Workload Partitioning Strategy for Improved Parallelism on FPGA-CPU Heterogeneous Chips

Amiri, S., Hosseinabady, M., Rodríguez, A., Asenjo, R., Navarro, A. & Nunez-Yanez, J.

DOI 10.1109/FPL.2018.00071
ISSN 1946-147X
ESSN 1946-1488

Publisher: IEEE

Workload Partitioning Strategy for Improved Parallelism on FPGA-CPU Heterogeneous Chips

Sam Amiri*, Mohammad Hosseinabady*, Andres Rodriguez†, Rafael Asenjo†, Angeles Navarro† and Jose Nunez-Yanez*
*University of Bristol, UK. E-mail: {ma17215, m.hosseinabady, j.l.nunez-yanez}@bristol.ac.uk
†Universidad de Malaga, Spain. E-mail: {andres, asenjo, angeles}@ac.uma.es

***Abstract*—In heterogeneous computing, efficient parallelism can be obtained if every device runs the same task on a different portion of the data set. This requires designing a scheduler which assigns data chunks to compute units proportional to their throughputs. For FPGA-CPU heterogeneous devices, to provide the best possible overall throughput, a scheduler should accurately evaluate the different performance behaviour of the compute devices. In this article, we propose a scheduler which initially detects the highest throughput each device can obtain for a specific application with negligible overhead and then partitions the dataset for improved performance. To demonstrate the efficiency of this method, we choose a Zynq UltraScale+ ZCU102 device as the hardware target and parallelise four applications showing that the developed scheduler can provide up to 94.06% of the throughput achievable at an ideal condition, with comparable power and energy consumption.**

***Index Terms*—scheduling, heterogeneous, parallelisation, throughput, energy, power, FPGA, ARM, SoC, ZCU102**

I. Introduction

Heterogeneous computing can solve performance and/or power-energy consumption limitations associated with traditional processing using a single type of device. Two common heterogeneous chips are the integrated CPU-GPU (Graphics Processing Unit), and CPU-FPGA (Field-Programmable Gate Array). FPGAs offer bit-level parallel computing suitable for certain designs which cannot easily be parallelised with traditional methods. Hardware Description Languages (HDLs) are generally used to program FPGAs requiring expert hardware knowledge. Recent advancements in high-level synthesis tools have made it possible to describe hardware at high level languages such as C++ and OpenCL, and achieve performance comparable to when HDLs are used as entry point for some applications [1].

The general approach of utilising a heterogeneous FPGA-CPU device is to use the CPU as the scheduling and management unit which offloads the tasks to an FPGA accelerator for processing. Recent heterogeneous devices contain more CPU cores with increased efficiency which makes CPU involvement in actual task processing beneficial. A method for parallel execution is to implement the entire application in both FPGA and CPU devices, and schedule the workload for processing in them such that every device consumes a different portion of dataset. In this article, we introduce a strategy to partition the data chunks among the devices proportional to their performance. This requires exploring the performance behaviours of CPU and FPGA devices for an application when the data chunk size (i.e. data subset size) assigned to each device varies. The applications are programmed in C/C++ for both CPU and FPGA using Xilinx SDSoC framework for higher productivity. The novelty of this work can be summarised as follows:

- Exploring the performance behaviour of tightly integrated CPUs and FPGA when processing an application for different data chunk sizes;
- Proposing a strategy to measure the data chunk sizes required for CPU and FPGA to improve the execution performance of an application;
- Implementing the scheduler based on the integration of the pipeline pattern of Intel TBB (Threading Building Blocks) with Xilinx SDSoC framework;
- Applying the proposed scheduler to four benchmarks with different throughput behaviours implemented through SDSoC framework on a Xilinx Zynq UltraScale+ MPSoC ZCU102 board composed of FPGA and CPU compute units, and analysing the performance and power/energy consumption results obtained.

