Duet: Creating Harmony between Processors and Embedded FPGAs

Ang Li
Princeton University
Princeton, NJ 08544, USA
angl@princeton.edu

August Ning
Princeton University
Princeton, NJ 08544, USA
aning@princeton.edu

David Wentzlaff
Princeton University
Princeton, NJ 08544, USA
wentzlaf@princeton.edu

***Abstract*—The demise of Moore's Law has led to the rise of hardware acceleration. However, the focus on accelerating stable algorithms in their entirety neglects the abundant fine-grained acceleration opportunities available in broader domains and squanders host processors' compute power.**

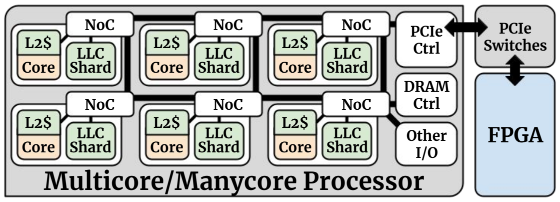
This paper presents Duet, a scalable, manycore-FPGA architecture that promotes embedded FPGAs (eFPGA) to be equal peers with processors through non-intrusive, bi-directionally cache-coherent integration. In contrast to existing CPU-FPGA hybrid systems in which the processors play a supportive role, Duet unleashes the full potential of both the processors and the eFPGAs with two classes of post-fabrication enhancements: *fine-grained acceleration*, which partitions an application into small tasks and offloads the frequently-invoked, compute-intensive ones onto various small accelerators, leveraging the processors to handle dynamic control flow and less accelerable tasks; *hardware augmentation*, which employs eFPGA-emulated hardware widgets to improve processor efficiency or mitigate software overheads in certain execution models.

An RTL-level implementation of Duet is developed to evaluate the architecture with high fidelity. Experiments using synthetic benchmarks show that Duet can reduce the processor-accelerator communication latency by up to 82% and increase the bandwidth by up to 9.5x. The RTL implementation is further evaluated with seven application benchmarks, achieving 1.5-24.9x speedup.

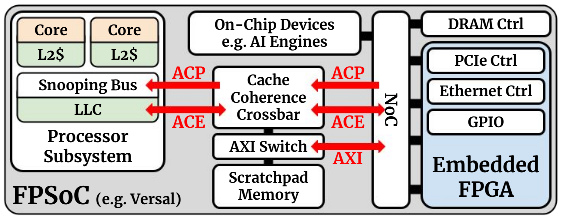
I. Introduction

The demise of Moore's Law [33] and the stagnation of processor performance growth [44] have given rise to hardware acceleration [12], [22], [34]. Since ASIC-based accelerators suffer from high non-recurring engineering costs and low versatility, field-programmable gate arrays (FPGAs) are gaining steam due to their post-fabrication, gate-level reconfigurability and close-to-ASIC performance [16], [20], [8]. FPGAs can be integrated either as standalone devices (Fig. 1a) or into field-programmable system-on-chips (FPSoC) (Fig. 1b). The conventional, FPGA-based acceleration paradigm is **coarse-grained acceleration** (Fig. 2b), which offloads an algorithm in its entirety onto the FPGA. Despite offering substantial improvements in performance and energy efficiency, coarse-grained acceleration is only practically applicable to limited algorithms that are both stable (to justify the accelerator design costs) and sufficiently large (to justify the control and data transfer overheads).

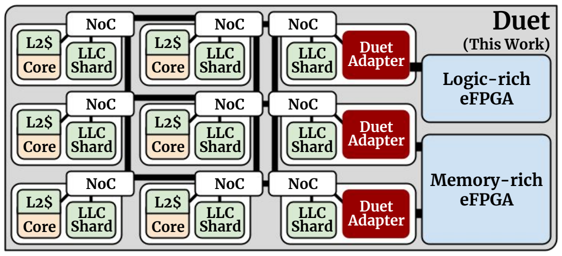
This work proposes **Duet** (Fig. 1c, Sec. II), a novel, cache-coherent, manycore-FPGA architecture that is tailored to enable two novel paradigms of hardware acceleration.



(a) Multicore/Manycore Processor and Standalone FPGA
L2$=Private cache; **NoC**=Network-on-chip; **LLC**=Last-level cache



(b) Field-Programmable System-on-Chip (FPSoC), e.g. Versal [19]
AXI [3]: non-coherent interconnect; ACP [3]: uni-directional coherent interconnect; ACE [3]: fully coherent interconnect.



(c) Duet (This Work)
Each Duet Adapter can be configured independently to support uni- or bi-directional coherence in addition to a memory-mapped non-coherent interface. Duet enables scalable integration of: 1) any number of processors; 2) multiple independent embedded FPGAs (eFPGAs); 3) multiple NoC access points per eFPGA.

Fig. 1: CPU-FPGA Systems

Fine-grained acceleration (Fig. 2c, Sec. III-A) partitions an algorithm into smaller tasks and offloads only the frequently-invoked, compute-intensive ones onto a variety of small accelerators. Processors still play a critical role by handling dynamic control flow, memory/IO-bound tasks, or any other less accelerable computations. For example, fine-grained accelerators can be used for special instructions like tangent, basic algorithms like sorting, or inner loop bodies like the compute payload per node/edge during a graph traversal. **Hardware**

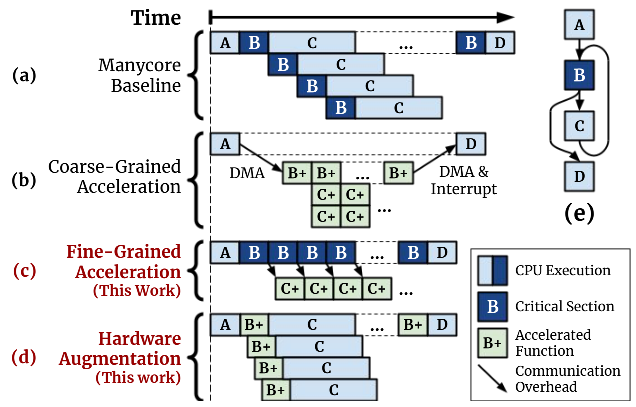


Fig. 2: Accelerating a Hypothetical Program
(a-c) Execution time of a manycore baseline and different acceleration paradigms; (d) An example of hardware augmentation in which the embedded FPGA emulates a lock-free task scheduler; (e) Control flow graph of the program.

augmentation (Fig. 2d, Sec. III-B) takes an application-agnostic approach — it employs FPGA-emulated hardware widgets to reduce processor idle time or mitigate software overhead in certain execution models. For example, hardware-implemented, lock-free data structures can reduce synchronization overhead in shared-memory, multi-thread programs; hardware task schedulers enable task-level parallelism with lower overhead than software schedulers.

Existing CPU-FPGA systems are inefficient for fine-grained acceleration and hardware augmentation due to two reasons: first, the centralized, bandwidth-optimized CPU-FPGA interconnect performs poorly when lots of cores and accelerators communicate via short, frequent messages; second, even on state-of-the-art FPSoCs which support bi-directional cache coherence between the processors and the FPGA, cacheline-granular memory sharing incurs non-trivial overhead. The overhead is justifiable for coarse-grained acceleration due to extremely high compute-to-memory ratio, but becomes critical for small, frequently-invoked, fine-grained accelerators.

Duet is tailored to the distinct architecture requirements posed by fine-grained acceleration and hardware augmentation. In essence, **Duet promotes eFPGAs to be equal peers with processors and integrates them as first-class citizens on the network-on-chip (NoC) through the novel, lightweight, Duet Adapters**. In particular, Duet contains the following novelties:

- **Scalability**: Duet enables tight integration of one to multiple eFPGAs of various resource compositions into a scalable manycore system. Each eFPGA may be connected to multiple Duet Adapters for higher aggregate memory bandwidth.
- **Hybrid Cache Coherence**: Duet adopts a hybrid scheme to coherently integrate the eFPGAs. Each Duet Adapter contains one to multiple private, local, hardware **Proxy Caches** (Sec. II-C) that translate the platform-dependent, cache coherence protocols into simple memory interfaces for the eFPGA. Furthermore, each Proxy Cache can be configured at eFPGA programming time to support an optional, bi-directionally coherent, soft cache built out of eFPGA resources. The use of soft caches improves accelerator performance by exploiting data locality while coherently sharing data with the processors.
- **Non-Intrusive Integration**: For fine-grained acceleration and hardware augmentation, a sizable amount of compute is still run on the processors and is critical to the overall performance. However, the direct participation of eFPGA-emulated, soft caches in cache coherence may slow down the cache system because eFPGAs run at a lower clock frequency and suffer from clock-domain-crossing overheads. Addressing this challenge, the Duet Adapters operate in the processors' (fast) clock domain and move FPGA-side coherence maintenance off of the critical path. As a result, the cache system performs equally fast as if each Duet Adapter was an additional processor-owned private cache.
- **Plug-and-Play Integration**: Duet requires little to no hardware changes to existing manycore systems because the Duet Adapters transduce between the eFPGAs and the NoC. Such decoupling is critical because modifying a mature processor design often leads to performance degradation or even hardware bugs, especially as design complexity and verification costs skyrocket on advanced technology nodes [23].

