Power Adaptive Computing System Design in Energy Harvesting Environment

Qiang Liu*, Terrence Mak†, Junwen Luo†, Wayne Luk* and Alex Yakovlev†
*Department of Computing, Imperial College London, UK, SW7 2AZ.
Email: {qiang.liu205, w.luk}@imperial.ac.uk
†School of Electrical, Electronic and Computing Engineering, Newcastle University, UK, NE1 7RU.
Email: {terrence.mak, j.w.luo, alex.yakovlev}@ncl.ac.uk

***Abstract*—Energy harvesting systems provide a promising alternative to battery-powered systems and create an opportunity for architecture and design method innovation for the exploitation of ambient energy source. In this paper, we propose a two-stage optimization approach to develop power adaptive computing systems which can efficiently use energy harvested from solar source. At design time, an SPMD (single process, multiple data) computation structure with multiple parallel processing units is generated, and a convex optimizer runs at run-time to decide how many processing units can operate simultaneously subject to the instant power supplied from the harvester. The approach is evaluated on three embedded applications. The results show that the proposed approach can develop and manage a computing system for each application to adjust its power consumption with respect to the power supply while maximizing speed. Compared to static systems without adaptability, our power adaptive computing system improves the harvested energy utilization efficiency up to 28.8%. These computation systems can be applied to distributed monitor networks to improve computation capability at nodes. In our experiments, the throughput per watt in a node with a ARM9 processor can be improved 19 times by adding the developed adaptive computing system to the node.**

I. Introduction

Traditional battery-powered systems work under limited energy supply. For applications that require long working duration, such as distributed sensing or distributed monitor systems, supply of reliable energy becomes a critical concern and much effort has been devoted to energy efficient or low-power system design [1]. With advances in energy harvesting technologies, it is possible to implement a self-powered system that harvests ambient energy from the environment [2], [3], [4]. Particularly, harvesters provide a spectrum of power delivery subject to various environmental conditions and systematic volumetric, including solar [2], electromagnetic [3], mechanical piezoelectric vibration [4] and so on. Such energy harvesting systems provide a promising alternative to battery-powered systems and create an opportunity for architecture and design method innovation for the exploitation of ambient energy source.

In this paper, we propose an approach to develop power adaptive computing systems which can efficiently use energy harvested from solar source (15mW/cm^2 [5]). These systems can be applied to distributed monitor networks to provide computation capability at sensory nodes. In this way, collected raw data are preprocessed and only desired information is transferred on the networks.

The design criteria for systems using energy harvesting sources are fundamentally different from those using a battery. The battery-based system benefits from a relatively predictable metric of energy residual, suffices to characterize the energy availability, and is seemingly an unbounded power supply. For an energy harvesting system, rather than a limited energy supply, it has a limit on the power at which the energy can be used, and the power supply from energy harvesters varies with time. Although power regulators aim to stabilise and deliver a constant power supply, there is an upper bound for the transient power delivered to the computational electronics.

In contrast to low-power circuit design principles, it is desirable that the computational load in an energy harvesting system consumes energy at an appropriate rate that is compatible with the harvester, in which the computational performance is maximized while the power consumption of the computational load is not greater than the power supplied from the harvester. Therefore, an intelligent control of the computational load that adapts the computation performance to the transient power constraints from the energy harvester is required.

This paper presents a two-stage design optimization approach to achieve optimized utilization of harvested energy. Specifically, at design-time, given the characteristics of the solar energy harvester, an SPMD (single process, multiple data) structure of the computational load is determined for an application, maximizing the computation speed on a target hardware platform. The parallel computation structure contains multiple homogeneous processing units (PUs), each with an enable control signal. When the system energy is sufficient, all PUs are enabled and the computation system runs at the highest speed; otherwise some of the PUs are disabled correspondingly. This can be realized using *clock gating* [6] technique and can adjust the system power consumption. At run-time, a convex optimization model is presented to intelligently determine which PUs are enabled at the same time (*clock gating schemes*), subject to the existing state of energy harvester. The convex model is customized for each specific application, resulting in fast and globally optimal solutions. The contributions of this paper are:

- a two-stage optimization approach for designing power adaptive systems;

- a custom convex model used at run-time to determine clock gating schemes, adjusting system power consumption to instant power supplied from a solar harvester; and
- evaluation of the approach on a platform with a ARM9 processor and an FPGA for three embedded applications; results show that our power adaptive system can improve harvested energy utility efficiency up to 28.8%, compared to static designs without adaptability; and the computing system can provide the ARM processor with improvements in FLOPS/W up to 19 times.

The rest of this paper is arranged as follows. Section II introduces related work. Section III describes the energy harvesting system. Section IV presents the proposed two-stage design optimization approach. Experimental results are shown in Section V and are followed by the conclusion Section VI.

II. Related Work

Harvesting energy from environments to power electronic devices has been a hot research topic. Especially in the domain of sensor networks, design methodologies, including power transfer [7], energy prediction [8], [9], energy storage [2], [5], and power management [2], [10], [11], [12], [13], [14], have recently been investigated.