The rest of this paper is organized as follows. Section II presents an overview of related work. Section III summarises the development tools and the benchmarks. Section IV covers the novelties of the article in detail, including the behaviour of different compute units and the new scheduling strategy and its application to four benchmarks to evaluate performance and power/energy results. Finally, Section V concludes the paper.

II. Previous Work and Background

The combination of CPUs and FPGAs, integrated on the same die or externally, has received increasing attention recently. Xeon-FPGA platform for the data centre by Intel [2], Catapult fabric for accelerating large-scale data centre services by Microsoft [3], and integration of POWER8 processor architecture with external FPGAs by IBM [4] are several examples.

Workload balancing for a heterogeneous device at compile- or run-time has been explored in the literature around mainly two types of implementations, vertical and horizontal. An application might be vertically segmented into subtasks each optimally mapped onto a different device, hence building a streaming pipeline. In [5], an FPGA-GPU-CPU architecture is used where steps of the algorithm requiring deep pipelining and small buffers are mapped to FPGA, steps requiring massively parallel operations with large buffers leverage GPU, and the CPU is used for coordination, low throughput and branching dominated tasks.

In a horizontal implementation, different compute units run the same kernel on different portions of the dataset. In [6], the workload is manually partitioned with two-third of dataset assigned to an FPGA and the rest to a GPU. LogFit scheduling method, introduced in [7], [8] for GPUs and FPGAs respectively,

This work was funded by the Engineering and Physical Sciences Research Council (EPSRC) through the ENEAC (EP/N002539/1) grant.

examines the throughput for different accelerator chunk sizes and applies a logarithmic fitting function to find the accelerator chunk size that maximises the throughput. CPU chunk sizes are adaptively computed based on the accelerator chunk size. Instead of manually splitting the workload among the participating compute units or examining various chunk sizes with large overhead, we propose a solution here to estimate the throughputs at run-time with low overhead. Our method explores the performance of the units within a short period at run-time and then finds the optimal split in order to improve execution overlap and hence throughput for a specific application running on an FPGA-CPU device.

III. Development Environment and Benchmarks

To develop the scheduler and applications, we have used the platforms, libraries and tools explained briefly in the following.

A. Development Tools

Every application is developed for two targets, FPGA and CPU. For FPGA execution, the application is developed in SDSoC environment, and using SDSoC tools (version 2017.1 [9]), we port the implementation to a Linux environment for compilation by sds++ to generate hardware and drivers. This gives a bitstream for uploading in the FPGA and a C-callable hardware library to be linked with the CPU host code. The CPU version of the application and the scheduler are compiled using standard g++ (with C++11 support) directly in the target board running Ubuntu 15.04. The evaluation board used in our experiments is Xilinx Zynq UltraScale+ ZCU102 featuring a XCZU9EG FPGA and ARM Cortex-A53 v8 architecture-based 64-bit quad-core processor.

B. Scheduling and Parallelisation

In a heterogeneous execution by dataset partitioning, the whole dataset is divided into many fixed-length (e.g. matrix rows) *blocks* and at each iteration, a number of blocks called a *data chunk* is assigned to a compute unit. A property of such heterogeneous execution at run-time is that once a chunk of data is assigned to a compute unit, only that subset can be loaded to the unit, and the unit's input ports cannot be predictively filled with extra chunks to save in time. This is because the next data chunk to be processed is assigned at run-time to only a compute unit in idle state. The cut-off time between data chunk assignments causes filling and emptying ports and pipelines hence introducing overheads. A scheduler should reduce this overhead among other tasks.

To apply and control parallelism, we use `pipeline` and `filter` classes of Intel Threading Building Blocks (Intel TBB) [10]. Implementing a scheduler requires at least two pipeline stages. The first is the main scheduling unit which is a serial filter to select the next idle compute unit and calculate the data chunk size to be assigned to it (explained in Section IV). The iteration space is composed of the data subsets already assigned to compute units and the ones remaining and waiting to be assigned. The second stage is a parallel filter to communicate and process data chunks simultaneously in all the compute units (including the FPGA and four CPU cores).

