Introducing the NAIL Accelerator Interface Layer for Low Latency FPGA Offload

Edward Grindley*, Thurstan Gray*, James Wilkinson*, Chris Vaux*, Adam Ardron*, Jack Deeley*
Alexander Elliott*, Nidhin Thandassery Sumithran†, and Suhaib A. Fahmy†
*The Alan Turing Institute, London, United Kingdom
Email: nailacceleration@gmail.com
†School of Engineering, University of Warwick, Coventry, United Kingdom
Email: s.fahmy@warwick.ac.uk

***Abstract*—We present the NAIL Accelerator Interface Layer (NAIL), a framework for offloading to Field Programmable Gate Arrays. NAIL has been optimised for latency-sensitive applications, supporting isolated multi-user acceleration. It allows accelerators to be employed through a flexible host communication layer, using asynchronous operation while processing data anywhere in host memory. Multiple independent processors are supported with large numbers of concurrent tasks. NAIL has been deployed at significant scale, and is now released as open-source.**

I. Introduction

Most work on interfacing FPGAs with general purpose processing presents a host-centric view of the FPGA, with limited virtualisation [1], [2]. Integration of FPGAs in cloud platforms has enabled wider adoption of their benefits, but host-centric virtualisation limits them to serving as hidden accelerators for vendor services, or virtual resources for hardware designers.

NAIL provides an abstraction supporting ad-hoc use of FPGA accelerators over PCIe or datacentre networks, focusing on low-latency and high throughput while processing small datagrams with small functions. Servers hosting one or more FPGAs offer different accelerator cores, each of which is usable by any host thread. Multiple threads can access multiple cores simultaneously, with data streams being seamless and isolated to the user. FPGA developers can design new cores without worrying about offload infrastructure, and users unfamiliar with FPGAs can apply such acceleration in their applications by exploiting a library of cores in an abstracted manner. NAIL is released as open source to enable the community to leverage these abstractions in their own designs and extend the framework [3].

II. Firmware Design Approach

A. Core Within Processor

NAIL's design considers individual accelerator "cores", each providing operational behaviour, enclosed within a "processor" responsible for managing control flow and interaction with shared resources. Currently, the NAIL design supports up to four cores per FPGA, due to the size of available Base Address Registers (BARs) on the PCIe interface (larger numbers have been proven experimentally). The four cores operate independently and simultaneously.

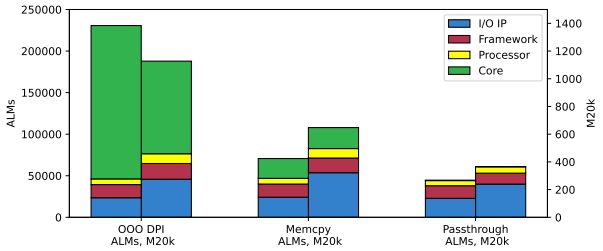

Fig. 1. NAIL resource consumption on Intel Arria10 GX 1150 FPGA.

Three example cores are evaluated in Fig. 1, that shows the overall usage of ALMs and M20Ks on the Intel Arria 10 GX 1150 FPGA for each of these cores:

- Simple passthrough core, resulting in a design that is dominated by PCIe and framework overheads with negligible core resources.
- Memcpy core which requires additional framework resources for connections to two banks of external DRAM.
- Out-of-order Deep Packet Inspection core that uses significant FPGA resources internally and requires large framework resources for out of order results handling.

I/O consumes 5–11% of resources available on the FPGA, with the NAIL base and processor framework accounting for a further 3–4%. Ultimately, this leaves significant resources available for core implementations (shown in green).

B. Generic Features

1) Common Interfaces: Dataflow in NAIL is based on a few consistent interfaces, including flat-vector, packetised, and memory-mapped communication typically with valid/ack-based flow-control. Crucially, this standardisation enables reuse of common sub-components, such as buffers, multiplexers and memories, for easier development, as well as simpler integration of larger components by eliminating conversion and adaptation functionality in RTL. Reuse also covers key testing and simulation functionality.

2) Equivalence to Software Functions: NAIL accelerator functions operate with a simple view of being issued a command and providing a result for it, in a 1-to-1 relationship. This mirrors the traditional software approach of calling a function

and receiving a returned value, providing easier integration with drivers and higher-level software.

