Runtime Support for Adaptive Power Capping on Heterogeneous SoCs

Runtime Support for Adaptive Power Capping on Heterogeneous SoCs

Yun Wu
School of Electrical, Electronic and Computer Science
Queen's University Belfast
Belfast, United Kingdom
Email: yun.wu@qub.ac.uk

Dimitrios S. Nikolopoulos
School of Electrical, Electronic and Computer Science
Queen's University Belfast
Belfast, United Kingdom
Email: d.nikolopoulos@qub.ac.uk

Roger Woods
School of Electrical, Electronic and Computer Science
Queen's University Belfast
Belfast, United Kingdom
Email: r.woods@qub.ac.uk

***Abstract*—Power capping is a fundamental method for reducing the energy consumption of a wide range of modern computing environments, ranging from mobile embedded systems to datacentres. Unfortunately, maximising performance and system efficiency under static power caps remains challenging, while maximising performance under dynamic power caps has been largely unexplored. We present an adaptive power capping method that reduces the power consumption and maximizes the performance of heterogeneous SoCs for mobile and server platforms. Our technique combines power capping with coordinated DVFS, data partitioning and core allocations on a heterogeneous SoC with ARM processors and FPGA resources. We design our framework as a run-time system based on OpenMP and OpenCL to utilise the heterogeneous resources. We evaluate it through five data-parallel benchmarks on the Xilinx SoC which allows fully voltage and frequency control. Our experiments show a significant performance boost of 30% under dynamic power caps with concurrent execution on ARM and FPGA, compared to a naive separate approach.**

***Index Terms*—OpenCL; ARM; FPGA; Power Capping; DVFS; Streaming; Data Partition**

I. Introduction

Energy consumption is the most significant limitation of modern servers for high-performance and cloud computing. Despite advances in heterogeneous systems architecture and programming support, effective management of the limited power and energy resources available to servers remains a key challenge [21]. Recently, the server market witnesses an increased penetration of embedded heterogeneous SoCs as server substrates. Such solutions are becoming attractive in both industry and academic settings [11]. For example, ARM-based SoCs with Field-Programmable Gate Array (FPGA) logic have shown significant advantages in power consumption and efficiency in executing highly parallel computation [14]. Other examples of the efficient use of heterogeneous SoCs in server setups are the ZCluster [18] and Zedwulf [19], which improves performance over similar clusters based solely on either ARM CPUs or FPGA enabling elasticity to trade power with performance. Compared to commodity servers, these SoC solutions improve energy-efficiency out of the box. However, optimal use of the limited power resources of these SoCs requires significant involvement from software, which is the problem that we investigate in this paper.

By restricting the peak power consumption of a compute node, power capping is a fundamental technique to achieve better energy efficiency on servers [17]. Largely based on Dynamic Voltage-Frequency Scaling (DVFS), power capping has been widely adopted for homogeneous systems [17] [8] while recent work on heterogeneous systems with CPUs and GPGPUs system has shown good potential from power capping on such systems without compromising performance [16]. For a fixed, static power cap set on a system, performance can be maximised by controlling the application degree of parallelism [7], controlling voltage & frequency, or optimising the sharing of hardware resources [12]. Our own earlier work [25] has demonstrated performance optimisation under fixed power caps hybrid ARM/FPGA SoCs. Unfortunately, static power capping is insufficient as it fails to capture workload variation that enable elastic allocation and allocation of resources for minimising energy consumption. Furthermore, prior work on power capping for heterogeneous platforms assumed workload allocations that leveraged a single type of computational resources (e.g. ARM or FPGA), instead of dynamic partitioning and simultaneous execution of workload tasks on all heterogeneous resources.

In this work, we propose a new adaptive power capping technique for OpenCL kernels on heterogeneous SoC-based servers based on ARM and FPGA accelerators. By using OpenMP as a higher-level abstraction for orchestrating the partitioning and concurrent execution of OpenCL kernels between hard cores and reconfigurable accelerators, we combine DVFS, thread control and data partitioning, to maximise performance under an adaptive constraint. Specifically, this paper makes the following contributions:

1) A run-time system of both hardware and software framework with concurrent OpenCL streaming execution model for hybrid ARM/FPGA SoCs.
2) An adaptive power capping method at run-time based on combination of DVFS, data partitioning and resource allocation.
3) An experimental campaign of adaptive power capped performance optimisation with five applications on a commercial Xilinx Zynq platform, showing up to 30%

performance improvement under a dynamic power cap, also scaled proportionally to the power cap.

The rest of this paper is organised as follows. Section II investigates related works. The proposed hardware and software frameworks are introduced in Section III. Our adaptive power capping method is introduced in Section IV, along with our method for power measurement and modeling. We present the implementation and experimental evaluation of our works in Section V. Section VI summaries our findings.

II. Background

High performance computing clusters and datacentres are using increasingly more embedded systems components and software methods to reduce their power draw and cooling costs [6], [15]. Unfortunately, common techniques to reduce power draw, such as power scaling of hardware components, power capping and duty cycling come at a performance cost [22].

Heterogeneous SoCs used in embedded systems have gained traction as building components of high performance computing systems [11]. Research on the use of ARM processors in cloud datacentres has demonstrated significant advantages in energy efficiency compared to other architectures [24]. Low-power ARM processors achieve better energy efficiency than well provisioned Intel processors designed for the server market [24] [20].

Systems based on FPGAs have also achieved higher energy efficiency than both general-purpose CPUs and Graphics Processing Units (GPUs) [23], in a range of applications where algorithms can both tolerate and leverage variable precision. The integration of FPGA fabrics with general purpose processors and the advent of high-level parallel programming languages as hardware synthesis tools have also improved substantially the programmability of systems with FPGAs. As an example, the Xilinx Zynq platform boasts an ARM processor for running Linux and common software stacks and Programmable Logic (PL) for acceleration. The platform supports data transmission through the AMBA AXI bus into the FPGA fabric [26]. This allows for efficient and workload-specific designs of accelerator-enabled servers and mobile systems [9]. Clusters of hybrid ARM-FPGA SoCs using ARM processors as a data transfer controller for distributed FPGA processing such as the ZCluster, are also beginning to emerge and demonstrate superior performance than homogeneous clusters for data-intensive applications by elasticity, e.g. based on Hadoop [18]. Zedwulf is another SoC cluster which allows exploration of the performance-power trade-off for sparse graph applications [19].

