Power-Efficient Accelerator Allocation in Adaptive Dark Silicon Many-Core Systems

Muhammad Usman Karim Khan, Muhammad Shafique, Jörg Henkel
Chair for Embedded Systems, Karlsruhe Institute of Technology, Germany
{muhammad.khan, muhammad.shafique, henkel}@kit.edu

Abstract— **Modern many-core systems in the dark silicon era face the predicament of underutilized resources of the chip due to power constraints. Therefore, hardware accelerators are becoming popular as they can overcome this problem by exercising a part of the program on dedicated custom logic in an energy efficient way. However, efficient accelerator usage poses numerous challenges, like adaptations for accelerator's sharing schedule on the many-core systems under run-time varying scenarios. In this work, we propose a power-efficient accelerator allocation scheme for adaptive many-core systems that maximally utilizes and dynamically allocates a shared accelerator to competing cores, such that deadlines of the executing applications are met and the total power consumption of the overall system is minimized. The experimental results demonstrate power minimization and high accelerator utilization for a many-core system.**

I. Introduction and Related Work

Reduced transistor sizes in the modern fabrication technologies have led to new unforeseen challenges for system designers. Specifically, the failure of Dennard's Scaling [1] has resulted in the emergence of the Dark Silicon age, where the chip's real-estate cannot be 100% utilized continuously, at full capacity. In fact, current predictions [2] suggest that only 30%-50% of the chip's available resources will be *bright* (fully utilized) for 8nm technology, while the rest will be kept *dark* (unutilized) or *gray* (partially utilized or underutilized). This forced underutilization emerges from the fact that power per unit of area is increasing monotonously with increasing transistor density. Therefore, the temperature of the chip may reach levels which will not be contained by the available state-of-the-art coolants and result in permanent damage of the chip. Thus, power-efficient designs are of primary importance for modern systems [3]. However, high power-efficiency can be gained via ASIC or FPGA implementations, at the cost of reduced flexibility and high time-to-market limitations.

In order to combine the advantages of both programmable and application-specific custom architectures, accelerators based many-core systems are becoming increasingly popular in industry [4][5]. Accelerators are usually high complexity parts of programs (called tasks) implemented in custom hardware, and a programmable core[1] can offload its tasks to these accelerators. Accelerators naturally lend themselves to occupy the underutilized chip's area. In addition to increasing the Bright Silicon, accelerators are designed to quickly process the assigned tasks. Therefore, accelerators are fundamental to high complexity, deadline-conscious applications. Examples include video encoding and decoding [6] (also see Intel's Quick Sync Technology), software defined radios [7] etc.

For ease of discussion, we broadly classify accelerators into three categories, based upon their flexibility and access mechanisms. First are the in-core accelerators, which are embedded as a part of the programmable core's computation pipeline (e.g. Nios II custom instructions [8][9], vector instructions [10]). However, note that these accelerators can only be accessed by the corresponding core, and they are a part of the execution stage of the computational pipeline. Therefore, these accelerators exhibit the least flexibility as these accelerators can only be accessed by their cores.

The second category is clustered accelerators, where an accelerator can be accessed by only a specific set of cores and resides in vicinity of these cores. Such accelerators are also called tightly-coupled accelerators. Schemes like [11][12] are available to schedule the accelerator's sharing with the corresponding cores by offloading their tasks, using past-predicts-future paradigms and dynamic programming. However, these schemes do not consider the complete power consumption of the system, and neither do they account for the deadlines of the running applications.

The third and the most flexible category of accelerators can be accessed by all the cores (e.g. via a Network on Chip, NoC) and are called decoupled accelerators or loosely-coupled accelerators. It is evident that the clustered and decoupled accelerators are the most versatile and offer maximum advantages. However, state-of-the-art scheduling schemes presented in the literature [13][14][15][16][17] for decoupled accelerators usually try to reduce the resources used, maximize the processing speed, or, reuse the accelerators' memory as cache or reconfigurable logic. However, no reference to the power consumption, frequency tuning of the cores and deadlines of the applications is made.

