Dynamic Reconfiguration in Sensor Networks with Regenerative Energy Sources

Ani Nahapetian[1], Paolo Lombardo[2], Andrea Acquaviva[3], Luca Benini[2], Majid Sarrafzadeh[3]

[3]Computer Science Department, University of California, Los Angeles (UCLA), Los Angeles, California, USA
[2]Dipartimento di Elettronica, Informatica e Sistemistica (DEIS), Università Bologna, Bologna, Italy
[3]Information Science and Technology Institute (ISTI), Università di Urbino, Urbino, Italy

ABSTRACT

In highly power constrained sensor networks, harvesting energy from the environment makes prolonged or even perpetual execution feasible. In such energy harvesting systems, energy sources are characterized as being regenerative. Regenerative energy sources fundamentally change the problem of power scheduling for embedded devices. Instead of the problem being one of maximizing the lifetime of the system given a total amount of energy, as in traditional battery powered devices, the problem becomes one of preventing energy depletion at any given time.

Coupling relatively computationally intensive applications, such as video processing applications, with the constrained FPGAs that are feasible on power constrained embedded systems, makes dynamic reconfiguration essential. It provides the speed comparable to a hardware implementation, but it also allows the dynamic reconfiguration to meet the multiple application needs of the system. Different applications can be loaded on the FPGA, as the system's needs change over time. The problem becomes how to schedule the dynamic reconfiguration to appropriately make use of the regenerative energy source, to ensure the proper availability of energy for the system over time.

In this paper, we present a methodology for carrying out dynamic reconfiguration for regenerative energy sources, based on statistical analysis of tasks and supply energy. The approach is evaluated through extensive simulations. Additionally, we have evaluated our implementation on our regenerative energy, dynamically reconfigurable prototype, known as the MicrelEye. Our approach is shown to miss 57.7% less deadlines on average than the current approach for reconfiguration with regenerative energy sources.

1. INTRODUCTION

In an effort to provide long, or even perpetual, lifetime distributed embedded nodes, with the fast and flexible functionality of dynamically reconfigurable systems, systems are being developed that utilize both low power and dynamically reconfigurable components. Hardware execution is known to be more energy efficient, and with the flexibility of FPGAs, the integration of FPGAs into sensor nodes is posed to present great benefits, especially since there are now low-power solutions that integrate FPGAs on chip (such as ATMEL), and there is continuing research into low-power FPGAs. Aside from limited energy availability, another key challenge facing sensor networks is the highly limited computational resources. Dynamic reconfiguration allows the execution of different types of task with the speed and the energy efficiency of hardware. The different tasks can be due to the fact that the nodes are equipped with multiple sensors or because the complexity of the task dictates that it is divided into subtasks, as is the case with our video sensor prototype presented in subsequent sections.

Capturing energy from the environment is referred to as energy harvesting or energy scavenging. Systems that obtain or supplement their energy supply with energy captured from the environment are characterized as having regenerative energy sources. These systems are fundamentally different from battery-powered systems because instead of working with a limited total available energy, they must optimize for proper operation given limited energy availability at any instance in time [7]. As a result, there are instances, where it may be beneficial to consume energy. Additionally there can be considerable variability in energy availability. This information may be predictable, depending on the characterizability of the energy sources. Regenerative energy for sensor networks is a novel approach for overcoming a critical problem of power consumption. For certain applications, especially networks deployed in open spaces, solar energy harvesting can be a very dependable source of energy for operation. Finally, with regenerative energy sources, sensor networks can adapt themselves to perpetual operation [8][4]. Energy can be harvested from various renewable sources. There exist several prototype systems with various regenerative energy sources. [12] gives a history and a survey of different types of energy harvesting systems, including their own ambulatory motion energy harvesting shoe prototype. [2] provides a system for vibration energy harvesting. The Prometheus project [4] and the Heliomote project [5] are prototypes for utilizing solar power. [14] discusses the feasibility of a network consisting of mobile nodes that roam the environment in search of energy for all the nodes, taking the concept to the robotics realm. A few papers have taken a look at the scheduling aspects of regenerative energy systems, including [1][15] which utilize dynamic voltage scaling to approach the problem and [10][11] which provide an online scheduling approach, which is independent of a dynamic voltage scaling approach.

