1 implemented processing elements using simple cores, SIMD units, GPUs, FPGA blocks, CGRA arrays, and custom accelerators. However, none of these approaches meets all the requirements in Section 2.2. To illustrate the issues, Figure 2 compares different logic options used in NDP systems for the PageRank scatter kernel shown in Figure 1. One memory stack has 8 channels and 128 GBps peak bandwidth in total, but, once memory controllers and interfaces are accounted for, only about 50 mm^2 of the logic layer can be used for compute units [28]. To maximize performance, we put as many logic units as the area and power budgets allow in the stack (see Section 6.3 for detailed description). The computational throughput is normalized to the maximum memory bandwidth, indicated by the solid line. Matching this line implies a balanced system with sufficient compute capability to take full advantage of the high bandwidth through TSVs. Once the memory bandwidth is saturated there is no point in scaling the computational throughput any further.

While optimal for flexibility, programmable cores, either multi-threaded or SIMD, fall short of utilizing all available memory bandwidth due to their large area overheads, therefore suffering from low performance. On the opposite extreme, custom accelerators (ASICs) can easily saturate the memory channels but lack programmability. The extra compute capability provides marginal benefits. It would be better to trade the computational throughput for flexibility.

Reconfigurable units provide a good tradeoff between performance and flexibility, but the two common configurable logic types, FPGA and CGRA, have shortcomings as well [15, 54]. FPGA achieves low power consumption through customization, but suffers from low clock speed and high area overheads due to the bit-level configurable blocks and routing fabric, which are too fine grained for arithmetic operations. The DSP blocks in modern FPGAs help, but their number and width do not match emerging big data applications. Hence, FPGA fails to saturate memory bandwidth with high performance under the area constraints.

CGRA is based on a large number of coarse-grained functional units which can significantly reduce the power and area overheads for computation. Due to the limitation of the simple interconnect, traditional CGRAs only target applications with regular computation patterns, such as matrix multiplication or image processing [40, 50]. A recent NDP system [15] used a new type of CGRA [21] with circuit-switched routing network as a powerful interconnect to support more complicated data and control flows, but also introduced significant power consumption due to the power-hungry pipelined registers and credit-based flow-control logic (see Figure 2). Stringent power constraints also limit the number of elements that can be used in the logic layer. Moreover, CGRA still suffers from data layout issues with simple IO alignment scheme, and it is not particularly efficient when special functions are needed, such as sigmoid and tanh functions for neural network processing.

3. HETEROGENEOUS RECONFIGURABLE LOGIC (HRL)

HRL provides an efficient and flexible logic substrate for NDP by combining coarse-grained and fine-grained logic blocks and routing networks. It achieves the best of the two configurable logic options: the area efficiency of the CGRA and the power efficiency and flexible control of the FPGA.

3.1 HRL Array

Figure 3 shows the organization of the HRL array. It consists of a number of heterogeneous components connected with a statically reconfigurable routing fabric. HRL uses CGRA-style, coarse-grained functional units (FUs) for common arithmetic operations. It also uses FPGA-style, fine-grained configurable logic blocks (CLBs) for irregular control logic and special functions. The output multiplexer blocks (OMBs) provide flexible support for branches. The number of each block type in the array should be determined according to the application characteristics, as discussed in Section 6.1. To reduce power overheads, HRL uses static reconfigurable interconnect, which consists of a coarse-grained, bus-based data network [57] and a fine-grained, bit-level control network. Finally, the array is sized to match the shallow compute pipeline and the wide data access width.

3.1.1 HRL Blocks

HRL includes a grid of *functional units* (FUs, Figure 4b), similar to those in CGRA [21]. They are used as a power and area efficient way for coarse-grained arithmetic and logic operations, including addition, subtraction, comparison, or multiplication, which would be expensive to construct with FPGA lookup tables. The comparison output `out` is a sepa-

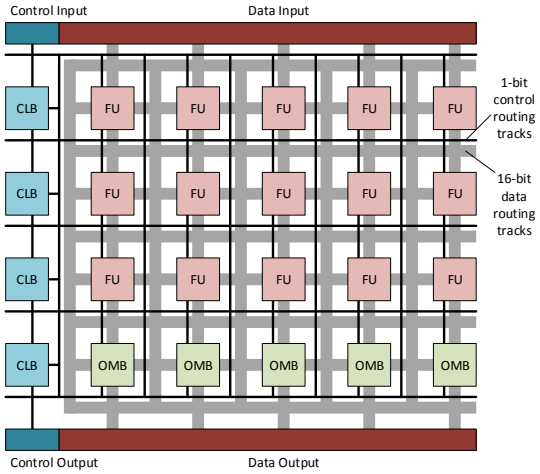
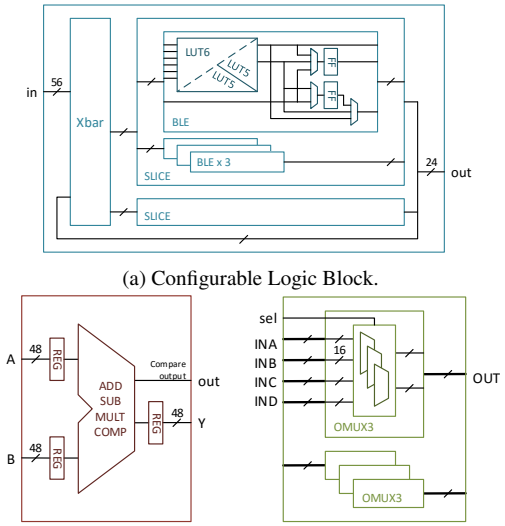


Figure 3: The organization of the HRL array.



(a) Configurable Logic Block.

(b) Functional Unit. (c) Output Multiplexer Block.

Figure 4: Block diagrams of the HRL blocks.

rate single-bit port. There are also data registers at the input and output of each FU to facilitate pipelining and retiming. Compared with the DSP blocks in FPGAs, our FU blocks support much wider arithmetic operations (e.g., 48-bit add or multiply) that directly match the analytics applications.

The *configurable logic blocks* (CLBs) shown in Figure 4a are similar to those in FPGAs [54]. CLBs use lookup tables (LUTs) for arbitrary, bit-level logic. They are used primarily to implement control units which are less regular. Having the control logic directly embedded into the array allows it to run independently, avoiding the overheads of feeding cycle-by-cycle instructions or coordinating with programmable cores as the case in many CGRA designs. CLBs are also useful for special functions, such as the activation functions in neural network processing (sigmoid or hyperbolic tangent). Traditional CGRAs use multi-cycle Taylor series [20]. A better and widely used solution in machine learning accelerators is piece-wise linear approximation [4], but it still requires multiple comparators (FUs). On the other hand, a CLB has sixteen 5-input LUTs to realize a 5-to-16 function for a 32-segment piece-wise linear function with 16-bit coefficients, which has adequate accuracy for neural networks [34]. The full activation function then requires only one CLB and two FUs (the multiplier and the adder).