In this paper, we focus on power management techniques previously used in energy harvesting systems. As mentioned earlier, power management in the energy harvesting systems aims at controlling power consumption subject to varying power supply to improve power efficiency. A well known dynamic power consumption model for CMOS circuits is

$$Power = \frac{1}{2} \times C \times V^2 \times F, \tag{1}$$

where C is the load capacitance, V is the supply voltage and F is the switching frequency. Therefore, by changing C, V and F, one could adjust the system power consumption.

Dynamic voltage scaling (DVS) is used in [10], [11] to improve energy efficiency. DVS technique drives processors to work at full speed when the energy is sufficient in the harvesting system; otherwise slows down the processors, by adjusting supply voltage and frequency. Liu *et al.* [10] schedule tasks in terms of system energy and task priority. Zhang *et al.* [11] formulate the process of selecting voltages and frequencies and use enumeration with respect to the system characteristics to find the appropriate solution. In energy harvesting systems, however, voltages are subject to instability in spite of the existence of power/voltage regulators (introducing additional overheads), and scaling of voltages may be restricted in practice.

Another widely used technique is duty cycling [2], [12], [13], [14]. Duty cycling changes operational frequency of system components, *i.e.* the ratio between the component active time period and the system time period. Reducing duty cycle corresponds to decreasing F in (1). Works in [2], [12], [13], [14] formulate the duty cycle control problem using linear program (LP), and solve the LPs periodically at run-time. Authors of [2], [12], [13] try to maximize average duty cycle, while [14] aims at reducing the difference of duty cycles at different time slots.

The idea of power adaptive computing and power elasticity were presented in [15], [16]. In particular, [16] proposed using methods of discrete event control, soft arbitration and concurrency management to adapt to power constraints. Stochastic analysis methods have been used to characterize the behavior of multi-core system in power-latency tradeoffs [17].

In this paper, we use the clock gating technique [6], [18], which disables a clock line and the functional units driven by this clock line in circuits, to reduce the system power consumption. The computing system developed in this paper contains multiple parallel processing units (PUs). Disabling some of the PUs corresponds to reducing the capacitive load C. We use a customized non-linear convex model to determine on-line the number of PUs to be enabled, given the harvesting system energy availability and computation tasks. This dynamic management can adjust the number of active PUs executed in a clock cycle, and thus modulates the peak power consumption more effectively. The non-linear convex model leads to more energy efficient clock gating schemes, compared to a linear model as shown in Section V, at the cost of longer solution time. We implement the computing system on an FPGA-based platform because hardware execution is more energy efficient. Exploitation of FPGAs in energy harvesting sensor networks has been previously carried out in [19].

III. Power adaptive system

Fig. 1 shows a typical energy harvesting system. In this paper, we focus on designing the adaptive computing system and developing a run-time power management method.

In this paper, the energy source is solar energy. Apparently the function of the energy harvester is to transfer ambient power (light) to the electrical power to supply the whole system. Fig. 2 shows the variation and density of harvested solar energy over a week in October 2010. The power varies in a time scale of minutes. Battery and ultra-large capacitors are two options for the energy storage. In this work, we assume that the energy storage only provides adequate energy to maintain the estimator and control system, when the harvested power is less than the least working power requirement of the computing system.

The estimation method uses the length of experiments days, sampled time, and change rate of weather and weight factor. Here we use the technique [2], [9] based on the history average data and previous data values to estimate current output. It is sensitive to the sampling frequency. Here we sampled every 5 minutes in the day [9].

The power adaptive computing system is shown in Fig. 3. The computing system executes in SPMD structure. All processing units (PUs) have the same functionality, work on different data sets and have enable control signals (Ena). The structure of each processing unit, the number of processing units and associated local memory size are customized for a specific application. The local controller communicates with

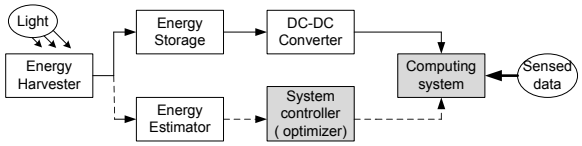

Fig. 1. The structure of an energy harvesting system.

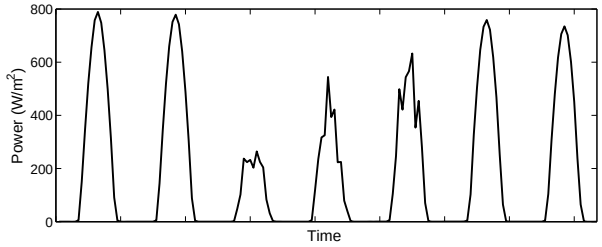

Fig. 2. Solar energy within a week in October 2010 [20].

the system controller in Fig. 1, enables/disables PUs independently, and manages data transfers between global memories and local memories. The dynamic power consumption of this computing system linearly varies with the number of working PUs, under the same system voltage supply and clock frequency. Therefore, clock-gating some of the PUs allows the system to adapt itself to the supplied power and thus to be used in an energy harvesting system.

The system controller contains a run-time optimizer, which determines the proper clock gating scheme for the computing system at run-time. The optimizer actually solves a convex optimization problem which will be presented in the next section.

