GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator

Jan Gray
Gray Research LLC, Bellevue, WA, USA
jsgray@acm.org

***Abstract*— GRVI is an FPGA-efficient RISC-V RV32I soft processor. Phalanx is a parallel processor and accelerator array framework. Groups of processors and accelerators form shared memory clusters. Clusters are interconnected with each other and with extreme bandwidth I/O and memory devices by a Hoplite NOC with 300-bit links. An example Kintex UltraScale 040 system has 400 RISC-V cores, peak throughput of 100,000 MIPS, peak shared memory bandwidth of 600 GB/s, NOC bisection bandwidth of 700 Gb/s, and uses 12-17 W.**

Keywords—FPGA; soft processor; GRVI; RISC-V; RV32I; cluster; accelerator; Phalanx; Hoplite; router; NOC.

I. Introduction

In this Autumn of Moore's Law, the computing industry is challenged to scale up throughput and reduce energy. This drives interest in FPGA accelerators, particularly in datacenter servers. For example, the Microsoft Catapult [1] system uses FPGA acceleration at datacenter scale to double throughput or cut latency of Bing query document ranking.

As computers, FPGAs offer parallelism, specialization, and connectivity to modern interfaces including 10-100 Gb/s Ethernet and many DRAM channels (soon HBM). Compared to general purpose CPUs, FPGA accelerators can achieve higher throughput, lower latency, and lower energy.

There are two big challenges to development of an FPGA accelerator. The first is software: it is expensive to move an application into hardware, and to maintain it as code changes. Rewriting C++ code in RTL is painful. High level synthesis maps a C function to gates, but does not help compose modules into a system, nor interface the system to the host.

OpenCL to FPGA tools are a leap ahead. Now developers have a true software platform that abstracts away low level FPGA concerns. But "OpenCL to FPGA" is no panacea. Much software is not coded in OpenCL; the accelerator is specialized to particular kernel(s); and a design spin may take hours.

To address the diversity of workloads, and for fast design turns, more of a workload might be run directly as software, on processors in the fabric. (Soft processors may also be tightly coupled to accelerators.) To *outperform* a full custom CPU requires *many* energy-efficient, FPGA-efficient soft processors.

The second challenge is implementation of the accelerator SOC hardware. The SOC consists of dozens of compute and accelerator cores, interconnected to each other and to extreme bandwidth interface cores e.g. PCI Express, 100G Ethernet, and, in the coming HBM era, eight or more DRAM channels. How is it possible to interconnect the many compute and interface cores at full bandwidth (50-150 Gb/s)?

GRVI Phalanx is a framework for building accelerators that helps address these problems. A designer can use any mix of software, accelerated software, or fixed function accelerators, composed together into clusters. With a cluster, cores can be directly coupled or communicate through shared memory. Between clusters and external interface cores, a wide Hoplite NOC [2] carries data as messages at full bandwidth.

The rest of this paper details a work-in-progress GRVI Phalanx system, bottom up: GRVI soft processor, GRVI cluster, Hoplite NOC, complete system, aspirational programming models, and status/next steps.

II. The GRVI RISC-V RV32I Core

Actual *acceleration* of a software-mostly workload requires an FPGA-efficient soft processor that implements a standard instruction set architecture (ISA) for which the diversity of open source software tools, libraries, and applications exist. The RISC-V ISA [3] is a good choice. It is an open ISA; it is modern; extensible; designed for a spectrum of use cases; and it has a comprehensive infrastructure of open source specifications, test suites, compilers, tools, simulators, libraries, operating systems, and processor and interface IP. Its core ISA, RV32I, is a simple 32-bit integer RISC. A question for the present work was: is it possible to devise an extremely FPGA-efficient implementation of RISC-V RV32I?

A processor design is an exercise in which features to include and which to leave out. A simple, austere core uses fewer resources, which enables more cores per die, achieving more compute and memory parallelism per die.

The design goal of the GRVI RV32I core was therefore to maximize MIPS/LUT. This is achieved by eliding inessential logic from each CPU. Infrequently used resources, such as shifter, multiplier, and byte/halfword load/store, are cut from the core. Instead, they are shared by a pair of cores in the cluster, so that their overall amortized cost is halved.