The *output multiplexer blocks* (OMBs) shown in Figure 4c are placed close to the output to support branches. Each OMB includes multiple multiplexers that can be configured in a tree, cascading, or parallel configuration. The branch condition, usually the output of a comparison in an FU, comes into the select bits `sel` to select the input data between multiple sources (`INA`, `INB`, etc.). Compared with dedicated FUs that support selection or predication operations in CGRA [19], OMBs are cheaper in power and area, and more flexible as they support multiple branches in *if-elseif-else*-like statements through tree or cascading configurations.

It is worth noting that HRL does not have distributed block RAM (BRAM). In-memory analytics workloads have little temporal locality to benefit from large caches. Nevertheless, we provide a centralized external scratchpad buffer for applications that requires explicit data buffering (see Section 4).

3.1.2 HRL Routing Fabric

The HRL routing fabric provides flexible but low-cost interconnect between logic blocks by decoupling the data path and control path. It consists of two networks: the multi-bit *data net* and the single-bit *control net*, both of which are statically configurable similar to the routing fabric in FPGAs. The data net leverages datapath regularity by using bus-based routing to achieve significant area and power savings [57]. We choose a 16-bit bus granularity as a good tradeoff between efficiency and alignment flexibility (see Section 3.1.3). This statically-configured routing fabric is cheaper than the circuit-switched network used in previous CGRA [21], and a small number of 16-bit bus tracks is sufficient for the kernel circuits in memory-bound applications given their simple computation patterns. The control net routes control signals only, which need fewer tracks (20% to 30%) compared to the conventional FPGA fabric. Overall, the two networks need a relatively narrow fabric to support flexible routing at low power and area overheads.

Figure 5 shows the connections between the routing tracks and different blocks. The data input/output ports (`A`, `B`, `Y` of the FU, and `INA`, `INB`, `OUT` of the OMB) are connected to the data net, while the FU comparison output `out` and the OMB `sel` bits are routed to/from the control net. When wide data types are processed at an FU, multiple 16-bit bus tracks can be concatenated and connected to the FU ports. The input/output ports of CLBs are mostly connected to the control net, but some can also be grouped into a bus and routed to the data net when used for a piece-wise linear function.

Note that the data net and control net are separate, meaning there are no configurable switches connecting the two in the fabric. The only interaction between them is through array blocks. This limitation reduces the switch box size in each network to save area and power, while still providing enough flexibility in control flow. For example, if we need to

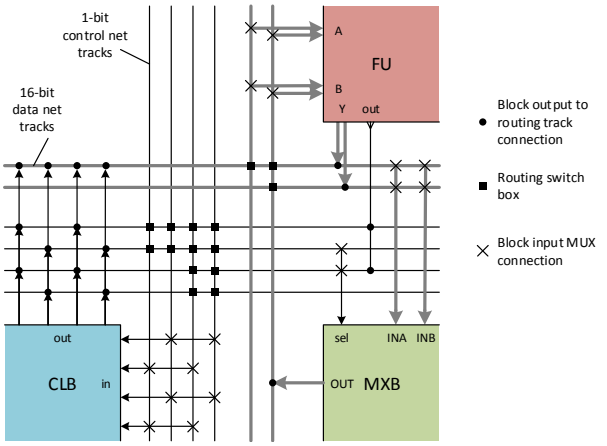


Figure 5: The HRL routing fabric and connections to blocks.

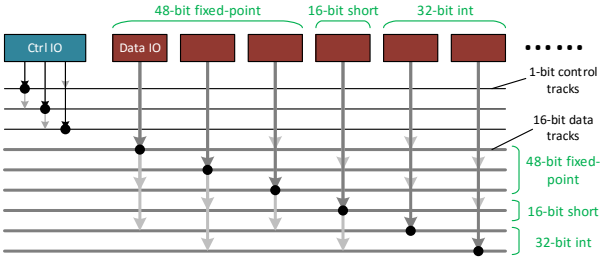


Figure 6: Configurable IO in the HRL array, shown with a data mapping example. Configured connections are shown in solid dots.

evaluate a branch condition containing a comparison `a < b`, the two operands from the data net are connected to the input ports of an FU, and the compare output `out` becomes a control signal on the control net, which can be further routed to the `sel` port of an OMB or an input of a CLB.

3.1.3 HRL Input/Output

To support wide and irregular data layouts with mixed data types, HRL uses the configurable IO blocks shown in Figure 6. The control signals are connected to the control net in the same way as in an FPGA. The data signals are split into 16-bit chunks, each of which is mapped to one of the data IO buffers. These IO buffers can be connected to one or more 16-bit tracks in the data net. Figure 6 shows an example, where one 48-bit fixed-point number, one 16-bit `short`, and one 32-bit `int` are connected to the IO, and further routed to other blocks through the data net.

Another issue is the ratio between the pipeline depth and the data access width in NDP logic. Conventional compute-bound applications typically need many FUs and deep pipelines to map the data-flow graph of a large computation region onto accelerators [20]. In contrast, the memory-bound applications targeted by NDP typically involve fairly simple computations on fairly wide data delivered from DRAM accesses (see Figure 1). Therefore, we size the HRL array to support shallow pipelines (no more than 4 FUs) with wide IO (up to 60 bytes) in order to avoid performance bottlenecks while maintaining high efficiency.

3.1.4 Array Summary

By combining both coarse-grained (CGRA-style) and fine-grained (FPGA-style) components, HRL meets all the requirements for NDP logic as summarized in Section 2. The key aspects that enable both high efficiency and high flexibility in HRL arrays include: (1) coarse-grained functional units provide power- and area-efficient implementation to evaluate computation; (2) embedded bit-level configurable logic blocks avoid control overheads in conventional CGRAs; (3) decoupled data and control networks simplify the routing fabric while maintaining sufficient flexibility for complicated data flows; (4) new output multiplexer blocks support efficient and flexible control flows; and (5) configurable IO blocks ensure flexible alignment and wide data access width.