Problem: In a nutshell, a single accelerator is shared by multiple cores in the clustered and decoupled accelerator categories. Since this accelerator can only be allotted to a single compute core at a given time, therefore, some of the applications running on these cores might miss their deadlines, or these applications might change their workload at runtime. Further, it is possible that the accelerator is not continuously utilized (i.e. accelerators are darkened) which defeats their purpose of providing power- and complexity-efficiency. In addition, it is also possible that in order to meet the deadlines, higher than required power is pumped to the cores. This will increase the power consumption of the system, and therefore, elevate the chip's temperature.

A. Our Novel Contributions

In this work, we address the following problem: *How to allocate the shared hardware accelerator among the competing cores, such that the hardware is fully utilized, all application deadlines are met and the power consumption of the complete system is minimized under a specific set of running frequencies?* We propose an adaptive accelerator allocation scheduling scheme, for utilizing a shared hardware accelerator by multiple, concurrently running applications. This schedule not only accounts for meeting the deadlines of the applications, it also helps to reduce the dynamic power by determining the voltages and frequencies of the cores. Once a core offloads its tasks to the accelerator, the core can go into sleep state which further reduces the power/temperature of the system. Summarizing, we propose:

Accelerator allocation schedule to allocate the shared hardware accelerator among the competing cores such that the hardware accelerator is maximally utilized.

Power-efficient frequency tuning of the cores such that deadlines are met by all running applications, by distributing the workload on the programmable cores and the hardware accelerator, and the power consumed by the many-core system is minimized.

[1] In the following text, a core refers to a programmable entity (a CPU), which can run the software code and access the accelerators.

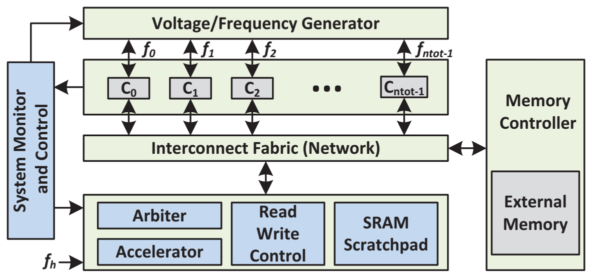


Fig. 1: Overview of our novel contributions

The outline of our novel contributions and the proposed architecture are shown in Fig. 1. As noted, the "System Monitor and Control" generates the appropriate signals to determine the frequencies of the cores and the accelerator allocation by deriving and solving an optimization problem based upon the system parameters. Cores, accelerator and the external memory are connected via interconnect fabric. The accelerator consists of an arbiter to determine the core accessing the accelerator, read write control circuitries to access on-chip or off-chip memories, and internal SRAM scratchpad memory. Note that we assume that the task which can be offloaded to the accelerator can also be done locally by the core (via software). Further, we assume that the frequencies of the cores can be independently adjusted and the clock frequency of the accelerator is constant and very low (i.e. the accelerator is always bright).

Paper Organization: A basic overview of the hardware acceleration and its association with the system dynamics is given in Section II. Section III discusses the system model, optimization problem and the resulting accelerator scheduling scheme. Experimental setup and evaluation is presented in Section IV and the paper is concluded in Section V.

II. System Analysis

For a given set of applications and their associated deadlines, there is a system power that can be pumped in which just satisfies all the deadlines. This minimum power (p_{min}) can determine the most efficient clock frequencies of the cores. Finding this power will result in maximum power savings for the system. Pumping a power lesser than p_{min} will result in deadline violations. Conversely, pumping more power than p_{min} will needlessly increase the power consumption and also lift the temperature of the chip. Therefore, one of the prime motives of this work is to find p_{min}.

This concept is graphically shown in Fig. 2(a). The figure shows the relationship of the time consumed by the application on the core for the given frequency of the core and the percentage of the total tasks offloaded to the hardware accelerator. As noticed, by increasing the offloading percentage, the time consumption of the application running on the core reduces significantly. At the same time, it must be noted that increasing frequency also causes a reduction in the time consumption. This means that a suitable amount of offloading percentage and frequency of the core can be determined, which will result in the deadline constraint being met (i.e. the execution time below some threshold value). Therefore, both these factors must be considered while designing shared accelerator architecture and appropriate selection of these factors can result in high power savings.