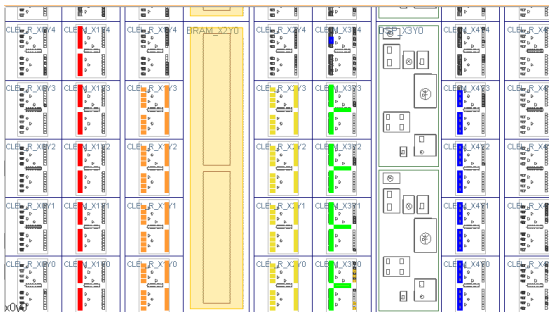

Fig. 1: GRVI processing element (PE) datapath RPM

The GRVI microarchitecture is unremarkable. It is a two or three stage pipeline (optional instruction fetch latch; decode; execute). The datapath (Fig. 1) includes a 2R/1W register file; two sets of operand multiplexers (operand selection and result forwarding) and registers; an ALU; a dedicated comparator for conditional branches and SLT (set less than); a PC unit for I-fetch, jumps, and branches; and a result multiplexer to select a result from ALU, return address, load data, optional shift and/or multiply. The color shaded LUT slices in Fig. 1, left to right, comprise the register file, first and second operand muxes and registers, ALU, and result mux – all 32-bits tall.

The datapath 6-LUTs are explicitly technology mapped and floorplanned into a relationally placed macro (RPM). The ALU, PC unit, and comparator use "carry logic". Each LUT in the synthesized control unit is scrutinized.

GRVI is small and fast. The datapath uses 250 LUTs; the core overall is typically 320 LUTs. It runs at up to 375 MHz in Kintex UltraScale (-2). Its CPI is not yet determined but is expected to be ~1.3 / ~1.6 (2 / 3 stage). Thus the figure of merit for the core is approximately 0.7 MIPS/LUT.

III. GRVI Clusters

As GRVI is relatively compact, it is possible to implement many PEs per FPGA – 750 in a 240,000 LUT KU040. Besides PEs, the system needs memories and interconnects. A KU040 has 600 dual-ported 1Kx36 BRAMs – one per 400 LUTs.

How might all these cores and memories be organized into a useful, fast, easily programmed multiprocessor? It depends upon workloads and their parallel programming models. The present design targets data, task, and process network parallel programs (SPMD or MIMD) with small compute kernels.

For data memory, it is difficult to build fast cache coherent shared memory for hundreds of cores. Also, caches consume resources better spent on *computation*. Thus: no data caches.

Another option is an uncached shared memory design. Here BRAMs are grouped into 'memory segments' distributed about the FPGA; any PE or accelerator at any site on the FPGA may issue remote store and load requests, and load responses, which traverse a NOC to and from the addressed memory segment. This is easy to build and to program correctly, but since GRVI is not memory latency tolerant, a non-local load might stall a PE for 10-20 cycles as the load request and response traverse the NOC. Here shared memory programs are slow programs.

Instead, the Phalanx architecture partitions FPGA resources into small *clusters* of processors, accelerators, and a cluster shared memory (CRAM) of 4-64 KB in size. CRAM accesses have fixed low latency, and so as long as a workload's data can be divided into small working sets (≤ 64 KB), developers will naturally wind up in the "Pit of Success". (Looking ahead, new Xilinx UltraScale+ FPGAs with 288 Kb UltraRAM blocks will greatly improve per-cluster RAM capacities.) Table 1 lists some of the better GRVI cluster configurations. The present design uses the last configuration in the table.