To evaluate Duet with high fidelity, we build Dolly, a prototype instance of Duet at the RTL level (Verilog & SystemVerilog). In addition, we assemble a full toolchain for software development and accelerator design, leveraging an array of open-source projects, including OpenPiton [6], BYOC [5], PRGA [31], Yosys [54], and VTR [38]. **To encourage further research in tightly-integrated, hardware cache-coherent CPU-eFPGA systems, we have open-sourced Dolly and its toolchain, available at https://github.com/PrincetonUniversity/Duet**.

Experiments show that the Duet Adapter introduces negligible hardware overhead for the CPU-eFPGA integration (Sec. V-B). The Proxy Cache reduces the latency of processor-accelerator communication up to 82% and increases the bandwidth up to 9.5x compared to having the eFPGA participating directly in the coherence protocol (Sec. V-C). The latency reduction and bandwidth increase are stable across different eFPGA clock frequencies. On selected benchmarks (Sec. V-D), Dolly improves the overall performance up to 24.9x in comparison to corresponding processor-only baselines and up to 4x in comparison to other CPU-FPGA systems.

The major contributions of this work are:

- Presenting Duet, a manycore-FPGA architecture which employs novel Duet Adapters to integrate embedded FPGAs in a scalable, non-intrusive, cache-coherent manner.
- Identifying and demonstrating with examples two novel paradigms of hardware acceleration enabled by Duet, namely fine-grained acceleration and hardware augmentation.
- Presenting Dolly, an RTL-level prototype of Duet and a full toolchain for software development and accelerator design.
- Evaluating Dolly's silicon area consumption and performance with synthetic and application benchmarks.
- Releasing Dolly and its toolchain for open-source access.

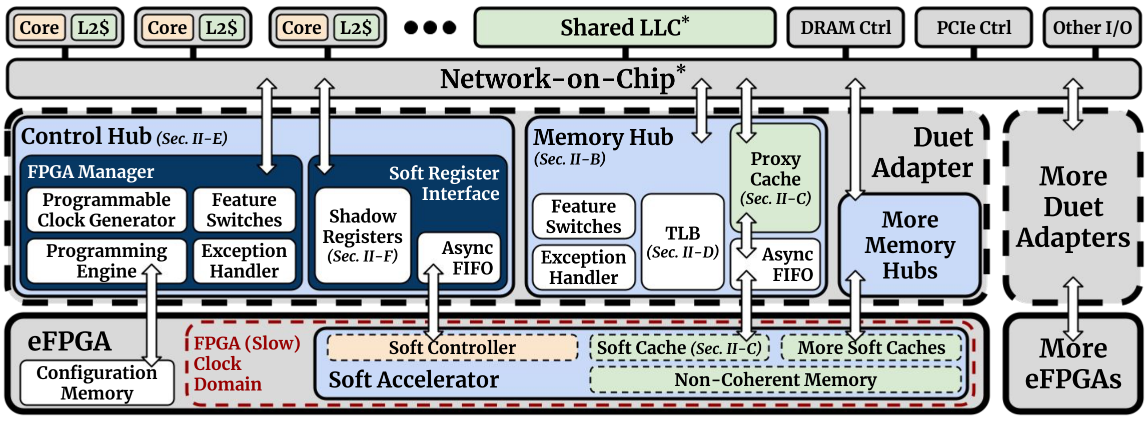


Fig. 3: Duet Architecture and an Emulated Soft Accelerator

* For simplicity, the figure shows a bus-based NoC and a centralized LLC. Duet can be adapted to other NoC topology and distributed LLC. For example, Dolly (Sec. IV) uses a 2D-mesh and a distributed LLC.

II. Architecture

Before diving into the architecture of Duet, we want to clarify several terms that are used throughout this paper: **software** refers to the program running on the processors; **hardware**, **hard** cache, etc. refer to the components that are fixed at fabrication time, in contrast to the **soft** components that are emulated with the reconfigurable resources in the eFPGAs. In addition, we use the term **soft accelerator** to refer to both fine-grained accelerators and hardware augmentation widgets.

A. Overview

Fig. 3 shows an overview of the Duet architecture. Duet integrates eFPGAs as first-class citizens on the NoC through novel, hardware **Duet Adapters**. Each Duet Adapter contains one to several **Memory Hubs** and one **Control Hub**. The Memory Hubs enable bi-directionally coherent memory accesses of the soft accelerators and transduce between the memory interfaces of the eFPGAs and the NoC. The Control Hubs present the eFPGAs as on-chip devices that are accessible via memory-mapped I/Os (MMIOs).

The fact that eFPGA-emulated soft accelerators typically run at a much slower clock than the rest of the system poses a challenge to minimizing processor-accelerator communication overhead. In particular, any traffic entering or leaving the slow clock domain must pay for the clock-domain-crossing (CDC) overhead, and every slow clock cycle significantly penalizes the total communication time (Fig. 5a & Fig. 6a). A few dozen cycles seem negligible, but since the accelerated tasks are short and invoked frequently, the accumulated overhead can chip away a large fraction of the effective speedup. Addressing this challenge, the Duet Adapter adopts various strategies to move the logic in the slow clock domain off of the critical path.

B. Memory Hub

Each Duet Adapter may contain multiple Memory Hubs, each attached to the NoC using an independent connection. The soft accelerator may use all or any subset of them to access the memory. Each Memory Hub consists of an exception handler, a set of feature switches, and a **Proxy Cache** (Sec. II-C), all implemented in hardware. Besides the hardware Proxy Cache, each memory hub can support one optional, bi-directionally coherent, **Soft Cache** built out of eFPGA resources. The exception handler as well as all the feature switches can be configured by the processors via on-chip MMIOs.

The exception handler employs timeout and parity checks to monitor eFPGA outputs. When an exception is detected, e.g., due to an RTL or software bug, it asserts an error code and deactivates all Memory Hubs in the same Duet Adapter. Once deactivated, the Memory Hubs stop accepting any memory requests from the eFPGA, but the Proxy Caches remain functional, so that any in-flight coherence messages are processed properly. This mechanism prevents accelerator bugs from halting the system at the micro-architecture level.

The feature switches allow the processors to configure the Memory Hubs according to the state of the eFPGA and the specifications of the soft accelerator. For example, the Memory Hubs should be deactivated during eFPGA reconfiguration; if soft caches are used, the Memory Hubs should be configured to forward invalidation requests into the eFPGA.

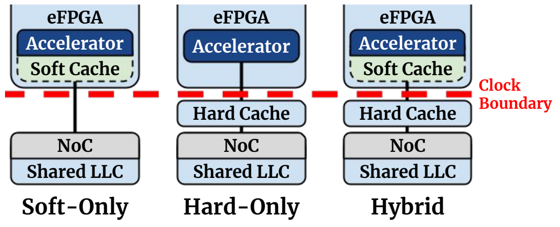
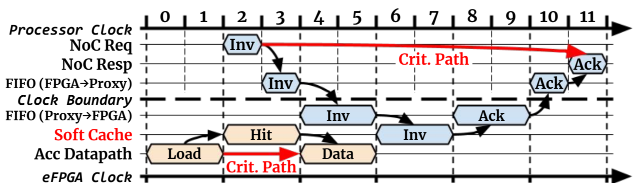


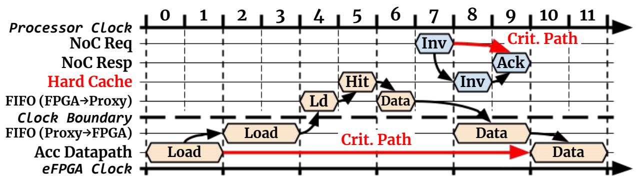
Fig. 4: FPGA-Side Cache Organization Options

C. Proxy Cache

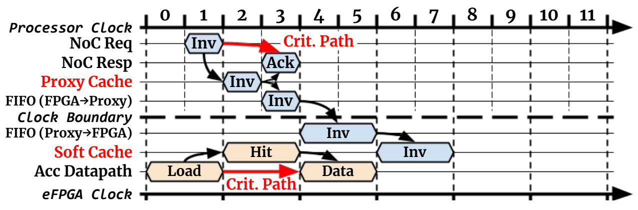
A key challenge in building coherently-integrated CPU-FPGA systems is how to organize the FPGA-side caches. Fig. 4 shows the options:



(a) Cache Operations with Only **Soft** Caches
Inv=Invalidation from a remote cache; **Ack**=Acknowledgement



(b) Cache Operations with Only **Hard** Caches



(c) Cache Operations with Duet (This Work)

Fig. 5: Cache Operations with Different Cache Organizations
For simplicity, this figure shows single-stage async FIFOs, and the FPGA clock runs at half of the system clock frequency. On real hardware, async FIFOs typically take two to four stages, and the FPGA clock could be as slow as 1/10 of the system clock.