Further, as previously mentioned in the Dark Silicon perspective, a core should not continuously run at a high frequency. This concept is shown in the Fig. 2(b), where x-axis denotes the accelerator usage percentage and y-axis is the energy consumption of the system. Note that increasing the amount of offloading results in reduced energy. Further, the trend between the dynamic energy consumption and the percentage of offloading is linear. Thus, the tasks can be offloaded to the hardware accelerator and the core can go into sleep mode, by which the dynamic power and temperature of the core can be reduced. This also points out to the important observation that maximum power savings of the system can be achieved if the associated hardware accelerator is fully utilized.

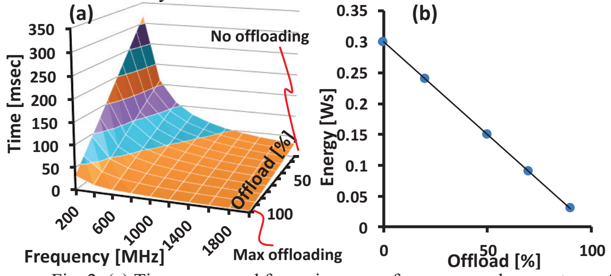
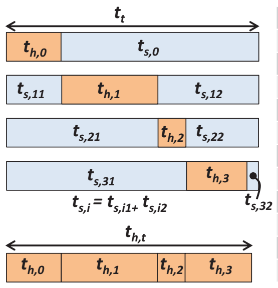


Fig. 2: (a) Time consumed for a given core frequency and percentage of offloading and (b) dynamic energy consumed by the core for a given percentage of offloading the task "4×4 Mean/Variance" to the accelerator.

III. Power-Efficient Accelerator Allocation

Based upon the system analysis presented in the previous section, we propose our power efficient acceleration allocation scheme. For the following discussion, assume that several applications (or independent threads of an application) are concurrently running on each compute core. In the coming text, we will only use applications for demonstrating the proposed scheme. However, our scheme is equally applicable to concurrently running threads of an application. Each application has an associated set of tasks, which must be finished within the given deadline. These tasks can either be run on software or hardware. Our objective is to allocate the accelerator to these applications, such that the total power is minimized and system meets the application(s) deadline(s). We begin with modeling the system, and then present a scheduling scheme to determine the best accelerator allocation.

A. System Modeling and Objectives



t_t	Total time for an epoch
$t_{s,i}$	Time on core by app *i*
$t_{h,i}$	Time on acc by app *i*
$t_{h,t}$	Time on acc by all apps
$n_{s,i}$	Tasks per sec run on the core *i*
$n_{h,i}$	Tasks per sec on acc by app *i*
n_i	Total tasks required per sec
$c_{s,i}$	Cycles per task of app *i*
$c_{h,i}$	Cycles per task on acc by app *i*
n_{tot}	Total number of cores/apps
$f_{s,i}$	Frequency of core *i*
f_h	Frequency of the acc

Fig. 3: (Left) Breakdown of an example execution time on a 4-core system and a shared accelerator, (right) variables used in this work.

Consider that the system has n_{tot} cores, competing for an accelerator. The application i consumes $t_{s,i}$ seconds and $t_{h,i}$ seconds when its corresponding task is run on software and hardware respectively. An example accelerator allocation is diagrammatically shown as an example in Fig. 3(left). This figure shows the time consumption of each application on the programmable core and the hardware accelerator. Notice that for an epoch of t_t seconds, the total time for which the accelerator is engaged by the cores is given by $t_{h,t} \leq t_t$. For the set of variables used in this paper, refer to Fig. 3(right).

