EPROF: An Energy/Performance/Reliability Optimization Framework for Streaming Applications

Yavuz Yetim
Princeton University
yyetim@princeton.edu

Sharad Malik
Princeton University
sharad@princeton.edu

Margaret Martonosi
Princeton University
mrm@princeton.edu

Abstract— Computer systems face increasing challenges in simultaneously meeting an application's energy, performance, and reliability goals. While energy and performance tradeoffs have been studied through different dynamic voltage and frequency scaling (DVFS) policies and power management schemes, tradeoffs of energy and performance with reliability have not been studied for general purpose computing. This is particularly relevant for application domains such as multimedia, where some limited application error tolerance can be exploited to reduce energy [7].

In this paper, we present EPROF, an optimization framework based on Mixed-Integer Linear Programming (MILP) that selects possible schedules for running tasks on multiprocessors in order to minimize energy while meeting constraints on application performance and reliability. We consider parallel applications that express (on task graphs) the performance and reliability goals they need to achieve, and that run on chip multiprocessors made up of heterogeneous processor cores that offer different energy/performance/reli-ability tradeoffs. For the StreamIt benchmarks [16], EPROF can identify schedules that offer up to 34% energy reduction over a baseline method while achieving the targeted performance and reliability. More broadly, EPROF demonstrates how these three degrees of freedom (energy, performance and reliability) can be flexibly exploited as needed for different applications.

I. Introduction

Continuing the trajectory of Moore's Law scaling has pushed microprocessors into a realm where satisfying application performance goals on constrained energy budgets is increasingly challenging. One approach to addressing this has been to explore tradeoffs in which gate or circuit level reliability is sacrificed in order to achieve higher performance at acceptable energy levels (e.g., [5]). Sacrificing reliability for performance may be acceptable if these errors can be corrected [5], or if they do not always impact functional correctness. For example, some gate-level errors are not necessarily visible at the architecture or software level, e.g. single-bit errors in a branch predictor or bit-flips in portions of the data cache that are currently unused [13]. Further, many current software applications can also tolerate infrequent low-level errors. For example, image processing and audio applications can both tolerate a modest numbers of bit-flips in their picture or sound data without any detectable effect on the application behavior. For both these reasons, prior work has explored particular instances of this tradeoff between energy and resiliency (e.g., [1, 7, 10, 11, 14]).

Increasingly, chip multiprocessors are likely to be heterogeneous collections of processor cores with different energy, performance, and reliability characteristics. This heterogeneity might arise by design (e.g., ERSA assumes 1 reliable core and N cheaper but less-reliable ones [11]) or might arise through technology variations that result in heterogeneous processor behaviors. Our goal is to exploit this heterogeneity to run applications with "just enough" reliability, and to meet performance constraints with the least energy. We focus here on applications where this can be accomplished through static scheduling of application components on such heterogeneous multiprocessors. Streaming media applications (such as the StreamIt [16] benchmarks) are ideal candidates for this.

This paper proposes EPROF and evaluates its effectiveness for static scheduling. The static scheduling is done using the EPROF framework. One input to EPROF is the energy, performance, and reliability characteristics of each core. The second is an application task graph that sets performance targets (e.g. real-time or soft-real-time deadlines) and reliability requirements. An example might be "This portion of the application can tolerate no more than E errors per data set.". EPROF uses these inputs in a novel Mixed Integer Linear Programming (MILP) [3, 4] formulation to identify the best possible schedule. The optimal schedule output by the MILP solver will exploit low-energy processors whenever possible, as long as using them does not violate timing deadlines (if they are slow), reliability targets (if they have more frequent errors), or mutual exclusion rules (if resources cannot also be in use by another task within the graph). Our scheduling constraints encompass the aspects of the execution of an application on an occasionally-faulty multi-processor system while still providing linearity and simplicity in the formulation.