Soft-Only The eFPGA is directly connected to the NoC. The soft cache, if used, must implement the system-wide cache coherence protocol.

Hard-Only A hardware, private cache is added between the eFPGA and the NoC. Cache coherence is hidden from the eFPGA, so soft caches are not supported.

Hybrid A hardware, private cache participates in the system-wide cache coherence on behalf of the eFPGA and supports the use of soft caches with a local cache coherence protocol.

Option 1, *soft-only*, has three main drawbacks. First, it becomes the accelerator developers' burden to design a cache in compliance with a platform-dependent, complex cache coherence protocol. The sophisticated control logic consumes more logic resources, has higher access latency, and is rarely reusable across devices — a soft cache designed for ACE [3] would hardly work with CHI [4]. Second, since the soft caches have access to the micro-architectural state (e.g., can block a NoC message indefinitely), the system cannot fully contain faulty or malicious behaviors of the soft caches. Third, as shown in Fig. 5a, the soft caches may slow down the hardware cache system due to CDC overheads and the slow cycles spent in the FPGA clock domain. Nonetheless, *soft-only* is the de facto design on most commodity FPSoCs because these FPGA-centric architectures are designed for coarse-grained acceleration. Specifically, coherent memory sharing is rare and mainly offered to simplify programming; soft cache IPs are licensable from the vendors; and accelerator bugs are expected to fail the entire system.

Option 2, *hard-only*, addresses all the drawbacks of the soft-only approach but has other limitations. First, the CDC overhead is now imposed on the accelerator datapaths (5b). Second, the cache implementation is fixed at fabrication time and may be suboptimal for certain accelerators.

Given the downsides of the soft-only and hard-only options, Duet takes a *hybrid* approach. A private, local, hardware **Proxy Cache** implements the platform-dependent, system-wide cache coherence protocol and provides a simple memory interface to the eFPGA. Each Proxy Cache can be configured through feature switches to support an eFPGA-emulated, soft cache which can be tightly integrated into the accelerator datapaths. Moreover, the Proxy Cache contains two key novelties when compared to a naïve implementation of the hybrid approach:

First, **the Proxy Cache neither requires nor accepts any acknowledgements from the soft cache**. As a result, the Proxy Cache always responds to coherence messages promptly (Fig. 5c), insulating the cache system from the slow eFPGA clock. To maintain coherence, the Proxy Cache requires the soft cache to be write-through but allows write buffering (the Proxy Cache itself can be write-back or write-through, depending on the coherence protocol of the LLC). Furthermore, because the asynchronous FIFOs deliver messages in order, the soft cache always receives invalidations, line fills, and write acks in the same order as they are sent by the Proxy Cache. Rigid proof of the correctness of such protocols is out of the scope of this paper, but the PCX protocol in OpenSPARC T1 [41] and its extension, the TRI protocol in BYOC [5], are two examples that meet the above requirements.

Second, the protocol is simple yet flexible. In the common case, the soft cache only needs to support two request types (`Load` and `Store`) and three response types (`LoadAck`, `StoreAck` and `Inv`alidation). It is up to the accelerator designer whether to use a write buffer, how many entries the write buffer has, and whether read-after-write forwarding is compatible with the consistency assumptions of the application. The Proxy Cache can be configured to work with either a write-allocate or a write-no-allocate soft cache, as well as be configured to enable atomic operations which require the soft cache to support incrementally more message types.

D. *Memory Protection and Virtualization*

Another key challenge for CPU-FPGA systems is memory protection and virtualization. The Proxy Cache is physically-indexed, physically-tagged because it is implemented in hardware and is closer to the LLC. However, it depends on the type of soft accelerator whether the eFPGA should use virtual or physical addresses. Hardware augmentation widgets may be trusted firmware and can be granted access to physical memory. On the contrary, application-specific, fine-grained accelerators are like user programs and can be faulty or malicious, so they are better restricted to virtual addresses.

To enable virtual memory accesses of the soft accelerator, Duet adds a translation look-aside buffer (TLB) to each

Memory Hub. The TLB can be disabled if the soft accelerator is granted access to the physical memory space; otherwise, accelerator-initiated memory accesses must be translated by the TLB while being speculatively processed by the Proxy Cache. On a page fault, the TLB sends an interrupt to a processor, then the kernel-level interrupt handler either updates the TLB using MMIOs or kills the accelerator if the page access is deemed invalid.

One special case is when soft caches are used but the accelerator only has access to virtual addresses, i.e., the soft caches are virtually-indexed, virtually-tagged. To enable reverse-mapping from physical address to virtual address when the Proxy Cache forwards an invalidation into the soft cache, the Proxy Cache stores the virtual page number beside the physical tag of each cacheline. This also rules out the coexistence of synonym aliases (different virtual addresses mapping to the same physical address) in the soft cache. In particular, the Proxy Cache can invalidate the existing virtual address before responding to a load request for the same physical cacheline through a different virtual address.

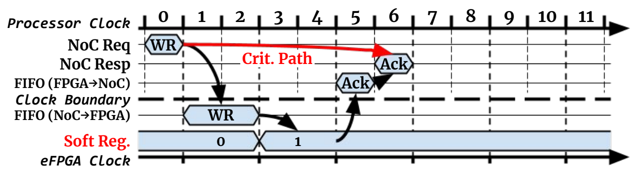
E. Control Hub

The Control Hub consists of two submodules: the **FPGA Manager** and the **Soft Register Interface**. The FPGA Manager provides necessary hardware support for programming and monitoring the eFPGA. The programming engine loads the bitstream into the configuration memory, and performs integrity checks to detect data corruption. The programmable clock generator either divides the system clock, or integrates a separate PLL for finer control over the generation of the FPGA clock. The exception handler and the feature switches are similar to those in the Memory Hubs. In particular, the feature switches can be used to set the timeout limit, reset the soft accelerator, or clear previously-logged error codes.

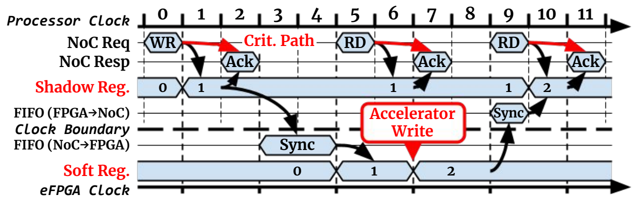
The Soft Register Interface enables the soft accelerator to implement a soft "device controller" similar to those commonly seen on off-chip peripheral devices. The soft registers emulated by the eFPGA are accessible via on-chip MMIOs and often have additional effects rather than simply holding a value. For example, reading a soft register may dequeue from a FIFO inside the accelerator so that repetitive reads return different values. When the Control Hub is deactivated, the Soft Register Interface returns bogus data to all processor accesses so that the system is not halted.

F. Shadow Registers

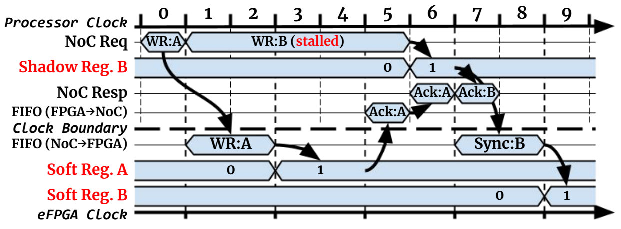
To prevent unwanted side effects, MMIOs typically adhere to a strict memory ordering model, e.g., I/O ordering. Unfortunately, as stressed in Sec. II-A, these strictly ordered accesses may stall the processors' pipelines because the eFPGA runs at a slower clock (Fig. 6a). Addressing this inefficiency, the Soft Register Interface is augmented with several types of **Shadow Registers** residing in the fast clock domain (Fig. 6b). When a processor writes to a shadowed soft register, the Shadow Register acknowledges the request before forwarding the write into the eFPGA. Conversely, the soft accelerator actively



(a) Accesses to a Normal Soft Register



(b) Accesses to a Shadowed Soft Register (This Work)



(c) Strict Ordering of Two Soft Register Writes (This Work)

Fig. 6: Accessing Soft Registers and Shadow Registers

synchronizes shadowed soft registers over the asynchronous FIFO, so that the Shadow Registers can respond to processor reads immediately without notifying the eFPGA.

Duet supports four types of Shadow Registers: *plain*, *FPGA-bound FIFO*, *CPU-bound FIFO*, and *token FIFO*. Plain Shadow Registers only keep the last value of multiple writes, which are ideal for passing constant parameters. FPGA-bound and CPU-bound FIFOs, as their names suggest, record all writes from one side and allow the other side to read in order. CPU-bound FIFO is *blocking*, i.e., a processor read is stalled until the soft accelerator pushes into the FIFO or the request times out. In contrast, CPU-bound *token* FIFO is a dataless, *non-blocking* FIFO that consumes a token or returns *"empty"* in response to a processor read. Token FIFO is designed particularly to emulate the non-blocking `try_join` semantic in parallel programming models.

