WRAITH: A Resource-Efficient Dataflow Accelerator

ECE 427 Project Proposal

Prakhar Gupta prakhar7@illinois.edu
Ingi Helgason ingih2@illinois.edu
Pradyun Narkadamilli pradyun2@illinois.edu
Sam Ruggerio samuelr6@illinois.edu
Pratyay Rudravaram pratyay2@illinois.edu

***Abstract*—This work presents WRAITH, a compute substrate that attempts to exploit available TLP and DLP in a power-efficient manner by colocating a set of ”virtual” RISC-V processors and a dataflow accelerator on a shared datapath.**

I. Problem Statement

Dataflow accelerators are a broad class of compute engines that allow for parallelizing regular computational patterns with low control divergence. However, in cases where the compute workload is unable to take full advantage of the available compute in the accelerator, a large portion of the chip remains unused. One such case of this is applications with high *TLP* but not high *DLP*. These kind of applications are characterized by differing compute patterns, high control divergence, and operational independence. We believe that unused or underutilized compute in a dataflow accelerator can be used to exploit this TLP, even if not to the extent of a set of fully-featured CPU pipelines, when the accelerator is colocated with a set of high-powered CPUs to extract peak performance on ”complex” threads.

II. Architecture

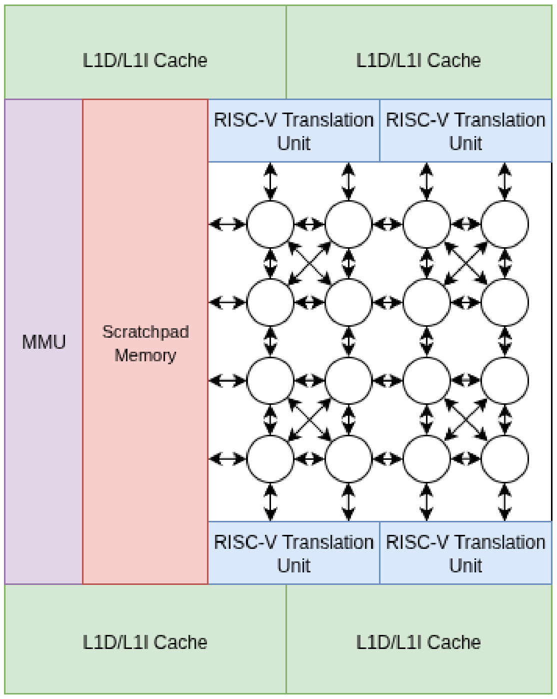

Fig. 1. WRAITH Architectural Block Diagram

We present **WRAITH**, a **W**orkload-**R**econfigurable **A**ccelerator with **I**ntegrated **T**hread **H**andling. Despite the long name, at a high level our architectural goal is simple - to add a set of RISC-V translation units (*RVTUs*) to a dynamic dataflow accelerator substrate that can convert traditional RISC-V instructions to modified instructions accepted by the compute network. This compute network would be a fixed network topology with *routing set at load-time* as opposed to a fixed-route network as popularized by the systolic array class of accelerators[3, 2]. An example of such an architecture is shown in Figure 1. Using such an approach awards us some big opportunities for compute efficiency:

- Sharing large functional units between multiple virtual cores and the dataflow path
- Retaining the scalability of typical dataflow accelerators by using a generic network topology
- Being able to ”allocate” functional units via routing paths rather than a specialized network architecture[1, 4]
- Potential for instruction coalescing in the processor front-ends to map onto longer network paths

Given the current block diagram, we expect our I/O pins to be split over three big categories.

- Memory Bus (16-bit data, R/W pins, potential data/address bit sharing)
- Inter-Die Network Connections (Minimum I/O for SerDe connection)
- Controller Configuration Pins (Master/Slave mode, RVTU state, etc.)
- Standard Clock and Power

A. Interconnect Protocol

This section of our proposal details the **WRAITH Interconnect Mesh Protocol (WIMP)**.