C. Benchmarks

Four benchmarks with different features are used in our experiments to expose the required scheduling decisions: Advanced Encryption Standard (AES), HotSpot, Nbody, and General Matrix Multiplication (GEMM). Our AES implementation uses a 256-bit key to encrypt blocks of 16-bytes, totalling 16MB of data. 15 rounds of byte manipulation functions (subbytes, mixcolumn, addroundkey, and shiftrow) are repeated according to the standard. HotSpot application, extracted from Rodinia benchmark collection, is used for thermal simulation on the surface of a chip. It is applied to a chip with 1024x1024 points here. Nbody is a physics simulation to measure particles (bodies) behaviour under the influence of a force. 10K bodies are examined using a brute force algorithm. GEMM application multiplies two dense matrices each having 1024x1024 floating-point values.

IV. Heterogeneous Execution

As mentioned in Section II, the partitioning for parallelisation can be applied at task level (vertical) or data level (horizontal). For this work, we implement the entire application for each target element, and the scheduler controls parallelism at run-time by the size of data chunk assigned for processing by each device. The emergence of C-based full-system design tools such as Xilinx SDSoC has made this approach favourable as certain attributes, directives, or pragmas can be used to convert a program coded for a processor into the one targeting FPGA, although some hardware knowledge is still required.

Every type of processing element present in a heterogeneous system communicates and processes data in a different fashion, hence exhibiting different throughput/energy-consumption performance depending on the circumstances. The throughput behaviour is important as it will affect the scheduling decisions.

A. Behaviour of a CPU-Only System

The cache-based architecture and relatively shallow pipeline of a processor makes it perform at almost the same throughput level irrespective of the data chunk sizes assigned to it. Each of the four Cortex-A53 processors in ZCU102 device has separate 32 KB L1 caches for instruction and data, and all the cores share a 1 MB L2 cache in Cache Coherent Interconnect (CCI) domain. Access to DDR Memory Subsystem is provided through CCI. With the L1 caches part of a processor's 8-stage pipeline, whether the processor is assigned many smaller data chunks or few larger ones does not have a noticeable impact due to the random memory access and cache filling. Empirically every data block itself is composed of multiple finer data units (e.g. elements in one row of a matrix) and changing the data chunk size does not have a significant effect on memory locality. Although considering processor cache-effects would increase the accuracy of initial assessment in our proposed scheduler (Section IV-C), it would also increase its overhead, hence it is ignored here.

B. Behaviour of an FPGA-Only System

Contrary to the behaviour of a CPU, as the chunk size increases, an FPGA-only execution generally provides an increasing performance up to a point (which we call Saturation Level, SL) where it flattens out, as shown in Fig. 1. The FPGA execution time to process a specific volume of data chunk is mainly composed of the following components:

- Pipeline initialisation and flushing times (T_{IO}): This is the latency to initially locate and transfer data chunk from memory through the chip ports and fill the pipelines (ramp up) as well as the latency to empty the pipeline stages and

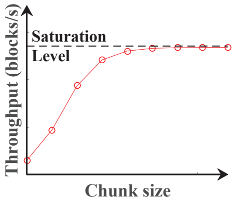


Fig. 1: Saturation level of an FPGA-only execution

finalise the processing of the data chunk assigned (ramp down). T_{IO} depends on the speed and width of the ports communicating data, and is almost constant and independent of the assigned data chunk size for an application. This delay repeats for every iteration of FPGA execution.

- Pipeline processing time (T_{Proc}): This is the time to process data chunk inside FPGA fabric. If the ratio of the number of data units to pipeline depth is high, the average T_{Proc} for one unit is almost equal to delay of the worst case pipeline stage. As in a practical application the number of data units is much higher than the number of pipeline stages, T_{Proc} is directly proportional to the size of data chunk processed.