Note that normal soft registers are still available in case the software requires *non-bufferable* accesses that must be sent all the way to the endpoint. For example, a soft register can be dedicated as a barrier for synchronizing the processor and the eFPGA. The processor signals its arrival at the barrier by reading the soft register, while the eFPGA signals its arrival at the barrier by acknowledging the read. To maintain I/O ordering, Shadow Register accesses are processed and responded to in order with respect to other shadowed or normal register accesses, as shown in Fig. 6c.

III. Applications

Duet enables two novel paradigms of hardware acceleration, namely fine-grained acceleration and hardware augmentation. In this section, we illustrate both paradigms with examples and discuss why Duet is the ideal architecture for them.

Listing 1: The key compute loop of the Barnes-Hut simulation algorithm [7]. The highlighted functions are static, compute-intensive, and ideal for fine-grained acceleration.

```
// Calculate the net force on particle "p"
void CalculateNetForce (BHTreeNode node, Particle p):
  if (Distance (node, p) > node.radius * THRESHOLD)
    // "p" is far enough from the particles in the subtree of
    // "node", so we approximate the net force with a low-order
    // multipole expansion and stop traversing the subtree
    p.force += ApproxForce (node, p);
  else if (IsLeafNode (node))
    for (Particle q : node.children)
      p.force += CalcForce (q, p);
  else
    // traverse the subtree via recursive calls
    for (BHTreeNode child : node.children)
      CalculateNetForce (child, p);

// Calculate the net forces on all particles in parallel
void CalculateNetForceOnAllParticles (
    BHTreeNode root, vector<Particle> allParticles):
  parallel_for (Particle p : allParticles)
    CalculateNetForce (root, p);
```

A. Fine-Grained Acceleration

1) Overview: At its core, fine-grained acceleration and coarse-grained acceleration are similar because they both offload computation onto application-specific accelerators. In fact, there is a spectrum rather than a clear boundary between the two paradigms — the shorter the accelerated function is, or the more frequent the accelerator interacts with the processors, the closer it is to the fine-grained end of the spectrum.

Fine-grained acceleration has the following advantages: first, it fully utilizes the compute power of the processors by running the less accelerable fractions of an algorithm on them, for example dynamic control flow, memory/IO-bound tasks, managing complex data structures, etc.; second, it reduces software and accelerator design costs by moving less logic onto the FPGA; third, in comparison to a monolithic accelerator, a collection of small accelerators are more composable, reusable, and more resilient to software updates.

2) Example: Barnes-Hut (BH) algorithm [7] is an approximation algorithm for simulating a dynamic system of particles interacting via certain physical forces such as gravity. In each time step, the simulation volume is divided into hierarchical, cubic cells stored in an octree (for three-dimensional space), so that the total force exerted by particles in distant cells can be approximated through a center-of-mass abstraction or a low-order multipole expansion. Listing 1 shows the key functions that calculate the net force on all particles.

Prior work [17] has proposed an FPGA-based, coarse-grained BH accelerator which contains replicated pipelines to parallelize force calculation on multiple particles. However, despite the use of a special *traversal cache* that facilitates data reuse between the pipelines, control flow divergence often leads to low pipeline utilization. Furthermore, the accelerator relies on the host processor to serialize the BH-tree and preload the node pointers into the *traversal cache*, making it inapplicable if the dataset exceeds the FPGA's BRAM capacity. In summary, building a coarse-grained BH accelerator can be an intimidating task without a satisfactory result.

Fig. 7 shows the timeline of BH running on a dual-

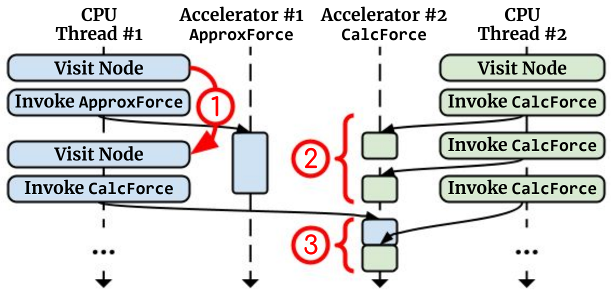


Fig. 7: Multi-Threaded BH with Fine-grained Acceleration ① Loop-carry dependencies and dynamic control flow are handled by the processors; ② Processors and accelerators run in parallel through software pipelining; ③ Accelerators are time-multiplexed by multiple CPU threads to increase utilization.

core system with two fine-grained accelerators. Besides being easy to program, this software-hardware co-design is scalable and flexible: multiple accelerators of each type can be instantiated to increase parallelism; if THRESHOLD is high, i.e., ApproxForce is called for less times, the system can allocate more eFPGA resources to implement more and/or faster CalcForce accelerators, and vice versa if THRESHOLD is low.

The fine-grained BH accelerators would be inefficient without Duet's architectural support. First, the accelerators access just a few cachelines at **random** memory locations per execution, resulting in poor utilization of the block-oriented, bandwidth-optimized memory system in most existing CPU-FPGA architectures. In contrast, Duet's hybrid cache organization minimizes accelerators' memory access latency, especially for random accesses at cacheline granularity. Second, the accelerators are invoked frequently and time-multiplexed by multiple CPU threads. A centralized CPU-FPGA interconnect may be congested by the MMIOs issued by the processors to invoke the accelerators. Duet's scalable integration makes it possible to split the workload across multiple eFPGAs, and the shadow registers can further minimize MMIO latency.

B. Hardware Augmentation

1) Overview: Hardware augmentation allows application developers to add various hardware widgets to the system after chip fabrication. These widgets help improve processor efficiency and/or mitigate software overheads in certain execution models, providing application-agnostic acceleration. Different hardware augmentation widgets often exhibit unique behaviors and provide distinct software APIs. For example, decoupled access-execute engines [49] hide the latency of memory redirects by dereferencing software-specified pointers and collect the data into a hardware queue; hardware garbage collectors [32] run autonomously in the background and alleviate software overhead of the runtime environment. Since hardware augmentation widgets collaborate closely with the processors, Duet's scalable, tight, cache-coherent integration is critical to maximize performance.

2) Example: Discrete event simulation (DES) is a well-known challenge for parallel execution [18]. All the simu-

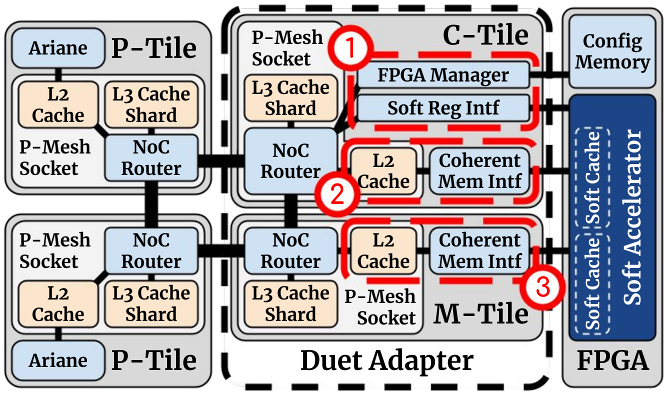


Fig. 8: Architecture of Dolly-P2M2
Dolly-P2M2 has 2 processors, 1 eFPGA, and 2 Memory Hubs. P-tile, C-tile and M-tile are physical tiles in a 2D mesh network. P-Mesh Socket is a physical wrapper for common components in all physical tiles, including an L2 cache, a NoC router, and a shard of the shared L3 cache. ① is the Control hub (Sec. II-E). ② and ③ are two Memory Hubs (Sec. II-B). Note that ① and ② reside in the same physical C-tile.

lating threads must either progress in lock step (conservative method), which suffers from high synchronization overheads, or exploit speculative execution (optimistic method), which requires a carefully designed software runtime or an architecture that supports thread-level speculation, e.g., SWARM [27].

Duet opens up another possibility through hardware augmentation: an eFPGA-emulated task scheduler can be loaded as part of the simulator software while offering hardware-level performance. Consider a possible implementation of such a task scheduler. Processors schedule new events by pushing memory pointers to the events into a *FPGA-bound FIFO*, after which the task scheduler fetches the event data from shared memory and adds the pointer into the proper event queue. Once certain events are ready to be processed, the task scheduler pushes the pointers into an *CPU-bound FIFO* so that the processors can streamline event processing with minimal communication latency with the scheduler. The task scheduler can support task speculation by fetching the cache-lines that may be modified by a speculative event and storing versioned copies of them in its non-coherent memory. On a mis-speculation, the task scheduler rolls back the cachelines to the most up-to-date, non-speculative versions, then reschedules the mis-speculated events.