BRAMS	LUTs	PEs	IRAM	CRAM	Clusters
1I + 2D = 3	1200	2	4 KB	8 KB	200
2I + 4D = 6	2400	4	4 KB	16 KB	100
4I + 8D = 12	4800	2	16 KB	32 KB	50
4I + 8D = 12	**4800**	**8**	**4 KB**	**32 KB**	**50**

Table 1: Some GRVI cluster configurations

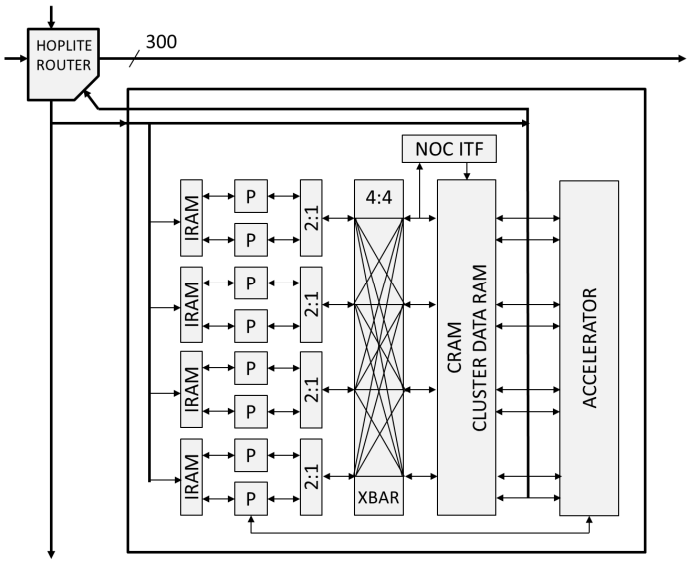

Fig. 2: Eight GRVI Cluster and 288-bit payload Hoplite router

The no. of BRAMs per cluster determines its LUT budget. In a KU040, twelve BRAMs correspond to 4800 6-LUTs.

In this configuration, eight PEs share 12 BRAMs. Four BRAMs are used as small 4 KB kernel program instruction memories (IRAMs). Each pair of processors share one IRAM.

The other eight BRAMs form a 32 KB cluster shared memory (CRAM). This has twelve 32-bit wide ports. Four ports provide a 4-way banked interleaved memory for PEs. Each cycle, up to four accesses may be made on the four ports. The eight PEs connect to the CRAM via four 2:1 concentrators and a 4x4 crossbar. This requires fewer than half of the resources of an 8x8 crossbar. See Fig. 2.

In case of simultaneous access to a bank from multiple PEs, an arbiter grants it to one PE and halts the others' pipelines.

The remaining eight ports provide an 8-way banked interleaved memory for accelerator(s), and also form a single 256-bit wide port to send or receive 32 byte messages, per cycle, to any NOC destination, via the cluster's Hoplite router.

To send a message, one or more PEs prepare a 32 B message in CRAM, then one PE stores the Phalanx Address (PA) of the message destination to the NOC interface's memory mapped I/O region. The NOC interface receives the request and atomically loads, and sends, a 32 B message to some client of the NOC. The PA of the message destination encodes the NOC address (x,y) of the destination, as well as the local address at the destination. If the destination is a GRVI cluster, the arriving message is immediately written into that cluster's CRAM and/or its accelerator(s).

Message send also enables fast *local* memcpy and memset. Aligned data may be copied at 32 B per two cycles, by sending a series of 32 B messages from a cluster, via its router, to itself.

If a cluster is configured with one or more accelerators, they may communicate with the PEs via shared memory or directly via a PE's ALU, store-data, and load-data ports. Accelerators may also use the router to send/receive messages.

(Soon) message send/receive will be used to store/load 32 B to DRAM, to send/receive an Ethernet packet (as a series of messages) to/from an Ethernet NIC, and to send/receive data to/from AXI4 Stream endpoints.