Therefore, the execution time (T_{Exe}) for a data chunk processed in one iteration can be estimated as:

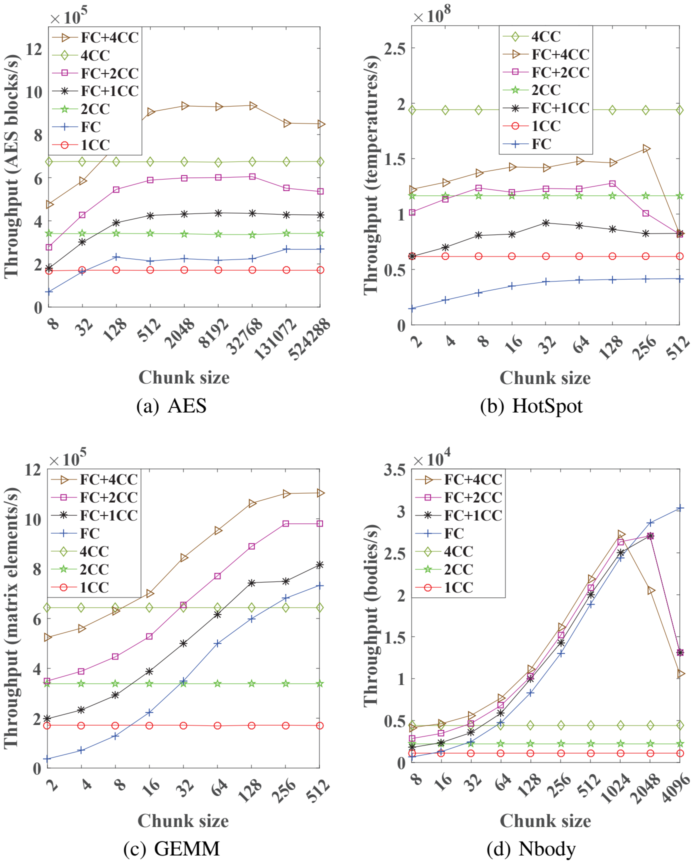
$$T_{Exe} = T_{IO} + T_{Proc} \tag{1}$$

Considering CS_{Tot} and CS_F as respectively the total chunk size to be processed in an application and the chunk size assigned to the FPGA at each iteration, as CS_F is made smaller, the number of iterations to process CS_{Tot} becomes larger and the time T_{IO} is repeated at a higher rate (once per iteration). However, with rising data chunk sizes, T_{IO} repetitions are decreased causing the gradual increase of throughput in the plot of Fig. 1. The best FPGA throughput is at SL where the ratio of combined T_{IO} to T_{Proc} for all the iterations is reduced to almost zero. An optimised scheduler should measure CS_F such that it falls within SL.

C. Proposed Scheduling Strategy for FPGA-CPU Device

For efficient scheduling, the data chunk size assigned to a device should be proportional to its throughput [11]. Given a heterogeneous device composed of an FPGA with Th_{F-SL} throughput at SL and four CPU units each having Th_C for an application and the throughput ratio of Th_{Rat} (Th_{F-SL}/Th_C), then the processing of enough number of remaining data blocks can be efficiently parallelised if the relationship between the data chunk size assigned to FPGA (CS_F) and to one CPU core (CS_C) is $CS_C = CS_F/Th_{Rat}$. For this scheduling strategy to work, if one device is considered as base and its chunk size is known, the data chunk size for the other devices can be adaptively set according to the base device's chunk size. We take FPGA as base here due to its higher performance than CPU in general, however the same results should hold if the base device is changed. When processing the final few remaining data blocks, for each of the nC CPUs the chunk sizes determined should be based on the remaining chunk size CS_{Rem} as $CS_C = CS_{Rem}/(Th_{Rat} + nC)$ to make sure that CPUs are assigned proportionally and if the FPGA is freed, there are enough blocks for balanced execution.

$$CS_C = min\Big(\frac{CS_F}{Th_{Rat}}, \frac{CS_{Rem}}{Th_{Rat} + nC}\Big) \tag{2}$$