IV. Dolly

To evaluate Duet with high fidelity, we built Dolly[1], a prototype system at the RTL level (Verilog/SystemVerilog) with a full toolchain for software development and accelerator design. Dolly is configurable in the number of processors, the logic capacity of the eFPGA, the number of memory hubs, etc. For simplicity, we name each Dolly instance PpMm, where p specifies the number of processors, and m specifies the number of memory hubs available to the eFPGA. For example, Fig. 8 shows the architecture of Dolly-P2M2. **To encourage further research in tightly-integrated, hardware cache-coherent CPU-eFPGA systems, we have open-sourced Dolly and its toolchain, available at https://github.com/PrincetonUniversity/Duet.**

Dolly is based on the OpenPiton P-Mesh cache coherence system [6] in a 2D mesh configuration. The cache system is organized in three levels: the L1 caches are tightly interwoven into the processors; one private, write-back, 8KB L2 cache per physical tile interfaces with the corresponding L1 cache through the transaction-response interface (TRI) [5]; the shared L3 cache is distributed among all physical tiles, 64KB per shard, and runs a directory-based MESI protocol together with the private L2 caches. The NoC offers point-to-point ordering of message delivery and supports additional message types besides the coherence messages, enabling on-chip MMIOs required by Dolly.

Dolly contains three types of physical tiles and an eFPGA. Each *P-Tile* hosts an Ariane [57] processor, a 6-stage, single-issue, in-order CPU which implements the 64-bit RISC-V instruction set. Each Ariane processor has a private hardware FPU, an 8KB L1 instruction (L1I) cache, and an 8KB L1 data (L1D) cache. *C-Tiles* and *M-Tiles* compose the Duet Adapter: each *C-Tile* contains one Control Hub (Sec. II-E) and one Memory Hub (Sec. II-B); each *M-Tile* includes one Memory Hub. To demonstrate Duet's adaptability, Dolly implements the Proxy Cache by adding a *coherent memory interface* to the *unmodified* P-Mesh L2 cache. The L2 cache, L3 cache shard and NoC router are the same across all physical tile types, so we wrap them into one module, the *P-Mesh socket*.

The eFPGA is built with PRGA [31] and employs a standard island-style architecture composed of logic blocks and routing blocks. The eFPGA includes all of the common logic resources available on modern FPGAs, including look-up tables (LUT), bypassable flip-flops, hard adder chains, Block RAMs (BRAM), and hard multipliers. Dolly employs a non-synthesizable clock generator to generate the eFPGA clock whose frequency can be specified in software. All the asynchronous FIFOs are implemented with dual-clock RAMs and Gray-coded, 2-stage synchronizers.

V. Evaluation

A. Overview

We conduct our evaluation of Duet through RTL simulations of Dolly instances and focus on three perspectives: first of all, we report the area consumption and the typical frequency of the hard components of Dolly (Sec. V-B); second, we run synthetic benchmarks to measure the CPU-FPGA communication latency and bandwidth achieved by the Duet Adapter under different conditions (Sec. V-C); third, we run seven application benchmarks and compare their performance against processor-only and FPSoC baselines (Sec. V-D).

To show the lower bound of Duet-enabled improvements, we give the processors various advantages throughout our evaluation. Specifically, we boost the clock frequency of the

[1]Named after *Dolly Suite, Op. 56*, a collection of pieces for piano duet composed by Gabriel Fauré.

TABLE I: Area and Typical Frequency of Dolly Components

Component	Device Technology	Area (mm^2)	Freq. (MHz)	**Scaled Area*** **(mm^2)**	**Scaled Freq.*** **(MHz)**
Ariane [58]	GlobalFoundries 22nm FDX	0.39	910	**1.56**	**455**†
P-Mesh Socket [6]	IBM 32nm SOI	0.55	1000	**1.1***	**711***†
FPGA Mgr + Soft Reg Intf	FreePDK45	0.21	925	**0.21***	**925***
Coherent Memory Intf		0.04	1250	**0.04***	**1250***

* Scaled to 45nm with a linear MOSFET scaling model
† To emulate a higher-performance multi-core, our evaluation simulates the Ariane cores and the hardware cache system at 1GHz

Ariane processors and the P-Mesh cache system to 1GHz, despite that previous works reported their maximum frequencies (after scaling to 45nm) to be 455MHz and 711MHz, respectively. For the application benchmarks, processor-only baselines always start with a warm cache, while the soft accelerators always start with a cold cache.

B. Area and Typical Frequency of the Hard Components

Table I summarizes the area and typical frequency of each hardware component of Dolly. The numbers for Ariane and P-Mesh components are retrieved from previous works. The submodules of the Control Hub and the Memory Hub are synthesized using Synopsys Design Compiler and Silvaco's Open-Cell Library [48] based on the NCSU FreePDK45 process design kit [50] using an area utilization rate of 70%. In summary, the Control Hub and the Memory Hub require minimal hardware resources. The area consumption of the eFPGA and the clock frequency of the emulated accelerator are reported on a per-benchmark basis in Sec. V-D.

C. CPU-eFPGA Communication Latency and Bandwidth

In this section, we evaluate the peak performance of Duet with a synthetic benchmark. The eFPGA emulates a simple scratchpad memory and a processor uses different mechanisms to access it. To show that the eFPGA's clock frequency has a significant impact on performance, we sweep the eFPGA clock from 20MHz to 500MHz while fixing the system clock at 1GHz. In particular, we study the following CPU-eFPGA communication mechanisms and show how Duet improves them over existing CPU-FPGA systems:

Soft registers are memory-mapped, eFPGA-emulated registers which the processor can read and write via non-coherent MMIOs. These soft registers can be implemented inside the eFPGA (**Normal Registers**), in which case processor accesses must pay the Clock-Domain-Crossing (CDC) overhead. **Shadow Registers** address this inefficiency. In this study, we use FPGA-bound FIFOs to handle processor writes and CPU-bound FIFOs for processor reads.

Alternatively, the processor can store the data in shared memory and pass the memory address to the eFPGA via a soft register write. The soft register write not only signals the eFPGA to load the data (**eFPGA Pull**), but also works as a synchronization to ensure that all processor stores are committed into the cache system before the eFPGA starts

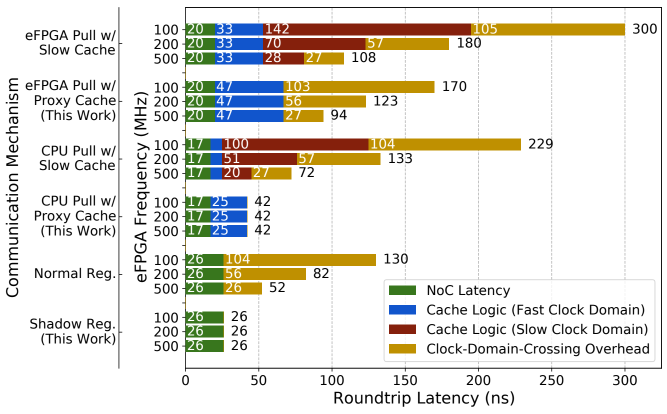


Fig. 9: CPU-eFPGA Communication Latency
(Single processor; Single transaction; Lower is better)

loading. The eFPGA can send data to the processor in a similar way (**CPU Pull**). As described in Sec. II-C, commodity FPSoCs typically emulate FPGA-side caches using eFPGA resources (**Slow Cache**), while Duet employs the novel **Proxy Cache** to improve cache performance.

Latency Study

We first measure the minimum round-trip latency of the aforementioned communication mechanisms on Dolly-P1M1. Fig. 9 shows the breakdown of the CPU-eFPGA communication latency into four parts: NoC latency, cache processing time in the fast clock domain, cache processing time in the slow clock domain, and the CDC overhead. All numbers are collected in an ideal scenario, that is, single processor (no contention or NoC congestion), single transaction (no buffer clogging). Note that the eFPGA pulls and CPU pulls are guaranteed to miss in the requesting cache and to hit in the other party's private cache in a modified state. Both the NoC latency and the cache processing time in the fast clock domain include the time for the distributed directory to send and process the secondary write-back requests.

As we can see, the CDC overhead and the slow clock penalty on cache processing constitute over half of the round-trip latency for a slow cache, even when the eFPGA runs at 50% of the CPU clock frequency. For eFPGA pulls, the Proxy Cache can reduce the latency by 13% to 43%, and the reduction increases as the eFPGA runs slower. For CPU pulls, the Proxy Cache achieves a constant latency regardless of the eFPGA clock frequency, reducing the latency by 42% to 82%. The Shadow Registers also have a fixed latency, reducing CPU-eFPGA communication latency by 50% to 80%. Note that the Proxy Cache and the Shadow Registers only move the eFPGA off of the critical path, and the eFPGA still needs time to issue memory accesses and to handle soft register accesses.