The system works as follows. The sensed data are first stored in the global memories. The system controller sends messages to the energy estimator and run-time optimizer to bring them into work. The estimator estimates the supply power, and the optimizer determines a proper clock gating scheme based on the estimation. Then the system controller sends control signals to the local controller of the computation system, including starting operation signal, input data parameters (such as image size), and clock gating signals (Ena). The local controller then triggers data transfers and computation. When the computational system completes its job, the local controller sends a finish signal back to the system controller and the latter handles results. After that, the system controller starts the power estimation and run-time optimization again. The whole process is repeated. It is assumed that the power stable time interval is greater than the time required by the computation system for processing one data set. In the context of this paper, solar energy is sampled every 5 minutes, while processing one data set for the applications completes within a few seconds.

IV. Power Optimization Approach

To develop and manage the above described power adaptive system, we propose a two-stage design optimization approach. In the first stage, a parallel computation structure of the work load is determined. In the second stage, a clock gating scheme is determined dynamically subject to supplied power.

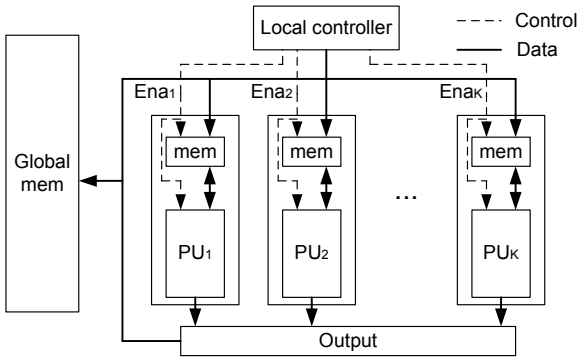

Fig. 3. A power adaptive enabled computing system.

A. Design-time optimization

The task in this stage is to design a parallel computing system for an application, given a programmable hardware platform. The target applications contain loop structures with potential for parallelization. The objective is to minimize the system execution time, while design time is not crucial at this stage. To this end, we apply several design optimization techniques and explore design space to determine an SPMD computation structure as shown in Fig. 3. We adopt the approach [21] to automate the design space exploration at compile time with exploitation of data reuse, loop pipelining and loop parallelization. Unlike [21], the design optimization problem is the following:

$$\begin{aligned} \min \quad & T(\vec{\rho}, \vec{k}, ii) \\ \text{subject to} \quad & R_{mem}(\vec{\rho}, \vec{k}, ii) \leq Res_{ram} \\ & R_{comp}(\vec{\rho}, \vec{k}, ii) \leq Res_{comp} \end{aligned} \qquad (\mathcal{P}_1)$$

where the execution time model T, the memory resource utilization model R_{mem} and the computational resource utilization model R_{comp} are formalized in [21]; $\vec{\rho}$ is a data reuse variable vector determining the local memory space for each processing unit, $\vec{k} = (k_1, k_2, \ldots, k_N)$ is a loop parallelization variable vector indicating k_l iterations of loop l in a N-level nested loop structure are executed in parallel and ii is the number of clock cycles of the loop pipelining initiation interval; Res_{ram} and Res_{comp} are the availability of memory resources and computational resources in hardware.

In the computation system generated by $\mathcal{P}_1$, each processing unit (PU) is customized with respect to arithmetic operations involved in the application and processes data in pipeline ii, and there are $\prod_{l=1}^{N} k_l$ homogeneous PUs in total. Note that the customization and parallelism are determined simultaneously, leading to more efficient designs [21].

Each processing unit just performs arithmetic computations and could contain adders/subtracters, multipliers or comparators for the target embedded applications. Each processing unit only accesses its own local memory.

After the parallel computation structure is determined, the design can be synthesized, placed and routed, and mapped onto a programmable hardware. This whole process could take several hours to complete.

This stage decides the computation structure with the peak speed. However, this peak speed may not always be achievable due to the changing power supplier in the energy harvesting environment. Therefore, in the next section, clock gating schemes are dynamically determined corresponding to the supplied power.

B. Run-time optimization

To enable run-time power optimization, two key facilities are needed: a) a system power model and b) a fast optimizer. The system power model estimates the system power consumption with different clock gating schemes, while the optimizer selects the most power-efficient clock gating scheme. In this paper, we present an empirical power model for each application; the power model is simple and accurate enough to make sure the computation system works safely, *i.e.* the system power consumption is not greater than the supplied power during a fixed period. Meanwhile, we customize the optimization problem $\mathcal{P}_1$ for each application to derive a simplified optimization problem, leading to a fast solution.

1) Power model: For the parallel computation structure derived in the previous section, the system power consumption variation with different clock gating schemes can be expressed in a model as below:

$$P_c(m) = P_{const} + m \times P_{pu}, \tag{2}$$

where P_{const} is the constant part of the system power consumption when experimenting with different clock gating schemes, P_{pu} is the power consumption of a PU when it is enabled, and m is the number of PUs enabled.

In our work, for each application, we experiment with the parallel computation structure on several different clock gating schemes to measure the system power consumption, and then fit the power values to the power model (2) to obtain the parameters P_{const} and P_{pu}. The instantiated power model is then used in the optimization problem described below.

Usually, the power values estimated by model (2) sometimes may be smaller than the real values that may cause problems where power consumption is greater than power supply. Therefore, in practice, a fine tuning value is added to P_{const} in (2) to guarantee safe operations.