(a) AES

(b) HotSpot

(c) GEMM

(d) Nbody

Fig. 2: Performance evaluations using Dynamic partitioning

As this partitioning strategy requires the FPGA chunk size to be known in advance, it cannot practically be used as a scheduler however it can be utilised for exploring and analysing FPGA and CPU performance and power/energy behaviours when different chunk sizes are selected. This partitioning strategy is applied to GPU-CPU devices in [11], called Dynamic partitioning, and we extend it to FPGA-CPU devices for device throughput behaviour analysis. Its application to our four benchmarks provides the throughputs shown in Fig. 2 for various FPGA chunk sizes. Each combination of the four available CPU cores (CCs) and the FPGA core (FC) are considered. As expected, CPU-only implementations are independent of data chunk sizes for all the four applications. When FPGA is involved in the heterogeneous process, the performance does not always increase with rising chunk sizes. Three types of anomalies can generally occur:

- As CS_F increases, Th_C remains constant and Th_F increases gradually to SL. The expectation is to obtain non-decreasing heterogeneous throughput Th_{Het} with rising CS_F due to less overhead for data movement and pipeline ramp-up and ramp-down. This might not happen creating a hill-like plot as occurs for AES application.
- The overall throughput Th_{Het} is expected to roughly equal the combined throughputs of the running units. However Th_{Het} might sharply drop as happens for FPGA data chunks of 2048 and 4096 in Nbody application.
- The heterogeneous performance could be much lower than the performance of an individual unit throughout all the chunk sizes examined, due to the negative effect of a unit's run on the others, in particular for memory access contention. This happens for HotSpot application.

Our aim is to propose a scheduler which finds the best chunk size for the base unit (FPGA) and resolves the first two issues mentioned above. The last anomaly is ignored here as its detection overhead would be high. These problems arise mainly when two non-overlapping periods occur with either FPGA working and the CPUs idling or vice-versa, which occur respectively when conditions of (3) or (4) hold true:

$$\frac{CS_F}{Th_F} > \frac{CS_{Rem}}{nC \times Th_C} \quad (3) \qquad \frac{CS_C}{nC \times Th_C} > \frac{CS_{Rem}}{Th_F} \quad (4)$$

Condition of (3) generally happens when $CS_F \times Th_C$ is large such that the CPUs process all the remaining data blocks before FPGA could handle its assigned data. AES (and also HotSpot, although its heterogeneous run fails to improve the performance) suffers this inefficiency at large FPGA data chunk sizes causing Th_{Het} to decrease. Condition of (4) occurs when $CS_C \times Th_F$ is large such that the FPGA processes CS_{Rem} before CPUs can handle their assigned data. Lack of accurate estimate of initial Th_{Rat}, $Th_{Init-Rat}$ at the start is the main reason for this inefficiency; if $Th_{Init-Rat}$ is set to a small value but the real Th_{Rat} is much larger, the measured and assigned CS_C would be too large for the CPUs to handle (initially, $CS_C = CS_F/Th_{Init-Rat}$). The performance collapse of heterogeneous execution for large FPGA chunk sizes in Nbody example is due to this problem. GEMM application does not suffer from the inefficiencies discussed.

An efficient scheduler delivering a high throughput should satisfy the following conditions:

1) CS_F gives FPGA throughput at SL in heterogeneous run;
2) Condition (3) is avoided;
3) Since a run-time scheduler initially needs a $Th_{Init-Rat}$ value, its negative effect (resulting in condition (4)) should be constrained and minimised.

The first two conditions can be satisfied if:

$$\frac{CS_F}{CS_{Rem}} \lessapprox \frac{Th_{F-SL}}{nC \times Th_C} \tag{5}$$

If (5) is set to equal, the CS_F measured gives the upper bound of SL region which is sufficient to overlap the FPGA and CPUs processes; if CS_F is set larger than that, CPUs will finally remain idle and if set to a smaller value, the operations will still overlap but FPGA operation *might* fall outside SL region. As SL in general stretches for a wide range of FPGA chunk sizes, in order to compensate for a possible marginal error for Th_{Rat} prediction, it is better to use a smaller value for CS_F than the one generated by (5) when set to equal.

