MANCHESTER 1824
The University of Manchester

The University of Manchester Research

SimAcc

DOI:
10.1109/FCCM.2019.00031

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
Iordanou, K., Palomar, O., Mawer, J., Gorgovan, C., Nisbet, A., & Luján, M. (2019). SimAcc: A Configurable Cycle-Accurate Simulator for Customized Accelerators on CPU-FPGAs SoCs. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)). IEEE. https://doi.org/10.1109/FCCM.2019.00031

Published in:
IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

OPEN ACCESS

Download date:25. Apr. 2024

SimAcc: A Configurable Cycle-Accurate Simulator for customized accelerators on CPU-FPGAs SoCs

Konstantinos Iordanou, Oscar Palomar, John Mawer, Cosmin Gorgovan, Andy Nisbet and Mikel Luján
School of Computer Science, The University of Manchester, UK
{first}.{last}@manchester.ac.uk

Abstract*—This paper describes a flexible infrastructure for fast computer architecture simulation and prototyping of accelerator IP. A trend for System-on-Chips is to include application specific accelerators on the die. However, there is still a key research problem that needs to be addressed: *How do hardware accelerators interact with the processors of a system and what is the impact on overall performance?

To solve this problem, we propose an infrastructure that can directly simulate unmodified application executables with FPGA hardware accelerators. Unmodified application binaries are dynamically instrumented to generate processor load/store and program counter events and any memory accesses generated by accelerators, that are sent to an FPGA-based out-of-order pipeline model. The key features of our infrastructure are the ability to code exclusively at the user level, to dynamically discover and use available hardware models at run time, to test and simultaneously optimize hardware accelerators in an heterogeneous system.

In terms of evaluation, we present a comparison between our system and Gem5 to demonstrate accuracy and relative performance, using the SPEC CPU benchmarks; even though our system is implemented on Zynq XC7045 which integrates dual 667MHz Arm Cortex-A9s with substantial FPGA resources, it outperforms Gem5 running on a Xeon E3 3.2 GHz with 32GBs of RAM. We also evaluate our infrastructure in simulating the interaction of accelerators with processors using accelerators taken from the Mach Benchmark Suite and other custom accelerators from computer vision applications.

I. Introduction

Simulators provide valuable solutions by giving clear insights into complex systems. A simulator is a powerful and important tool because it enables computer architects and software developers to evaluate alternative designs, plans or policies. Thus, "What if?" questions concerning system performance can be addressed without having the requirement of experimenting on a system that actually exists.

A recent trend of modern computer systems is the inclusion of hardware accelerators in heterogeneous System-on-Chips (SoCs). For example, the market offers SoCs (Xilinx Zynq) that combine FPGAs for acceleration and general purpose CPUs. It has been suggested that the trend will lead to heterogeneous systems with many specialised hardware accelerators, although only a small subset of them powered on at a time, to tackle the problem of Dark Silicon [1]. There is a need for providing support for this type of architecture in the simulation infrastructure and to consider interactions between the accelerators, the CPUs and the memory hierarchy. Except in straightforward cases, with extremely simple accelerators, the time it takes to complete a task in the accelerator is not constant, due to interactions with the rest of the system via the memory hierarchy. Additionally, often the actual input data values affect the time required by the hardware accelerator to finish. Thus, naive solutions that simply add a fixed latency are not very useful.

This paper proposes SimAcc, a methodology to combine the accuracy of synthesized IP (intellectual property) hardware models with the flexibility to simulate binaries using software based simulators. The synthesised IP hardware models include both actual hardware accelerators and models of the CPU for simulation. This methodology resolves the speed-complexity issue by partitioning the functional simulation from timing models that simulate the microarchitectural structures affecting performance. An event stream comprised of the loaded/stored data addresses, and the program control flow addresses taken by native execution of user-level threads, and any memory accesses by accelerator IP, is consumed by timing models that are implemented in Bluespec and synthesized to an FPGA. This stream of events is generated from the Dynamic Binary Instrumentation [2] of an unmodified application executable. Our current system targets out-of-order ARMv7 ISAs using a Zynq XC7045 which integrates dual 667MHz Arm Cortex-A9s with substantial FPGA resources.

The software simulation of synthesizable register transfer level (RTL) is very slow, yet computer architects desire for faster simulation and testing of accelerator IPs that modifies the instruction set architecture (ISA) and/or the internal microarchitecture is increasing. Accelerator IP can be directly plugged into our simulation system in the form of a synthesizable model, encapsulated in Bluespec or designed with High-Level Synthesis, and then test the overall system performance (timing/performance statistics) on an unmodified application executable using this accelerator IP.

The contributions of this paper are:

- We outline the design, flexible modelling capabilities, and performance of our dynamic binary instrumentation based on a simulation system that is constructed using a Bluespec library of composable instrumentation systems. We compare our simulation infrastructure with Gem5 system call emulation mode using the SPEC CPU2006 Benchmark suite and SPEC reference CPU benchmarks using fastforwarding. One of the advantages of our infrastructure is that it delivers cycle level accurate timing

faster than comparable simulators by implementing the timing model in FPGA hardware.
- We propose a simulation methodology that includes the interaction of processors and accelerators. We are able to use unmodified IP blocks, which are wrapped with logic to capture the memory accesses that they initiate and provide through our infrastructure a timing/performance monitoring simulation. The proposed system can save a significant part of development time, as end-users can study the timing/performance statistics to identify optimization opportunities, and then re-design their accelerators for testing again with the same executable.
- Our system, implemented on a Zynq XC7045 which is equipped with dual 667MHz Arm Cortex-A9s integrated with programmable logic, outperforms Gem5 running on a Xeon with 32GBs of RAM, and our fastforwarding feature is 60x faster than this on Gem5.
- We design accelerators for two well-known computer vision applications, ORB-SLAM2 and Semi-Direct Monocular Visual Odometry (SVO), in order to present the interaction between accelerators and processors with SimAcc.