3) Random Access Data: To enable the desirable capability of accessing data held in host memory, NAIL, as part of its command structure, enables the host to pass a reference to DMA-coherent memory, which can then be accessed by the FPGA. To eliminate the risk of data corruption, this particular data access is read-only. Other write-to-host interactions (such as returning results) do not have this restriction, as their interactions are constrained by the framework rather than the core.

4) DRAM Support: Where a core needs to utilise memory beyond the capacity of internal BRAMs or distributed RAM, NAIL gives cores access (read/write) to local board DRAM. This is done via the ”utility interface”, whereby each core is provided with zero or more ports connected to external resources on the device, depending on what is available.

C. Parallelisation

1) Concurrent Operation: Multithreaded utilisation of offloaded accelerators is not exclusive to NAIL, however NAIL compounds this with concurrent operation on a per-thread basis. A user of each implemented core can launch multiple jobs at once (the exact number constrained by the individual core design) before checking for completions as desired.

2) Out-of-Order Operation: When strict ordering is required by the design of a system, resources can often be left idle due to the risk of long-running tasks being ”overtaken” by faster operations. In the NAIL framework, out-of-order operation is supported, whereby parallel computations can be executed regardless of relative duration, to maximise throughput with results being restored in order after-the-fact to avoid corruption.

III. Software Design

A. Driver

NAIL's driver is traditionally written in C. Some specific responsibilities of the driver include:

- Allocate and release DMA-coherent user-space memory for firmware to interact with, including command and results buffers and required data memory.
- Notify the NAIL framework of new command packets.
- Gather results from NAIL to be made accessible to higher-level software.
- Manage virtual-memory addresses in the FPGA's address table.

B. Hardware Abstraction Layer (HAL)

The HAL's basic function (typically written in C++) is to provide software developers with the ability to utilise NAIL accelerators with understanding generally limited to the provided functionality rather than the method for manipulating hardware. Amongst other things, the HAL provides management of driver-based resources, formatting of command packets and polling of results. Specific cores may extend the HAL to adjust basic parameters, extra memory management, or custom results parsing. Examples of such code are included in the framework release.

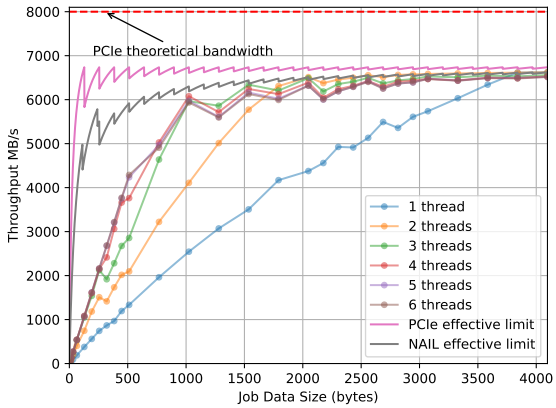

Fig. 2. Measured data throughput with respect to PCIe and framework limits.

IV. NAIL Performance

Fig. 2, shows the measured data throughput against modelled effective PCIe limit [4]. Additionally a simple model of NAIL's effective limit, taking into account the TLP overheads associated with both command and (non-line) data requests is shown. For larger data sizes per job, the framework reaches the practical limits of PCIe in use. Smaller job rate is limited by an Arria 10 PCIe endpoint limitation to available PCIe transfer credits, restricting concurrent TLP operations. For small (¡500 byte) data items the framework achieves over 8 million jobs per second despite this limitation.

V. Conclusions and Roadmap

NAIL is a latency focused FPGA offload framework that has been tested rigorously in production and is released as open source [3]. It provides an abstraction for managing FPGA accelerator cores that is suitable for non experts and offers low-latency offloading. NAIL is subject to several planned advancements:

- Porting to newer FPGAs and those from other vendors.
- PCIe interactions tested against novel PCIe connections; using devices such as DPUs for local interactions as well as RDMA technologies like RoCE.
- Partial Reconfiguration to replace individual cores.
- Data caching to allow multiple cores to work on the same data without repeated PCIe data transactions.

References

[1] A. Vaishnav, K. D. Pham, and D. Koch, “A survey on FPGA virtualization,” in *FPL*, 2018, pp. 131–137.

[2] K. Vipin and S. A. Fahmy, “DyRACT: A partial reconfiguration enabled accelerator and test platform,” in *FPL*, 2014.

[3] [Online]. Available: https://github.com/alan-turing-institute/nail-fpga

[4] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, “Understanding PCIe performance for end host networking,” in *SIGCOMM*, 2018, pp. 327–341.