The Processing Elements (PEs) and peripherals in WRAITH communicate via a ”hooked, packet-based” communication scheme. While official terminology varies on how to refer to such architectures, we obey two basic principles:

1) A PE is a *reactionary* device – it will arbitrate between the packets on its various ingress ports, optionally generate a response, then send that to the appropriate egress port.
   a) The expected response and action to a packet is fully encoded by its *packet ID*, an identifier associated with an input packet.

The response and action patterns alluded to above are encoded via an *action table*. One of these exist per PE, and they encode the expected action of the PE, the destination of the response, and what (if any) local register to index into as a source.

WRAITH is primarily designed as a ”register-immediate” computation platform - one source will be delivered as part of the packet, and one source can be pulled off of the regfile. As a result, WIMP’s packets are single-flit (they are transferred in a single cycle), and require only a `valid` bit to demarcate valid information. A WIMP packet consists of only two fields:

1) Packet ID (nominally 3 bits)
2) Immediate Operand (32 bits)

WIMP’s expectation is that a PE will use the Packet ID to index into its action table, which will be programmed to represent the dataflow pattern desired on the CGRA. This, in turn, means that the configuration bitstream for WRAITH is the initialization configuration for the action tables stored in each of the PEs. Our action table, in all fairness, is rather simple. We’ve listed the individual fields below and nominal bit widths.

Field	Bit Width
Response Packet ID	3
Next-Hop Destination (N/S/W/E/D)	3
Function Select	4
Register Source Index	3
Register Dest Index	3
Total	16

In order to be able to program the action table, we reserve packet ID `0` as a ”configuration” packet. When a PE receives this, it treats it as a ”write to action table” packet. Importantly, our action table entries are thin enough that the immediate field of a packet can encode not only new action table entry, but also the action table index to write to (3 bits) and the mesh node that this packet is targeting (4 bits). For the sake of programming, PEs will implement fixed 2D routing logic in response to configuration packets.

1) If the bottom two bits (X Coord.) are different from the PE’s coordinate, pass right
2) If bottom two bits (X Coord.) are equivalent to the PE’s X coordinate a. If the top two bits (Y Coord). are different from the PE’s own Y coordinate, pass down b. If the top two bits (Y Coord). are equivalent to the PE’s own Y coordinate, write to table as specified

Additionally, PID 0 will act as a ”functional bypass” (the PE passes through the immediate unharmed).

Overall, the simplicity of WIMP allows us to trade off design simplicity for more software complexity – the lack of predication or typical ”fixed routing” means that the compiler written for WRAITH must take into account fixed bound iterations, contention between multiple ingresses in cyclic paths, etc.

To handle backpressure and contention, it is expected that each of the ingress ports is supplanted by a low-depth FIFO (nominally 2/4 entries). Lack thereof and instead using registers to buffer the packet runs the risk of very inefficient handshaking between ingress/egress pairs or long stall paths.

B. RISC-V Translation Units

For our proof-of-concept WRAITH chip, our goal is for a simple RV32IM processor to reuse the multiplier functional units in our mesh. For completeness of the RV32IM spec, a multicycle divider will be shared between each opposite pair of RISC-V translation units (RVTUs).

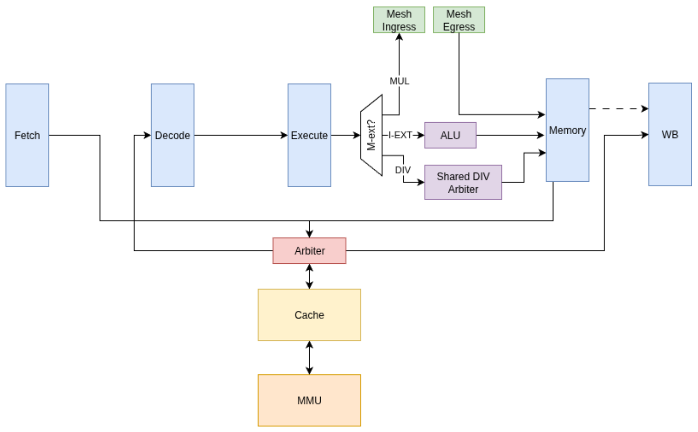

Fig. 2. The RISC-V Translation Unit with CGRA Modifications

For the most part, our initial prototype of an RVTU is going to be a typical 5-stage pipeline, albeit with the catch that our EX stage is able to generate packets for one of the Multiplier PEs, and block until it receives the appropriate response packet from the PE with the computed result. Divide instructions are sent to a multicycle divider arbited between a pair of RVTUs (one for the upper pair, one for the bottom pair).