II. Related Work

In this section we discuss the approaches of software-based simulators such as Gem5, Sniper and ZSim along with FPGA assisted simulators for evaluation. An in-depth survey of FPGA accelerated simulation of computer systems is contained in [3].

Gem5 [4] is currently the de-facto standard for architecture simulation, and the only open source simulator for Arm based systems. Gem5 provides a highly configurable simulation infrastructure handling multiple ISAs, and CPU models (ranging from functional-only atomic models to cycle-detailed out-of-order (OoO) models) in conjunction with a detailed and flexible memory hierarchy providing support for multiple cache coherency protocols and interconnect models. The simulation mode can be either system call emulation (SE) mode, concentrating on the simulation of user-level application code, or, full-system (FS) mode, where OS code is also simulated.

Sniper and ZSim [5], [6], [7] both utilise Intel's Pin infrastructure to dynamically rewrite the code executed by X86 applications using a pintool. Sniper and ZSim use custom pintools to implement their simulation infrastructures by extracting key information from an application's execution. ZSim decodes instructions to μ-ops related to the pipelined microarchitecture under simulation. A core's pipeline stages are simulated and evaluated at each μ-op. Sniper uses interval simulation based on high-abstraction analytical models of core performance. A core's instruction stream is divided into timing intervals delineated by miss events, such as branch mispredictions, and cache misses.

APTSim [8] is a novel FPGA-based infrastructure able to simulate ARMv7 binaries. APTSim proposes a methodology to combine the accuracy of synthesized IP hardware models with the flexibility to simulate binaries using software based simulators. APTSim fully decouples functional simulation from the timing models used to produce performance information. Dynamic binary instrumentation is used to generate an event stream comprised of the addresses of data loaded and stored, along with the program control flow addresses taken by native execution of user-level threads. The event stream is consumed by timing models that are synthesized to an FPGA. Timing models gather statistics about their behaviour, they do not store data, only state and counters and as a result this allows the behaviour of a model to be evaluated whilst the area of the model remains manageable.

FAST [9], [3] was one of the first systems to use FPGA accelerated timing models, implemented in Bluespec, that are driven by speculative functional execution of full-system simulation, using modified QEMU software [10]. QEMU determines the dynamic instruction trace passed to the timing models. However, at program execution points where a branch mis-speculation occurs, functionally incorrect instructions will be fetched in a real processor until a mis-speculation is resolved, then the functionally correct branch path is followed. Therefore, the modified QEMU software must use costly check-pointing at places where such mis-speculations can occur, and roll-backs to a checkpointed state when a mis-speculation occurs in order to follow mis-speculated instruction traces until they are resolved. Such modifications further slow down the execution of a vanilla QEMU that is already significantly slower than other dynamic binary instrumentation tools such as MAMBO and Pin. It is important to note that FS simulation is unnecessary for many applications, and that the infrastructure described in this paper enables significantly more flexible interfacing to timing models than FAST, as well as the integration of accelerator IP.

HAsim [11] is a FPGA-based simulator that simulates multicore architectures using a highly-detailed processor pipeline, cache hierarchy and detailed on-chip network using a single FPGA. Its main contribution is the use of fine-grain multiplexing of pipeline models to support detailed timing for multicore processors. Internal core state, such as the PC and register file, is duplicated but the combinational logic used to simulate each pipeline stage is not. HAsim's timing models track how many FPGA clock cycles represent a single target machine cycle for a given operation. In this scenario, structures such as cache memories that do not map efficiently to FPGA resources, are implemented using FPGA efficient features (such as Block-RAMs), and multiple FPGA clock cycles then represent a single target machine cycle.

HeteroSim [12] is a simulation platform for heterogeneous CPU-FPGA systems. It combines two simulators: Multi2Sim [13] which is a cycle-based simulation framework for CPU-GPU heterogeneous processors, and Verilator, a fast simulator based on hardware description languages, which is able to compile synthesisable Verilog into C++ or System C code. After that, it generates an executable to simulate a given design, returning cycle accurate execution time performance and timing statistics. The combination of these two simulators allows the generation of metrics for CPU-FPGA systems. In

general, an end user provides the x86 executable and Verilog code of kernels, which are executed by HeteroSim that then generates statistics about the CPU-FPGA system. The main contributions of HeteroSim are the effective Co-simulation of Verilog code and x86 executables, the Shared coherent memory sub-system of CPU and FPGA which used by the two simulators and the CPU-FPGA communication through different memory hierarchies.