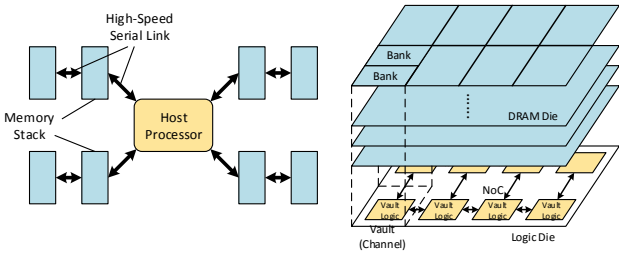
3.2 CAD Toolchain

While an HRL array uses coarse-grained FUs similar to CGRA, all its components, blocks and routing tracks, are statically configured. No runtime instructions are required to drive the logic and there is no need for instruction compiler support, which has been a challenging aspect for CGRA projects. The application mapping for HRL is similar to the FPGA design flow, which includes logic synthesis, packing, placement and routing. Since the control part includes general bit-level logic, it can utilize the same CAD tools. Moreover, current FPGA CAD tools already support coarse-grained DSP block extraction and mapping and it is straightforward to generalize them to support HRL FUs for the datapath. Thus, users can write conventional Verilog code or leverage high-level synthesis environments [2, 56], and a modified FPGA CAD flow will map the design on HRL. For this work, we rely on several modified open-source CAD tools [38, 53] to map Verilog designs onto the HRL arrays (see Section 5).

However, there are still several challenges to address. First, existing FPGAs do not have bus-based routing support, nor do they have the notion of separate control and data networks. Second, the packing and placement algorithms need to balance the routability and critical path latency between the two networks to achieve the overall best performance. Finally, the routing algorithm needs to take the bus structures into account. Recent advances in bus-based routing and coarse-grained FPGAs will help address these issues and further improve the HRL performance [25, 57]. We leave a detailed exploration to future work.

4. NEAR-DATA PROCESSING WITH HRL

We construct a scalable NDP system with multiple memory stacks and HRL-based processing units for analytics workloads. The key challenges are how to efficiently handle communication between HRL arrays within and across stacks and how to accommodate varying applications access patterns. Note that these problems are not specific to HRL. Other types of NDP logic (FPGA, CGRA, ASIC, and cores) must also address these issues. This section describes a simple and general hardware/software communication model that works well for the applications of interest and can be used with multiple types of NDP logic.



(a) System overview. (b) 3D memory stack.

Figure 7: The NDP system architecture.

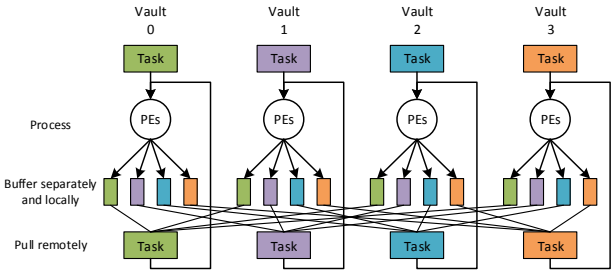


Figure 8: Communication model for the NDP system.

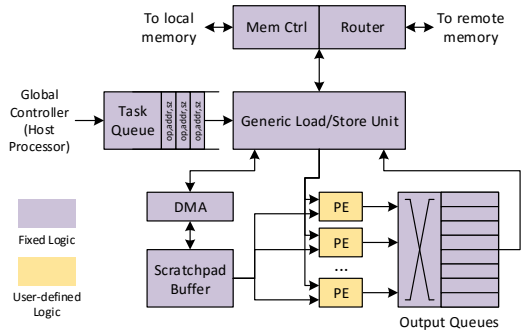


Figure 9: NDP vault hardware organization.

4.1 NDP System Overview

Figure 7a shows the NDP architecture. The host processor is a conventional out-of-order (OoO), multi-core chip, which connects to multiple chains of memory stacks through high-speed serial links. The memory stack integrates multiple DRAM dies on top of a logic die using TSVs, as shown in Figure 7b. The stack organization follows the HMC technology [27]. It presents several independent high-bandwidth, low-power channels (typically 8 to 16) to the logic die, where the memory controllers as well as off-stack SerDes links are implemented. A vertical channel together with its separate memory controller is usually called a *vault*. In this paper, we assume 8 vaults per stack and a 128-bit data bus within each vault. Although slightly different from HMC stack, this organization uses wider data bus that is optimized for streaming accesses presented in most applications (see Section 4.2).

Figure 9 shows that each vault in the memory stacks contains multiple processing elements (PEs)[1] to take advantage of the large bandwidth and high parallelism through 3D stacking. The PEs use physical address to directly access the local or remote vault memory, which is memory-mapped by the host processor to a non-cacheable region. This avoids the virtual memory translation and cache coherence problem between the NDP logic and host processors.

The host processor handles task initialization and coordination for the NDP logic. Before starting the PEs, the host processor partitions the input dataset and allocates the input data for each PE into the corresponding vault using special memory allocation routines. During the execution, the host processor has no access to the data. It only coordinates the communication by collecting and dispatching small task information messages (see Section 4.2). The host processor also handles necessary synchronization between PEs, which is not frequent in the following communication model.

4.2 Data Communication

The analytics domains we target—MapReduce, graph, and deep neural networks—process data in an iterative manner with separate *parallel phase* and *communication phase*. While it is easy to map the parallel processing phase onto any type of NDP logic, the communication phase is challenging. We leverage a previously proposed, *pull* based communication model to implement communication and synchronization while minimizing data copying [16]. Figure 8 summaries the execution flow. The PEs in each vault start with processing their assigned data in parallel. The output or intermediate data are buffered *locally and separately* in the local vault memory. After all input data are processed, the PEs notify the host processor about the completion. The host processor then groups the data assigned for each PE in the next iteration into *tasks*, mainly comprising the data address and size, and dispatches each task to the corresponding PE. Each PE will then process all the tasks assigned to it, by pulling the input data from local or remote vaults.

This generic model allows us to implement communication logic in a manner that is decoupled from the type of logic used for NDP computations. We add a network-on-chip (NoC) to each stack to support communication between vaults and stacks (see Figure 7b) [1, 16]. Figure 9 shows the vault hardware organization. The PEs are one or more units of any type (FPGA, CGRA, custom, or HRL). The other parts are generic hardware for communication. The load/store unit is essentially a DMA unit that accepts tasks from the global controller, and loads input data from local or remote vault memory for the PEs. Normally the input data will be streamed from/to DRAM, but we also support transfers to a 128 kByte scratchpad buffer shared between all PEs in a vault. The scratchpad is large enough to support long processing without frequent data transfers. Sharing the scratchpad is power- and area-efficient, and does not introduce performance bottleneck as the PEs are bounded by the streaming accesses directly to DRAM. Section 4.3 describes how graph applications can utilize the scratchpad. As we need to buffer the output data separately for each consumer, an output queue unit with a fixed number of physical queues is used (e.g., 64). We use longer queues (128 to 256 Bytes)

[1] A processing element can be an FPGA/CGRA/HRL array, a custom unit, or even a core.

to amortize the latency and energy cost to store back to memory. The queues support a few simple, in-place combine operations, such as sum, max and min. Queues can be shared between consumers, and the load/store unit at the consumer will filter out extraneous data while pulling.