Many simplifiying assumptions – including but not limited to a merged L1I/L1D cache, no Out-of-Ordering capabilities, and dedicated ALU/Branch HW – mean that our proof of concept here will not amortize much of the mesh area in the backends of the pipeline. The hope is that future iterations may be able to co-opt the mesh for more of the functional units in the pipeline, incorporate Out-of-Order scheduling, or emulate larger operations (like the V extension).

C. Software Interface

As with many experimental architectures, our current RTL progress is somewhat pipelined with further architectural decisions on the peripherals, in particular the MMU. Here we document the expected software API for WRAITH and how we expect the MMU to fulfill those necessities.

We intend WRAITH to be MMIO-based like most modern accelerators, using a DMA copy to load the configuration bitstream out of reset, with future DMA copies intended to transmit input and output blocks on/off the accelerator’s scratchpad memory block. The complication with WRAITH is that it acts as both a peripheral and master device – the RVTUs are still first-order devices that can make memory requests, but the CGRA mesh itself needs to be a peripheral for off-chip cores to do batch copies. Our current solution to this is to restrict WRAITH's ability to acccess the memory range that it is mapped into (thus preventing circular accesses) and to add a `master/slave` pin so to speak – this bit can be set when the memory bus detects that arbitration was *not* won by WRAITH and the target memory address is in WRAITH's MMIO range.

In order to prevent a simultaneous set of un-serviceable requests (from both the host CPU and WRAITH), an extra cycle of handshake is necessary. In addition to both parties signaling that they have a request of some kind that needs to be fulfilled using the shared bus, an additional pin is asserted on the next cycle after both parties signal that they have a read/write to complete. This way, the party not permitted to go first (statically-configured) can have its request squashed.

As discussed further in the subsequent MMU section, when there is a pending memory transaction which requires communication using the I/O bus, that

WRAITH requires many configuration settings, the most pertinent of which is which RVTUs to enable or disable, as well as various status signals that one may want to monitor at runtime (importantly whether WRAITH is done with the current computation). Our current thought is to assume that all of these signals can be made MMIO registers (i.e CSRs) at the moment, with potential to move them into the outer pinout if pin count allows. It is our expectation that the memory bus will occupy a majority of pins, with the first allocation of the remainin pins being pushed towards the RVTU enable/disables and a pin for "completion" that can be used as an interrupt on the host.

With regards to program interaction – in order to operate WRAITH, there are two distinct types of input DMA copies:

1) *Configuration Bitstream*: On the first DMA input copy after reset, the off-chip core must batch copy the contents of all the action tables on the chip in row-major order (including gaps if a PE is assumed to be unused, as it is claimed by an RVTU enabled in a CSR or by a pin).
2) *Input Data Blocks*: Subsequent DMA writes are expected to be input data – the expectation is that the first word of the data copied will contain the overall size of the input data and output data so that controllers streaming data into/off of the mesh will know when to terminate operation and signal completion via a CSR.

The expectation is that *any* DMA read off of WRAITH must only be triggered when WRAITH has a "complete" output block in its scratchpad memory (as indicated by the CSR or pin alluded to earlier). We are still nailing down the specifics of how we wish our DMA controller in the MMU to look like, and what failure tolerances and protections we want to build.

D. MMU

The on-board MMU (distinct from the memory management unit which will be associated with a RISC-V core used for system bring-up and validation in the Spring) possesses behavior somewhat resembling a past ECE 427 project in which a shared 16-bit bus was used to maintain cache coherency across 16 cores, for the purpose of transmitting address and data (the details of this past design are not relevant, they are mentioned for posterity's sake).