Power capping is a common and widely used power management method by maintaining the peak power consumption under a hard bound and allocating power capacity for either single components or entire platforms according to dynamic workload characteristics [10]. It can improve energy efficiency and reduce cost of ownership if system software is aware of and can achieve high performance under power constraints [17]. By adapting the power capping at the run-time, the adaptive power capping can deal with various power cap responsive to the system variation [13]. In homogeneous nodes and clusters, DVFS has been well investigated and established as a method to maintain power caps and improve energy-efficiency [17] [8]. Recent work on heterogeneous CPU-GPGPU systems has demonstrated power capping is viable on heterogeneous platforms via workload allocation between processors of different architectures [16]. Other adaptive techniques that maximize the performance under power caps include core allocation via thread packing or scheduling [7] and hardware resource management on multi-core processors in homogeneous systems [12]. Figure 1 shows the taxonomy of existing adaptive power capping techniques.

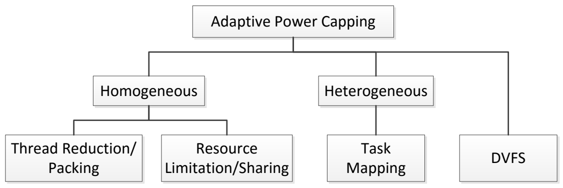


Fig. 1: Taxonomy of Adaptive Power Capping Techniques

Power capping on heterogeneous ARM-FPGA SoCs was also studied in our earlier work [25], where we used task mapping and DVFS either exclusively on ARM processors or exclusively on the FPGA. Our prior work did not exploit the possibility of power capping adapting kernel distribution with data partition and concurrent execution across ARM/FPGA components. Such a distribution would seemingly be feasible in software with parallel programming languages such as OpenCL. However, despite the release of toolkits from Xilinx and Altera to support OpenCL on PCI-e based FPGAs, these toolkits are designed based on Hardware Description Language (HDL) compiler and do not support any run-time power management [4] [27]. In this work we design and implement a new adaptive run-time systems which enables performance optimisation under power caps, via distributing data and kernel execution between hard cores and FPGAs.

III. Hardware and Software Deployment

We propose both new hardware and new software infrastructure to achieve performance optimisation under adaptive power caps on heterogeneous SoCs. Our framework comprises streaming accelerators of OpenCL kernels on the Xilinx Zynq PL, OpenCL device drivers and DVFS firmware for power management on both the Processing System (PS), ARM proc, and PL. By further combining OpenCL and OpenMP, we can achieve coordinated execution across all computational resources of the Zynq at run-time.

A. Hardware Deployment

Figure 2 shows the proposed hardware infrastructure of power capping using the streaming accelerators on PL.

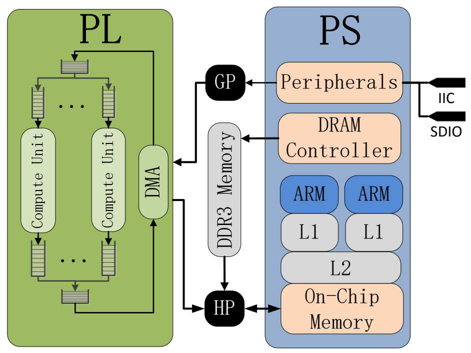


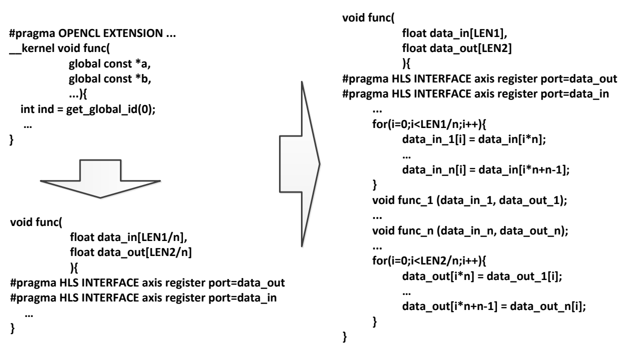
Fig. 2: Power Capping Hardware Infrastructure

On the PS side, the AXI Master General Port (MGP) is configured for writing data from the PS to the PL side, while the AXI Slave interface (High-Performance Port (HP)) is configured for transmitting data from the PL to the PS side. The I^2C peripheral I/O is configured to enable PS side PMBus access while the SD peripheral I/O is configured to allow Linux booting and data preservation through an SD card.

On the PL side, accelerators with multiple processing units (PU) are generated for different application kernels. This is achieved by generating multiple instances of OpenCL kernels with a streaming FIFO interface through the Vivado 2014.3 toolset. We generate varying numbers of PUs for each kernel, which is configured to execute on the PL through the OpenCL device driver. More specifically, we produce kernel versions with varying input block sizes for streaming data (e.g. 16, 64 and 256 bytes) and with varying data partitioning between the PS and the PL memories, under the constraint that these meet the power cap.

Our work aims at more efficient power management techniques on hybrid processor/FPGA platforms. We do not exhaustively optimize application design on the FPGA. We use the generic floating-point OpenCL kernel for generating accelerators on the Zynq PL through the Xilinx Vivado HLS. This process achieves satisfactory quality for small scale hardware accelerator design. Figure 3 illustrates an example of the automatic accelerator code generating process for the Xilinx Vivado HLS: 1) Assuming the input and output are divided for n PUs, the OpenCL kernel is transformed into C code with streaming pragma; 2) Multiple instance of C code with streaming pragma is combined into one for generate multiple PUs inside the accelerator.