2) Customized optimization model: The optimization model $\mathcal{P}_1$ formulates the design space of combined optimization techniques, determines the whole system structure, and thus has high complexity and needs from a few minutes to few tens of minutes to solve on a PC [21]. This optimization time is higher than the energy sample time interval, which is 5 minutes in this paper. Therefore, model $\mathcal{P}_1$ is not suitable to be applied to run-time optimization in the context of this paper.

Note that, however, at run-time the system structure is not changed and only the system clock gating scheme varies. The occupancy of the memory and computational resources keeps constant and thus the resource constraints in $\mathcal{P}_1$ can be removed. In addition, the pipelined operation of each PU is also fixed and thus the execution time model can be refined. Based on these discussions, a simplified optimization problem $\mathcal{P}_2$ is customized from $\mathcal{P}_1$ and is shown below:

$$\begin{aligned} \min \quad & (v-1) \times t + T_{in}(m) + T_{comp}(m) + T_{out}(m) \\ \text{subject to} \quad & \max(T_{in}(m), T_{comp}(m), T_{out}(m)) \le t \\ & P_c(m) \le P_s \\ & 1 \le m \le K \\ & L \times m^{-1} \times v^{-1} \le 1 \end{aligned} \tag{$\mathcal{P}_2$}$$

where $v = \lceil L/m \rceil$ is the number of loop strips (each strip contains m loop iterations executing in parallel), L is the total number of iterations of the parallelized loops, P_s is the instantly supplied power, $K = \prod_{l=1}^{N} k_l$ is the total number of PUs determined at design-time. The execution time is divided into three parts: the time for inputting data from global memories to local memories $T_{in}(m)$, the computation time $T_{comp}(m)$ taken by each PU, and the time for outputting results from local memories to global memories $T_{out}(m)$. These three stages are pipelined in the final system, resulting in the objective function.

Given a specific application, this optimization problem is instantiated and application-specific customization is carried out. Section V shows some examples.

This customized optimization model can be transformed into a convex model, and has fewer variables and simpler constraints compared to $\mathcal{P}_1$, leading to an optimal and fast solution. For the tested applications, the model can be solved in a few milliseconds on a PC. The solution to $\mathcal{P}_2$ tells us that m PUs can be enabled, such that the system speed is maximized while the system power consumption is not greater than the supplied power.

V. Experimental Results

As mentioned in Section I, one of the potential applications of the power adaptive system developed in this paper is a monitor network, where each node contains the power adaptive system as well as an energy harvester. Each node not just collects data, but also processes the data and sends desired information to the network. In this paper, we develop power adaptive systems for three applications: matrix multiplication, k-means clustering and Sobel edge detection algorithm, following the proposed two-stage optimization approach. These applications are capable of processing input data and generate reduced data sets. For example, in a monitor system, the Sobel edge detection can be used to extract edges of subjects in images, then the k-means clustering algorithm classifies subjects in terms of their edge properties, and finally the data of interested subjects are transferred to the network.

In our experiments, the power adaptive system is evaluated on a hardware platform shown in Fig. 4. The power estimator and system controller are on a ARM926EJ-S processor

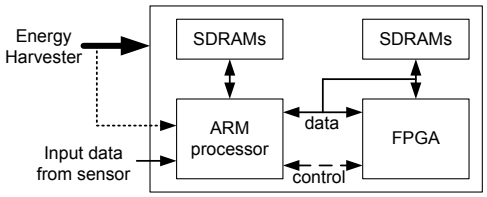

Fig. 4. Experimental hardware platform.

running at 160 MHz and having 64MByte SDRAMs, while the computation system is mapped onto a Virtex5-330t FPGA with 192 DSP blocks and 324 RAM blocks on-chip. In practice, other hardware platforms are equally applicable. In this experiment, we only implement a stand alone system, although it can be applied to a network. The real solar energy data from ORNL website [20] during day time from 6 am to 6 pm in October are used as the harvested power. The data are scaled down as harvested in a 10cm $\times$ 10cm solar cell panel to power the systems presented in this paper. All results shown in this section are obtained on the hardware platform, after synthesis, placement and routing, and mapping onto the hardware. The real power consumption values are obtained by measuring the voltage and current of the power source.

The measured peak power consumption of the ARM processor is about 2.4 W during work and is about 1.9 W when idle. The computation system is implemented on the FPGA with six clock regions; the clock tree in a region is disabled when all PUs in the region do not work, reducing dynamic power consumption. For each application, the power adaptive system developed following our approach is compared to two representative designs, statically determined at design-time, with all PUs always enabled, to show how the adaptability improves the harvested energy utility efficiency:

$$\text{Energy utility efficiency} = E_c/E_s, \tag{3}$$

where E_s is the generated energy and E_c is the consumed energy by the computation system. Moreover, to demonstrate the advantage of our run-time optimization approach, it is compared to another approach which uses the linear power model (2) to determine the clock gating scheme. The optimization problem $\mathcal{P}_2$ instantiated for the tested applications can be solved within 0.3 seconds on the ARM processor using OPT++ optimization library [22], which is negligible compared to the energy sample time interval.

a) Matrix multiplication: (MAT) in a key arithmetic operation in many application algorithms. Two input matrices A and B are multiplied to generate one output matrix C; the whole process runs in a 3-level loop nest. In our experiment, we test the multiplication of two 1024x1024 matrices and each matrix entry is a single precision floating-point number. Matrices A, B and C are stored in three SDRAMs separately.