In this work, however, we consider systems that require the speed of hardware, but because of complexity of the application or the limited nature of the hardware, need to have the flexibility of software. Dynamically reconfigurable resources bridge this gap, and provide the speed of an FPGA, but with the flexibility of being reconfigured on the fly at runtime, and hence allowing much more complicated applications on a limited FPGA resource.

With the increasing demand for real-time and computation intensive applications, such as sophisticated video and audio processing applications, on limited embedded systems such as sensor networks, the need to carry out dynamic reconfiguration becomes even greater. Reconfigurability in sensor networks has been explored by [16] for their PicoRadio low power, sensor network, to accommodate for different networking protocols. [9] examined dynamic software reconfiguration in sensor networks, using a constraint programming based approach. They verified their results using a simulation of a one-dimensional tracking application. The combination of a regenerative energy system,

978-3-9810801-2-4/DATE07 © 2007 EDAA

enabled with dynamic runtime reconfigurarabilty has first been examined by [3]. The paper, however, presents a simple approach to runtime reconfiguration, which we enhance in this paper with great improvements in terms of deadline misses. The paper shows that runtime reconfiguration is feasible and useful. They use a high power consuming system as a proof of concept. Here on the hand, we take a look at a real low power system equipped with the FPSLIC architecture and solar energy harvesting ability. Also, the paper [3] has worked under the assumption that if there is enough energy to reconfigure the hardware, then it is beneficial to reconfigure the hardware. This assumption is not necessarily true.

Table 1. Example

Task Type	SW Energy Requirement	HW Energy Requirement	Reconfiguration Cost
1	25	10	10
2	15	2	

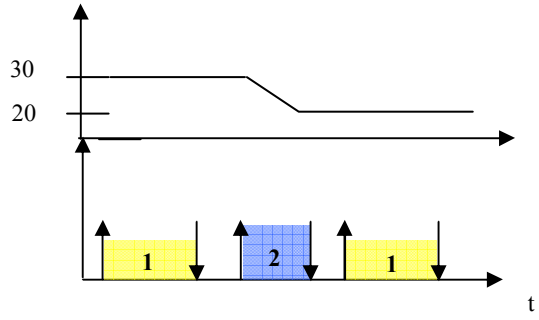

Figure 1. Example

Consider the following example where you are given two types of tasks, 1 and 2, and three tasks total. Table 1 gives their energy requirement for execution in software and in hardware. The reconfiguration cost for the hardware is also given. Figure 1 demonstrates the "reconfigure if able" approach, where the top curve represents the energy stored. During the start of execution of task 1 and 2, there is enough energy to reconfigure and run the tasks in hardware. During the execution of the second task, the energy stored is depleted and not replenished. As a result, there is not enough energy to execute the third task, either in hardware or in software. On the other hand, had we saved the reconfiguration obtained for the first task and run the second task in software, there would have been enough energy to execute the third task in hardware. Thus violating no deadlines. The example demonstrates that a reconfiguration of the hardware, even when there is a large availability of energy, can be more costly in the future, when there may be a limited amount of energy. Maintaining a valuable reconfiguration could be more valuable for low energy periods than a less valuable reconfiguration, even if it wastes energy in the short term.

2. PROBLEM FORMULATION

In this paper, we specifically address the problem of scheduling tasks onto hardware or software for execution, while manipulating the energy provided by regenerative sources. This is coupled with problem of determining when to reconfigure the FPGA.