4.3 Application Mapping

Table 1 summarizes the mapping methodology for all the three frameworks. Given the communication model, MapReduce [11] workloads can be trivially mapped to our NDP system. User-defined map and reduce phases execute on the configurable PEs by streaming input data from the local vaults, while the shuffle phase is handled by the communication model and the output queues. The in-place combine functions can also be used when combiners are provided in the programs.

For graph workloads, we leverage the streaming, edge-centric X-Stream system [48], which follows the gather-scatter processing pattern [17]. In the scatter phase, the edges are sequentially streamed in to generate the updates for destination vertices. The output queues allow these updates to be buffered separately according to the destination graph tile. Later, each PE will gather (pull) and apply the updates to the vertices in its graph tile. As the PEs need to randomly access the vertex data and to stream the edge/update lists simultaneously, we divide each graph tile into subpartitions that fit in the scratchpad buffer, and pre-load vertex data before processing each subpartition. The update lists are also pre-partitioned into sublists that match the vertex subpartitions. This method has been showed to be efficient as it leads to sequential DRAM accesses [7].

The neural network workloads work similarly to the graph processing, as the neurons can be treated as vertices and the synapses as edges between them. Besides that, we apply two additional optimizations. For fully connected networks, many synapses connect to the same destination neuron in a remote partition. We use local combiners to reduce communication cost. Convolutional and pooling layers use a few shared weights for all synapses, which can be cached in the scratchpad buffers.

4.4 Runtime Reconfiguration

As applications involve multiple parallel phases that run iteratively, the reconfigurable PE arrays need to support fast reconfiguration between phases. However, this is not a critical issue for our NDP systems. The configuration streams can be stored in local vault memory, which can be accessed quickly. Since all the arrays will implement the same logic, a single copy of configuration stream can be shared within each vault or stack, and the configuration can happen in parallel. Furthermore, the data communication between PEs and task assembling and dispatching at the host processor can take place in parallel with the reconfiguration.

The configuration cost for HRL is low. HRL has similar functional unit arrays as DySER [20], which requires 64 cycles to configure an 8×8 array. With the addition of the routing configuration, it takes less than 100 cycles to configure an HRL array, which is far less than the execution period of each phase, typically in the millions of cycles.

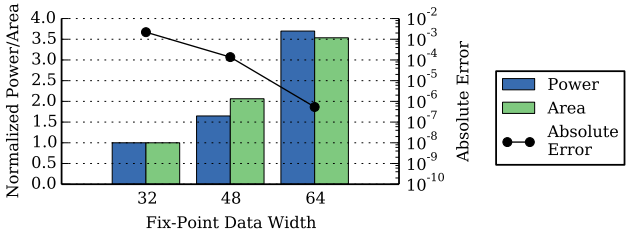


Figure 10: Cost and accuracy comparison between functional units with different data width. Errors are calculated against 64-bit floating-point numbers.

5. METHODOLOGY

5.1 Applications

We use a set of representative analytics workloads, namely GroupBy, Histogram (Hist), Linear Regression (LinReg) from in-memory MapReduce [11, 51], PageRank, Single Source Shortest Path (SSSP), Connected Component (CC) from graph processing [17, 18, 48], and Convolutional Neural Network (ConvNet), Multi-Layer Perceptron (MLP), Denoising Auto-Encoder (dA) from deep neural network processing [5, 10]. These workloads are known to be memory-intensive and can benefit from near-data processing [1, 16, 46]. We follow the methodology in Section 4.3 to divide these applications into coarse-grained phases called *kernels*, which are implemented in *kernel circuits* (KCs) and executed on the memory-side PEs. The communication between PEs and the host processor is handled the same way for all logic options as explained in Section 4.2.

The applications and their kernel circuits are summarized in Table 1. GroupBy involves no computation and utilizes the communication model to cluster the data with same keys. Hist and LinReg have simple summation reducers and use the in-place combine functionality of the output queues (no KCs). Each graph application has two KCs for scatter and gather. The gather KCs have branches to update the vertex data if all updates are collected (see Figure 1). Neural network applications share similar KCs but differ in the network connection structures. We evaluate two parameter propagation schemes: embedded (i.e., no-sharing) and shared [4]. The `neuron_update` KC uses piece-wise linear approximation to evaluate the sigmoid activation function.

For efficiency, we use 48-bit fixed-point arithmetics instead of floating-point operations. Figure 10 shows the comparison between different fixed-point data widths. 48-bit FU achieves sufficient accuracy ($< 10^{-3}$) while keeping the power and area costs low. The use of fixed-point operations biases results in favor of FPGA-based logic. For CGRA and HRL, it is straightforward to introduce wider FUs or floating-point units in the arrays if necessary.

5.2 CAD Flow

The applications are written in shared-memory, multi-threaded C/C++ code. We extract and convert the kernels into Verilog code using Vivado HLS [56]. While the complete programs contain complicated control and communication models, the kernels are fairly simple and the generated

Framework	MapReduce [11, 51]	Graph [17, 48]	Deep Neural Networks [5, 10]
Communication	Shuffle phase	Vertex messages to other graph tiles	Synapses to other network partitions
Stream Data	All input data	Edge list, message list	Synapses
Scratchpad Data	–	Vertex Data	Neurons
Example Applications	GroupBy, Histogram, Linear Regression	PageRank, Single Source Shortest Path, Connected Component	Convolutional Neural Network, Multi-Layer Perceptron, Denoising Auto-Encoder
Kernel Circuits	KC1:`hist_map`, KC2:`lr_map`	KC3:`pr_scat`, KC4:`pr_gath`, KC5:`sssp_scat`, KC6:`sssp_gath`, KC7:`cc_scat`, KC8:`cc_gath`	KC9:`shared_param_prop`, KC10:`embedded_param_prop`, KC11:`neuron_update`

Table 1: Framework applications.

Verilog code is nearly optimal.

We use VTR 7.0 to synthesize, pack, place, and route the kernel circuits on FPGA [38], as commercial tools do not allow necessary tuning and resizing for our FPGA array (see Section 6.2). For HRL, we use Yosys for coarse-grained block extraction, synthesis, and mapping [53]. Yosys also separates the control network apart from the data network. The routing for the two networks is done with VTR. We follow the 2-phase routing method used in [25] for bus-based routing, i.e., presenting the buses as 1-bit tracks to VTR and applying the same routine to other bits after routing. The power and area are scaled based on [57].