The generated Hardware Description Language (HDL) from HLS is packed into an IP block for the Vivado synthesis tool. By connecting the AXI DMA to the accelerator, it is mapped to a fixed address which allows the PS to drive the computation by writing data to AXI slave FIFO and reading data from the AXI master FIFO through HP back to the ARM processor. The Vivado script helps automate the process at compile-time for each benchmark with a variable number of PUs and streaming data size, e.g. input size $LEN1$ and output size $LEN2$ shown in Figure 3, where the OpenCL kernel is transformed into the input C code for Vivado HLS with pragma of streaming interface and multiple accelerators are instantiated.



```
#pragma OPENCL EXTENSION ...
__kernel void func(
            global const *a,
            global const *b,
            ...){
   int ind = get_global_id(0);
    ...
}
```

```
void func(
            float data_in[LEN1/n],
            float data_out[LEN2/n]
            ){
#pragma HLS INTERFACE axis register port=data_out
#pragma HLS INTERFACE axis register port=data_in
    ...
}
```

```
void func(
            float data_in[LEN1],
            float data_out[LEN2]
            ){
#pragma HLS INTERFACE axis register port=data_out
#pragma HLS INTERFACE axis register port=data_in
      ...
      for(i=0;i<LEN1/n;i++){
            data_in_1[i] = data_in[i*n];
            ...
            data_in_n[i] = data_in[i*n+n-1];
      }
      void func_1 (data_in_1, data_out_1);
      ...
      void func_n (data_in_n, data_out_n);
      ...
      for(i=0;i<LEN2/n;i++){
            data_out[i*n] = data_out_1[i];
            ...
            data_out[i*n+n-1] = data_out_n[i];
      }
}
```

Fig. 3: Streaming Accelerator Code Generation Example

We note that all the generated accelerators are processed at compile-time which produces bit files for the PL configuration. These bit files are saved on an SD card with an identity of kernel name and input data size, for later run-time reconfiguration based on data partitioning choices made by our framework. The resource utilization data and clock performance from Table II to VI are also recorded as plain text which is read at run-time for power and energy estimation of the PL.

B. Software Deployment

The proposed software infrastructure includes coordinated execution of OpenCL workgroups across the SoC and power management for improving performance under power caps.

1) Coordinated OpenCL Execution: We use OpenCL to support coordinated parallel execution across the PS and PL resources of the Zynq. We use the open-source PoCL library [3] and OpenMP library [2] constructs to orchestrate the distribution of OpenCL workgroups between the PS and the PL. We specifically use OpenMP to parallelize the kernel queuing loop for each OpenCL device in the platform. Figure 4 shows the execution model combining OpenCL and OpenMP on the Zynq based SoC.

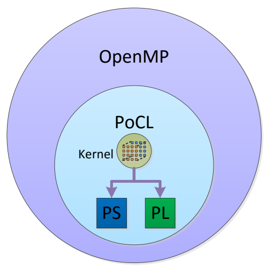


Fig. 4: Coordinated Execution Model

In the PoCL implementation, we use multi-threading on the PS side to implement naive OpenCL while using a loadable

OpenCL device driver based on PoCL for the streaming accelerator on the PL, including the HP and DMA interfacing. With this device driver, the kernel I/O arguments are aggregated into a serial data stream to send or receive from the PL accelerator as an OpenCL buffer before or after the kernel queuing. Figure 5 illustrates the device driver for OpenCL buffering and execution on streaming accelerators.

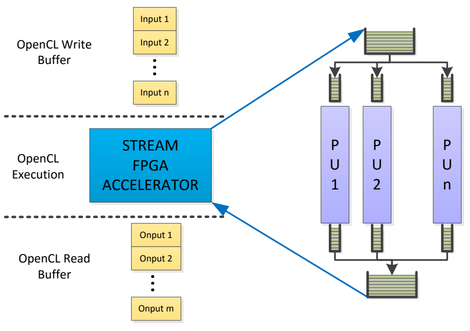


Fig. 5: OpenCL Execution on Streaming Accelerator

Notice that since there is no real kernel instruction generated for PL, the corresponding kernel compilation at run-time is a NULL function. As PL streaming accelerator is synthesized from Xilinx Vivado HLS, the stream data FIFO is utilized as I/O memory interface. Hence, the OpenCL memory is allocated on main Double Data Rate SDRAM (DDR) memory, where the generic OpenCL memory allocation function works for the PL.

With OpenCL support available on both the PS and the PL, multiple kernel workgroups can be concurrently queued on all computational components of the Zynq using OpenMP loops. We also use OpenMP critical sections to protect the OpenCL buffer from concurrent writers from both the PS and the PL.

2) Power and Performance Management: To simultaneously scale the power and performance on the PS and the PL, we built a custom Linux kernel to allow more than the nominal frequency tuning steps of the Zynq platform, specifically set to ten steps as $[112, 134, 149, 167, 191, 223, 267, 334, 445, 667]$ Mhz. This is achieved by searching and recording all voltage and frequency setting values of the Zynq SoC between the maximum and minimum nominal frequencies. We add a PMBus driver to record power samples from the PS through the $sysfs$ interface. We have also developed an I2C scaling interface for both the PS and the PL, to enable voltage tuning ranging from 0.86 to 1.00 $Volt$ with fine-grain 0.02 $Volt$ intervals. By choosing a different I2C page address of power rail, we can control DVFS of both the PS and the PL on the PS side.

To support power modeling of computational kernels for the purpose of power capping, we also obtain detailed hardware event rates for OpenCL kernel execution, including cycle/instruction rates and cache and memory accesses and misses at run-time. We achieve this by enabling the ARM coprocessor register 15 [5] from userspace and configuring $cp15$ inline assembly in the Linux kernel as a loadable kernel module. This module is loaded at the beginning of a profiling session and unloaded at the end of a profiling session. Performance information is obtained by the inline assembly code via accessing the ARM Performance Monitor Unit (PMU). The code for this is inserted into the OpenCL kernel at run-time in order to profile both kernel scheduling (queuing, data transfers) and kernel execution performance. By combining power estimation and performance profiling at run-time, we directly deduce energy consumption.