Single-Processor Bandwidth Study

Next, we study the maximum bandwidth of single-processor communications on Dolly-P1M1. In this study, we configure the synthetic benchmark to pass 512 quad-word (8 Bytes) integers from one processor to the eFPGA and then fetch them back. With the soft registers, this is done by having the processor execute a loop that writes/reads one integer

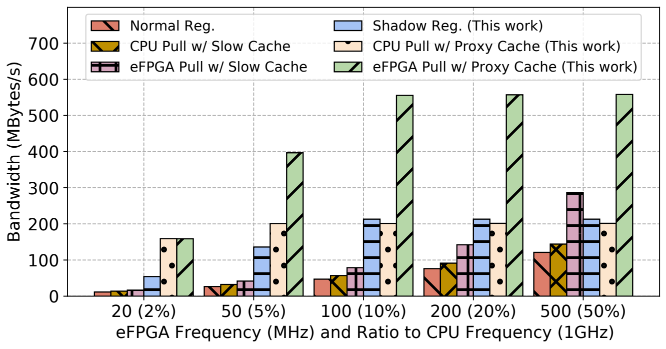


Fig. 10: Processor-eFPGA Communication Bandwidth vs. eFPGA Clock Frequency (Single processor; Higher is better)

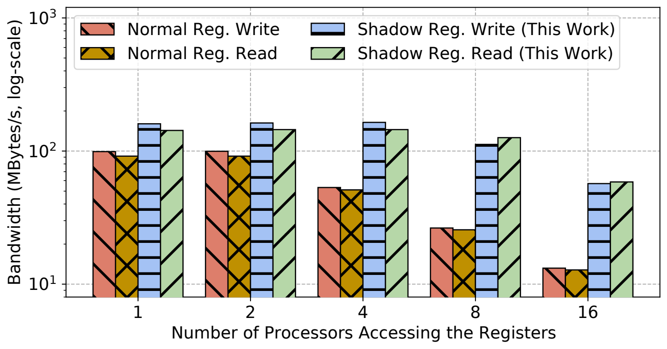


Fig. 11: Processor-eFPGA Communication Bandwidth (Per Processor) vs. Number of Contending Processors

per iteration. When using shared memory, the processor first allocates two 4KB buffers in the memory and passes the base addresses to the eFPGA using two plain shadow registers. Then, the processor stores all the integers into one of the two buffers and sends a *read* request to another normal soft register, awakening the eFPGA and blocking itself. The eFPGA proactively loads the entire array into its scratchpad memory, stores it back to the other buffer, and then unblocks the processor by acknowledging the processor's read request. Finally, the processor loads the array from shared memory.

Fig. 10 shows the results of this study. The Proxy Cache delivers the highest bandwidth across all eFPGA frequencies. In fact, the Proxy Cache reaches the peak bandwidth for eFPGA pulls (558MB/s) when the eFPGA runs faster than 100MHz (10% of the CPU clock frequency); for CPU pulls, the peak bandwidth (201MB/s) is reached at 50MHz. In comparison, the slow cache can only achieve 287MB/s for eFPGA pulls and 144MB/s for CPU pulls even when the eFPGA runs at 500MHz. The largest bandwidth gap is 9.5x when the eFPGA runs at 100MHz. The upper bounds of cache-based communications are determined by the NoC bandwidth and the number of concurrent, in-flight, memory requests supported by the Proxy Cache. Note that the upper bounds of CPU pulls are much lower than those of eFPGA pulls. This is because the cache line size is 16 Bytes and the eFPGA can load up to one line per cycle, but the L2 cache in Dolly only supports stores up to 8 Bytes, so the eFPGA must send two requests to store one cacheline, under-utilizing the asynchronous FIFOs.

The Shadow Registers also achieve a stable bandwidth (213MB/s) once the eFPGA clock frequency exceeds 10% of the CPU clock frequency. In comparison, normal registers can only reach 121MB/s at 500MHz. The upper bound of soft register accesses is limited by the strict consistency model of MMIOs. If the I/O consistency model is relaxed, the upper bound is then limited by the number of concurrent, in-flight MMIOs supported by the load-store queue of the processor.

Multi-Processor Scalability Study

One key difference between Duet and existing CPU-FPGA systems is the scalability. At the architecture level, Duet has no constraint on the number of processors, eFPGA fabrics, or the number of Memory Hubs per eFPGA. In this section, we study the effect of multi-processor contention on the soft registers. In particular, we run the synthetic benchmark on multiple Dolly instances with different number of processors which constantly writes/reads the same soft register. The eFPGA clock is fixed at 500MHz (50% of the CPU clock frequency).

Fig. 11 shows the results. The Shadow Registers can stably support up to 8 processors before the per-processor bandwidth starts to drop, while the normal registers can only support up to 2 processors. Considering that the processors need to perform other tasks between soft register accesses, the Shadow Registers can support even more processors in real applications.

Summary

In summary, the significant reduction in latency and the stable increase in bandwidth clearly justify the promotion of soft registers and private caches into the fast clock domain.

D. Application Benchmarks

In this section, we evaluate Duet with seven application benchmarks. For each benchmark, we first run a C implementation as the **processor-only** baseline and record the total runtime. Note that all benchmarks are run in bare metal due to limited simulation speed at the RTL level. We then design the soft accelerators for each benchmark, either directly at the RTL level, or by synthesizing an alternative C implementation with Catapult HLS [47]. Each accelerator is synthesized, placed and routed with the PRGA [31] workflow to get the accurate eFPGA clock frequency and an area estimation of the utilized FPGA resources. In particular, we map each benchmark onto the flagship FPGA model provided by VTR [38] (`k6_frac_N10_frac_chain_mem32K_40nm`) which resembles an Altera Stratix IV FPGA. With the accurate eFPGA clock frequency, we rerun the benchmarks on various **Dolly** instances and an **FPSoC**-like architecture, recording the total runtime which includes all the overhead in CPU-FPGA communication and synchronization. In particular, the FPSoC model moves the P-Mesh L2 cache into the eFPGA's (slow) clock domain and downgrades all shadowed soft registers to normal registers. We then compute the speedup over processor-only baselines and compare the results achieved by Dolly and the FPSoC model.

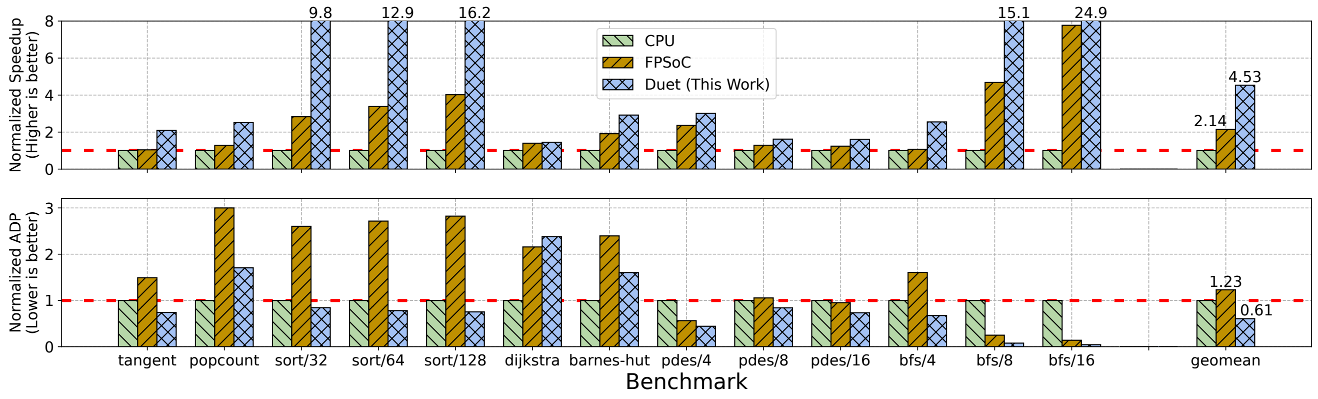


Fig. 12: Normalized Speedup and ADP of Application Benchmarks

`sort/N` indicates the array size of the sorting accelerator; `pdes/N` and `bfs/N` indicates the number of cores sharing the accelerator.

We use Area-Delay-Product (ADP) to quantify and compare area efficiency. When calculating area consumption, the processor-only baseline only counts the processors and the hardware cache system. The FPSoC-like architecture adds the silicon area of the FPGA on top of the processor-only baseline, and Dolly further includes the Duet Adapters.