For traditional CGRA, we use a DySER-like array [20,21] that has been efficiently used in previous NDP studies [15]. Although DySER is designed to work within a general-purpose processor and relies heavily on the processor's control and load/store unit, in our evaluation it is controlled by the generic vault structures discussed in Section 4.2 and no additional control is necessary. We apply an optimistic estimation on how many resources each KC occupies in DySER. We first count the minimum number of FUs needed for the computation. Based on the number of FUs and input/output ports, we estimate the minimum number of switches when these blocks are mapped to the smallest bounding box. This method estimates the best possible mapping results, as in practice this minimum mapping may be unroutable, or need time-sharing the switches which increases the latency. Thus, we are underestimating the potential benefits of HRL over CGRA.

5.3 System Models

We use zsim [49], a fast and accurate simulator supporting up to thousands of threads, to simulate the performance of the NDP systems with different PE types. Memory-side programmable cores use single-issue, in-order, multi-threaded cores at 1 GHz [16, 46]. Other PEs are modeled as streaming processors, with the throughput and latency numbers obtained from post-synthesis results for each logic type as described below. We extend zsim with a detailed DDRx memory model, which has been validated with DRAMSim2 [47] and against a real system. The timing parameters of the 3D memory stacks are conservatively inferred from publicly-available information [27, 52]. We also model the on-chip and inter-stack interconnect in zsim. The key system parameters are summarized in Table 2.

Power and area: We assume 45 nm technology process for any logic in the logic layer of the stack, including cores, FPGA, CGRA, HRL, custom units, vault controllers and interface circuits. The area budget for the logic die in the 3D stack is set to 70 mm^2 [28]. These technology and area numbers are close to the HMC prototype device. The peak power consumption of the stack is assumed to be 8 W based on an NDP system thermal study [12]. Excluding the memory components and SerDes interfaces leaves about 5 W for the PEs. This budget is moderate compared to previous work [16, 46].

Host Processor	
Cores	8 x86-64 OoO cores, 2 GHz
L1I cache	32 kB, 4-way, 3-cycle latency
L1D cache	32 kB, 8-way, 4-cycle latency
L2 cache	256 kB private, 8-way, 12-cycle latency
L3 cache	20 MB shared, 20-way, 28-cycle latency
Cacheline	64 Bytes
3D Memory Stack	
Organization	16 GB, 8 layers $\times$ 8 vaults $\times$ 8 stacks
Bus Width	128 bits
Timing Parameters	$t_{CK} = 2.0$ ns, $t_{RAS} = 28.0$ ns, $t_{RCD} = 14.0$ ns, $t_{CAS} = 7.0$ ns, $t_{WR} = 9.0$ ns, $t_{RP} = 14.0$ ns
Serial links	160 GBps bidirectional, 8-cycle latency
Vault Hardware	
Scratchpad	128 kB, 64-Byte block
Output Queues	64 queues, 128-Byte each
On-chip links	16 Bytes/cycle, 4-cycle zero-load delay

Table 2: The key parameters of the NDP system.

We model the FPGA arrays based on the Xilinx Virtex-6 device [26, 54]. The power numbers are extracted from Xilinx Power Estimator (XPE) [55], and are consistent with VTR's internal power model. The blocks used in CGRA and HRL, including the FUs, the CGRA switches, and the HRL OMBs, are synthesized using Synopsys Design Compiler 2014 on TSMC 45 nm technology. We optimize them for a 3 ns clock period, which provides reasonable performance without significantly increasing power and area. Higher frequencies lead to limited benefits as shown in Section 6.3. These area and power numbers are then used in VTR to estimate the total area and power. To assure model accuracy, we also compare the synthesized numbers for CGRA with DySER results [19, 20] scaled by size and technology, and they match very well.

The memory power is calculated using Micron DRAM power calculator [41]. We model the 3D memory by scaling the 2D DRAM static and dynamic power with different bank organization and the replicated peripheral circuits for each vault. Overall, our 3D memory power model results in roughly 10 to 20 pJ/bit, which is close to but more conserva-

KC	FU (w/ MULT)	OMB	CLB
1	8 (2)	0	1
2	9 (3)	5	1
3	7 (2)	0	1
4	15 (3)	7	2
5	8 (2)	0	1
6	9 (2)	6	1
7	5 (1)	0	1
8	6 (2)	4	1
9	4 (2)	0	1
10	8 (2)	0	1
11	18 (3)	6	1
max	18 (3)	7	2

Table 3: Block mapping results for all kernel circuits.

tive than the numbers reported in 3D memory literature [28].

For the other components in the system, the general programmable cores are modeled with McPAT 1.0 [35]; the scratchpad buffers and output queues are modeled using CACTI 6.5 [43]; the routers and interconnect are modeled using Orion 2.0 [31]; and the DMA controllers and load/store units are modeled using Synopsys IP blocks.

6. EVALUATION

This section evaluates the HRL array and NDP systems based on HRL. We first size the key parameters of HRL based on the workload characteristics in Section 6.1. Then, we compare the area and power of a single HRL array with alternative reconfigurable logic options with equivalent logic capacity in Section 6.2. In Section 6.3, we demonstrate that HRL provides better performance and power efficiency as an NDP logic option than the traditional reconfigurable logic. Finally, in Section 6.4, we present the full system comparison with all logic options—including general programming cores, traditional FPGA and CGRA, HRL, and custom units—for a practical NDP system with multiple PEs per vault, multiple vaults per stack, and multiple stacks and chains.

6.1 HRL Design Exploration

HRL is a heterogeneous configurable array with multiple types of blocks. Table 3 shows the numbers of different blocks used in the target kernel circuits. The numbers of FUs with multipliers are showed in parenthesis. As we can see, across all KCs the minimum array configuration is 18 FUs, 7 OMBs, and 2 CLBs. To provide some additional placement flexibility, we use a 6×6 HRL array with 20 FUs, 10 OMBs, and 6 CLBs with a layout similar to Figure 3.

Since the HRL FUs are more efficient than the LUT logic in FPGAs, HRL generally has shorter critical path delays. As the HRL critical path mainly consists of a 3 ns FU delay and about 1 ns routing delay, we target 200 MHz clock frequency for all KCs, by adding a few more routing tracks in addition to the minimum number for better routing results.