IV. Adaptive Power Capping

A. Overview

We model power and energy consumption of the system and propose a process to adapt the power cap at runtime by modifying the partitioning of data between the PS and PL, while also dynamically reconfiguring the PL to tune its buffer size and PU number. Figure 6 illustrates our adaptive power capping run-time system for the Zynq SoC.

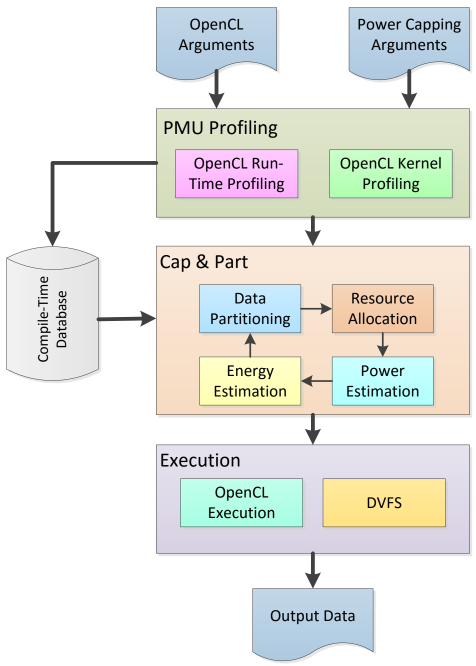


Fig. 6: Adaptive Power Capping Run-Time

The run-time involves three major steps. First, the OpenCL operational power on both PS and PL are estimated before workload execution based on compile-time power modeling and run-time PMU profiling. To reduce the PMU profiling overhead at run-time, the kernel is only executed once and the real-time performance is estimated based on the kernel execution rate calculated and OpenCL workgroup size. Following this step, we estimate the power consumption of a kernel on the PS or a PL with a varying number of PUs, and select a DVFS operating point for both the PS and PL that keeps overall power under the cap. By configuring the hardware resources of the PL, we evaluate alternative data partitioning between PS and PL, to achieve the best performance under the power cap. Finally, both PS and PL are scaled at the desired DVFS operating points while the desired distribution between the PS and PL is achieved via the OpenCL Application Programming Interface (API) of our developed framework.

B. Power Estimation

Power estimation is necessary for evaluating the impact of DVFS and data partitioning on power consumption. We propose a trained power model at compile-time and deployed using additional run-time information to estimate the power consumption. This is achieved through both compile-time and run-time profiling where the linear regression is adopted to generate the model.

Five benchmarks with different data size as well as different PU number are adopted as the workload for both PS and PL. At compile-time, we use linear regression to obtain a simple model for run-time average power estimation. Figure 7 illustrates the compile-time power profiling flow.

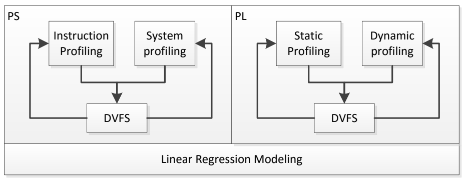


Fig. 7: Compile-Time Power Profiling and Modeling

We estimate PS power (P_{ps}) as the sum of Operating system (OS) power consumption (P_{os}) and workload power consumption ($P_{workload}$) with input from system profiling and instruction profiling, shown in Figure 7. The OS power is modeled when no workload runs in the system, while the workload power is modeled by running different workloads with variation in their instruction mixes. Equation 1 and Equation 2 depict the both modeling formulas.

$$\begin{aligned} P_{os} = \; & 154.23 + 571.48 \cdot v_{ps}^2 \\ & -510.95 \cdot v_{ps} + 0.173 \cdot v_{ps}^2 \cdot f_{ps} \end{aligned} \tag{1}$$

$$\begin{aligned} P_{workload} = \; & \gamma \cdot v^2 \cdot f \cdot IPC \\ & +(P_{alu} \cdot r_{alu} + P_{fpu} \cdot r_{fpu} \\ & +P_{neon} \cdot r_{neon} + P_{mem} \cdot r_{mem}) \end{aligned} \tag{2}$$

where P_{alu},P_{fpu},P_{neon},P_{mem} correspond to the power consumption of different instruction types, IPC is instructions per cycle, and r_{alu},r_{fpu},r_{neon},r_{mem} are ratios of each instruction type to the total number of instructions, which are calculated through PMU profiling for both kernel queuing and kernel execution.

Instruction power for ALU, FPU, NEON, and memory hierarchy is obtained from Equation 3.

$$P_{inst} = \alpha \cdot v_{ps}^2 \cdot f_{ps} + \beta \cdot v_{ps} \tag{3}$$

where v_{ps} is the voltage in V, f_{ps} is the frequency in MHz and corresponding α and β for different instruction types are given in Table I

TABLE I: Instruction Power Model Polynomials

Resource	α	β
alu	0.149	0.270
FPU	0.112	-2.740
neon	0.135	-3.840
Memory	0.718	4.900

We estimate PL power consumption as a sum of idle power and dynamic power, obtained from static profiling and dynamic profiling respectively (Figure 7). We use linear regression of the synthesized information, e.g. resource utilization and clocking rate, as shown in Equation 4

$$\begin{aligned} P_{pl} = \; & v_{pl}^2 \cdot (f_{pl}) \cdot (\varphi_1 \cdot r_{bram} + \varphi_2 \cdot r_{lut} + \varphi_3 \cdot r_{dsp}) \\ & +v_{pl} \cdot (f_{pl}) \cdot (\gamma_1 \cdot r_{bram} + \gamma_2 \cdot r_{lut} + \gamma_3 \cdot r_{dsp}) \\ & +P_{idle} \end{aligned} \tag{4}$$

where r_{bram}, r_{lut}, and r_{dsp} are the resource utilization ratios of block ram, look-up-table and DSP48e. The polynomials φ_{1-3} and γ_{1-3} are for dynamic power estimation related to specific design at run-time. P_{idle}, as shown in Equation 5, is the PL idle power consumption when the accelerator is not activated by data streaming:

$$P_{idle} = \delta_1 \cdot r_{bram} + \delta_2 \cdot r_{lut} + \delta_3 \cdot r_{dsp} + 26.598 \tag{5}$$

where δ_{1-3} are polynomials of various resource utilization to estimate P_{idle} at run-time.