At design-time, a performance-optimized design of this MAT, determined by the optimization problem $\mathcal{P}_1$ on the target FPGA, is $k_1 = 94, k_2 = 1, k_3 = 1$, $ii = 1$ and data of matrix A are buffered in on-chip RAMs to be reused. In this design, 94 iterations of the outermost loop of MAT are executed in 94 PUs in parallel, where one PU performs the multiplication of one row of matrix A and matrix B and generates one row of matrix C; the innermost loop of MAT is fully pipelined and a FIFO is needed for each PU to input matrix B in the column order. To buffer one row of A, one 32 Kbits RAM block is needed and thus each PU uses two RAM blocks as input data buffers to pipeline data input and computation. This design can obtain the final matrix $C_{1024\times0124}$ in 0.36 seconds at 100 MHz clock frequency, including the data transfer time between the global and local memories.

Once we implement the design on the FPGA, we could measure the power consumption of the computation system. Experimenting with several clock gating schemes, the trained power model for MAT is

$$P_c(m) = 4.56 + 0.027m. \tag{4}$$

Fig. 5 (a) shows the power estimated by this model and the corresponding measured power for MAT with different clock gating schemes. The maximum relative error of the estimation is 2.18%.

For the run-time optimization model, the three parts of the execution time in the number of clock cycles are refined as below:

$$T_{in}(m) = m \times numCol \tag{5}$$

$$T_{comp}(m) = lengthPipe \times numCol \tag{6}$$

$$T_{out}(m) = m \times numCol \tag{7}$$

where $numCol$ is the number of columns of the matrix and $lengthPipe$ is the scheduled length of pipelining the innermost loop. Bringing these functions into problem $\mathcal{P}_2$ instantiates the optimization problem for MAT. Moreover, since the structure of the PU is fixed, $lengthPipe$ is known at this stage; it is about 1024 clock cycles for pipelining the innermost loop of MAT. This value is much larger than the upper bound of m that is $K = 94$. Therefore, the run-time optimization problem for MAT is simplified as:

$$\min \quad v \times T_{comp}(m) + T_{in}(m) + T_{out}(m) \tag{8}$$

$$\text{subject to} \quad P_c(m) \le P_s \tag{9}$$

$$1 \le m \le 94 \tag{10}$$

$$numCol \times m^{-1} \times v^{-1} \le 1 \tag{11}$$

This is a geometric programming model, and can be transformed into a convex model, quickly converging to a globally optimal solution m, given the supplied power P_s. The run-time optimizer running on the ARM processor can solve this optimization problem in less then 0.3 seconds.

Fig. 6 shows the variation of power from the solar harvester using the dot line. With this power supply, the power adaptive system for MAT adjusts its power consumption by clock gating as shown in Fig. 7. Note that when the supplied power is less than the lowest working power of the adaptive system, the system stands idle. The idle power could be provided by a battery, which can be charged by the harvester when

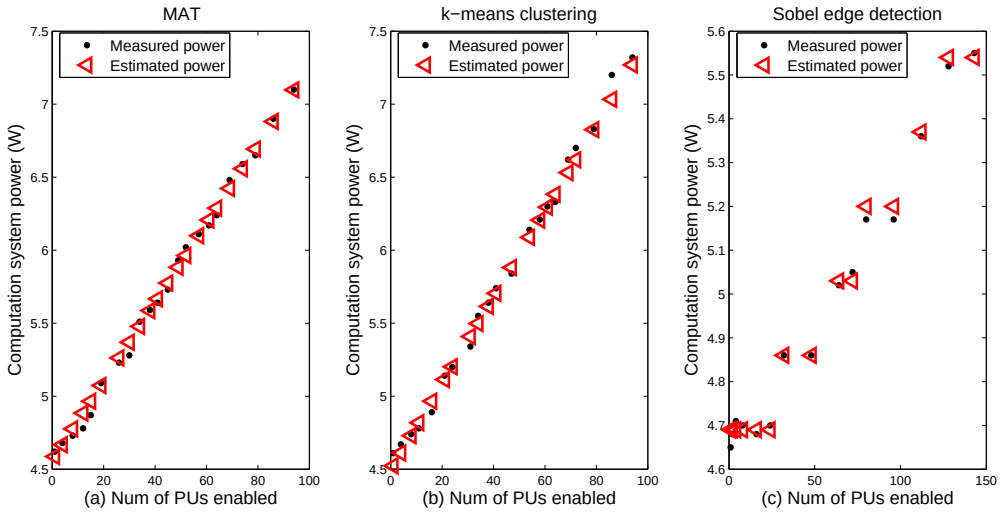

Fig. 5. Estimated power vs measured power for three different computational tasks.