Our proposed system uses 16 read-write pins in a statemachined configuration. During polling cycles, two pins may be asserted (one by the WRAITH chip, and the other by a connected CPU) by each party to indicate the presence of a request. Additional pins will be used to indicate the exact size of the subsequent streamed memory transaction (more on this later).

If either of these pins is asserted, the state machine shall enter one of two stages, depending on which party is granted control of the bus. Rather than a round-robin scheme, both parties cooperatively select the party deemed to be higher-priority (our design assumes that the transactions written to the WRAITH chip ("off-chip") will be higher priority, and thus handled first). From here, the bus is written to by the party selected, to indicate the address corresponding to the data which will follow. Finally, the bus will enter a state where that data can be streamed.

The MMU accommodates reading from main memory (off-chip), writing back cachelines, programming the scratchpad memory (to set-up a kernel on the CGRA), and writing the SPM region back to the CPU (in order to provide the results of the kernel to the main processing unit). For this reason, arbitration between caches and scratchpad memory, as well as the tagging of the size of transaction needed (single-cacheline, full scratchpad write(back)) need to be handled by a context-aware arbiter between the MMU's bus interface and the associated memories, so to speak.

III. Current State

Much of our current work has been planning out detailed architectural decisions, some of which is outlined above. We in-progress RTL for the mesh, MMU, and RISC-V translation units. Notably we have:

- The full PE design mapped out, including action table, arbiter, and physical regfile interaction
- The respective FUs located within the PEs
- PE allocation expectations
- MMU structures for getting off-chip data
- RISC-V Frontends (Fetch-Decode)
- SRAM blocks for SPM and RISC-V

We suspect much of the chip will be written and tested within 2 weeks from this report date. We will likely produce several kernel mappings and programs manually for simulation, while a full compiler will be developed leading up to bringup.

IV. Timeline and Milestones

We are planning against the following timeline, with each weekly goal mapped out. Specific day-of milestones are marked in bold.

Week of 9/1	Finish PE Mesh, MMU Start, and RISC-V Frontends
9/8	SPM and Memory Hierarchy finalized and tested
9/15	RISC-V Pipeline and Instruction streaming functional and tested. Initial CGRA kernels mapped and running
9/22	Total System tests to verify time sharing efficacy and correctness
9/29	Additional edits while starting PD and proper area estimates and constraints are encountered
10/2	**RTL Freeze**
10/6	Floorplanning of full netlist, identify good macro placements, aim for initial functional PNR baseline. Do power planning.
10/13	Iterative changes to PNR to address congestion, improve area (if necessary), bad routing. Check/fix timing. Ensure I/O pad connection to core is good.
10/20	Further static timing checks (hold/setup fixes), assess parasitics and address as needed
10/27	Catch-up on any unfinished steps in previous weeks, DRC & LVS
11/3	DRC & LVS fixes
11/6	**Initial GDS Check**
11/10	Continue with necessary work, as per feedback from Initial check
11/17	Trial GDS ready
11/19	**Trial GDS Due**
11/24	GDS Corrections (if any)
11/26	**Final GDS Due**

V. Main Challenges and Complexities

We anticipate a large amount of work being necessary during the physical design phase, due to the large routing congestion and area overhead associated with the routing and interconnect associated with the processing element array. This presents complexity due to area, congestion, and timing requirements associated with the large array.

Due to the large number of components involved in the design, as well as the large number of nonstandard blocks (most notably the RISC-V translation units, as well as their caches), addressing DRC and LVS issues which crop up could also be extremely cumbersome. On a similar note, we imagine that any late-discovered RTL issues would be nigh-impossible to address at the physical design stage, also due to the large number of components.

Another challenge we anticipate encountering is the potential difficulty of finding germane advice regarding specific subsystems; while none of the facitilies we intend to implement (CPU frontend, memories, CGRA) are unheard of in the context of this class, the specific combination which we aim to create is reasonably novel, and may introduce some unique constraints at the architecture and/or physical design level which lead to additional challenges.