Aladdin [14] is a power and performance simulator aimed at searching the design space search of accelerator-based systems. The main contribution of Alladin is that it takes high-level language descriptions of algorithms as inputs. It uses these descriptions to create dynamic data dependence graphs (DDDG) that represent the accelerators, without having to generate RTL. The dataflow behavior of accelerators makes them ideal to be represented with dynamic data dependence graphs. Aladdin is able to achieve more than 100x faster than traditional RTL design flows. The ability to co-simulate Verilog designs and x86 executable, the rapid simulation speed and the evaluation of a CPU-FPGA system with a wide variety of system configurations rank Aladdin as an ideal infrastructure to quickly and accurately model accelerators without generating RTL. However one of the key limitations of Aladdin is that the accelerators cannot be reconfigured, because Aladdin generates an accurate representation of accelerator and it is not using the actual generated hardware for the simulation.

LiME [15] is a hardware/software tool designed for memory system evaluation. LiME uses the Xilinx Zynq UltraScale+ MPSoC on the ZCU102 board to capture any/all memory access, either from the Processing System (PS) or the Programmable Logic (PL). LiME employs novel loopback circuitry in conjunction with address map relocation to pass memory references from the PS into the PL side.

III. Architecture of SimAcc and Methodology

SimAcc is composed from three main components. A C++ library and hardware interface standard called MAST [8] and the dynamic binary modification tool MAMBO [2] are responsible for the simulation side. Additionally, SimAcc includes the accelerator side, which includes accelerator IP for specific applications and wrappers to interact with the simulation side. SimAcc is able to simulate ARMv7 binaries using hardware models for memory system hierarchies and microarchitecture pipelines. Furthermore the simulation of applications combined with accelerator IPs can collect performance statistics from pipeline and accelerator models.

A. Simulator

Figure 1 illustrates an overview of the SimAcc simulation infrastructure and how performance estimates and statistics are generated. More specifically MAST enables all the Bluespec IP hardware models to be managed by a C++ userspace software driver library. MAST manages the event stream of memory accesses and PC-altering instructions, in order to feed the memory and pipeline models. The event streams

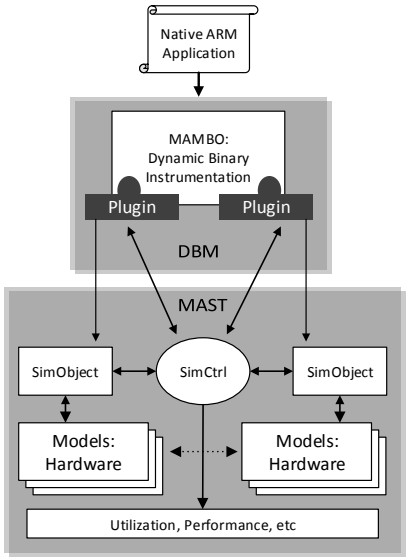

Fig. 1. Simulator side of SimAcc Infrastructure.

are generated from custom plugins that extend the Dynamic Binary Modification tool for ARMv7. The ability of our simulator to be extended and the flexibility it offers is one of the key benefits of the simulator part of SimAcc.

1) MAST BLUESPEC IP LIBRARY: MAST uses Bluespec to facilitate the rapid construction of highly parameterised models with well defined interfaces. Most MAST models have a single AXI interface, controlled by a library IP block that handles its identification, and locking, which can be used to control how the derivative models can accessed by a common software driver. Bluespec's atomic rule based coding style enables the synthesis of control systems that would be complex to specify using RTL languages. A MAST compliant IP component must adhere to a low-level IP interface containing identification, locking, data movement and I/O features such as burst controllers for fetching data from processor memory, and file based reading/writing IP. The high level models are currently: a) the out-of order Pipeline Model for the simulation of unmodified ARMv7 binaries; and b) the Cache System Model which obtains statistics for the cache hierarchies.

a) Out-of-order Pipeline model: implements most of the main features of out-of-order processors, including register renaming, memory aliasing checking and branch prediction. The functionality of the model is inspired by the Arm Cortex-A9 processor. The model does not functionally replicate the pipeline, i.e. it is not a functioning processor, it simply decodes instructions, counting individual instruction types, and registers used, and allocates instructions to a processor pipeline to obtain timing and utilisation statistics. The pipeline model communicates with a cache model to provide requests to the instruction and data caches. All the modules of the pipeline model are fully configurable.

b) Cache System: The cache model consists of three common cache structures which are included as additional models in the library. The first is a dual L1 cache, with a shared L2 cache, common in Intel-type multicore systems. The second is a cluster cache hierarchy, with a parameterised number of core caches (L1 instruction + L1 data) coupled via a Snoop Control Unit (SCU) to an L2 cache, this is the structure found in Arm's big.LITTLE style architecture. The third includes another dedicated cache in the second cluster cache which serves the requests directly from the accelerator of SimAcc only. One advantage of these combined models, when used as stand alone models, is that their components share a common ACP port, allowing larger memory systems to be configured on a device. It gathers statistics about the behaviour of a cache system. It is not a functional model, i.e. it does not store data, only address tags and states for cache lines; this allows the behaviour of a cache to be evaluated. The model always features an AXI slave that is used to lock the model to an application (the simulator), and to read back statistics; additionally, an optional ACP interface can be included, for burst transfers from the CPU. All the above cache systems are fully configurable in terms of cache-size, block-size and number of ways. End users can choose between the offered cache models to determine the most suitable model for their application or to extend the current Cache System implementation in order to replicate new cache hierarchies.