In order to measure CS_F and CS_{Rem} through (5) for efficient concurrent execution, we first need to find Th_{F-SL} and Th_C. If these throughputs could be measured initially at run-time using few blocks for FPGA and CPU, it would only incur a low overhead and also restrict the inefficiency of (4) caused by a random $Th_{Init-Rat}$ to this short period. Given two small FPGA chunk sizes j and k where $j < k$, and following (1):

$$T_{Exe(j)} = T_{IO(j)} + T_{Proc(j)} \tag{6}$$

$$T_{Exe(k)} = T_{IO(k)} + T_{Proc(k)} \tag{7}$$

If $k/j = l$, from (6) we have:

$$l \times T_{Exe(j)} = l \times (T_{IO(j)} + T_{Proc(j)}) \tag{8}$$

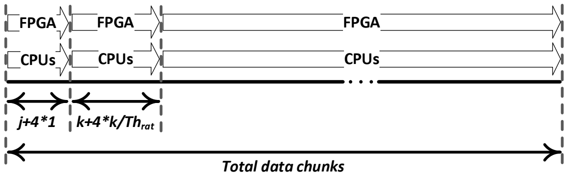


Fig. 3: Dataset partitioning by the proposed scheduler

Since $l \times T_{Proc(j)} = T_{Proc(k)}$, if we subtract (7) from (8):

$$\begin{aligned} l \times T_{Exe(j)} - T_{Exe(k)} &= l \times T_{IO(j)} - T_{IO(k)} \\ &= T_{IORed(k-j)} \end{aligned} \tag{9}$$

where $T_{IORed(k-j)}$ is the FPGA communication time reduced when data chunk size is increased by $k-j$. Since this reduction becomes almost zero at SL, the average total time required to execute one block at SL of FPGA can be measured as:

$$\begin{aligned} T_{Exe-SL(1)} &= \frac{T_{Exe(j)}}{j} - \frac{T_{IORed(k-j)}}{k-j} \\ &= \frac{T_{Exe(j)}}{j} - \frac{l \times T_{Exe(j)} - T_{Exe(k)}}{k-j} \end{aligned} \tag{10}$$

If $j = 2^n$ and $k = 2^{n+1}$, then (10) simplifies to (11) or (12):

$$T_{Exe-SL(1)} = \frac{T_{Exe(k)} - T_{Exe(j)}}{j} \tag{11}$$

$$Th_{F-SL} = \frac{j}{\frac{k}{Th_{F(k)}} - \frac{j}{Th_{F(j)}}} \tag{12}$$

and $Th_{F/C}$ which is the throughput ratio of FPGA to *all* CPU cores combined can be measured as:

$$Th_{F/C} = \frac{Th_{F-SL}}{nC \times Th_C} \tag{13}$$

Knowing $Th_{F/C}$ and the total data chunk size CS_{Tot} and following from (5), CS_F can be measured as:

$$CS_F = \frac{CS_{Tot} \times Th_{F/C}}{1 + Th_{F/C}} \tag{14}$$

Fig. 3 shows how the dataset is assigned to FPGA and CPU by the scheduler for processing. In summary:

- j data blocks are assigned to FPGA and one block to every CPU core, hence $Th_{Init-Rat} = j$. Within this period load imbalance could occur due to not knowing the throughput ratio and assigning a fixed number of tokens to different compute units. However as the chunk sizes assigned are low, the possible imbalance is marginal. $Th_{F(j)}$ is recorded.
- k data blocks are assigned to FPGA and with non-SL Th_{Rat} measured at the end of preceding step, a proportional number of tokens are assigned to each CPU core for rough balance. $Th_{F(k)}$ and Th_C are recorded. As CPU throughput is constant irrespective of the data chunk size assigned, the minimum chunk size would suffice to detect its throughput.
- With $Th_{F(j)}$, $Th_{F(k)}$ and Th_C measured, CS_F is calculated using (12), (13) and (14).