More formally, the problem can be formulated as given tasks, heterogeneous resources, and a regenerative energy source, the objective is to ensure the execution of the largest number of tasks, within their availability interval. (In the case of dependencies between tasks, without violating a dependency.) Each task i is characterized by an arrival time (a_i), a hard deadline (d_i), an energy requirement for execution on hardware (H_i) and on software (S_i), and a type distinguishing which reconfiguration profile they require from the library. Task types identify whether a reconfiguration is needed between the execution of two consecutive tasks. Tasks of different types require a reconfiguration. There is both a processor on which a software implementation can be executed, as well as hardware resources with a known reconfiguration cost (or costs). There is also the possibility of porting the reconfiguration data from an external source. Finally, there is a regenerative energy source with an energy buffer, whose energy loss over time we assume to be insignificant, and an external source of energy, which can vary significantly over time. The energy buffer, however, has a limited storage capacity, and after it reaches its capacity any additional energy is not captured by the system.

2.1 Assumptions

A key assumption in this work is that there exists both a software and a hardware version of tasks. The system can handle the case where certain tasks have only a single implementation, but the potential for energy savings is diminished if there is only a single implementation of tasks. From the viewpoint of the problem formulation, this only changes the reconfiguration cost that has to be characterized off-line. In addition to the reconfiguration cost, the energy consumption of hardware and software task executions must be profiled to determine the best execution strategy. If this information is not available, it may be collected during the execution of the first few occurrences of the task.

To complete the problem formulation, we impose two relationships among the hardware, the software, and the reconfiguration energy cost for each task, i, (H_i, S_i, R_i respectively). First, we assume that software execution is more convenient than performing reconfiguration followed by hardware execution. This is expressed by the following equation: $S_i \leq H_i + R_i$. The assumption ensures that software execution makes sense. Otherwise, there would be very little reason to use the software version, unless we were able to parallelize software and hardware executions. Reciprocally, the cost of running a task on hardware is less expensive, in terms of energy, than running a task on software, ignoring the cost of reconfiguration, as given the following equation: $H_i \leq S_i$. If this assumption were not valid, then hardware execution would be significantly less energy efficient than the software execution, and hence there would be no benefit to using the hardware from an energy viewpoint.

2.2 Key Observations

A few a key observations can be made about the problem, which can help in the formulation of the problem solution.

1. At any given moment, only the last reconfiguration is important for future reconfigurations and scheduling.
2. If there is a large supply of energy (i.e. larger than what is needed to fill the storage to capacity), then carrying out reconfigurations is valuable and can be carried out without negatively affecting the execution of tasks in the future.

3. If the current task has a large differential between its software execution and its hardware execution cost, then a reconfiguration is valuable.
4. If the current task is frequent, then a reconfiguration is valuable.

3. STATISTICAL APPROACH

The key observations presented in the previous section have lead to an approach that uses statistically gathered information about task arrival and energy availability characteristics to carryout appropriate reconfigurations. Recall that for tasks of different types, a reconfiguration is required between consecutive executions on hardware, whereas tasks of the same type do not require reconfiguration. By evaluating the expected energy after a number of future task executions, we can determine the benefit of carrying out a reconfiguration now. The expected energy availability can be estimated by the following equation:

$$Exp(E) = E_{current} - R - H_j + Exp(E_A) \cdot F - \left(Exp(E_{type \neq j}) + Exp(E_{type=j})\right) \cdot F \quad (1)$$

For the current task i of type j, $E_{current}$ is the current available energy, R is the reconfiguration cost (we assume without loss of generality that it is constant for each task), H_j is the cost of running tasks of type j on hardware, and $Exp(E_A)$ is the expected additional energy to be added to the energy storage because of the refill effect of the scavenger source. $Exp(E_{type \neq j})$ is the expected cost of running the next task, of a type other than j on software, and $Exp(E_{type = j})$ is the expected cost of running the next task of type j on hardware, scaled by the likelihood of such a task type occurring. F represents the number of tasks into the future the analysis is carried out for. This is formalized below.