The multiplier is an expensive component in the functional units. As not all FUs need multiply operation, having both FUs with and without multipliers is power- and area-efficient. However, routing will become more difficult as it limits the blocks to which the multipliers can be mapped, so more routing tracks are needed. Figure 11 shows the

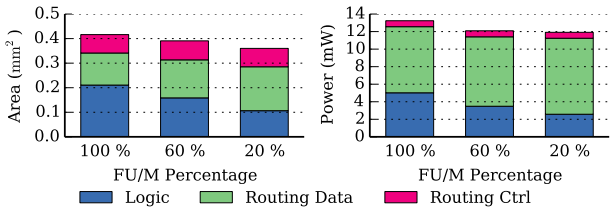
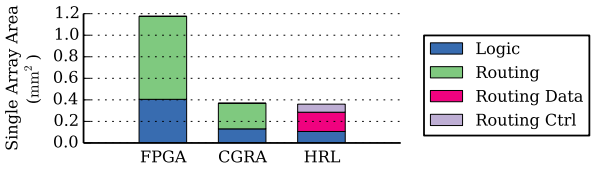


Figure 11: HRL power and area for different number of functional units with multipliers.



Logic	FPGA	CGRA	HRL
Grid Size	15×15	5×5	6×6
Blocks	165 CLBs 12 DSPs	25 FUs 36 Switches	20 FUs 6 CLBs 10 OMBs
Routing Tracks	250 x1	–	26 x16 Data 66 x1 Ctrl

Figure 12: Size and area comparison of logic arrays.

area and power breakdown for the HRL arrays with different percentages of FUs with multipliers (FU/M's). FU/M's are placed in one or more columns evenly across the array (e.g., 60% means three of the five columns of FUs are FU/M's). Both the area and power of the logic blocks decrease significantly when fewer FU/M's are used, but the routing part increases and offsets most of the benefits. Using only 20% FU/M's is slightly better than the other two cases. We use this design for the rest of the evaluation. The resulting routing fabric contains 26 16-bit bus tracks and 66 single-bit control tracks.

6.2 Reconfigurable Array Comparison

We now compare HRL with the two common reconfigurable array types, FPGA and CGRA (DySER). We size each array to have the same logic capacity. This means that the array can implement approximately the same amount of logic. To support the applications in Section 5, all arrays must accommodate the largest kernel circuit. For FPGA, we also increase the ratio of DSP blocks and remove BRAM blocks. Based on the critical path analysis, we run all kernel circuits at 100 MHz on FPGA and at 200 MHz on both HRL and CGRA. Although CGRA can run at a higher frequency, the frequency is kept low to avoid thermal problems within the 3D stack. Section 6.3 shows that higher frequency provides little performance benefit anyway.

Area: Figure 12 summarizes the size and area of each array. The FPGA array incurs large area overhead for its flexible bit-level logic and routing resources. The CGRA and HRL arrays, on the other hand, leverage the coarser granularity and only occupy about one third area of the FPGA array. HRL's data routing path is slightly smaller than that

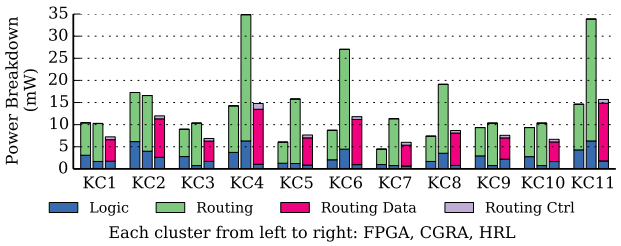


Figure 13: Power consumption for 11 kernel circuits on different arrays.

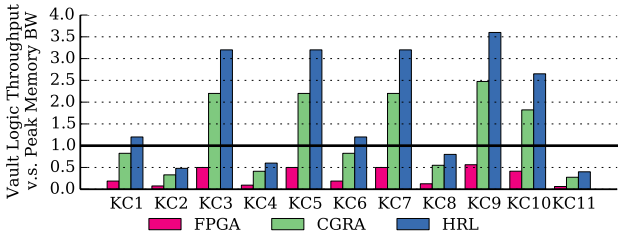


Figure 14: The total throughput of PEs in one vault. Normalize with the peak vault memory bandwidth.

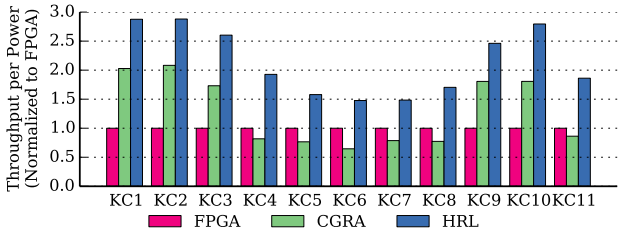


Figure 15: Power efficiency comparison for three PE types.

for CGRA, mainly because CGRA uses expensive switches that contain data registers.

Power: Figure 13 shows the power breakdown of each KC on the three configurable arrays. In general, the FPGA array consumes the least power in logic due to its lower frequency. The CGRA and HRL arrays have similar power consumption for logic since they use similar functional units, but sometimes HRL logic power is larger due to the inefficient packing algorithm which doesn't always achieve the best results and uses more blocks. However, for KC2, KC4, KC6, K8 and KC11, CGRA consumes much more power than HRL for logic. These kernel circuits involve branches, for which HRL uses the OMBs (see Table 3) that consume less power than CGRA's select or ϕ-function operations [19].

For routing power, CGRA's circuit-switched routing is very power-hungry. As we discussed, these switches have multiple data registers for different routing directions that consume high power [21]. KC4, KC6, KC8 and KC11 have significantly higher routing power due to the complicated data flows. On the other hand, HRL's data and control routing are quite efficient as they use fewer and cheaper routing tracks with fewer connections between coarse-grain blocks and the routing fabric. The routing power consumption is close to FPGA which runs at half speed. KC4 has complicated branch at output, and KC11 requires data from the CLB output for the piece-wise linear function coefficients. Thus they consume high routing power on HRL.

On average, the HRL array consumes almost the same power as the FPGA array while running at double frequency and saves 47.5% power compared to the CGRA array at the same frequency.

6.3 HRL Vault Analysis

We now compare the three reconfigurable logic options in a single vault in an NDP system. As we described in Section 4, each vault contains multiple PEs of any type. Based on the 70 mm^2 overall area constraint for the logic layer, we assume 6 mm^2 per vault can be used for the PEs, which leaves about 3 mm^2 for the other components in each vault. The scratchpad buffer is the largest one and takes 1.2 mm^2; others are less than 1 mm^2 in total. Similarly, the PE power budget of each vault is about 625 mW according to the overall power constraints for the stack. The peak PE power is estimated as 1.5 times of the largest power consumption over all considered KCs. Within these practical area and power constraints, we fit as many PEs of each type as possible, to exploit the high memory bandwidth. According to Figure 12 and 13, we can have 5 FPGA arrays, 11 CGRA arrays, or 16 HRL arrays for each vault. FPGA is limited by area and CGRA is limited by power.