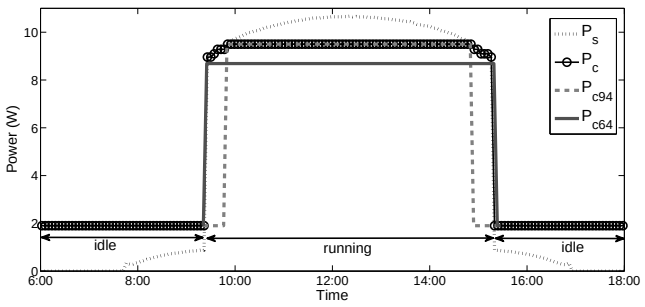

Fig. 6. Power variation of MAT. P_s: the supplied power. P_c: the power consumption of the adaptive system. P_{c94}: the power consumption of the system with 94 PUs always enabled. P_{c64}: the power consumption of the system with 64 PUs always enabled.

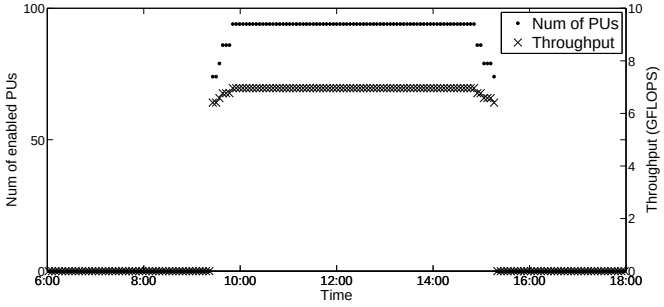

Fig. 7. Throughput and clock gating scheme variation of MAT.

power is residual. When the supplied power is greater than the lowest power requirement and keeps increasing, the computing system starts running and speed increases until all PUs are enabled as shown in Fig. 7. This dynamic performance can improve power efficiency. In Fig. 6 the power consumption of another two designs are also shown. These two designs do not exploit run-time optimization and operate with a fixed number of PUs; one has 94 PUs and another has 64 PUs. Compared to these two designs, our adaptive design improves the energy utility efficiency by about 16.19% and 8.58%, respectively. These are shown by the areas outlined by the circle line (adaptive design), the dashed line (94 PUs) and the solid line (64 PUs).

b) K-means clustering algorithm: classifies a multi-dimensional vector set into k sets, where the closest vectors in terms of the Euclidean distance belong to the same set [23]. The algorithm is a heuristic algorithm and usually runs iteratively to reach to the final clusters. This algorithm is commonly used for border detection and object recognition. In this experiment, we implement the algorithm for partitioning 10^5 64-dimensional vectors into 128 clusters, using 10 heuristic iterations; each iteration performs clustering operations within a 3-level loop nest. The entries of the vectors are single precision floating-point numbers.

The design-time optimization problem $\mathcal{P}_1$ maps the clustering operations in 96 PUs on the FPGA and each PU executes in full pipeline. The number of PUs is limited by the available FPGA on-chip DSP blocks. The parallel computation structure is similar to the one of MAT. The trained power model is

$$P_c(m) = 4.49 + 0.030m. \tag{12}$$

Fig. 5 (b) shows the power estimated by this model and the corresponding measured power for k-means with different clock gating schemes. The maximum relative error of the estimation is 2.32%.

The run-time optimization model can similarly be obtained. Similar results are shown in Figs. 8 and 9. The energy utility efficiency improvements are 28.8% and 9.63%, respectively, compared to two designs without run-time clock gating.

c) Sobel edge detection algorithm: detects object edges in images [24]. Two 3×3 mask windows move over an image pixel by pixel to calculate the gradients of each pixel in both row and column directions. The algorithm contains 4-level nested loops. In this experiment, the image size is CIF 288×352 pixels, and each pixel is represented by an 8-bit integer. The integer arithmetic operations in the edge detection

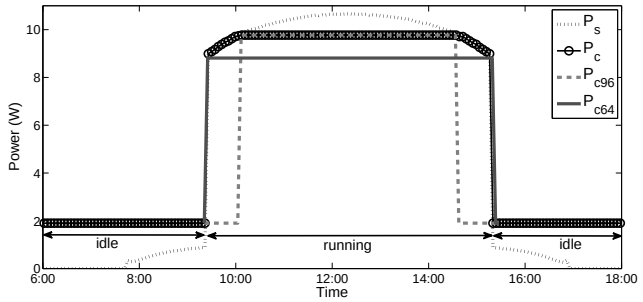

Fig. 8. Power variation of k-means clustering. P_s: the supplied power. P_c: the power consumption of the adaptive system. P_{c96}: the power consumption of the system with 96 PUs always enabled. P_{c64}: the power consumption of the system with 64 PUs always enabled.

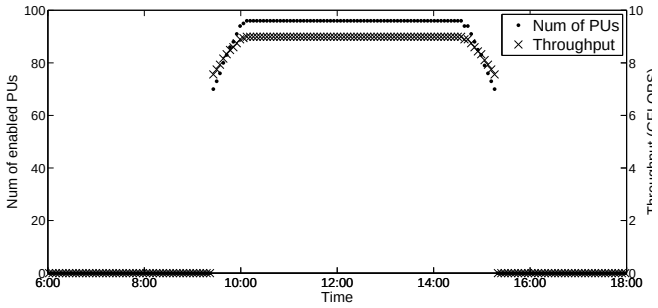

Fig. 9. Throughput and clock gating scheme variation of k-means clustering.