D. Performance Evaluation

The measurements and results of applying the proposed scheduler to the four applications are reported in Table I. For each application, the obtained heterogeneous throughput Th_{Het} can be compared against the Dynamic scheduling results where all

TABLE I: Measurements and results by the proposed scheduler (throughputs are in blocks/sec)

	AES	HotSpot	GEMM	Nbody
$\mathbf{Th_{F(j)}}$ [*]	224046	29077504	129135	655
$\mathbf{Th_{F(k)}}$ [*]	252568	36170957	221740	1287
$\mathbf{Th_{C}}$	165563	47080448	161913	1092
$\mathbf{Th_{SL}}$	289409	47842099	783877	37430
$\mathbf{Th_{F/C}}$	0.44	0.25	1.22	8.57
$\mathbf{CS_{F}}$	31872	179	503	8847
$\mathbf{Th_{Het}}$	806597	131072000	1038194	30303

[*] For better results, j and k are roughly decided based on total data chunk size; for GEMM, HotSpot and Nbody: $j = 8$ and $k = 16$, and for AES: $j = 128$ and $k = 256$.

TABLE II: Throughput comparison of the proposed scheduler vs. the best results obtained with Dynamic scheduling (Fig. 2)

	AES	HotSpot	GEMM	Nbody
Performance Comparison	86.23%	82.50%	94.06%	111.51%

compute units are involved and all possible chunk sizes are examined and the best Dynamic throughput is selected (illustrated in Fig. 2). This comparison is also reported in Table II.

The performance obtained by the proposed scheduler is comparable to the best performance of Dynamic partitioning. For Nbody application, the overall throughput is even increased because our scheduler resolves the condition of (4) which generally occurs when FPGA throughput is much higher than CPU throughput. For GEMM application, a very high heterogeneous throughput is achieved especially when taking into account the small overhead initially incurred in the proposed scheduler to find the devices throughputs. As can be observed in Table I, the small drop in the performance of AES and HotSpot is due to less accurate initial timing measurements of FPGA or CPU executions, hence resulting in less accurate evaluations of Th_C, $Th_{F(j)}$ or $Th_{F(k)}$. The advantage of the proposed scheduler is that the chunk size of compute units is determined initially with a low overhead while Dynamic partitioning needs to test all chunk sizes and select the best.