The seven benchmarks, their corresponding Dolly instance, and the acceleration paradigm (**FG** for fine-grained acceleration, **HA** for hardware augmentation) are the following:

- **`Tangent`** (P1M0, FG): A floating-point Tangent accelerator is implemented with Catapult HLS using a piece-wise linear approximation algorithm with a maximum error rate of 0.3% compared to the C math library (`libm`). An FPGA-bound FIFO is used to pass the argument to the accelerator and invoke it. Results are returned through an CPU-bound FIFO.
- **`Popcount`** (P1M1, FG): Popcount counts the number of ones in a long bit vector (512 bits). Since the Ariane processor does not support the RISC-V BitManip Extension, we use a byte look-up algorithm for the processor-only baseline. The accelerator is hand-written in Verilog and uses one Memory Hub to load the bit vector from coherent memory.
- **`Sort`** (P1M2, FG): We use the SPIRAL Project [60] to generate 3 sorting networks in Verilog for sorting 32, 64, 128 double-word (4-Byte) integers. The accelerator uses two memory hubs, one for reading the input array from coherent memory and one for writing the sorted array back, so that the accelerator can be pipelined to sort fixed-length slices of a larger array which can then be merge-sorted by the processor. The processor-only baseline runs quicksort on the entire array.
- **`Dijkstra`** (P1M1, FG): We implement an accelerator for Dijkstra's Shortest Path algorithm with Catapult HLS and use a soft cache to exploit data locality between consecutive calls to the accelerator.
- **`Barnes-Hut`** (P4M1, FG): As explained in Sec. III-A2, the two key kernels in the Barnes-Hut algorithm are implemented as soft accelerators using Catapult HLS. Both accelerators are pipelined and time-multiplexed by four processors. The processor-only baseline also parallelizes force calculation for different particles across four processors.
- **`PDES`** (P4M1/P8M1/P16M1, HA): As described in Sec. III-B2, a non-speculative, hardware task scheduler is designed in Verilog to accelerate Parallel Discrete Event Simulation (PDES) of a digital circuit. The processor-only baseline uses MCS locks [35] to arbitrate accesses to the shared event queue, and the lock contention can be severe as the number of cores increases.
- **`BFS`** (P4M0/P8M0/P16M0, HA): We implement multiple hardware, lock-free queues in Verilog to alleviate the synchronization overhead in parallel Breadth-First Search (BFS). The processors traverse the graph in barrier-synchronized steps and use the queues to store the current and next search frontiers. Similar to the PDES benchmark, the processor-only baseline suffers from synchronization bottlenecks.

TABLE II: Clock Frequency and Area of Soft Accelerators

Benchmark	Max. Freq (MHz)	Norm. Area *	Resource Utilization	
			CLB †	BRAM
Ariane + P-Mesh Socket	1000	1.0	-	-
Tangent	282	0.47	0.84	0
Popcount	189	2.77	0.83	0.56
Sort (32)	228	6.29	0.30	0.76
Sort (64)	234	8.10	0.27	0.92
Sort (128)	228	10.27	0.27	0.92
Dijkstra	127	1.94	0.96	0.31
Barnes-Hut	85	14.22	0.99	0.05
BFS	208	1.24	0.61	0.75
PDES	126	2.77	0.47	0.56

* Total silicon area needed by the eFPGA to implement the accelerator, normalized to 1x Ariane + 1x P-Mesh Socket

† *Configurable Logic Block*, including LUTs and flip-flops

Table. II lists the maximum clock frequency and eFPGA resource utilization of the soft accelerators. Note that the soft accelerators can only run at 8% - 28% of the processors' clock frequency. As studied in Sec. V-C, Duet achieves peak bandwidth in this frequency range and outperforms the conventional FPSoC communication mechanisms by at least 2x.

Fig. 12 shows the normalized speedup and ADP of the application benchmarks. Duet outperforms FPSoC across all benchmarks and achieves a geometric mean of 4.53x speedup over processor-only baselines, in comparison to 2.14x achieved by

FPSoC using the same accelerators. The sorting accelerators provide up to 16.2x speedup on Duet but only 4.0x on FPSoC, because the FPGA-side caches in FPSoC are penalized by the slow clock cycles. Some may argue that the FPGA-side caches are unnecessary because sorted arrays are consecutive blocks for which DMAs are efficient. However, due to the small array sizes (128-512B), even LLC-coherent DMA incurs too much control overhead in cache flushing. The lock-free queues used in BFS provide up to 24.9x speedup on Duet but only 7.8x on FPSoC, because the processor-accelerator communication latency on FPSoC is much higher without the shadow registers. Note that the BFS benchmark shows a superlinear speedup when scaling from a 4-core system (Dolly-P4M0) to an 8-core system (Dolly-P8M0). This is because the processor-only baseline sees a performance decrease when scaling from 4 cores to 8 cores due to intensifying lock contention.

In terms of area efficiency, Duet achieves a geometric mean of 0.39x lower ADP than the processor-only baseline (lower is better), while the FPSoC is 0.23x higher. Duet outperforms FPSoC on most benchmarks except for `Dijkstra`. This is because the FPSoC model hardens the FPGA-side cache in the slow clock domain, so that soft caches become unnecessary and can be removed to save eFPGA resources.

In summary, Dolly achieves superior speedup with the same eFPGA-emulated accelerators than FPSoCs. Therefore, we can conclude that Duet provides better support for fine-grained acceleration and hardware augmentation.

VI. Related Work

A. Standalone FPGA-Based Accelerators

FPGAs are being used in a fast-growing range of application domains due to their programmability and close-to-ASIC performance. On a smaller scale, standalone FPGAs are used to accelerate database queries [51], image processing [14], network filtering [52], and most popularly, deep learning applications [21], [46], [59]. On the other end of the spectrum, FPGAs are massively deployed in the Cloud, either offered directly as an on-demand computing service such as the Amazon AWS F1 instances [2], or used to build cloud-scale accelerators such as Microsoft Catapult [10] and BrainWave [15]. In these systems, the FPGAs are often integrated as PCIe peripherals or separate machines on the datacenter network. As explained in Sec. I, such systems excel in accelerating at the application level, but are insufficient for fine-grained acceleration due to high CPU-FPGA communication overhead.

B. Same-Package CPU-FPGA Hybrids and FPSoCs

Addressing the CPU-FPGA communication bottleneck, academia and industry have explored tighter integration of the two. Intel Harp [13] combines an Intel Xeon multi-core CPU and an Altera FPGA in one package, and bridges the two with QPI, a point-to-point interconnect with cache coherence support. Field-Programmable System-on-Chips (FPSoC) achieves even tighter integration by integrating processors and FPGAs on the same chip, similar to Duet at a high level. Many commodity FPSoCs [26], [36], [37], [42], [55] and academic FPSoCs [28], [45], [53] support full or partial cache coherence. For example, the Xilinx Zynq-7000 employs the AXI4 ACP interface [3] which supports uni-directional cache coherence (I/O coherency). The Xilinx Zynq UltraScale+ MPSoC [56] and Versal ACAP [19] provide full coherence support with the ACE protocol [3].

There are two key differences between Duet and FPSoCs. First, as shown in Fig. 1b, the processor subsystem and the eFPGA in FPSoCs are separated by a centralized interconnect, and the hardware cache hierarchy resides in the processor subsystem. Such organization is reasonable for an FPGA-centric, dual-core architecture, but cannot scale to a larger number of cores. Second, in the absence of FPGA-side hardware caches, it becomes the accelerator designers' burden to design and verify soft caches that comply with the coherence protocol, for example ACE [3]. Besides increasing design complexity, the soft caches' direct participation in cache coherence slows down the entire cache system because they run in the slow clock domain. Moreover, since the soft caches have access to the micro-architecture states of the NoC, the system cannot restrain faulty or malicious behaviors of the soft caches.

C. Near- and In-Processor Reconfigurable Computing

Dating back to the early days of reconfigurable computing, computer architects have proposed to integrate Reconfigurable Fabrics (RF) very close to or even into processors. Garp [9] places an RF between a processor and its private cache, making the two compute units share the entire memory system. Chimaera [24] and PRISC [43] embed Reconfigurable Functional Units (RFUs) directly into a processor's datapath, enabling post-fabrication customization of the Instruction Set Architecture (ISA). The Post-Fabrication Microarchitecture [29] couples an RF with a superscalar core and allows the RF to observe and microarchitecturally intervene at key pipeline stages. These systems have their advantages but conflict with our *plug-and-play integration* goal as they mandate the redesign of the processors.

D. Coherence Protocols for Accelerators

Hardware cache coherence has been widely adopted in multi-processor systems as they provide not only programmability but also performance. In the context of hardware acceleration, recent studies have also shown benefits of employing hardware cache coherence and have proposed a variety of solutions. FUSION [30] is a hierarchical, timestamp-based coherence protocol emphasizing data transfer between accelerators; Crossing Guard [39] proposes the use of a hard cache transducer to decouple the accelerator coherence protocol and the processor (host) protocol, as well as to incorporate fault-tolerance and memory translation; Spandex [1] offers flexible write strategy and granularity, adapting to highly heterogeneous architectures. While all of these proposals achieve hardware cache coherence, they are not optimized for FPGA-based accelerators in that they all require direct participation of the accelerators. However, through the use of the Proxy Cache, Duet can be adapted to these coherence protocols, pretending to be a "fast" accelerator to the NoC and hiding the slow clock domain from the coherence protocol.

Accelerator coherence standards and interconnects are also emerging in industry, e.g., CCIX [11], OpenCAPI [40], and CXL [25]. These proposals target the board level or above and are really optimized for coarse-grained acceleration.