2) MAST USERSPACE DRIVER LIBRARY: The hardware is managed by a C++ software library that acts as an entirely userspace driver for any IP blocks configured onto the FPGA. The library is able to recognize any IP blocks that are present, and to enable the appropriate plugins in MAMBO [2], to ensure correct instrumentation is performed during program execution. It consists of two main classes that are used within any application accessing the FPGA, as shown in Figure 1. SimCtrl is responsible for managing the system as a whole and SimObject for controlling an IP block. SimCtrl has a number of features allowing easy system development. On creation the object probes the FPGA to ascertain what hardware is configured on it; it creates a SimObject derived object for each IP block, that allows an IP block to be used by the application.

3) MAMBO Instrumentation: MAMBO [2] is an efficient dynamic binary modification tool for Arm architectures that transparently modifies the machine code of 32 bit and 16 bit instructions during execution. Its performance is 2.8 times faster on average than Valgrind [16], and 14.9 times faster than QEMU [10]. To drive our hardware models we use MAMBO plugins to add new functionality. They consist of a set of callbacks that are executed at various points of program execution. An initialisation function is used to assign end user provided functions to MAMBO events, in the case of SimAcc we also use it to call SimCtrl to ensure that we have any hardware required by the plugin and thus enable plugin events. We have extended MAMBO with the feature of fastforwarding the simulated program, until the desired point is reached and the plugins start sending event traces to the FPGA. The idea is that initially the start of each code cache fragment is instrumented only with a call to an assembly function, using the number of instructions in the fragment as the argument. When the desired threshold is reached, the fastworwarding instrumentation is discarded and regular instrumentation starts.

As an illustrative example, we measured how long it takes to reach the desired point of simulation in the mcf SPEC CPU2006 reference benchmark. While SimAcc fastforwards to the desired simpoint in 5 seconds, Gem5 using the DerivO3 CPU needs more than 5 minutes to reach the same point.

B. Accelerators + Simulator

One of the key contributions of our infrastructure is the ability to co-simulate processors and accelerators and their interaction using unmodified binaries. SimAcc allows to implement conventional FPGA-based accelerators, such as image processing filters, and access these from regular applications. Figure 2 illustrates how SimAcc models the interaction of specialised hardware with CPU and caches in an SoC. On one hand, CPU-initiated transactions are easily captured by the simulator, since it captures all memory accesses already. On the other hand, accelerators that include a master port can initiate transactions independently from the CPU. In our environment, this is achieved by an accelerator directly accessing the memory of the processor over the ACP port, this enables dramatic performance increases over CPU managed transfer. Moreover, the ability to use a block of memory makes the migration of CPU based code to FPGA accelerators a straightforward task, without having to deal with kernel level drivers. Nevertheless, combining accelerators and simulated hardware requires that the infrastructure is able to capture all accesses to the memory hierarchy, including those via the ACP port. Moreover, the accesses to the actual ACP port must still happen, as the behaviour and outcome of the simulated run could differ if the accelerator is not reading or writing the expected data.

In SimAcc we have tackled the problem of capturing the traffic in the ACP port by developing a number of IP blocks (namely, the ACP Snoop module, the Access Generator module and the Cache Arbiter) that wrap the accelerator and allow it to interact with the simulation infrastructure. The ACP Snoop module intercepts the ACP accesses generated by the accelerator, creates a descriptor for them and sends the original request to the actual ACP port. This enables using unmodified hardware blocks for the accelerators. Responses from the ACP port are sent directly to the accelerator. An interface of the ACP Snoop returns the most recent descriptor, if any, and it keeps a small buffer of recent descriptors. The Accesses Generator is responsible for retrieving descriptors from the ACP Snoop, and breaking them down into individual memory accesses to send to the cache model. The Cache Arbiter takes the requests from the Accesses Generator and the pipeline model (we do not need to if the simulated system does not include an accelerator). The Cache Arbiter also has an input interface from the Pipeline model that notifies when there is a new simulated cycle. We use this to sync the speed of the requests from the Access Generator and the Pipeline Model.

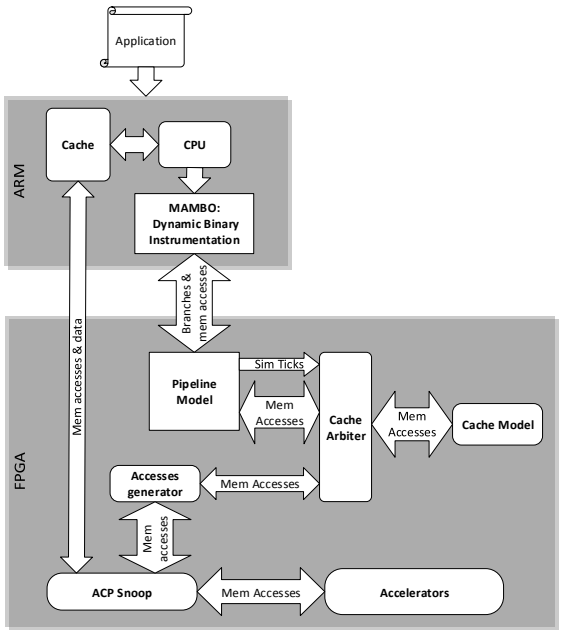

Fig. 2. Overview of the SimAcc Infrastructure.