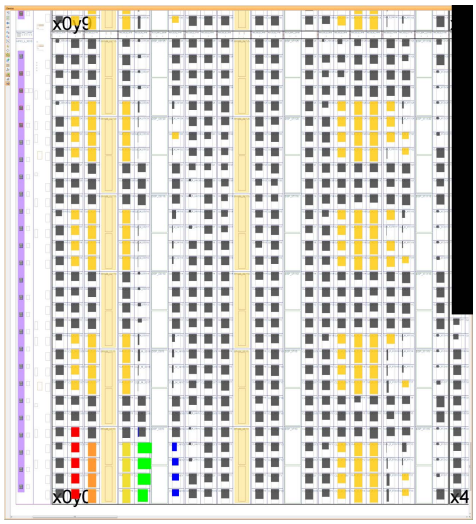

Fig. 3: Cluster implementation: 8 GRVI + 12 BRAM + router

Figure 3 is the floorplanned layout of a cluster of 8 GRVI PEs, 12 BRAMs (4 IRAMs, 1 CRAM), 0 accelerators, local interconnect, Hoplite NOC interface, and Hoplite NOC router. A cluster may be configured with more/fewer PEs and more or less IRAM and CRAM, to right-size resources to workloads.

As with the GRVI core, the cluster 'uncore' is implemented with care to conserve LUTs. For example, there are no FIFOs or elastic buffers in the design. From a compute/LUT or compute/Joule perspective, buffers are overhead. Instead NOC ingress flow control of message sends is manifest as wait states (pipeline holds) in the PE(s) attempting to send messages. Back pressure from the NOC, through the arbitration network, to each core's pipeline clock enable, is the critical path in the design, and *currently* it caps the clock period at about 300 MHz (small NOCs) and 250 MHz (die spanning SOCs).

IV. Hoplite NOC

Hoplite [2] is a configurable directional 2D torus router that efficiently implements extreme bandwidth NOCs on FPGAs. A Hoplite router has a configurable routing function and a switch with three message inputs (XI, YI, I (i.e. from client)) and two outputs (X, Y). At least one of the output message ports serves as the client output. Routers are composed on unidirectional X and Y rings to form a 2D torus network. See Fig. 4.

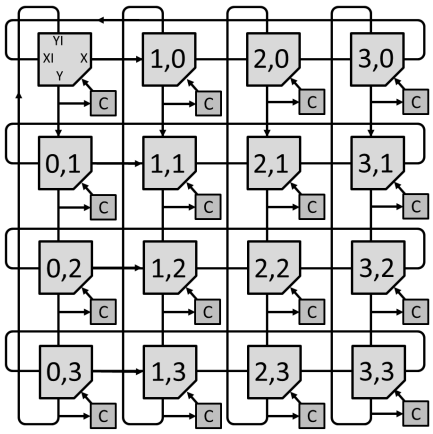

Fig. 4: A 4x4 Hoplite NOC with 16 client cores

A Hoplite router is simple, frugal, wide, and fast. In contrast with prior work, Hoplite routers use unidirectional, not bidirectional links; no buffers; no virtual channels; local flow control (by default); atomic message send/receive (no message segmentation or reassembly); client outputs that share NOC links; and configurable { ultra-wide links, workload optimized routing, multicast, in-order delivery, client I/O specialization, link energy optimization, link pipelining, and floorplanning }.

Hoplite was devised as a compact 2D router for an earlier MPPA limit study on how many 32-bit soft processor cores might fit in one XC6V240T. To conserve LUTs, FPGA-inefficient buffered virtual channel router design orthodoxy was set aside, to ask instead "what is *essential* to carry a *w*-bit message from any source to any destination in O(√N) hops?"

Hoplite was inspired by Kim's dimension sliced router [4] but is more austere. The unidirectional torus reduces a router's 5x5 crossbar to 3x3. The client output message port is infrequently used and inessential, and may be elided by reusing an inter-router link as a client output. This further simplifies the switch to 3x2. Since there are no buffers, when and if output port contention occurs, the router deflects a message to a second port. It will loop around its ring and try again later.

A one-bit slice of a 3x2 switch and its registers may be technology mapped into a fracturable Xilinx 6-LUT or Altera ALM, with a one wire+LUT+FF delay critical path through a router. (This is area and delay optimal for such a 2D router.) For die-spanning NOCs, inter-router wire delay is typically 90% of the clock period. This can be reduced by using pipeline registers in the inter-router links. Altera Stratix 10 HyperFlex interconnect pipeline flops seem well suited to Hoplite NOCs.