$$Exp(E_{type \neq j}) = \sum_{l \neq j, l=1}^{TT} \frac{N_l}{\sum_{k=1}^{TT} N_k} S_l \quad (2)$$

$$Exp(E_{type=j}) = \frac{N_j}{\sum_{k=1}^{TT} N_k} H_j \quad (3)$$

In the previous equations, TT refers to the number of task types, N_i refers to number of occurrences of tasks of type i, S_i refers to the energy cost of running task i on software, and H_i refers to the energy cost of running task i on hardware. Equations (2) and (3) calculate the expected energy cost of running the next task. To determine the proper expected cost, we have to estimate the probability that the next task type is other than the current one. This is obtained by multiplying the actual cost (S or H) by the probability that the task is (or is not) of the same type of the current one. The probability is obtained by averaging the sum the number of occurrences observed for a certain task type.

This sort of analysis can be used to determine if it is appropriate to reconfigure the hardware or not. If it is observed that the expected energy cost is negative, or even below a certain threshold, then the reconfiguration is not carried. Equation (2) and (3) assume that task of the same type have the same hardware and software energy costs. This may not necessarily be the case. The energy cost, may be determined by other parameters. Our model is robust enough to handle this case. When determining the expected cost of task types, the energy cost of the tasks can be calculated using the individual task characteristics, instead of the task type characteristics.

This analysis can be extended to order-two statistics, that is where we maintain statistics on the possibility of a task following another task. We maintain statistics on the pairs of tasks, instead of individual tasks. As a result, Equations (2) and (3) will be replaced by Equations (4) and (5), respectively.

$$Exp(E_{type \neq j}) = \sum_{l \neq j, l=1}^{TT} \frac{N_{j,l}}{\sum_{k=1}^{TT} N_k - 1} S_l \quad (4)$$

$$Exp(E_{type=j}) = \frac{N_{j,j}}{\sum_{k=1}^{TT} N_k - 1} H_j \quad (5)$$

There are various ways to compute the expected additional energy, and this question extensively studied in [8]. As shown in Equation (6), we use the product of the expected length of time until the arrival of the next task, D, and the estimated available power, $P_{expected}$.

$$Exp(E_A) = P_{\exp ected} \cdot Exp(D) \quad (6)$$

4. SIMULATION RESULTS

Simulations were conducted to determine the effectiveness of our approach. A comparison with five other approaches, detailed below, is presented.

Random – The random approach aims to run a task on hardware 50% of the time. If there is not enough energy, then it attempts to run the task on software. If the random approach does not run the task on hardware, it attempts to run the task in software. If there is not enough energy, the task misses its deadline.

All-hardware – The all-hardware approach always runs the task on hardware. If there is not enough energy, it misses the deadline.

Reconfigure-if-able – The reconfigure if you have enough energy approach aims to run the tasks on hardware, by reconfiguring if needed. If there is not enough energy to reconfigure, then the task is run in software, and if there is not enough energy to run the task in software, then the deadline is missed. This approach was the one taken by [3].

All-software – The all-software approach always runs the task on software. If there is not enough energy, it misses the deadline.

Statistical – The statistical approach calculates the expected energy after the execution of two tasks, with the approach presented in section 5. If the task is not run on software, as in the random and the reconfigure-if-able approaches, it attempts to run the task in software, and if there is still not enough energy it misses the deadline.

Oracle – The oracle approach is aware of the immediate harvested energy profile and future tasks. It is used to provide a lower bound, as it has the greatest amount of information to make the

decisions. However, it has limited foresight, and hence is not optimal.