VII. Conclusion

In this work, we present Duet, a scalable, manycore-FPGA architecture with non-intrusive, coherently-integrated, embedded FPGAs that is optimized for *fine-grained acceleration* and *hardware augmentation* in the broad general-purpose domains. By promoting the eFPGA to a first-class citizen on chip, Duet enables the eFPGA to access the NoC and to participate in bi-directional cache coherence just as any other processor. The novel, lightweight, Duet Adapters reduce critical communication overheads by placing the Proxy Caches and Shadow Registers in the faster processor clock domain. Our evaluation shows that Dolly, an RTL-level implementation of Duet, can reduce the latency of processor-accelerator communications by up to 82% and increase the bandwidth by up to 9.5x, stably across a wide range of eFPGA clock frequencies. Selected benchmarks leveraging Duet-enabled soft accelerators show up to 24.9x speedup over processor-only baselines and up to 4x over FPSoCs. Dolly and its toolchain have been open-sourced and available at https://github.com/PrincetonUniversity/Duet.

Acknowledgements

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7852. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1823222 and the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2039656. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] J. Alsop, M. D. Sinclair, and S. V. Adve, "Spandex: A Flexible Interface for Efficient Heterogeneous Coherence," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 261–274. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00031

[2] Amazon, "Amazon EC2 F1 Instances," https://aws.amazon.com/ec2/instance-types/f1/.

[3] ARM Limited, "AMBA AXI and ACE Protocol Specification," https://developer.arm.com/documentation/ihi0022/e/.

[4] ——, "AMBA CHI Architecture Specification," https://developer.arm.com/documentation/ihi0050/c/.

[5] J. Balkind, K. Lim, M. Schaffner, F. Gao, G. Chirkov, A. Li, A. Lavrov, T. M. Nguyen, Y. Fu, F. Zaruba, K. Gulati, L. Benini, and D. Wentzlaff, "BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 699–714. [Online]. Available: https://doi.org/10.1145/3373376.3378479

[6] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton: An Open Source Manycore Research Framework," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016, pp. 217–232. [Online]. Available: http://doi.acm.org/10.1145/2872362.2872414

[7] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, 1986.

[8] A. Boutros, S. Yazdanshenas, and V. Betz, "You Cannot Improve What You Do Not Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, dec 2018. [Online]. Available: https://doi.org/10.1145/3242898

[9] T. Callahan, J. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler," *Computer*, vol. 33, no. 4, pp. 62–69, 2000.

[10] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A Cloud-Scale Acceleration Architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13.

[11] CCIX Consortium, "Cache Coherent Interconnect for Accelerators (CCIX)," https://www.ccixconsortium.com/.

[12] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.

[13] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE Press, 2016, p. 1–6. [Online]. Available: https://doi.org/10.1145/2897937.2897972

[14] N. Chugh, V. Vasista, S. Purini, and U. Bondhugula, "A DSL Compiler for Accelerating Image Processing Pipelines on FPGAs," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, ser. PACT '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 327–338. [Online]. Available: https://doi.org/10.1145/2967938.2967969

[15] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. El Husseini, T. Juhasz, K. Kagi, R. K. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Yi Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[16] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?" in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 225–236.

[17] J. Coole, J. Wernsing, and G. Stitt, "A traversal cache framework for fpga acceleration of pointer data structures: A case study on barnes-hut n-body simulation," in *2009 International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 143–148.

[18] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Commun. ACM*, vol. 33, no. 10, p. 30–53, Oct. 1990. [Online]. Available: https://doi.org/10.1145/84537.84545

[19] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[20] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A Quantitative Analysis of the Speedup Factors of FPGAs over Processors," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162–170. [Online]. Available: https://doi.org/10.1145/968280.968304

[21] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. B. J. Dally, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 75–84. [Online]. Available: https://doi.org/10.1145/3020078.3021745

[22] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 243–254. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30

[23] Handel Jones, "Strategies in Optimizing Market Positions for Semiconductor Vendors Based on IP Leverage," https://www.ibs-inc.net/white-papers, 2014.

[24] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera Reconfigurable Functional Unit," in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*, 1997, pp. 87–96.

[25] Intel Corporation, "Compute Express Link™ (CXL)," https://www.intel.com/content/www/us/en/io/cxl-cache-mem-protocol-interface-cpi.html.

[26] ——, "Cyclone V SoC," https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html.

[27] M. C. Jeffrey, S. Subramanian, C. Yan, J. Emer, and D. Sanchez, "A Scalable Architecture for Ordered Parallelism," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 228–241.

[28] D. Koch, N. Dao, B. Healy, J. Yu, and A. Attwood, "FABulous: An Embedded FPGA Framework," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 45–56. [Online]. Available: https://doi.org/10.1145/3431920.3439302

[29] C. Kumar, A. Seshadri, A. Chaudhary, S. Bhawalkar, R. Singh, and E. Rotenberg, "Post-Fabrication Microarchitecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1270–1281. [Online]. Available: https://doi.org/10.1145/3466752.3480119

[30] S. Kumar, A. Shriraman, and N. Vedula, "FUSION: Design Tradeoffs in Coherent Cache Hierarchies for Accelerators," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 733–745.

[31] A. Li and D. Wentzlaff, "PRGA: An Open-Source FPGA Research and Prototyping Framework," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 127–137. [Online]. Available: https://doi.org/10.1145/3431920.3439294

[32] M. Maas, K. Asanovic, and J. Kubiatowicz, "A Hardware Accelerator for Tracing Garbage Collection," *IEEE Micro*, vol. 39, no. 3, pp. 38–46, 2019.

[33] C. A. Mack, "Fifty Years of Moore's Law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 202–207, 2011.

[34] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "ASIC Clouds: Specializing the Datacenter," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 178–190.

[35] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Trans. Comput. Syst.*, vol. 9, no. 1, p. 21–65, feb 1991. [Online]. Available: https://doi.org/10.1145/103727.103729

[36] Microchip Technology Inc., "SmartFusion 2 SoC," https://www.microsemi.com/product-directory/soc-fpgas/1692-smartfusion2.

[37] Microsemi Corporation, "PolarFire SoC," https://www.microsemi.com/product-directory/soc-fpgas/5498-polarfire-soc-fpga.

[38] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, May 2020. [Online]. Available: https://doi.org/10.1145/3388617

[39] L. E. Olson, M. D. Hill, and D. A. Wood, "Crossing Guard: Mediating Host-Accelerator Coherence Interactions," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 163–176. [Online]. Available: https://doi.org/10.1145/3037697.3037715

[40] OpenCAPI Consortium, "OpenCAPI™," https://opencapi.org/.

[41] Oracle Corporation, "OpenSPARC™ T1 Microarchitecture Specification," https://www.oracle.com/servers/technologies/opensparc-t1-page.html.

[42] QuickLogic Corporation, "EOS S3," https://www.quicklogic.com/products/soc/.

[43] R. Razdan and M. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," in *Proceedings of MICRO-27. The 27th Annual IEEE/ACM International Symposium on Microarchitecture*, 1994, pp. 172–180.

[44] K. Rupp, "48 Years of Microprocessor Trend Data," https://github.com/karlrupp/microprocessor-trend-data, 2019.

[45] P. D. Schiavone, D. Rossi, A. D. Mauro, F. Gurkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, "Arnold: an eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End-Nodes," 2020.

[46] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[47] Siemens Digital Industries Software, "Catapult High-Level Synthesis and Verification," https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/.

[48] Silicon Integration Initiative, Inc., "15NM OPEN-CELL LIBRARY AND 45NM FREEPDK," https://si2.org/open-cell-library/.

[49] J. E. Smith, "Decoupled Access/Execute Computer Architectures," *SIGARCH Comput. Archit. News*, vol. 10, no. 3, p. 112–119, Apr. 1982. [Online]. Available: https://doi.org/10.1145/1067649.801719

[50] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "Freepdk: An open-source variation-aware design kit," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007, pp. 173–174.

[51] B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Iyer, B. Brezzo, D. Dillenberger, and S. Asaad, "Database Analytics Acceleration Using FPGAs," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 411–420. [Online]. Available: https://doi.org/10.1145/2370816.2370874

[52] N. Weaver, V. Paxson, and J. M. Gonzalez, "The Shunt: An FPGA-Based Accelerator for Network Intrusion Prevention," in *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 199–206. [Online]. Available: https://doi.org/10.1145/1216919.1216952

[53] P. N. Whatmough, S. K. Lee, M. Donato, H.-C. Hsueh, S. Xi, U. Gupta, L. Pentecost, G. G. Ko, D. Brooks, and G.-Y. Wei, "A 16nm 25mm2 SoC with a 54.5x Flexibility-Efficiency Range from Dual-Core Arm Cortex-A53 to eFPGA and Cache-Coherent Accelerators," in *2019 Symposium on VLSI Circuits*, 2019, pp. C34–C35.

[54] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[55] Xilinx, Inc., "Zynq-7000 SoC," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[56] ——, "Zynq UltraScale+ MPSoC," https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[57] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019.

[58] F. Zaruba, F. Schuiki, S. Mach, and L. Benini, "The Floating Point Trinity: A Multi-modal Approach to Extreme Energy-Efficiency and Performance," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 767–770.

[59] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[60] M. Zuluaga, P. Milder, and M. Püschel, "Streaming Sorting Networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, May 2016. [Online]. Available: https://doi.org/10.1145/2854150