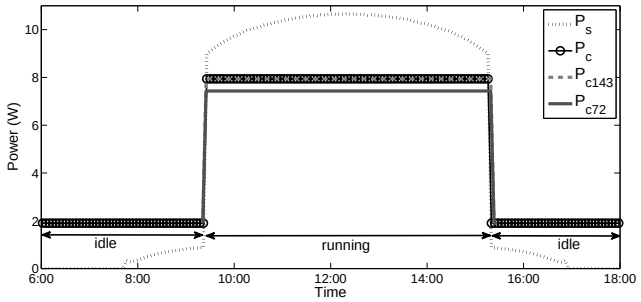

Fig. 10. Power variation of Sobel edge detection. P_s: the supplied power. P_c: the power consumption of the adaptive system. P_{c143}: the power consumption of the system with 143 PUs always enabled. P_{c72}: the power consumption of the system with 72 PUs always enabled.

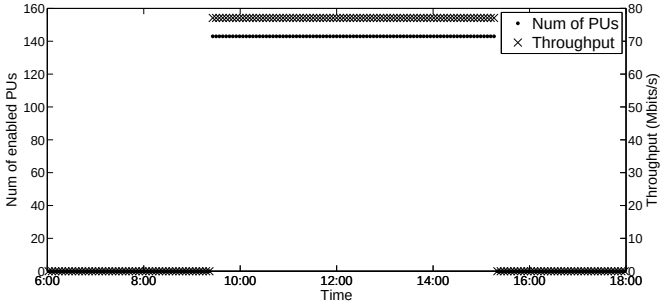

Fig. 11. Throughput and clock gating scheme variation of Sobel edge detection.

algorithm are realized using logic circuits rather than DSP blocks.

The design-time determined parallel structure is $k_1 = 143, k_2 = k_3 = k_4 = 1$ and $ii = 1$, where 143 rows of pixels are processed in parallel and the innermost two loops are fully pipelined. The degree of parallelism is limited by the on-chip RAM resources. Since the PU structure is relatively simple with just logic circuits, the power model for this design is

$$P_c(m) = 4.52 + 0.17\lceil m/24 \rceil. \tag{13}$$

where 24 is the number of PUs in each clock region. The maximum relative error of this estimation is 0.86%, as shown in Fig. 5 (c).

The working status of the adaptive system for Sobel edge detection is shown in Figs. 10 and 11, using the same power supply. Unlike the previous applications, the variations of the system power consumption and speed are not significant for most of the clock gating schemes. As a result, the energy utility efficiency of the adaptive system is similar to one of the design with 143 PUs always enabled, and is 6.86% improvement compared to one of the design with 72 PUs.

In addition, to demonstrate how the power adaptive computing systems improve the throughput/watt for the ARM processor, we experiment with two cases: running the three applications on the processor alone and running them on the processor plus the FPGA shown in Fig. 4. The results are shown in Table I. It is shown that the computing systems provide up to 19 times improvement in MFLOPS/W when compared to the processor.

We also compare two different approaches to determine the clock gating scheme. In this paper, we use a convex optimization problem as described in Section IV-B to find the global optimal clock gating scheme, given the supplied power. Alternatively, we can use the linear power model (2) to determine m as $\lfloor (P_s - P_{const})/P_{pu} \rfloor$. However, this approach may lead to a design, which uses more power than the convex optimization problem for the same speed. Table I compares the average power efficiency of the adaptive systems for the three benchmarks, using these two approaches. The supplied power varies from 6 W to 10 W with a 0.1 W increment. We can see that on average for MAT and Sobel the convex model can lead to designs with 1% higher power efficiency (corresponding to 0.08 MFLOPS/W and 0.01 Mbits/W) than the linear power model approach. The advantage of using the linear power model to determine m is the optimization response time (time for a division and a subtraction), while the convex optimization problem needs to solve a system of equations. If an adaptive system requires quick response time which cannot be met by the convex optimization problem, the linear power model approach is more promising. Nevertheless, the convex optimization is more general, because the power consumption model of a system may not have the linearity.

VI. Conclusion

In this paper, we propose a two-stage optimization approach for designing power adaptive computing systems applied to energy harvesting environments. The purpose is to provide

TABLE I
POWER EFFICIENCY COMPARISON. THE ARM PROCESSOR PERFORMANCE IS USED AS THE BASE LINE.

	ARM	ARM + Adaptive computing	
Application		Linear power model	Convex model
MAT	0.78 MFLOPS/W (1x)	14.65 MFLOPS/W (18.8x)	14.73 MFLOPS/W (18.9x)
k-means	0.62 MFLOPS/W (1x)	11.79 MFLOPS/W (19.0x)	11.79 MFLOPS/W (19.0x)
Sobel	0.13 Mbits/W (1x)	1.27 Mbits/W (9.8x)	1.28 Mbits/W (9.8x)

computation capability to nodes in distributed sensor networks. The power adaptive computation system contains multiple parallel processing units, each enabled/disabled independently. A run-time optimizer solves a convex model to decide how many processing units can be enabled, such that the system power consumption is not greater than the power supply from the harvester and the system computation speed is maximized. We develop the power adaptive computation systems for three applications following the proposed approach on a platform with a ARM9 processor and an FPGA, and evaluate the systems with harvested solar energy data. The developed power adaptive system can improve harvested energy utility efficiency up to 28.8%, compared to some static designs without adaptability; and can provide up to 19 times improvements in FLOPS/W to the ARM processor.