In order to determine the exact characteristics of the system that would make a specific approach more useful, we carried out extensive simulations. In these simulations, we utilized a randomly generated harvested energy profile, and then varied one component of the system: software execution cost, hardware execution cost, or reconfiguration cost. The experiments used two types of tasks, whose arrival times and deadlines were randomly generated. The reconfiguration cost, the hardware cost, and the software cost of the two tasks were determined by a ratio based on the real measurements made on the prototype system described in the next section. The ratio of the reconfiguration cost to task 1's software cost to task 1's hardware cost to task 2's software cost to task 2's hardware cost is 30:30:6:6:4. In the graphs the average deadline misses are presented, for the six different approaches. The statistical approach almost consistently outperforms the other approaches, by missing fewer deadlines. It misses only marginally more deadlines than the oracle, which has perfect knowledge of future tasks and the future harvesting energy profile.

Figure 4 presents the results of varying the software costs. The two tasks' software costs were obtained by multiplying the baseline software costs by a factor, which is plotted along the x-axis. The ratio of the software costs of the two tasks was 5:1. The hardware and reconfiguration costs were kept constant. In these simulations, we varied the factor by which all the software costs were multiplied. As is expected, the all-hardware approach performs poorly, regardless of the software cost. When software costs are very low, all of the approaches, except the all-hardware approach, perform well missing no or very few deadlines. However, the statistical approach outperforms all the other approaches, except the oracle, for the full range of software costs.

Figure 5 presents the results from varying the hardware costs in a similar fashion, except that the x-axis represents the factor multiplied to the baselines hardware costs. In these simulations, we varied the factor by which all the baseline hardware costs were multiplied, as we did with the software costs in the previous figure. The ratio of 3:2 was maintained for the hardware costs of the two tasks. The all-software approach's performance is same for the different hardware costs, as would be expected. The hardware based approaches, however, do not perform as well as the all-software approach in the previous figure, because there is a reconfiguration cost that must be paid, even if the cost of running the task on hardware is 0. The statistical approach performs very well compared to the other approaches and is very close to the oracle in all of the test cases. On average the statistical approach misses 35.3%, 82.6%, and 87.4% less deadlines than the all-software, the reconfigure-if-able, and the all-hardware approaches, respectively.

In Figure 6, we keep all the variables of the problem constant and vary the ratio of the reconfiguration cost to the task execution costs. As would be expected, Figure 6 shows that the performance of the all-software approach is the same regardless of the reconfigurations cost. As the reconfiguration cost becomes too large to make hardware execution feasible, the reconfigure-if-able, statistical, and oracle approaches converge to the performance of the all-software approach. The all-hardware approach performs very poorly throughout, because it misses tasks when there is not enough energy to reconfigure, even if there is enough energy to run that task in software. The statistical approach is almost equal to the oracle, which has both task and energy information for the near future. Also, the algorithm consistently outperforms the reconfigure-if-able, the all-software, and the all-hardware approaches, with the exception of one test case where there is a reconfiguration cost of 0. On average the statistical approach misses 42.7%, 57.7%, and 68.2% less deadlines than the all-software, reconfigure-if-able, and all-hardware approaches, respectively.