Our objective is to minimize the power consumption of the complete system. If the power of a core p_i is a function of its frequency $f_{s,i}$, then, mathematically, our objective is:

$$\min\left(\sum_{i=0}^{n_{tot}-1} p_i\left(f_{s,i}\right)\right) \qquad (1)$$

At the same time, we want to maximize the hardware utilization, i.e. the difference between the epoch time and the time for which hardware is engaged (t_t-$t_{h,t}$ є $\mathbb{R}^+$, positive real) should be as small as possible. In order to do so, we proceed by writing the total cycles processed per second on the accelerator, by all the cores to be equal to:

$$c_{h,0}n_{h,0}+c_{h,1}n_{h,1}+\ldots+c_{h,n_{tot}-1}n_{h,n_{tot}-1}=\sum\nolimits_{i=0}^{n_{tot}-1}c_{h,i}n_{h,i} \quad (2)$$

In this equation, $c_{h,i}$ is the number of cycles per task and $n_{h,i}$ is the number of tasks per second for application *i* on the accelerator. Therefore, if the difference t_t-$t_{h,t}$ needs to be minimized, we need to match the number of cycles processed per second on the accelerator to its clock frequency f_h. Note that the hardware is running at a fixed frequency. Mathematically, this constraint can be written as:

$$\sum\nolimits_{i=0}^{n_{tot}-1}c_{h,i}n_{h,i}=f_h \quad (3)$$

Additionally, since the deadlines should be met, therefore, the additional constraint is:

$$n_{s,i}+n_{h,i}\geq n_i \quad \forall i\in\{0,\cdots,n_{tot}\text{ -}1\} \quad (4)$$

This equation shows that the number of tasks per second on the hardware ($n_{h,i}$) and software ($n_{s,i}$) should at least equal the number of total tasks of the application *i* per second (n_i).

Moreover, the clock frequencies of the cores should be bound. Thus, we have an additional constraint that:

$$f_{\min}\leq f_{s,i}\leq f_{\max} \quad (5)$$

B. Optimization Algorithm

In order to optimize the above function given in equation (1), we are required to derive $p_i(f_{s,i})$ in terms of the system parameters which we can tune. If the cycles per task ($c_{h,i}$) and the number of tasks per second ($n_{h,i}$) on accelerator by applications *i* are known, we can determine the time spent by application *i* on accelerator by using:

$$t_{h,i}=\frac{c_{h,i}n_{h,i}}{f_h}t_t \quad (6)$$

Thus, the time consumed on core *i* ($t_{s,i}$) can be determined by using the following relation:

$$t_{s,i}=t_t-t_{h,i}=t_t\left(1-\frac{c_{h,i}n_{h,i}}{f_h}\right) \quad (7)$$

Using the above equation, the frequency of the core can be determined by:

$$f_{s,i}=\frac{c_{s,i}(n_i-n_{h,i})}{t_{s,i}} \quad (8)$$

In this equation, $c_{s,i}$ is the number of cycles per task of application *i* in the software. Here, we used the identity given in equation (4).

Now, by inserting equation (8) in equation (1), the power consumed by a core can be written as:

$$p_i\left(f_{s,i}\right)=p_i\left(\frac{c_{s,i}(n_i-n_{h,i})}{t_t\left(1-\left(c_{h,i}n_{h,i}/f_h\right)\right)}\right)=p_i\left(\frac{\alpha-\beta n_{h,i}}{1-\gamma n_{h,i}}\right) \quad (9)$$

In this equation, *α*, *β* and *γ* are constants given by:

$$\alpha=c_{s,i}n_i/t_t \qquad \beta=c_{s,i}/t_t \qquad \gamma=c_{h,i}/f_h \quad (10)$$

Therefore, the complete objective function with constraints can be collectively written as:

$$\min\left(\sum\nolimits_{i=0}^{n_{tot}-1}p_i\left(\frac{\alpha-\beta n_{h,i}}{1-\gamma n_{h,i}}\right)\right)$$

subject to: (11)

$$\sum\nolimits_{i=0}^{n_{tot}-1}c_{h,i}n_{h,i}=f_h$$

$$n_{s,i}+n_{h,i}\geq n_i \quad \forall i\in\{0,\cdots,n_{tot}\text{ -}1\}$$

$$f_{\min}\leq\frac{\alpha-\beta n_{h,i}}{1-\gamma n_{h,i}}\leq f_{\max} \quad \forall i\in\{0,\cdots,n_{tot}\text{ -}1\}$$

As noted, the optimization objective given in equation (11) is to find an appropriate number of tasks which are offloaded to the accelerator ($n_{h,i}$), for all applications. However, the optimization algorithm presented in the above equation is non-linear, even if we consider that power and frequency approximately forms a linear relationship for the given set of frequencies.