There are some limitations in our approach. One of the main drawbacks concerns the achieved performance of SimAcc. The accelerator IP in SimAcc is configured to run at its usual frequency. Moreover the simulated CPU is much slower than the actual CPU. Combining the above, the relative speed of the accelerator compared to the CPU is much higher when it is simulated. In order to synchronize the simulator with the accelerators in SimAcc infrastructure, we slow down the modelling of the memory accesses in the cache arbiter module with the input of simulated cycles. This approach fits in a large proportion of accelerators. There are two basic scenarios in which this approach is not ideal. First, this approach is not applicable for accelerators which are always running, that is, an accelerator which is not waiting for the CPU to send a start command. Second, the approach of SimAcc may affect the operation when there is time-dependent behaviour, either on the software or the accelerator. An example would be if the accelerator times out waiting for something to be provided by the CPU. Since the program running on the simulated CPU will progress much slower than on the actual CPU, a time out may be triggered in the simulation only.

C. SimAcc flow for accelerator development and co-simulation

SimAcc is an infrastructure which allows to test and simultaneously optimize accelerators in a heterogeneous system. Figure 3 presents the flow for developing FPGA-based accelerators and including them for simulation in SimAcc. For end users of SimAcc the process of running a simulation for an application with an accelerator is simple. As an initial step, they must produce a hardware configuration to match their desired memory hierarchy and pipeline models parameters. Running the source through the Bluespec compiler will produce a Verilog file that is suitable for creating a Vivado IP block (it is also possible to ship pre-compiled Vivado IP blocks that can be used directly, with the configuration parameters already set). After creating the desired models with the Bluespec compiler, the system is ready to be loaded to Vivado. From the accelerator side, end users can create accelerators using High Level Synthesis or Bluespec with the limitation that the created accelerator uses the ACP port. After that stage a simple Vivado TCL script is used to integrate accelerators and simulator models, producing a bitstream file for FPGA configuration. As illustrated in Figure 3, after getting the statistics, users either re-design their accelerator(s) through HLS or optimize their accelerator(s) prior to testing with the same executable.

IV. Accelerator Use-Cases

In this section we present use-cases of accelerators from the Mach Benchmark Suite [17] and accelerators for computer vision applications [18][19] that are designed and evaluated using our simulation system.

1) ORB-SLAM2: ORB-SLAM2 [18] is a state-of-the-art SLAM implementation. After carefully profiling the application, we developed an accelerator for one of the key functions of ORB-SLAM: FAST corner detector. This accelerator can be typically implemented with a sliding window over the current frame. This was challenging in this case, as ORB-SLAM divides the frame into multiple "cells" and independently in each one uses two variants: first a more restrictive one (higher threshold), and if not enough corners are found, the second version is used. The hardware implementation required buffering the image at the cell level and implementing both versions in parallel, selecting which one is used according to the results. This implementation results in a sequential memory access pattern that reads the input frame from a buffer and another that writes the output into another buffer.

We have implemented the accelerator for corner detection using Bluespec and we have leveraged MAST to package this accelerator for use in SimAcc. Images are passed to and from the main applications using the SimCtrl unit to map pages from virtual to physical addresses and to store page mappings on the FPGA, as a translation lookaside buffer (TLB). This use-case demonstrates that only user-space coding is required in MAST. ORB-SLAM2 has been modified to use the MAST software library capabilities to detect if a FAST accelerator exists in the hardware. We have demonstrated this works robustly even if the FPGA is reconfigured during the application runtime.

The simulator hardware blocks and the accelerator hardware blocks co-exist peacefully in the FPGA, and MAST can handle both without any problem. Note that since both ORB-SLAM2 and SimAcc use MAST, when simulating ORB-SLAM2 there are two instances of MAST: one linked with the simulator and one used by ORB-SLAM2. Both instances see all the hardware IP, but the simulator one manages the hardware models of the pipeline and caches and the other manages the accelerators. An interesting observation is that the MAMBO

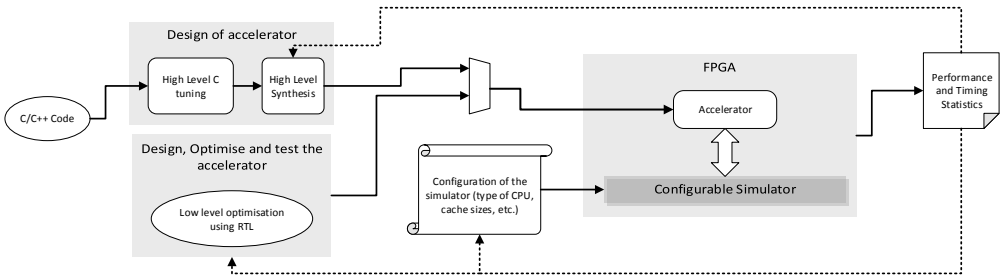

Fig. 3. SimAcc flow for accelerator development and co-simulation.

plugin runs functions of MAST directly, but that the simulated program executes instrumented versions of the same functions seamlessly.