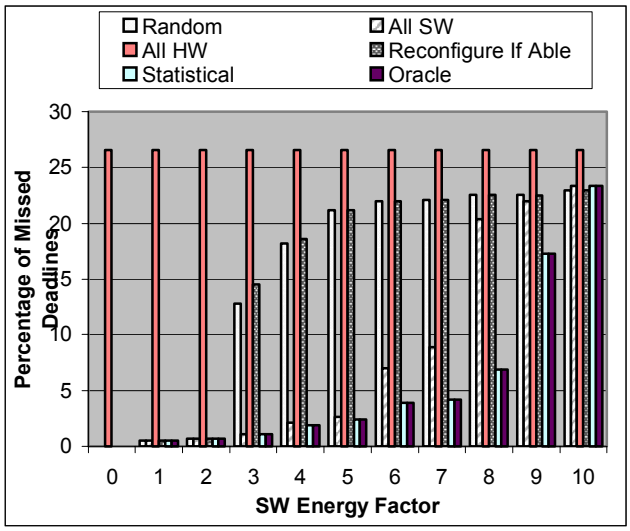

Figure 4. Deadline Misses for Various Software Energy Costs

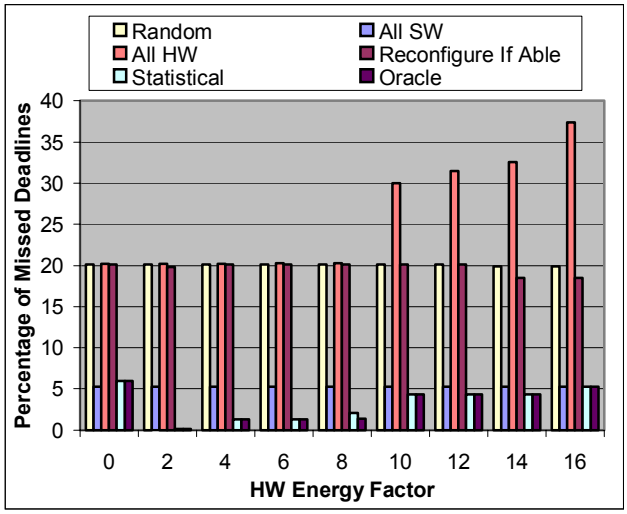

Figure 5. Deadline Misses for Various Hardware Energy Costs

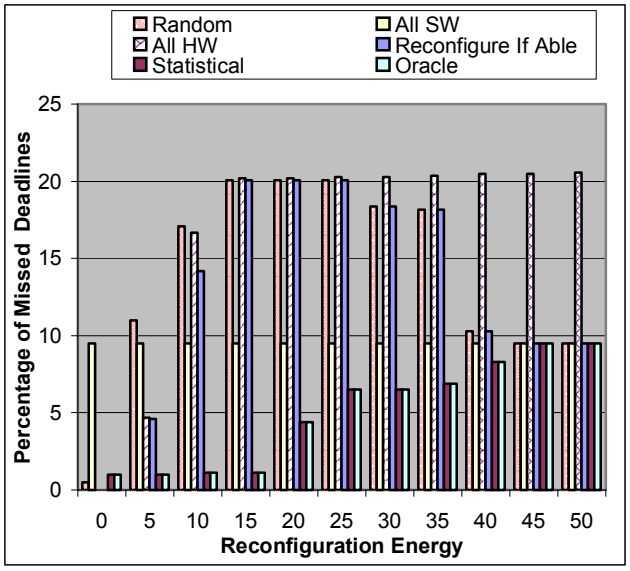

Figure 6. Deadline Misses for Various Reconfiguration Costs

5. CASE STUDY – MICRELEYE

This work has been targeted to the MicrelEye platform, shown in Figure 7. This node harvests power from a single solar cell and is equipped with a battery, which is used when the solar cell's energy availability is not sufficient. The MicrelEye features an Omnivision 7640 video sensor, which requires computation-intensive processing on the captured images. Also, image recognition is real-time constrained: processing time should match the frame rate of the video sensor. The system has a Bluetooth transceiver and hence is capable of networking with other MicrelEyes or with a network gateway. This implies that reconfiguration bitstreams can be downloaded from the network. Thus, the reconfiguration cost can include the cost of downloading the bitfile from the network. Finally, the system is equipped with an ATMEL FPSLIC configurable platform, featuring an AVR microcontroller and 40K gate FPGA. FPSLIC is the one of the lowest power consuming reconfigurable systems on the market. The key feature of the system is that there are both hardware and software versions of tasks. The problem is one of partioning the work between the two resources, given the new model of regenerative energy. Each task can be run on the microcontroller or on the FPGA. Thus the MicrelEye has access to both the bitfile necessary for hardware reconfiguration and the executable necessary for software execution. At runtime, using the algorithm presented, the approach will pick between the two versions to use to run tasks as they arrive dynamically. The challenge however is that the tasks do not fit on the hardware at the same time, and thus task executions must be scheduled onto the resources.