NelderMeadFunctionCost ( ):
Input*:* Epoch time t_t; $n_{h,i}$, n_i, $c_{s,i}$, $c_{h,i}$, for all applications; Accelerator frequency f_h; Minimum core frequency f_{min}; Maximum core frequency f_{max};
Output: Cost of the objective function for the given inputs *v*;

```
1.  $v \leftarrow 0;$
2.  $\forall i\in\{0 \text{ to } n_{tot}\text{-1 cores}\}\{$
3.      $t_{h,i} \leftarrow \text{ComputeTimeOnAcc}(t_t, n_{h,i}, c_{h,i}, f_h);$
4.      $t_{s,i} \leftarrow t_t - t_{h,i};$
5.      $f_{s,i} \leftarrow \text{ComputeCoreFreq}(n_i, n_{h,i}, c_{s,i}, t_{s,i});\}$
6.  $v \leftarrow v + \exp(|t_t - \sum t_{h,i}|);$
7.  $\forall i\in\{0 \text{ to } n_{tot}\text{-1 cores}\}\{$
8.      $v \leftarrow v + |n_i - n_{h,i}|;$
9.      $\text{if}(n_{h,i} < 0)\{v \leftarrow v + |n_{h,i}|;\}$
10.     $\text{if}(f_{s,i} > 0)\{v \leftarrow v + f_{s,i};\}$
11.     $\text{if}(f_{s,i} > f_{max})\{v \leftarrow v + (f_{s,i} - f_{max});\}$
12.     $\text{if}(f_{s,i} < f_{min} \text{ AND } t_{s,i} > 0)\{v \leftarrow v + (f_{min} - f_{s,i});\}$
13. }
```

Fig. 4: Nelder-Mead algorithm to find the best accelerator allocation

In our case, the optimization (finding the value of $n_{h,i}$ for all applications) is achieved using Nelder-Mead method [18]. Nelder-Mead is a greedy algorithm that iteratively determines the value of the given objective function (*v*) for the given inputs and moves towards an optimum. The advantage of using Nelder-Mead method is that it does not require the derivatives of the objective function to be calculated.

In each iteration of the Nelder-Mead algorithm, the objective function is evaluated for the set of inputs ($n_{h,i}$ in this case) to determine the "cost" of these inputs. That is, for the given $n_{h,i}$, the value of $v=\sum p_i(f_{s,i})$ is computed. Since in our case, constraints bound the search trajectory to find the optimum, therefore, we have modified the original constrained optimization problem into an unconstrained problem. Specifically, we use penalty function method. The function is evaluated as given in Fig. 4. Here, the cost of the function increases if the difference between t_t and $t_{h,t}$ increases (maximum accelerator utilization, line 6, equations(2)-(3)). Since we require maximizing the accelerator utilization, therefore, we introduce an additional penalty, depending upon the difference between the number of hardware operations and the total operations (lines 8-9). Further, if the core frequencies that will support $n_{s,i}$ are lesser than the minimum frequency (f_{min}), or larger than the maximum frequency (f_{max}), it will also result in increasing the cost of the function (bounded frequencies, line 10-11, equation (5)). Once the cost of the function *v* is determined, it is compared with the previous costs and algorithm moves in the direction of the lowest cost.

Note that in our proposed approach, the cores and the accelerator populate their caches and scratchpad respectively by fetching the data directly from the external memory. Further, even if the size of data processed by a core and the accelerator is equal, the accelerator will still generate higher throughput, due to its custom logic implementation.