Figure 5 is an example KU040 floorplanned die-spanning 6x4 Hoplite NOC with 256-bit message payloads that runs at 400 MHz and uses <3% of the device. This torus is not folded, so uses two sets of pipeline registers in the Y rings and one set in the X rings. Link bandwidth is 100 Gb/s. Bisection bandwidth is 800 Gb/s. Sans deflection, average latency from anywhere to anywhere is about 7 cycles / 17.5 ns.

Compared to FPGA-optimized buffered VC routers [5], Hoplite has an orders of magnitude better area × delay product. (Torus16, a 4x4 torus with 64-bit-flits and 2-VCs uses ~38,000 LUTs and runs at 91 MHz. A 4x4 Hoplite NOC of 64-bit messages, uses 1230 LUTs and runs at 333-500 MHz.) It is far cheaper to build two Hoplite NOCs than one 2-VC NOC!

FPGA-efficient NOCs may change the way large SOCs are designed. A NOC makes it easy to interconnect each core to each external interface; and as long as a core can connect to *some* nearby router, and tolerate a few cycles of NOC latency, its particular site in the FPGA does not matter very much.

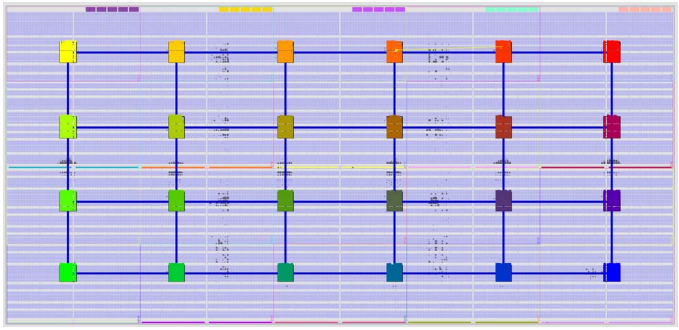

Fig. 5: A 6x4 Hoplite NOC with 100 Gb/s links (KU040)

Fig 6: A 400 GRVI Phalanx. 10x5 clusters of 8 PEs (KU040)

V. GRVI Phalanx

Fig 6. is a floorplanned 400 GRVI Phalanx implemented in a Kintex UltraScale KU040. It has ten rows by five columns of clusters (i.e. on a 10x5 Hoplite NOC); each cluster with eight PEs sharing 32 KB of CRAM. It uses 73% of the device's LUTs and 100% of its BRAMs. The 300-bit-wide Hoplite NOC uses ~6% of the device's LUT (~40 LUTs/PE).

In aggregate, the 400 PEs have a peak (will not exceed) throughput of about 100,000 MIPS. Total bandwidth into the CRAMs is 600 GB/s. The NOC has a bisection bandwidth of about 700 Gb/s.

Preliminary power data, measured via SYSMON, is 12-17 W (30-43 mW/core) during a parallel matrix multiply test workload spanning the 400 cores, sending and receiving matrix multiply work items and results over the NOC.

Fig. 7 shows a Verilog RTL snippet to instantiate a GRVI Phalanx, i.e. to instantiate the NOC and NX × NY array of clusters and interconnect NOC routers' inputs/outputs to each cluster. (It employs `XY etc. macros to mitigate Verilog's lack of 2D array ports.) A SOC/NOC floorplan generator (not shown) produces an FPGA implementation constraints file to floorplan the SOC/NOC into a die-spanning 10x5 array of tiles.