2) Semi-Direct Monocular Visual Odometry (SVO): Semi-Direct Visual Odometry (SVO) [19] has a semi-direct approach that eliminates the need of feature extraction and robust matching techniques for motion estimation. The algorithm operates directly on pixel intensities, which results in subpixel precision at high frame-rates. A probabilistic mapping method that explicitly models outlier measurements is used to estimate 3D points, leading to fewer outliers and more reliable points. The profiling of the algorithm shows that the most time-consuming part of the application is the feature alignment function. In order to evaluate the functionality of our simulation infrastructure the feature alignment function was hardware implemented, using HLS. This function can be explained as a relaxation step, that achieves a higher correlation between the feature-patches. This alignment is solved using the inverse compositional Lucas-Kanade algorithm [20].

3) Mach Benchmark Suite: MachSuite [17] is a collection of nineteen benchmarks for evaluating high-level synthesis tools and accelerator-centric architectures. MachSuite spans a broad application space, captures a variety of different program behaviors, and provides implementations tailored towards the needs of accelerator designers and researchers, including support for high-level synthesis. We illustrate the potential of our infrastructure by implementing six accelerators (aes encryption, fft strided, fft Transposed, bfs, bfs Queue, kmp) from Mach Suite and simulating them with our infrastructure.

V. Evaluation

A. Experimental setup

SimAcc uses a Zynq XC7045 which consists of an integrated processing system (PS) and programmable logic (PL), on a single die. The PS integrates dual 667MHz Arm Cortex-A9s. Additionally, we use Gem5 in system call emulation mode to generate statistics and compare against SimAcc. We currently evaluate single-threaded applications.

We use the SPEC CPU 2006 benchmark suite [21] to evaluate the accuracy of the out-of-order CPU model of SimAcc, compared with Gem5. We also use accelerators generated automatically from the Mach benchmark suite [17] to test the ability of SimAcc to model the interaction of the simulated CPU and arbitrary accelerators. In all cases, bear in mind that we must fit our simulation runtime infrastructure within the 1GB RAM space. Some basic configuration parameters used in the experiments, for both SimAcc and Gem5, are presented in Table II. First, we run the benchmarks with the test input in order to be able to run applications to completion without the fastforwading option, and keep simulation time reasonable. Allowing the simulation to execute until the end gives more confidence in the implementation and results. In a second stage of testing, we simulate using the reference input benchmarks in order to evaluate intervals of execution more representative of actual programs. In this case, to keep simulation time reasonable, we used SimPoint [22] to select the most representative 100M instruction interval and simulated only that (plus a 1M instructions warm up period), using the option of the fastforwarding both in SimAcc and in Gem5 [4]. In the case of Gem5, we also create a checkpoint, so we only need to fastforward once.

TABLE I
FPGA utilization for the evaluation system.

Component	LUT	BRAM
L1 I+D & L2	6%	2%
Snoop Controller	1%	–
Pipeline model	35%	1%
- IEU	6.4%	
- LSU	8.5%	
- ROB	2%	
DMA Engine	1%	–
Cache Arbiter	1%	–

We have used the flexibility of our infrastructure to simulate the accelerator use-cases mentioned before, using different cache configurations. **SmallCache**, **MediumCache** and **LargeCache** referred to different cache configurations. **SmallCache** defined as 16k,512k,32k for dcache, L2, icache respectively, **MediumCache** referred to 64k,2M,32k and **LargeCache** defined as 125k,4M,32k. Bear in mind that the presented results are normalised to the configuration SmallCache.

B. Results and Discussion

1) Resources Utilization: In terms of resources Table I shows the utilization reported by Vivado (version 2017.2) after

TABLE II
CONFIGURATION PARAMETERS.

CPU		Branch Configuration		Cache Configuration			
					L1	L2	instr
CPU type	DerivO3CPU	Type	Tournament	associativity	2	8	2
ROB entries	192	BTBentries	4096	size	64K	2M	32K
LQ/SQ entries	16	BTB tag size	16	block size	64	64	64
Phys Registers	56	RAS size	16	TLB entries	32	32	32

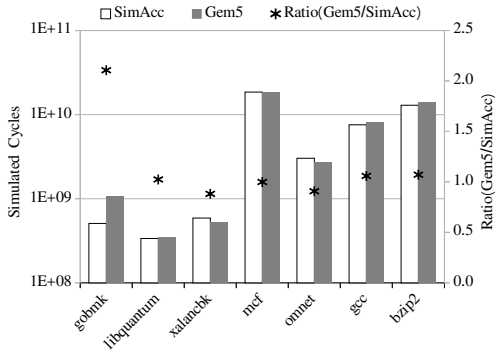

Fig. 4. Simulated cycles (SPEC test input).

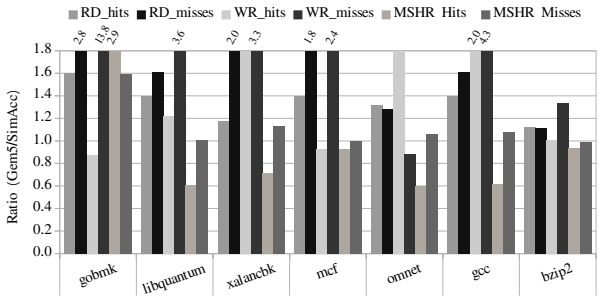

Fig. 5. L1 data cache statistics (SPEC test input).