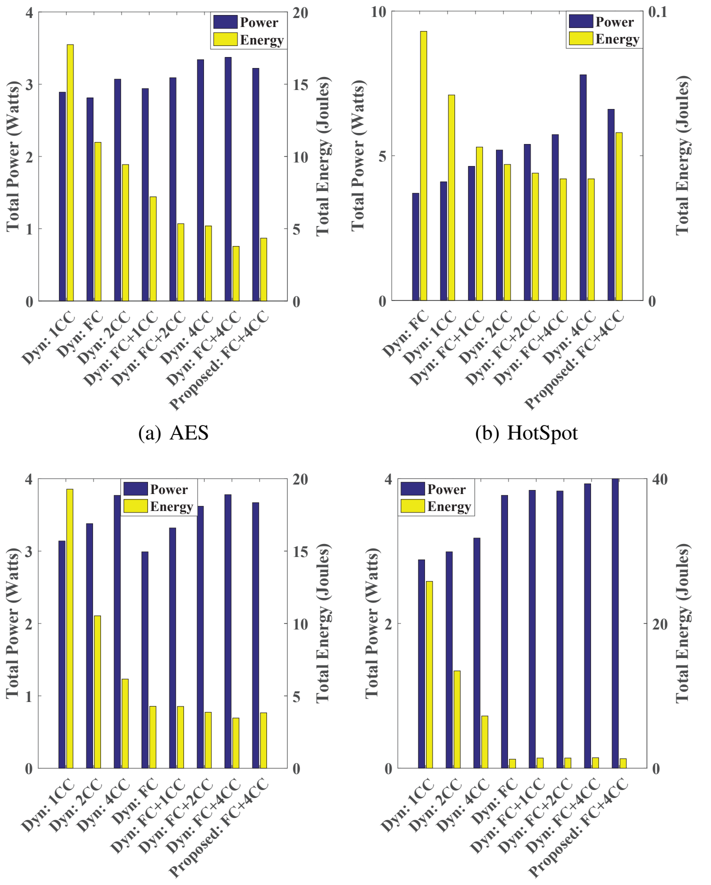
E. Power and Energy Evaluation

ZCU102 board has a PMBus (Power Management Bus) power control and monitoring system for reading power and current values every 0.05 sec. The power values we measure are for the combined power consumptions of LPD (low-power domain), FPD (full-power domain), and PLPD (PL power domain). Energy consumption is the integration of power readouts within the run time. Fig. 4 illustrates power/energy evaluations for the four applications on ZCU102. Dynamic scheduling results are shown for all platform configurations and the best FPGA chunk size for each configuration. The results for our proposed scheduler are for full utilisation of the compute units. In general, with every extra unit added to the configuration, the energy consumption improves despite the power increase caused by the extra unit.

Overall, the power/energy costs obtained by our scheduler are comparable to the best power/energy costs obtained by Dynamic partitioning. HotSpot and AES show more noticeable higher energy costs due to the lower throughputs obtained for these applications.

V. Conclusion

This paper introduces a dataset partitioning strategy for parallel execution of applications implemented in a heterogeneous device composed of FPGA and CPUs. The proposed scheduler highly restricts the overhead of device throughput estimations in order to improve the heterogeneous performance. The throughput, power, and energy evaluations obtained by the application of the proposed scheduler on a ZCU102 board prove comparable to the best evaluations when all the data chunk sizes are examined. As future work, we plan to explore how the proposed scheduler could be improved for irregular applications whose behaviour depends not only on the data set size, but also on its contents.



(a) AES

(b) HotSpot

(c) GEMM

(d) Nbody

Fig. 4: Power/energy evaluations

References

[1] O. Arcas-Abella *et al.*, "An empirical evaluation of High-Level Synthesis languages and tools for database acceleration," in *FPL '14*, 2014, pp. 1–8.

[2] "Xeon+FPGA platform for the data center," www.ece.cmu.edu/~calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf, accessed: 2018-04-02.

[3] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," ser. ISCA '14, 2014, pp. 13–24.

[4] "Managing reconfigurable FPGA acceleration in a POWER8-based cloud with FAbRIC," https://openpowerfoundation.org/blogs/fabric-fpga-cloud/, accessed: 2018-04-02.

[5] P. Meng *et al.*, "FPGA-GPU-CPU heterogeneous architecture for real-time cardiac physiological optical mapping," in *FPT '12*, 2012, pp. 37–42.

[6] K. H. Tsoi and W. Luk, "Axel: A heterogeneous cluster with FPGAs and GPUs," ser. FPGA '10, 2010, pp. 115–124.

[7] A. Vilches *et al.*, "Adaptive partitioning for irregular applications on heterogeneous CPU-GPU chips," *Procedia Computer Science*, vol. 51, pp. 140–149, 2015.

[8] J. Nunez-Yanez *et al.*, "Simultaneous multiprocessing in a software defined heterogeneous FPGA," *The Journal of Supercomputing*, 2018.

[9] "SDSoC user guide," www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1027-sdsoc-user-guide.pdf, accessed: 2018-04-06.

[10] "Intel TBB," www.threadingbuildingblocks.org, accessed: 2018-04-06.

[11] A. Navarro *et al.*, "Strategies for maximizing utilization on multi-CPU and multi-GPU heterogeneous architectures," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 756–771, 2014.