Finally, while not a technical challenge, we would be to remiss to mention that we are already on a delayed schedule relative to where we would like to be, and are currently short one group member due to some external exacerbating factors. We anticipate the need to spend much time and effort guaranteeing the correctness of our RTL models, and ensuring that their timing, power, and other requirements are feasible for the required form factor.

VI. Required IPs

We may require the underlying register file IPs provided by ARM's SRAM IPs to produce smaller than intended SRAM Blocks. We do not foresee requiring extra IPs outside of the provided IPs from ARM, Synopsys (including the DW multiplier and divider), and Cadence.

VII. Team Composition

For the sake of planning, we separate the efforts of this project into 5 broad categories. Each category is assigned two coordinators for redundancy who are exppected to maintain to-dos and short-term deliverables on each front. In practice, we expect all members of the group to be touching collateral across the project.

Category	Coordinators
Architecture/RTL	Pradyun, Sam
Physical Design	Pradyun, Ingi
Design Verification	Prakhar
Modeling and Software Support	Sam, Prakhar
Hardware Tooling	Ingi, Sam

We expect that at minimum 2/4 group members (Ingi, Pradyun) will be available to pursue bringup in the spring. Below are some of the qualifications of each group member.

- **Prakhar** - Senior in Computer Engineering
  - *Relevant Coursework*: ECE 411
  - *Experience*: Verification Intern at ARM, EE Intern at Siemens, ECE 391 CA
  - *Technical Strength*: Benchmarking, Verification, RTL, Scripting
- **Ingi** - Senior in Electrical Engineering
  - *Relevant Coursework*: ECE 411, ECE 425, ECE 340, ECE 477, ECE 483
  - *Experience*: ECE 391/411 CA, Data Acquisition / Radio Systems with Illini Electric Motorsports
  - *Technical Strength*: Physical Design, Tooling, System Integration
- **Pradyun** - 1st year M.S ECE
  - *Relevant Coursework*: ECE 511, ECE 411, ECE 425, ECE 482, ECE 438
  - *Academic Experience*: ECE 411 TA/CA, ECE 385 CA, ECE 391 CA
  - *Industry Experience*: FPGA Design Intern at IMC Trading, ASIC Design Intern at Microsoft, CPU Design Intern at SiFive

    - *Technical Strength*: Architecture, RTL, Physical Design, Tooling
- **Sam** - 3rd year PhD in Computer Science
    - *Relevant Coursework*: ECE 411
    - *Industry Experience*: Vulnerability Researcher at Battelle
    - *Technical Strength*: RTL, Tooling
- **Pratyay** - Junior in Computer Engineering
    - *Relevant Coursework*: ECE 411, ECE 391
    - *Experience*: ECE 411 CA, ECE 385 CA
    - *Technical Strength*: Benchmarking, Verification, RTL, Scripting

Across the board, each of our group members have a diverse range of experiences with high technical depth. We expect each individual member to have a great deal of flexibility when working with a project with high architectural and implementation complexity, hence the lofty proposal laid out in this document.

References

[1] Esam El-Araby, Vikram K. Narayana, and Tarek El-Ghazawi. "Space and Time Sharing of Reconfigurable Hardware for Accelerated Parallel Processing". In: *Reconfigurable Computing: Architectures, Tools and Applications*. Ed. by Phaophak Sirisuk et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–230. ISBN: 978-3-642-12133-3.

[2] G. Haug and W. Rosenstiel. "Reconfigurable hardware as shared resource for parallel threads". In: *Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.98TB100251)*. 1998, pp. 320–321. DOI: 10.1109/FPGA.1998.707935.

[3] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects". In: *SIGPLAN Not.* 53.2 (Mar. 2018), pp. 461–475. ISSN: 0362-1340. DOI: 10.1145/3296957.3173176. URL: https://doi.org/10.1145/3296957.3173176.

[4] Zhongyuan Zhao et al. "Towards Higher Performance and Robust Compilation for CGRA Modulo Scheduling". In: *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020), pp. 2201–2219. DOI: 10.1109/TPDS.2020.2989149.