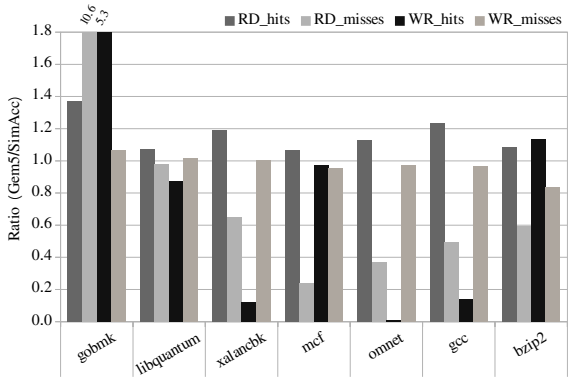

Fig. 6. L2 cache statistics (SPEC test input).

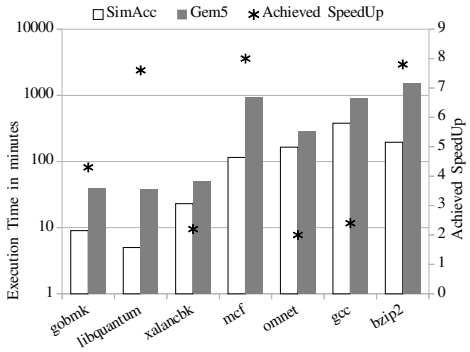

Fig. 7. Simulation time in minutes (SPEC test input).

the implementation step. The pipeline model seems to occupy most area, but note that a significant amount of this area is occupied from the IEU, LSU and ROB modules of the pipeline model which are fully dependent from their size parameters. For instance, choosing a large LSU leads to a dramatic increase in the resources utilization of the pipeline model.

2) Accuracy: Figure 4 shows the number of simulated cycles of the SimAcc infrastructure compared with Gem5, and the ratio between them. Figures 5 and 6 plot the statistics for the data L1 cache and the unified L2 cache respectively. The most significant difference is found in benchmark gobmk. At this stage of development, we have not fully implemented support for multiple accesses instructions in our infrastructure, and an in-depth study revealed that gobmk makes extensive use of multiple accesses instructions, which causes this discrepancy.

The other benchmarks present differences in the range 0.4 to 20. While these are apparently large discrepancies, they typically appear in benchmarks where the absolute value is quite small, and thus the relevance of these is small in the overall simulation.

When simulating without the fastforwarding option, we compare the number of simulated instructions as well as the number of cycles, together with key statistics (e.g. from the caches and branch predictor) that help us to understand the discrepancies between the two simulators.

When simulating intervals of the reference input, we simply present results for IPC (Instructions Per Cycle), as the number of simulated instructions is fixed in Figure 8.

3) Simulation speed: Figure 7 reports the execution time in minutes for the SPEC benchmarks running on our infrastructure and Gem5. We can yield that the achieved speedup of SimAcc compared with Gem5 ranges from 2x - 8x. Although our current work has been on accuracy of simulation, rather than performance and there is significant scope for performance improvement. Firstly the use of page sized RAM buffers in the timing model means that if code is not present in a RAM buffer then an entire page is written to the FPGA RAM; in many applications more than 4 pages of RAM will

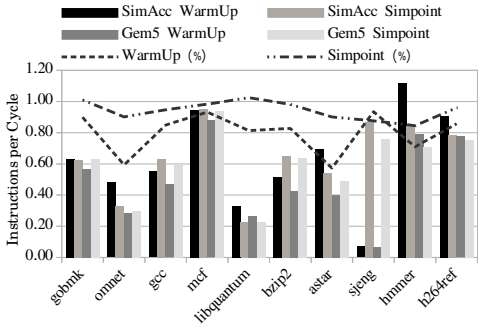

Fig. 8. Instructions per Cycle (SPEC ref input).

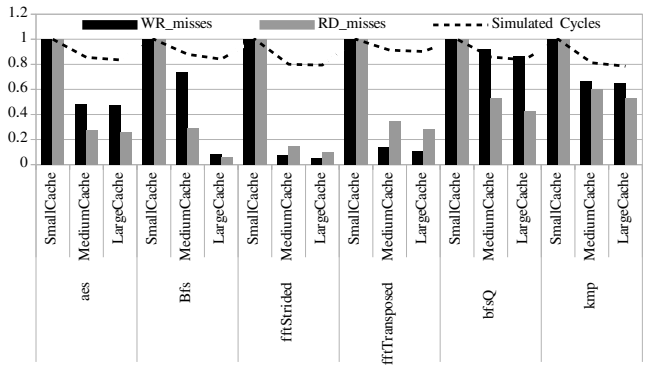

Fig. 9. L1 data cache statistics normalised to SmallCache and Simulated Cycles normalised to 1 (MachSuite accelerators).

be used in relatively tight blocks of code meaning time is wasted on moving data to the FPGA.

A logical alternative solution is to directly access the specific memory address over the ACP port, using the processors cache as a code cache, this could significantly reduce the amount of communications to main memory and cause less pollution of the processor's cache, both of which should improve overall performance. Moreover, the memory model is currently transferring accesses one at a time to the cache model, a more rapid solution would be to double buffer these in the MAMBO plugin and allow the cache to DMA them from the buffers as they become full; this would allow overlap of cache modelling and data gathering and the use of ACP offers around an order of magnitude improvement in data transfer rates compared with direct writes.