The tasks that we have developed for execution are *thresholding* and *edge detection*. Thresholding converts a frame from its full 8-bit, gray scale representation (or 24-bit, RGB representation) to a single bit representation for each pixel. It converts each pixel to either white or black, depending on a threshold value. The resulting images can be thought of as a silhouetted version of the original image. This is useful for object detection. Edge detection is a key step in object detection and tracking. In our case it is carried out by multiplying nine neighboring pixels with the Laplacian matrix, taking their absolute value and determining if it is above a certain threshold. This technique is refereed to as Laplacian edge detection.

Figure 7. MicrelEye: Video Node Prototype

With our prototype, we first characterized the task execution and hardware reconfiguration costs, which are summarized in Table 2. We, then, implemented a vision application control flow, shown in Figure 8, that is used for object detection and tracking, using the thresholding and edge detection tasks. A threshold value, τ, is used to detect the percentage of successfully tracked objects during the execution of the application to determine the next task that needs to be executed. We ran the random, all-software, all-hardware, and reconfigure-if-able approaches on nice different sequences of test frames, captured by the video sensor. Each sequence leads to a different number of detected objects over time, triggering the thresholding task with different frequencies given a fixed threshold, which in our case was 40%.

Table 2. Measured Energy Values for MicrelEye System

Application	Reconfiguration Energy (mJ)	SW Energy (mJ)	HW Energy (mJ)
Thresholding	25.0	8.93	4.48
Edge Detection	37.4	28.08	6.60

The numerous experiments on this platform are summarized in Figure 9, where the percentage of missed deadlines is shown for the six different approaches on the nine different frame sequences. The experiments show that the statistical approach presented in the previous sections is indeed close to or identical to the percentage of missed deadlines by the oracle, even on real video applications. Also the approach far outperforms the random, all-software, all-hardware, and reconfigure-if-able approaches. The all-hardware and the reconfigure-if-able approaches perform worse than expected, because of their bias towards using the hardware implementation over the software implementation. The version of FPSLIC used in the prototype does not support partial reconfiguration, and hence reconfiguration is costly, causing reconfiguration favoring approaches to deplete the stored energy quickly. Additionally, due to limited I/O capabilities of the FPGA the potential to fully parallelize the hardware implementation is limited. However, our approach also outperforms the all-software approach, since it intelligently uses a combination of hardware and software.