IV. Experimental Setup and Evaluations

For testing our proposed approach, x86 based Sniper many-core simulator [19] is used. This simulator is also coupled to McPAT power estimation framework [20], which is used for generating power and energy numbers. For evaluation purposes, we focus on the image and video related applications, due to their high complexity. Additionally, these applications process the image as set of blocks, have a high tendency to benefit from parallelization, and have a tight deadline constraint, defined as frames per second (*fps*, frames that must be processed in one second). Therefore, tasks of such applications are suitable candidates for accelerators. In the following, we briefly describe the accelerator used in this work.

4×4 Mean/Variance: For illustrative purposes, the mean and variance computation application is considered. This application will fetch a block of 4×4 pixels from the image and generate the mean and variance of the region. This type of block-based variance computation is significant for texture classification and efficient image/video compression. For this task, our simulations show that c_s=492 and c_h=70. The image is stored in the external memory. The cores process different sized parts of the same image and the arbiter partially shares the workload of each core, by allocating it the hardware accelerator according to our proposed scheme. A part of image is brought to the internal SRAM scratchpad of the accelerator. Further, if the size of the image a core needs to process or *fps* requirement increases, the number of tasks per second (n_t) also increases, and thus, more power and/or accelerator schedule should be allocated to that core. Table I shows the resource consumption of implementing the accelerator on an Altera's Arria II GX260 FPGA.

Fig. 5 shows the relationship of the power consumed by the system, by changing the fps and the total number of cores competing for the hardware accelerator. For this evaluation, a FullHD image of size 1920×1080 pixels is considered, and regions of this image are distributed among the applications. This way, the total number of operations per second (n_t) of each application are different, and thus, the accelerator demand varies for all these applications. Further, we consider t_t=2sec and f_h=100MHz.

Fig. 6(a) shows an example distribution of the hardware accelerator to the cores. Notice that some of the cores (e.g. 2-4) mostly run their tasks on the software. Further, summation of the time consumed by the hardware accelerator will be almost equal to t_t, which shows that the accelerator is 100% utilized. Fig. 6(b) shows the corresponding frequencies of the cores for the proposed schedule. The cores with considerable accelerator allocation are usually running at a much lower frequency. Additionally, the frequency of a core also depends upon n_t and a larger n_t either requires more offloading or a higher core frequency, determined by our optimization algorithm given in Fig. 4.

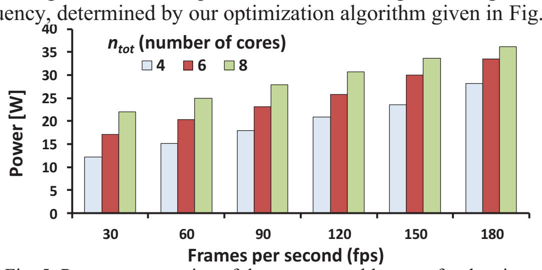


Fig. 5: Power consumption of the programmable cores for the given *fps* requirement and different number of competing cores.

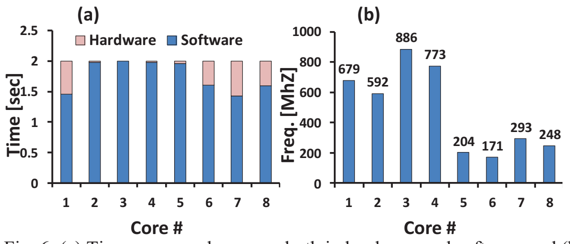


Fig. 6: (a) Time consumed per core both in hardware and software and (b) the corresponding frequency of the cores for n_{tot}=8, *fps*=120.

V. Conclusion

This work addresses the problem of allocating a shared hardware accelerator to multiple applications such that the deadlines of all applications are met and the power consumption of the system is minimized. Specifically, we propose an accelerator allocation scheme, which allocates the hardware accelerators to the applications, in a round-robin fashion, for different time intervals. This way, a core which can offload its tasks to the accelerator can be darkened, and save power and lower its temperature. In addition, the frequencies of the competing cores are tuned by deriving a constrained optimization problem such that the total system power is minimized.