Figure 14 shows the total computational throughput in one vault, normalized to the peak vault memory bandwidth. The solid line (1.0) indicates a balanced design, i.e., computational throughput matches memory bandwidth. Above the line implies the system is memory-bound and there is waste in the compute capability; below the line means that memory bandwidth is underutilized. FPGA cannot provide sufficient processing throughput. Both CGRA and HRL are able to exploit the abundant bandwidth in most cases, but only HRL achieves balance in KC1 and KC6. Also, our CGRA mapping is optimistic; realistic CGRAs will have smaller throughput if the design is hard to route. KC2, KC4, and KC11 have lower computational throughput in all cases because they are not fully pipelined by HLS tool and have high initialization intervals (IIs, the number of cycles between two data inputs). It is also obvious that optimizing CGRA and HRL to run at higher clock frequencies is not particularly useful as they already hit the power constraint or match the memory bandwidth.

Figure 15 compares power efficiency as in throughput per Watt. Both the computational throughput and power consumption of CGRA are higher than those of FPGA, which results in slightly better efficiency. HRL is 2.15x more power efficient than FPGA and 1.68x than CGRA on average, and outperforms the alternatives for *all* considered kernel circuits. This allows HRL to provide more processing capability without increasing the power, which results in higher performance with similar power consumption for the full NDP systems (see Section 6.4).

6.4 HRL System Analysis

We now look at the performance of the full NDP systems

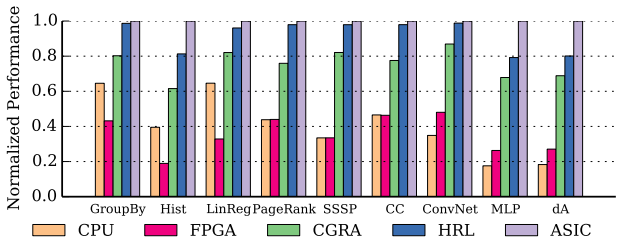


Figure 16: Application execution performance on all PE types.

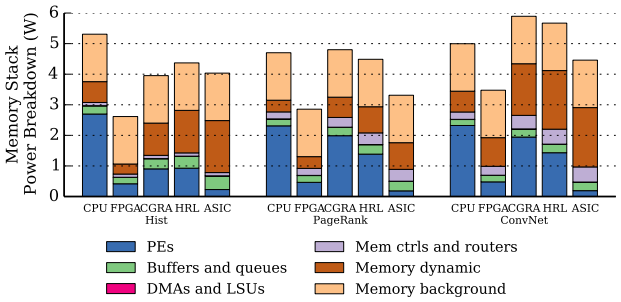


Figure 17: Average power breakdown for one memory stack.

including all phases of each application, most of which execute on the memory-side PEs, while some execute on the host processor. The overall NDP system includes 8 stacks with 8 vaults each. We compare all PE options, including in-order multi-threaded cores, FPGA, CGRA, HRL, and custom circuits (ASIC). The number of PEs per vault for each logic type is determined as explained in Section 6.3. In all cases, the system with custom engines can easily saturate the memory bandwidth in each vault, thus serves as a upper bound of efficiency. Figure 16 shows that the NDP systems based on multi-threaded cores, FPGAs, or CGRAs can only achieve about 30% to 80% of the custom-based system performance due to the limited number of PEs that can fit within the available area and power budgets. In contrast, HRL comes within 92% of the performance limits and has no more than 20% slowdown at the worst case. Compared with the other alternatives, HRL provides 1.2x to 2.6x speedup. This is a direct result from Figure 14: HRL has sufficient PEs to saturate DRAM bandwidth, while others are limited by area and/or power.

Figure 17 shows the power breakdown of one memory stack for three representative applications, one per application domain. We ignore the host processor power here. The memory components consume more than half the power. DRAM dynamic power is proportional to the bandwidth utilization, thus it is higher in the systems with HRL and custom units. The power for scratchpad buffers and output queues is not significant. The PE power matches the results in Figure 13. Multi-threaded core is not power-efficient due to its general structure and high clock frequency (1 GHz). It consumes small amount of power in memory since the bandwidth is seriously underutilized. Custom circuit is the opposite, which has very low power in the PEs while the high bandwidth results in high memory dynamic power. FPGA consumes low power but the performance is also bad. HRL uses more PEs than CGRA to achieve better performance, while the power consumption is slightly lower. Overall, with near-peak performance, HRL-based systems consume only 23% more power than the ideal custom-based systems.

Figure 17 also shows the power overhead of placing processing near data compared to a baseline 3D memory system without compute capability. The cost is roughly 1 W to 2 W per stack for HRL, which is acceptable for thermal consideration as discussed in [46].

7. CONCLUSION

This paper focuses on compute logic options for near-data processing systems, which provide significant performance and energy improvements for in-memory analytics applications. Conventional options, including programmable cores, reconfigurable logic, and custom engines, are limited either in their power and area efficiency or in their flexibility. We propose a novel reconfigurable array called Heterogeneous Reconfigurable Logic (HRL), that combines the advantages of FPGA and CGRA designs to provide a logic substrate with high performance, high efficiency and high flexibility for NDP systems. We use HRL to build a scalable multi-vault, multi-stack NDP system with efficient communication for MapReduce, graph processing, and deep neural networks workloads. HRL improves performance per Watt by 2.15x over FPGA and 1.68x over previously-proposed CGRA. For NDP systems with practical constraints, HRL achieves 92% of the peak performance of an NDP system based on custom units on average for all evaluated applications.

8. ACKNOWLEDGMENT

The authors want to thank Raghu Prabhakar, Christina Delimitrou, and the anonymous reviewers for their insightful comments on earlier versions of this paper. This work was supported by the Stanford Pervasive Parallelism Lab, the Stanford Platform Lab, the Center for Future Architectures Research (C-FAR), Samsung, and NSF grant SHF-1408911.

9. REFERENCES

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," in *ISCA-42*, pp. 105–117, 2015.

[2] Altera Corporation, "Altera SDK for Open Computing Language (OpenCL)." https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[3] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-Data Processing: Insights from a MICRO-46 Workshop," *Micro, IEEE*, vol. 34, pp. 36–42, July 2014.

[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ASPLOS-19*, pp. 269–284, 2014.

[5] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System," in *OSDI-11*, pp. 571–582, 2014.