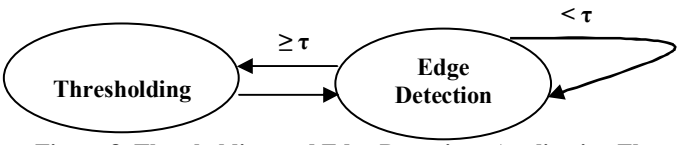

Figure 8. Thresholding and Edge Detections Application Flow

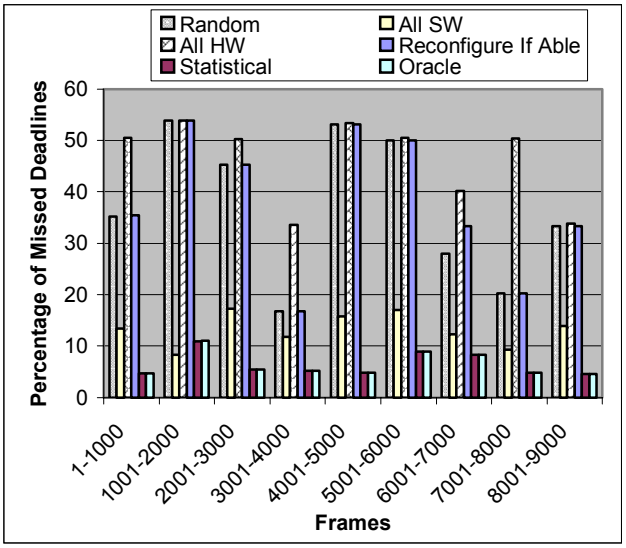

Figure 9. Deadline Misses for Various Frame Sequences

6. CONCLUSION

In this paper, we discussed the paradigm shift caused by regenerative energy sources, as compared to traditional battery powered systems, in power aware scheduling of task executions. Additionally, we presented the need to integrate reconfigurable devices into sensor networks nodes, given the growing number of low-power FPGAs coupled with the need for fairly sophisticated fast computation on sensor network nodes. We presented a statistically based approach to schedule tasks onto hardware and software and to reconfigure hardware. We evaluated the approach using extensive simulations, where our approach was found to be very close in value to the oracle, which is aware of the near future in terms of task arrival and energy availability. Finally, we presented a prototype system, for which our system has been developed and implemented. Again we showed the large advantage in terms of deadline misses that our approach has over the random, the all-software, the all-hardware, and the reconfigure-if-able approaches.

7. REFERENCES

[1] A. Allavena and D. Mossé, Scheduling of Frame-based Embedded Systems with Rechargeable Batteries. In *Proceedings of IEEE Workshop on Power Management for Real-Time and Embedded Systems (in conjuction with RTAS'01)*, 2001

[2] Y. Ammar, A. Buhrig, M. Marzencki, B. Charlot, S. Basrour and M. Renaudin, Wireless sensor network node with asynchronous architecture and vibration harvesting micro power generator. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient intelligence: innovative Context-Aware Services: Usages and Technologies*, 2005.

[3] I. Folcarelli, A. Susu, T. Kluter, G. De Micheli, A. Acquaviva, An opportunistic reconfiguration strategy for environmentally powered devices. In *Proceedings of the 3rd Conference on Computing Frontiers (CF '06)*, 2006.

[4] X. Jiang, J. Polastre, and D. Culler, Perpetual Environmentally Powered Sensor Networks. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, 2005

[5] HelioMote Project. http://research.cens.ucla.edu/portal/page?_pageid=56,55124,56_55125&_dad=portal&_schema=PORTAL

[6] A. Kansal, D. Potter and M.B. Srivastava, Performance Aware Tasking for Environmentally Powered Sensor Networks. In *Proceedings of ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2004.

[7] A. Kansal, J. Hsu, M. B. Srivastava, V. Raghunathan, Harvesting Aware Power Management for Sensor Networks. *Proceedings of the 43rd Design Automation Conference (DAC '06)*, 2006.

[8] A. Kansal, J. Hsu, S. Zahedi, M. B. Srivastava, Power Management in Energy Harvesting Sensor Networks. *ACM Transactions on Embedded Computing Systems (in revision)*, May 2006.

[9] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti. Constraint-guided dynamic reconfiguration in sensor networks. *Proceedings of the Third international Symposium on information Processing in Sensor Networks (IPSN '04),* 2004.

[10] C. Moser, D. Brunelli, L. Thiele and L. Benini. Real-time Scheduling with Regenerative Energy. In *The Proceedings of 18th Euromicro Conference on Real-Time Systems (ECRTS '06)*, 2006.

[11] C. Moser, D. Brunelli, L. Thiele and L. Benini. Lazy Scheduling for Energy Harvesting Sensor Nodes. In *The Proceedings of Fifth IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES '06)*, 2006.

[12] J.A. Paradiso, T. Starner, Energy Scavenging for Mobile and Wireless Electronics. Pervasive Computing, pp. 18-27, January-March, 2005.

[13] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M.B. Srivastava, Design Considerations for Solar Energy Harvesting Wireless Embedded Systems. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, 2005.

[14] M. Rahimi, H. Shah, G. Sukhatme, J. Heidemann, and D. Estrin. Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003.

[15] C. Rusu, R. Melhem, and D. Mossé, Multi-version Scheduling in Rechargeable Energy-aware Real-time Systems. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS '03)*, 2003.

[16] T. Tuan, S.F. Li, J. Rabaey. Reconfigurable platform design for wireless protocol processors. *Proceedings 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings,* 2001.