At run-time we obtain PMU information of both the OpenCL API and kernel executions for the polynomials in power model (Equation 2) built at compile-time. We adopt the simple fitting of Equation 4 by reading a saved PL resource utilization value. Through estimating the average power for both PS and PL at run-time for OpenCL kernels the energy consumption is obtained by integrating power and execution time.

C. Adaptive Capping

We execute the OpenCL kernel using data-level parallelism by partitioning the kernel between PS and PL. The term data size indicates the streaming data block size used to transfer data between PS and PL while the number of PUs indicates the number of instantiated computational units used for the part of the kernel executed on the PL.

For a given peak power level, P_{cap}, set by the system designer or administrator, our run-time adapts DVFS settings to data partitioning between PS and PL and to the choice of different numbers of PUs on the PL. Equation 6 shows the formulated adaptive power capping problem using non-linear programming.

$$\begin{cases} P_{ps}(F_{ps}, V_{ps}, IPC_{ps}, D_{ps}) + P_{pl}(F_{pl}, V_{pl}, R, D_{pl}) \leq P_{cap} \\ \min P_{ps} \cdot T_{ps} + P_{pl} \cdot T_{pl} \\ \min max(T_{ps}, T_{pl}) \end{cases} \tag{6}$$

where P_{ps} and P_{pl} are power model functions for the PS and PL, D is the input data partition size corresponding to the implemented resource allocation on the PL, F is frequency, V is voltage, IPC is the instructions per cycle, R is the PL resource utilization described in Section III, T is the executing time calculated by cycle over frequency and P_{cap} is the capping threshold.

Algorithm 1 describes the entire adaptive power capping process after run-time PMU profiling. $|SIZE|$ is the cardinality of the set of all data size options. We do not support arbitrary data partitioning due to the explosive synthesis time for PL implementation. We support a fixed set of data buffer size opions instead. $INST$ and CYC are the instruction and cycle count of both PS and PL execution as recorded by the PMU. By involving the APMonitor Optimization Suite (APM) [1], the Cap & Part algorithm behaves as follows:

Algorithm 1: Cap & Part

```
Data: $SIZE$, $P_{ps}$, $P_{pl}$, $INST$, $CYC$, $R$, $V$, $F$,
      $P_{cap}$, $D_{ps}$, $D_{pl}$
Result: $PT$, $VF$
begin
    $PT \longleftarrow \emptyset, VF \longleftarrow \emptyset, T \longleftarrow \emptyset$
    for $j \leftarrow 1$ to $|SIZE|$ do
        $D_{pl} \leftarrow SIZE$
        for $i \leftarrow 1$ to $|D_{pl}|$ do
            $D_{ps}(i) \longleftarrow SIZE_j - D_{pl}(i)$
            if $D_{ps}(i) \geq 0$ then
                $(vf, flag)$ = APM($P_{ps}$, $P_{pl}$, $INST$,
                $CYC$, $R$, $V$, $F$, $D_{ps}(i)$, $D_{pl}(i)$, $P_{cap}$)
                if $flag\ is\ true$ then
                    $pf_{ps} = D_{ps}(i)/(CYC_{ps} * f_{ps})$,
                    $f_{ps} \in vf$
                    $pf_{pl} = D_{pl}(i)/(CYC_{pl} * f_{pl})$,
                    $f_{pl} \in vf$
                    $pf = min\{pf_{ps}, pf_{pl}\}$
                    $PF = PF \cup pf$
                    $PT = PT \cup \{D_{ps}(i), D_{pl}(i)\}$
                    $VF = VF \cup vf$
                end
            end
        end
    end
    $ind$ = $\max(PF)$
    $PT$ = $PT_{ind}$, $VF$ = $VF_{ind}$
end
```

By involving the APMonitor Optimization Suite (APM) [1], the Cap & Part algorithm behave as:

1) Firstly, three empty sets, PF, VF and PT, are created to record performance, DVFS operating point and data partition. The $SIZE$ is assigned to D_{pl} and each time the $D_{ps}(i)$ is obtained by subtracting $D_{pl}(i)$ from the current overall data size $SIZE_j$, which enables ergodic of all partitioning combinations between PS and PL. The data partition is based on OpenCL workgroup as shown Figure 8, where x, y, z represents the size of each workgroup dimension and the partitioned size is of subscript $_{PS}$ and $_{PL}$. We support one dimension dynamic partition at the moment.

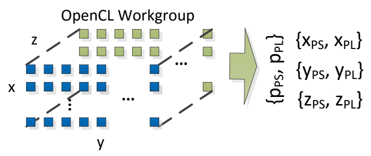


Fig. 8: OpenCL Workgroup Partitioning

2) The operating point of PS and PL for a given data partitioning is written into a nonlinear programming model file. An APM modeling template for APM solver is created which consists of:
   - Parameters section with coefficients in Equation 1-5.
   - Variables section with the range of voltage, frequency and data size.
   - Equation section with objective function in Equation 6.

   Having updated the model parameter with the input of function $APM(\bullet)$, we call the APM solver at run-time to produce a flag of solver success and DVFS options vf for the PS and the PL.
3) If the flag indicates successful solution, the data partitioning is recorded to set PT as well as DVFS options. The performance pf is calculated by partitioned data per cycle and recorded in set PF.
4) Finally, after going through all the data partition combinations, the performance of all the candidates in PF is compared and the index of maximum performance is used to select the corresponding data partition in PT and DVFS operating point in VF.