4) Accelerators: Figure 9 presents the simulated cycles for the six accelerators from MachSuite. The simulated cycles decreased as the cache sizes becoming larger, with MediumCache yielding significant speedup while LargeCache having limited effect for some accelerators. Figure 9 shows statistics from the cache model. Obviously as the L1 cache size becomes larger, we observed a smaller cache miss rate. Concerning the L2 cache statistics, the smallest cache size already fitted the whole workload of the benchmarks, and all configurations present a steady cache miss rate.

Additionally, Figures 10 and 11 illustrate the simulated cycles, L1 and L2 data statistics for FAST and Feature Alignment

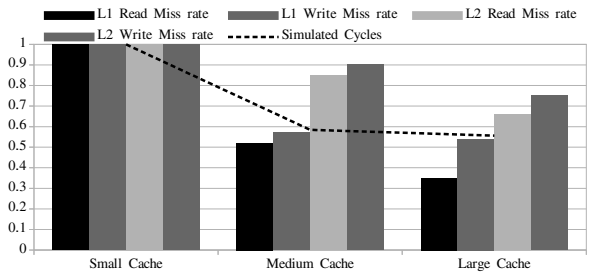

Fig. 10. Simulated cycles and Data Cache Statistics normalised to SmallCache (FAST accelerator for ORB-SLAM2).

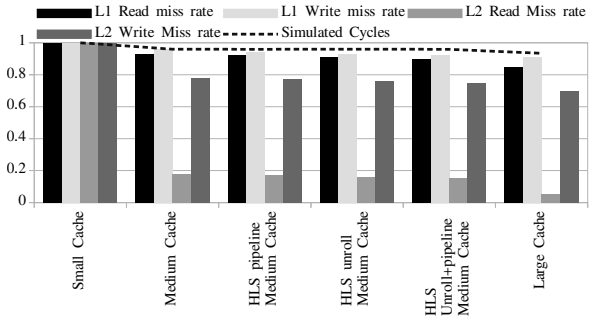

Fig. 11. Simulated cycles and Data Cache Statistics normalised to SmallCache (Feature Alignment accelerator for SVO).

accelerators for ORB-SLAM2 and SVO applications respectively. We can observe that the simulated cycles decrease as the cache size becomes larger for both applications. From the graph 11 we can see minor differences in simulated cycles and cache statistics when we apply different HLS optimizations.

VI. Conclusions and Future Work

We have demonstrated the potential of combining a flexible IP hardware library, a user-level driver library and dynamic binary instrumentation for microarchitecture simulation and prototyping. We exploit the advantages of an FPGA SoC to accelerate at a very fine granularity (instructions), rather than large blocks of code. With this, we can benefit from the accuracy and speed of FPGA-based modelling and the ability to run binaries. Moreover, this paper contributes the first FPGA-based simulator for Arm combined with accelerators, significantly extending the options for simulating Arm processors. As future work, we will implement the system on an Ultrascale+ Zynq board. Moreover, we plan to extend the models to include more microarchitectural features and alternatives (e.g. more features in the out-of-order execution model, or different pipelines in parallel which simulate each core independently to model multi-core systems).

VII. Acknowledgements

This work was partially supported by EPSRC grants PAMELA EP/K008730/1 and RAIN Hub EP/R026084/1, and EU H2020 project ACTiCLOUD agreement no. 732366. Mikel Luján is supported by an Arm/RAEng Research Chair.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, May 2012.

[2] C. Gorgovan, A. d'Antras, and M. Luján, "Mambo: A low-overhead dynamic binary modification tool for arm," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 1, pp. 14:1–14:26, Apr. 2016.

[3] H. Angepat, D. Chiou, E. S. Chung, and J. C. Hoe, *FPGA-Accelerated Simulation of Computer Systems*, 2014.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC*, 2011, pp. 52:1–52:12.

[6] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled simulation of multi-threaded applications," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2013, pp. 2–12.

[7] W. Heirman, S. Sarkar, T. E. Carlson, I. Hur, and L. Eeckhout, "Power-aware multi-core simulation for early design stage hardware/software co-optimization," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2012, pp. 3–12.

[8] J. Mawer, O. Palomar, C. Gorgovan, A. Nisbet, W. Toms, and M. Lujn, "The potential of dynamic binary modification and cpu-fpga socs for simulation," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines*, April 2017, pp. 144–151.

[9] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. H. Reinhart, D. Eric Johnson, and Z. Xu, "The fast methodology for high-speed soc/computer simulation," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, Nov 2007, pp. 295–302.

[10] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2005, pp. 41–41.

[11] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 406–417.

[12] L. Feng, H. Liang, S. Sinha, and W. Zhang, "Heterosim: A heterogeneous cpu-fpga simulator," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 38–41, Jan 2017.

[13] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012.

[14] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 97–108.

[15] A. K. Jain, S. Lloyd, and M. Gokhale, "Microscope on memory: Mpsoc-enabled computer memory system assessments," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines*, April 2018, pp. 173–180.

[16] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2007, pp. 89–100.

[17] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *2014 IEEE International Symposium on Workload Characterization*, Oct 2014, pp. 110–119.

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct 2015.

[19] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation*, May 2014, pp. 15–22.

[20] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *Int. J. Comput. Vision*, vol. 56, no. 3, pp. 221–255, Feb. 2004.

[21] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[22] A. A. Nair and L. K. John, "Simulation points for spec cpu 2006," in *2008 IEEE International Conference on Computer Design*, Oct 2008, pp. 397–403.