Table I: Resource consumption of the 4x4 Mean/Variance accelerator for n_{tot}=4 on Altera Arria II GX FPGA

Comb. ALUTs	Memory ALUTs	Registers	Memory (bits)	DSP blocks	fmax (MHz)
645 (<1%)	0	1011 (<1%)	8192 (<1%)	7 (<1%)	180.28

Acknowledgement

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" [SFB/TR 89]; http:/invasic.de.

References

[1] R. H. Dennard et al., "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," in *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[2] H. Esmaeilzadeh, E. Blem, R.S. Amant, K. Sankaralingam, D. Burger, "Dark silicon and the end of multicore scaling," in *International Symposium on Computer Architecture* (*ISCA*), pp. 365–376, 2011.

[3] M. Shafique, S. Garg, J. Henkel, D. Marculescu, "The EDA Challenges in the Dark Silicon Era," in *Design Automation Conference* (*DAC*), 2014.

[4] L. Seiler et al., "Larrabee: A Many-Core x86 Architecture for Visual Computing," in *IEEE MICRO*, pp. 10–21, 2009.

[5] M. Mattina, "Architecture and Performance of the Tile-GX Processor Family," *white paper*, pp. 1–18, 2014.

[6] M. Shafique, L. Bauer, J. Henkel, "Optimizing the H.264/AVC Video Encoder Application Structure for Reconfigurable and Application-Specific Platforms," in *Journal of Signal Processing Systems (JSPS)*, vol. 60, issue 2, pp. 183–210, 2010.

[7] C. Liu, O. Granados, R. Duarte, J. Andrian, "Energy Efficient Architecture Using Hardware Acceleration for Software Defined Radio Components," in *Journal of Information Processing Systems*, vol. 8, no. 1, pp. 133–144, 2012.

[8] Altera, "Nios II Custom Instruction User Guide," 2011.

[9] M. U. K. Khan, M. Shafique, J. Henkel, "Hardware-Software Collaborative Complexity Reduction Scheme for the Emerging HEVC Intra Encoder," in *Design, Automation and Test in Europe (DATE)*, pp. 125–128, 2013.

[10] H. Shojania, L. Baochun, "Parallelized Progressive Network Coding with Hardware Acceleration," in *IEEE International Workshop on Quality of Service*, pp. 47–55, 2007.

[11] D. Sheldon, A. Forin, "An Online Scheduler for Hardware Accelerators on General Purpose Operating Systems," *Tech. report, Microsoft Research*, 2010.

[12] C. Huang, D. Sheldon, F. Vahid, "Dynamic Tuning of Configurable Architectures: the AWW Online Algorithm," in *International Conference on Hardware/Software Codesign and System Synthesis*, pp. 97–102, 2008.

[13] T. Majumder, P. P. Pande, A. Kalyanaraman, "High-throughput, Energy-Efficient Network-on-Chip-Based Hardware Accelerators," in *Journal of Sustainable Computing: Informatics and Systems*, vol. 3, issue 1, pp. 36–46, 2013.

[14] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, G. Reinman, "Architecture support for accelerator-rich CMPs," in *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*, 2012.

[15] E. Cota, P. Mantovani, M. Petracca, M. Casu, and L. Carloni, "Accelerator Memory Reuse in the Dark Silicon Era," in *IEEE Computer Architecture Letters*, pp. 1–4, 2012.

[16] J. A. Clemente, I. V. Beretta, V. Rana, D. Atienza, D. Sciuto, "A Mapping-Scheduling Algorithm for Hardware Acceleration on Reconfigurable Platform," in *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 2, 2014.

[17] S. Paul, R. Karam, S. Bhunia, R. Puri, "Energy-Efficient Hardware Acceleration through Computing in the Memory," in *Design, Automation and Test in Europe (DATE)*, 2014.

[18] J. A. Nelder, R. Mead, "A Simplex Method for Function Minimization," in *The Computer Journal*, vol. 7, no. 4, 1965.

[19] T. E. Carlson, W. Heirman, L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Multi-core Simulation," in *SC*, pp. 1–12, 2011.

[20] S. Li, J. H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, pp.469–480, 2009.