V. Experiments

We use five benchmark kernels as use cases to demonstrate our approach: Binomial Tree Vector computation (BT), FIR Filter (FF), Image Convolution (IC), Sparse Matrix-Vector Multiplication (SPMV), and Square Matrix Multiplication (MM). For BT the data size is the vector length; for FIR and IM it is the pixel number where each pixel is of size 7; for MM and SPMV it is the matrix size. We use a different number of PUs on the PL to process the same amount of data in parallel. The number of PUs for a given processing data size leads to varying resource utilization, performance and power consumptions on the PL. We used the XC7Z020 CLG484 - 1 AP SoC on Zynq 702 evaluation board with hardware and software details listed in Section III. We set power caps to the sum of both PS and PL power ranging from 300 to 600 mW with 100 mW intervals. The voltage is tuned between 0.86 V and 1.00 V with 0.02 V intervals.

A. Accelerator Implementation

Tables II-VI illustrate the resource utilization and performance of various applications kernels on the PL. Using our software support, the number of PUs and the streaming data block size are adapted to the power cap at run-time.

TABLE II: Binomial Tree Vector Addition Resource Utilization

Number	Size	LUT	BRAM	DSP48	Clock (MHz)
	16	2662 (5.00%)	6 (2.14%)	25 (11.36%)	150.51
1	64	2687 (5.05%)	6 (2.14%)	25 (11.36%)	164.47
	256	2723 (5.11%)	6 (2.14%)	25 (11.36%)	124.95
	16	10727 (20.16%)	24 (8.57%)	100 (45.45%)	164.47
4	64	10769 (8.57%)	24 (8.57%)	100 (45.45%)	169.29
	256	10827 (20.35%)	36 (12.86%)	100 (45.45%)	144.51
	16	21053 (39.57)	40 (14.29%)	200 (90.9%)	167.17
8	64	21128 (39.71%)	44 (8.57%)	200 (90.9%)	175.93
	256	21215 (39.88%)	72 (12.86%)	200 (90.9%)	170.5

TABLE III: FIR Filter Resource Utilization

Number	Size	LUT	BRAM	DSP48	Clock (MHz)
	16	7893 (14.84%)	2 (0.72%)	9 (4.09%)	170.91
1	64	7924 (14.89%)	5 (1.78%)	9 (4.09%)	170.09
	256	7955 (14.95%)	18 (6.43%)	9 (4.09%)	151.54
	16	15667 (29.45%)	3 (1.07%)	18 (8.18%)	169.58
2	64	15702 (29.52%)	9 (3.21%)	18 (8.18%)	166.78
	256	16151 (20.35%)	36 (12.86%)	18 (8.18%)	145.45
	16	31340 (58.91%)	5 (1.79%)	36 (16.36%)	165.26
4	64	31386 (58.99%)	18 (6.43%)	36 (16.36%)	168.43
	256	32261 (60.64%)	72 (25.71%)	36 (16.36%)	151.63

TABLE IV: Image Convolution Resource Utilization

Number	Size	LUT	BRAM	DSP48	Clock
	16	1213 (2.28%)	1 (0.35%)	3 (1.36%)	164.66
1	80	1204 (14.89%)	4 (1.42%)	3 (1.36%)	147.28
	16	4599 (8.64%)	7 (2.5%)	12 (5.45%)	176.03
4	80	4641 (8.72%)	11 (3.93%)	12 (5.45%)	169.06
	16	18448 (34.67%)	17 (6.07%)	48 (21.81%)	171.29
16	80	18643 (35.04%)	41 (14.64%)	48 (21.81%)	157.41

TABLE V: Sparse Matrix Vector Resource Utilization

Number	Size	LUT	BRAM	DSP48	Clock(MHz)
	16	3251 (6.11%)	6 (2.14%)	14 (6.36%)	163.37
1	64	3670 (6.89%)	8 (2.85%)	14 (6.36%)	167.67
	256	5079 (9.55%)	8 (2.85%)	14 (6.36%)	140.47
	16	11645 (21.89%)	6 (2.14%)	56 (25.45%)	169.41
4	64	11637 (21.87%)	8 (2.85%)	56 (25.45%)	169.18
	256	12045 (22.64%)	8 (2.85%)	56 (25.45%)	143.27
	16	23745 (44.63%)	10 (3.57%)	112 (50.9%)	142.17
8	64	23877 (44.88%)	10 (3.57%)	112 (50.9%)	143.67
	256	24297 (45.67%)	10 (3.57%)	112 (50.9%)	128.07

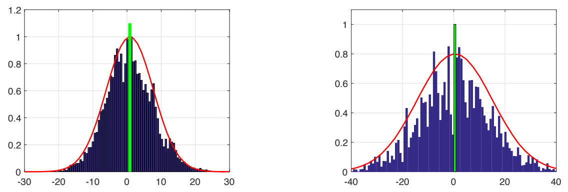
TABLE VI: Matrix Multiplication Resource Utilization

Number	Size	LUT	BRAM	DSP48	Clock (MHz)
	4	1223 (2.29%)	1 (0.35%)	3 (1.36%)	168.52
1	16	1107 (2.08%)	3 (1.07%)	3 (1.36%)	166.25
	64	2689 (5.05%)	9 (3.93%)	3 (1.36%)	142.49
	4	2493 (4.68%)	1 (0.36%)	12 (5.45%)	146.18
4	16	2892 (5.43%)	3 (1.07%)	12 (5.45%)	152.49
	64	9016 (16.94%)	11 (3.93%)	12 (5.45%)	126.79
	16	5524 (10.38%)	3 (1.07%)	24 (10.9%)	143.08
8	64	7243 (13.61%)	9 (3.21%)	24 (10.9%)	121.59

The floating-point implementation of all five benchmarks maintains relatively high performance with a clock rate of around 150 MHz. The overall resource utilization, ranging from $2-45\%$ of LUTs, $1-25\%$ of BRAM and $1-90\%$ of DSP, covers the variety of PL usages which is relatively robust for either power estimation and performance/power trade-off options. Ideally, arbitrary input data sizes should be supported on the PL and the PS. However, due to the limitation of time consuming synthesis, we only support up to three different input data sizes and PU number for each benchmark and these are used in this analysis to verify our adaptive power capping technique.