GRVI Phalanx is extensively parameterized and easy to retarget to 7 Series and UltraScale devices. A 2x2x8=32 PE port to a Digilent Arty board (XC7A35T) took two hours, mostly to tweak the floorplan generator to target the FPGA's irregular layout and resource mix, and to integrate the board's pinout constraints. Here the clock frequency is 150 MHz, so even in a modest FPGA NOC link bandwidth exceeds 40 Gb/s.

```
wire`XY    i_rdy; // client input accepted this cycle
wire`XY    o_v;   // client output valid this cycle
wire`MsgXY i;     // client input channels
wire`MsgXY o;     // client output channels

NOC #(.MCAST(MCAST), .NX(NX), .NY(NY),
      .D_W(D_W), .X_W(X_W), .Y_W(Y_W), ...)
  noc(.clk, .rst_in, .ce, .i_rdy, .i, .o_v, .o);

genvar x, y;
generate
  for (y = 0; y < NY; y = y + 1) begin : ys
    for (x = 0; x < NX; x = x + 1) begin : xs
      Clu #(.MCAST(MCAST), .NX(NX), .NY(NY), .X(x), .Y(y),
            .D_W(D_W), .X_W(X_W), .Y_W(Y_W), ...)
        c(.clk, ...,
          .no_v(o_v[`xy(x,y)]), .no(o[`mxy(x,y)]),
          .ni_rdy(i_rdy[`xy(x,y)]), .ni(i[`mxy(x,y)]));
    end
  end
endgenerate
```

Fig. 7: Verilog RTL for Hoplite NOC/cluster generation

VI. Accelerated Parallel Programming Models

GRVI Phalanx aspires to make it easier to develop and maintain an FPGA accelerator for a parallel software workload. Some workloads will fit its mold, i.e. highly parallel SPMD or MIMD code with small kernels, local shared memory, and global message passing. Here are some parallel models that should map fairly well to a GRVI Phalanx framework:

- OpenCL kernels: run each work group on a cluster;
- 'Gatling gun' parallel packet processing: send each new packet to an idle cluster, which may exclusively work on that packet for up to (#clusters) packet-time-periods.
- OpenMP/TBB: run MIMD tasks within a cluster;
- Streaming data through process networks: pass streams as messages within a cluster, or between clusters;
- Compositions of such models.

Since GRVI Phalanx is implemented in an FPGA, these and other parallel models may then be further accelerated via custom GRVI and cluster function units; custom memories and interconnects; and custom standalone accelerator cores on cluster RAM or directly connected on the NOC.

VII. Status and Future Work

GRVI Phalanx is a new work-in-progress. Currently GRVI cores implement the RV32I instruction set (sans perf counters) plus MUL/MULH* and LR/SC. At time of writing, 400 cores boot, run a monitor, a Thoth [6] message passing system upon which billions of messages have been sent/received, and parallel test workloads including AES encryption and integer matrix multiply. The system may be configured with a 1080p (320x102 character) VGA display console; its 32 KB VRAM is a Hoplite NOC client, writeable from any core or cluster.

Looking ahead the system will be elaborated with hardware and software to enable increasingly complex workloads. Near term work includes: debug/trace over the NOC; remote reads over NOC; interrupts; Hoplite-AXI4 and AXI4-Stream bridges to interface to Zynq, DRAM, Ethernet, PCI Express, and other IP; energy savings (e.g. GRVI sleep mode, awaiting a message); an Altera Arria 10 port to harness its many floating point DSP blocks; and an OpenCL implementation.

Acknowledgements

Thanks to Nachiket Kapre and Robert Ferguson for their interest and help with Hoplite and GRVI Phalanx.

References

[1] A. Putnam, et al, A reconfigurable fabric for accelerating large-scale datacenter services, in 41st Int'l Symp. on Comp. Architecture (ISCA), June 2014.

[2] N. Kapre, J. Gray, "Hoplite: building austere overlay NoCs for FPGAs", 25th Int'l Conf. on Field-Programmable Logic and Applications, Sept. 2015.

[3] K. Asanović, D. Patterson, "Instruction sets should be free: the case for RISC-V". Technical Report No. UCB/EECS-2014-146, August 2014.

[4] J. Kim, Low-cost router microarchitecture for on-chip networks, in 42nd IEEE/ACM Int'l Symp. on Microarchitecture, MICRO-42, Dec 2009.

[5] M. Papamichael, J. Hoe, "Connect: Re-examining conventional wisdom for designing nocs in the context of fpgas," in Proc. ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays, FPGA 2012, 2012.

[6] D. Cheriton, M. Malcolm, L. Melen, G. Sager. Thoth, a portable real-time operating system. Commun. ACM 22, 2 Feb. 1979.