[6] K. Choi, "Coarse-Grained Reconfigurable Array: Architecture and Application Mapping," *IPSJ Transactions on System LSI Design Methodology*, vol. 4, pp. 31–46, 2011.

[7] E. S. Chung, J. D. Davis, and J. Lee, "LINQits: Big Data on Little Clients," in *ISCA-40*, pp. 261–272, 2013.

[8] Computing Community Consortium (CCC), "Challenges and Opportunities with Big Data." http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf, Feb. 2012.

[9] S. Das, A. Fan, K.-N. Chen, C. S. Tan, N. Checka, and R. Reif, "Technology, Performance, and Computer-aided Design of Three-dimensional Integrated Circuits," in *ISPD '04*, pp. 108–115, 2004.

[10] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," in *NIPS*, pp. 1223–1231, 2012.

[11] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI-6*, pp. 10–10, 2004.

[12] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal Feasibility of Die-Stacked Processing in Memory," in *WoNDP-2*, December 2014.

[13] D. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational RAM: Implementing Processors in Memory," *IEEE Des. Test*, vol. 16, pp. 32–41, Jan. 1999.

[14] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA-38*, pp. 365–376, 2011.

[15] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *HPCA-21*, pp. 283–295, Feb 2015.

[16] M. Gao, G. Ayers, and C. Kozyrakis, "Practical Near-Data Processing for In-memory Analytics Frameworks," in *PACT-24*, pp. 113–124, Oct 2015.

[17] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Power-Graph: Distributed Graph-parallel Computation on Natural Graphs," in *OSDI-10*, pp. 17–30, 2012.

[18] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework," in *OSDI-11*, pp. 599–613, Oct. 2014.

[19] V. Govindaraju, *Energy Efficient Computing through Compiler Assisted Dynamic Specialization*. PhD thesis, University of Wisconsin-Madison, WI, 2014.

[20] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing," *IEEE Micro*, vol. 32, pp. 38–51, Sept. 2012.

[21] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically Specialized Datapaths for Energy Efficient Computing," in *HPCA-17*, pp. 503–514, 2011.

[22] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WoNDP-2*, 2014.

[23] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park, "Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture," in *Supercomputing '99*, p. 57, 1999.

[24] M. Horowitz, "Computing's Energy Problem (and What We Can Do about It)," in *ISSCC'14*, pp. 10–14, Feb 2014.

[25] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu, "Elastic CGRAs," in *FPGA '13*, pp. 171–180, 2013.

[26] E. Hung, F. Eslami, and S. J. E. Wilton, "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices," in *FCCM-21*, pp. 45–52, 2013.

[27] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 1.0," 2013.

[28] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube New DRAM Architecture Increases Density and Performance," in *VLSIT 2012*, pp. 87–88, June 2012.

[29] JEDEC Standard, "High Bandwidth Memory (HBM) DRAM." JESD235, 2013.

[30] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *ISCA-40*, pp. 404–415, 2013.

[31] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-stage Design Space Exploration," in *DATE '09*, pp. 423–428, 2009.

[32] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "FlexRAM: Toward an Advanced Intelligent Memory System," in *ICCD-30*, pp. 5–14, Sept 2012.

[33] P. M. Kogge, "EXECUBE-A New Architecture for Scaleable MPPs," in *ICPP'94*, vol. 1, pp. 77–84, Aug 1994.

[34] D. Larkin, A. Kinane, V. Muresan, and N. O'Connor, "An Efficient Hardware Architecture for a Neural Network Activation Function Generator," in *ISNN '06*, pp. 1319–1327, 2006.

[35] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO-42*, pp. 469–480, 2009.

[36] G. H. Loh, "Extending the Effectiveness of 3D-stacked DRAM Caches with an Adaptive Multi-Queue Policy," in *MICRO-42*, pp. 201–212, Dec 2009.

[37] G. H. Loh and M. D. Hill, "Supporting Very Large DRAM Caches with Compound-Access Scheduling and MissMap," *Micro, IEEE*, vol. 32, pp. 70–78, May 2012.

[38] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," vol. 7, pp. 6:1–6:30, June 2014.

[39] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," in *ISCA-27*, pp. 161–171, 2000.

[40] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *FPL 2003*, pp. 61–70, 2003.

[41] Micron Technology Inc., "TN-41-01: Calculating Memory System Power for DDR3." http://www.micron.com/support/power-calc, 2007.

[42] M. Motoyoshi, "Through-Silicon Via (TSV)," *Proceedings of the IEEE*, vol. 97, pp. 43–48, Jan 2009.

[43] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *MICRO-40*, pp. 3–14, 2007.

[44] M. Oskin, F. T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," in *ISCA-25*, pp. 192–203, 1998.

[45] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM," *IEEE Micro*, vol. 17, pp. 34–44, Mar 1997.

[46] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the Impact of 3D-Stacked Memory+ Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.

[47] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, pp. 16–19, Jan 2011.

[48] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-Stream: Edge-centric Graph Processing Using Streaming Partitions," in *SOSP-24*, pp. 472–488, 2013.

[49] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems," in *ISCA-40*, pp. 475–486, 2013.

[50] H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Trans. Comput.*, vol. 49, pp. 465–481, May 2000.

[51] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: Modular MapReduce for Shared-memory Systems," in *MapReduce '11*, pp. 9–16, 2011.

[52] C. Weis, N. Wehn, L. Igor, and L. Benini, "Design Space Exploration for 3D-stacked DRAMs," in *DATE 2011*, pp. 1–6, March 2011.

[53] C. Wolf, "Yosys Open SYnthesis Suite." http://www.clifford.at/yosys/.

[54] Xilinx Inc., "UG364: Virtex-6 FPGA Configurable Logic Block." http://www.xilinx.com/support/documentation/user_guides/ug364.pdf, February 2012. v1.2.

[55] Xilinx Inc., "Xilinx Power Estimator (XPE)." http://www.xilinx.com/products/technology/power/xpe.html, 2012. v14.3.

[56] Xilinx Inc., "Vivado High-Level Synthesis (HLS)." http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, 2014.

[57] A. Ye and J. Rose, "Using Bus-based Connections to Improve Field-programmable Gate-array Density for Implementing Datapath Circuits," *VLSI Systems, IEEE Transactions on*, vol. 14, pp. 462–473, May 2006.

[58] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing," in *NSDI-9*, pp. 2–2, 2012.

[59] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-oriented Programmable Processing in Memory," in *HPDC-23*, pp. 85–98, 2014.

[60] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *3DIC '13*, pp. 1–7, Oct 2013.