B. Power Estimation

We record the estimated power and compare it with the real-time physical power measurement. The power model is calculated for each combination of voltage and frequency, using also information from profiling. The real-time physical measurement is recorded separately with the PMU profiling process for both the PS and the PL throughout kernel execution. By checking the distribution of mean square error between estimated average power and the real-time physical measurement, Figure 9a shows the overall verification of PS power model for the five benchmarks with different data sizes while Figure 9b shows the overall verification of PL power model for the five benchmarks with both various data sizes and PU number.



(a) PS (b) PL

Fig. 9: Overall Power Model Verification

Figure 10 and Figure 11 show the comparison of estimated power with physical measurement for individual benchmarks.

The mean square error between estimated average power and real-time physical measurement fits the normal distribution. Through over 40 scenarios for the tested five benchmarks with different data sizes and PU numbers, we achieve less than 7% estimation error for the PS and no more than 15% error for the PL. We note that for the entire system we achieve less than 5% average power estimation error. Nevertheless, we use additional slack of 10% to cap the peak power and defend from errors in average power estimation at run-time.

Table VII shows the overhead of power monitoring and DVFS at nominal frequency, 666 MHz, on the PS. This overhead would be higher as frequency is scaled down. Table VIII shows the overhead of obtaining PMU information for power estimation at nominal frequency. The overheads include both OpenCL API and kernel execution on PS or PL. On the PS the kernel is executed once while for the PL the kernel is executed as many times as needed for processing all input

TABLE VII: Power Monitor and DVFS Overhead

	Power Monitor (μs)	Voltage Scaling (ms)	Frequency Scaling (ms)
Time	51.0	0.64	5.70

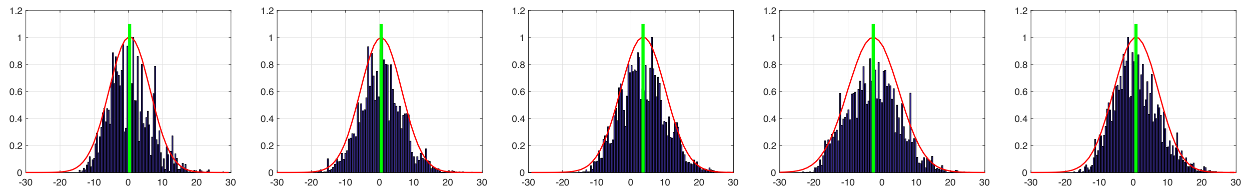

(a) BT (b) FIR (c) IM (d) MM (e) SPMV

Fig. 10: PS Power Model Verification

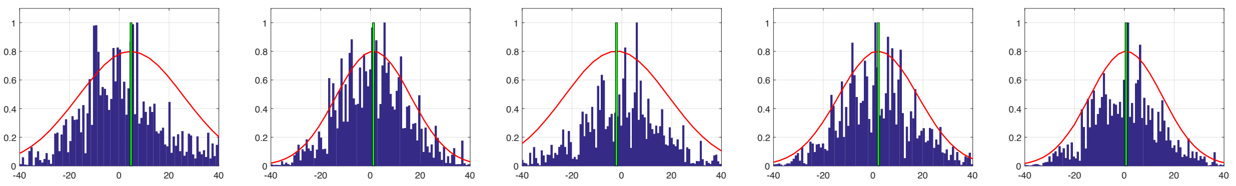

(a) BT (b) FIR (c) IM (d) MM (e) SPMV

Fig. 11: PL Power Model Verification

data. The result indicates significant overhead of OpenCL API when data streaming to the PL is used, both for run-time PMU profiling and the execution stage.

TABLE VIII: Run-Time PMU Overhead

	Benchmark	Size	PS (ms)	PL (ms)
Time	BT	16	0.41	47.16
		64	0.43	135.53
		256	0.45	337.94
	FIR	16	6.42	63.53
		64	6.43	175.47
		256	6.41	433.24
	IM	16	0.25	64.07
		80	0.25	219.38
	MM	4	4.6	11.79
		16	4.54	188.64
		64	4.8	542.12
	SPMV	4	0.39	1.47
		16	0.40	5.145
		64	0.50	18.0075

C. Adaptive Power Capping

We enforce caps on the sum of PS and PL power through DVFS and different data partitionings between the PS and PL, as shown in Figure 6. Table IX-XII illustrate the capped power and performance of five benchmarks using the proposed approach.

The 'Size' column indicates the input data size, where single value is the input data size of PS only, whilst the two added values $PSsize + PLsize$ represent concurrent kernel execution between the PS and the PL for a given data partitioning. The recorded DVFS operating points of the Zynq 702 evaluation board are shown in 'PS' and 'PL' columns. The selected device (PS or PL) for kernel execution is indicated by the annotation *. The performance column records the calculated normalized ratio of comparison between the computation times with adaptive power capping and the execution time on the PS only without DVFS.

We make the following observations on adaptive power capping results:

TABLE IX: Power Capping Parameter with 300 mW Cap

Benchmark	Size	PS	PL	PU No.	Performance
SPMV	256	(267 0.86)*	(29 0.86)	1	0.40
MM	64	(267 0.86)*	(166 0.90)	1	0.40
IM	80	(194 0.86)*	(33 0.86)	16	0.29
FIR	256	(267 0.88)*	(30 0.86)	1	0.67
BT	256	(267 0.88)*	(24 0.86)	1	0.67

TABLE X: Power Capping Parameter with 400 mW Cap

Benchmark	Size	PS	PL	PU No.	Performance
SPMV	256	(444 0.88)*	(25 0.86)	1	0.67
MM	48+16	(667 0.86)*	(166 0.90)*	1	1
IM	80	(333 0.90)*	(33 0.86)	16	0.5
FIR	256	(666 0.86)*	(30 0.86)	1	1
BT	192+64	(444 0.86)*	(65 0.88)*	1	0.89