Future work includes investigating the efficiency of the energy estimator, studying the detailed behavior of the computation system and sensed data, integrating other dynamic power optimization techniques into the approach, and extending the design approach to other energy harvesting sources and to work at the network level.

Acknowledgement. This work was supported in part by UK EPSRC, by the European Union Seventh Framework Programme under Grant agreement number 248976 and 257906, by the HiPEAC NoE, by Alpha Data, by Celoxica, by nVidia, and by Xilinx."

REFERENCES

[1] S. Henzler, *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies (Springer Series in Advanced Microelectronics)*. Springer-Verlag New York, Inc., 2006.

[2] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, September 2007.

[3] J. Paradiso and T. Starner, "Energy scavenging for mobile and wireless electronics," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 18–27, 2005.

[4] P. Mitcheson, E. Yeatman, G. Rao, A. Holmes, and T. Green, "Energy harvesting from human and machine motion for wireless electronic devices," *Proceedings of the IEEE*, vol. 96, no. 9, pp. 1457–1486, 2008.

[5] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Information Processing in Sensor Networks,*, 2005, pp. 457–462.

[6] H. Li, S. Bhunia, Y. Chen, K. Roy, and T. Vijaykumar, "DCG: deterministic clock-gating for low-power microprocessor design," *IEEE Transs on VLSI*, vol. 12, no. 3, pp. 245–254, 2004.

[7] C. Alippi and C. Galperti, "An adaptive maximum power point tracker for maximising solar cell efficiency in wireless sensor nodes," in *IEEE International Symposium on Circuits and Systems*, 2006.

[8] J. Recas Piorno, C. Bergonzini, D. Atienza, and T. Simunic Rosing, "Prediction and management in energy harvested wireless sensor nodes," in *Proceedings of the conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology*, vol. 1, 2009, pp. 6–10.

[9] M. Ali, B. Al-Hashimi, J. Recas, and D. Atienza, "Evaluation and design exploration of solar harvested-energy prediction algorithm," in *DATE*, 2010, pp. 142–147.

[10] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," in *DATE*, 2008, pp. 236–241.

[11] B. Zhang, R. Simon, and H. Aydin, "Energy management for time-critical energy harvesting wireless sensor networks," in *Stabilization, Safety, and Security of Distributed Systems*, ser. Lecture Notes in Computer Science, S. Dolev, J. Cobb, M. Fischer, and M. Yung, Eds. Springer Berlin / Heidelberg, 2010, vol. 6366, pp. 236–251.

[12] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management in energy harvesting systems," in *DATE*, 2007, pp. 1–6.

[13] ——, "Robust and low complexity rate control for solar powered sensors," in *DATE*, 2008, pp. 230–235.

[14] D. Noh, L. Wang, Y. Yang, H. Le, and T. Abdelzaher, "Minimum variance energy allocation for a solar-powered sensor system," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, vol. 5516, pp. 44–57.

[15] A. Yakovlev, "Energy-modulated computing," in *DATE11, Invited Paper in Special Day on Intelligent Energy Management*, 2011. [Online]. Available: http://async.org.uk/tech-reports/NCL-EECE-MSD-TR-2010-167.pdf

[16] F. Xia, A. Mokhov, Y. Zhou, Y. Chen, I. Mitrani, D. Shang, D. Sokolov, and A. Yakovlev, "Towards power elastic systems through concurrency management," in *Tech report, Microelectronic System Design Group, School of EECE, Newcastle University*, 2010. [Online]. Available: http://async.org.uk/tech-reports/NCL-EECE-MSD-TR-2010-155.pdf

[17] Y. Chen, I. Mitrani, D. Shang, F. Xia, and A. Yakovlev, "Stochastic analysis of power, latency and the degree of concurrency," in *IEEE International Symposium on Circuits and Systems*, 2010, pp. 4129–4132.

[18] A. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2000, pp. 114–119.

[19] A. Nahapetian, P. Lombardo, A. Acquaviva, L. Benini, and M. Sarrafzadeh, "Dynamic reconfiguration in sensor networks with regenerative energy sources," in *DATE*, 2007, pp. 1054–1059.

[20] "Oak Ridge National Laboratory (ORNL) RSR web site," (2010 Oct.). [Online]. Available: http://www.nrel.gov/midc/ornl_rsr/

[21] Q. Liu, T. Todman, and W. Luk, "Combining optimizations in automated low power design," in *DATE*, 2010, pp. 1791–1796.

[22] J. C. Meza, R. A. Oliva, P. D. Hough, and P. J. Williams, "OPT++: An object-oriented toolkit for nonlinear optimization," *ACM Trans. Math. Softw.*, vol. 33, June 2007.

[23] (2010). [Online]. Available: http://www.eee.bham.ac.uk/russellm/ee3j2/k-means.c

[24] (2006). [Online]. Available: http://www.pages.drexel.edu/ weg22/edge.html