TABLE XI: Power Capping Parameter with 500 mW Cap

Benchmark	Size	PS	PL	PU No.	Performance
SPMV	256	(667 0.90)*	(25 0.86)	8	1
MM	48+16	(667 0.88)*	(168 0.90)*	1	1.33
IM	48+16	(444 0.86)*	(171 0.86)*	16	0.83
FIR	192+64	(667 0.86)*	(136 0.88)*	1	1.33
BT	48+16	(667 0.86)*	(150 0.88)*	1	1.33

TABLE XII: Power Capping Parameter with 600 mW Cap

Benchmark	Size	PS	PL	PU No.	Performance
SPMV	48+16	(667 0.90)*	(169 0.90)*	4	1.33
MM	48+16	(667 0.96)*	(168 0.92)*	1	1.33
IM	48+16	(667 0.86)*	(154 0.86)*	16	1.25
FIR	48+16	(667 0.94)*	(170 0.86)*	1	1.33
BT	192+64	(667 0.94)*	(164 0.86)*	1	1.33

1) Co-processing is adopted with data partitioning between PS and PL given enough power budget beyond 500 mW. In this case performance goes up by a factor of 1.33× compared to execution on the PS only without DVFS.
2) PS only execution is mostly chosen with capping thresh-

old under 500 mW and PL is tuned with minimum voltage and frequency. By disassembling and debugging our run-time, we found that the OpenCL API and in particular the command queue mechanism dominates the overhead and calls for further optimizations.

3) A small PU number on the PL is chosen for coordinated execution due to the significant power increment related to resource utilization. More optimized fixed-point PL implementation could improve this situation by reducing resource utilization and precision, which can also be the factor of point 2).
4) Larger data block sizes are chosen for most of the cases. The data streaming is invoked through micro-code of AXI DMA from PS. Therefore, a reduced number of data transmissions reduces the overhead.

VI. Conclusions

We proposed a new power capping technique adapted to data size scaling with resource allocation for heterogeneous ARM/FPGA SoC based on OpenCL run-time with OpenMP enabling power management using both ARM processor and streaming accelerators on FPGA concurrently. By using a compile-time profiling assisted, run-time estimated power consumption, the combined DVFS with data partitioning between ARM and FPGA is implemented to maximize the performance under power caps using non-linear programming model. The experimental results demonstrate up to 30% performance improvement , from power cap of $500mW$, compared to ARM only based processing without any power management. Therefore, we are confident that our run-time system improves performance and reduces power consumption at the same time, enabling better energy efficiency.

Acknowledgement

This work was supported by EPSRC grants EP/L004232/1 (ENPOWER), EP/L000055/1 (ALEA) and EP/K017594/1 (GEMSCLAIM).

References

[1] http://apmonitor.com/, accessed by January 2016.
[2] http://openmp.org/wp/, accessed by January 2016.
[3] http://pocl.sourceforge.net/, accessed by January 2016.
[4] Altera. *Altera SDK for OpenCL Programming Guide*. November 2015.
[5] ARM. *ARM v7-M Architecture Reference Manual*. www.arm.com, 2010.
[6] L. Barroso, J. Clidaras, and U. Hoelzle. *The Datacenter as a Computer:An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2013.
[7] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 175–185, New York, NY, USA, 2011. ACM.
[8] H. David, E. Gorbatov, and U. R. Hanebutte. RAPL: Memory Power Estimation and Capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194, August 2010.
[9] F. Eberli. Next Generation FPGAs and SOCs - How Embedded Systems Can Profit. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 610–613, June 2013.
[10] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.
[11] R. Griessl, M. Peykanu, J. Hagemeyer, M. Porrmann, S. Krupop, M. Berge, T. Kiesel, and W. Christmann. A Scalable Server Architecture for Next-Generation Heterogeneous Compute Clusters. In *2014 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 146–153, Aug 2014.
[12] C. Hankendi and A. Coskun. Adaptive Power and Resource Management Techniques for Multi-threaded Workloads. In *2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, pages 2302–2305, May 2013.
[13] C. Hankendi, S. Reda, and A. K. Coskun. vcap: Adaptive power capping for virtualized servers. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 415–420, Sept 2013.
[14] X. inc. [online]. In *Available: www.xilinx.com*, 2016.
[15] S. Kamil, J. Shalf, and E. Strohmaier. Power Efficiency in High Performance Computing. In *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.* , pages 1–8, April 2008.
[16] T. Komoda, S. Hayashi, and T. Nakada. Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 349–356, October 2013.
[17] C. Lefurgy, X. Wang, and M. Ware. Power Capping: a Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, 2007.
[18] Z. Lin and P. Chow. ZCluster: A Zynq-based Hadoop Cluster. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 450–453, Dec 2013.
[19] P. Moorthy and N. Kapre. Zedwulf: Power-Performance Tradeoffs of a 32-Node Zynq SoC Cluster. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 68–75, May 2015.
[20] Z. Ou, B. Pang, Y. Deng, J. Nurminen, A. Yla-Jaaski, and P. Hui. Energy- and Cost-Efficiency Analysis of ARM-Based Clusters. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 115–123, May 2012.
[21] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 48–59, New York, NY, USA, 2008. ACM.
[22] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, and P. Bouvry. An Overview of Energy Efficiency Techniques in Cluster Computing Systems. *Cluster Computing*, 16(1):3–15, 2011.
[23] W. Vanderbauwhede and K. Benkrid. High-Performance Computing Using FPGAs. In *TUCS Technical Report*. Springer Publishing Company, Incorporated, 2013.
[24] O. S. Winter, S. Lafond, and J. Lilius. Evaluation of the Energy Efficiency of ARM Based Processors for Cloud Infrastructure. In *TUCS Technical Report*, number 991, 2010.
[25] Y. Wu, J. Nunez-Yanez, R. Woods, and D. Nikolopoulos. Power modelling and capping for heterogeneous ARM/FPGA SoCs. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 231–234, Dec 2014.
[26] Xilinx. Zynq-7000 All Programmable SoC Overview. www.xilinx.com, 2013.
[27] Xilinx. *SDAccel Development Environment User Guide*. September 2015.