While other research has explored time-only or time-energy schedulers (e.g., [4, 6]), or considered reliability in the scheduling through duplication [17], ours is the first to develop a technique for optimizing the three-dimensional energy, performance and reliability schedule that exploits intrinsic limited error tolerance in application classes such as multimedia. The corresponding MILP formulation is experimentally validated through scheduling the StreamIt applications on representative heterogeneous chip multiprocessors (CMPs).

The contributions of this work are as follows:

1. The EPROF scheduling framework is the first to manage errors in a hardware-software system to exploit energy, performance and reliability tradeoffs. Deadlines and error requirements can be specified for portions of an application's task graph, in order to pinpoint which parts of a program are more or less error-tolerant.
2. The MILP formulation has several novel components. Our strategy of making an "infinite" pipeline into a single schedulable stage manages complexity and allows us to schedule full-sized streaming programs. In addition, we have developed linear constraints for enforcing mutual exclusion of critical sections in parallel programs. This allows us to schedule task graphs for interesting real-world parallel programs. Finally, we show how reliability constraints can be naturally captured in linear form.
3. Through experimentation on a range of hardware configu-

TABLE I
EXAMPLE POWER AND ERROR SETTINGS FROM [10]

Core Settings	Error Rate (errors/ns)	Busy Power (W)
Perfect	0	13.20
Moderate Error	0.067	10.74
High Error	0.212	9.69

rations and benchmarks, our work demonstrates the high leverage available to system designers by exploiting energy, performance and reliability tradeoffs. For the six StreamIt applications studied, energy savings of up to 34% are possible while meeting performance and reliability targets.

The remainder of this paper is structured as follows. Section II describes the formulation of our scheduling and optimization framework. Section III describes our experimental methodology, and gives experimental results. Section IV discusses related work, and Section V offers our conclusions and directions for future work.

II. MILP SCHEDULING FORMULATION

The scheduler takes as input the application and hardware parameters and achieves a minimum energy schedule that satisfies the deadline and reliability constraints. To achieve this, it assigns tasks to different cores with different power, performance, and reliability characteristics. The scheduler achieves both pipeline and data parallelism. Its output is the core assignment, schedule and pipeline stage numbers for each task.

A. Hardware Assumptions

We model the hardware as a number of heterogeneous cores where heterogeneity manifests itself in differing error rates, clock frequencies and busy/idle powers. The error rate is the average number of errors that occur in a unit of time. This is assumed to be an intrinsic property of the core and fixed for the duration of an application. Due to time dependent factors, such as aging, error rates may need to be recharacterized and the tasks will need to be rescheduled for the new settings. Example settings for three cores are given in Table I. The data that relate power to error rates are taken from [10] and are based on errors introduced due to voltage over-scaling. Note that these are gate level errors and may be masked at the software level [13]. Clock frequency is used as a proxy for core performance. For power dissipation, there are two components, busy and idle. On any given cycle, if a core is performing an operation, it is "charged for" busy power, else it consumes idle power. For inter-core communication, we assume a general model that transfers one byte of data in a known time and with a known amount of energy between any two cores.

B. Application Assumptions

We focus on streaming applications which can be represented as a collection of `tasks` that are indivisible units of computation. These tasks and their data dependencies form a dependency graph with tasks as vertices and edges representing the dependencies. These applications execute a stream of data through the tasks in the order specified by the dependency graph. A `job` is a set of tasks. In addition to the deadline and reliability constraints for the entire application, additional deadline/reliability constraints may be specified for a `job`.

TABLE II
ABBREVIATIONS OF VARIABLES, CONSTANTS, SETS

Term	Description	Type
C	Cores	Set
J	Jobs	Set
T_j	Tasks belonging to job j	Set
F_t	Follower Tasks of task t in the dependency graph	Set
LT_j	Last Tasks of job j (Leaves of dep. graph)	Set
LK	Lock variables	Set
CS_l	Set of Tasks that acquire lock l to execute	Set
PB_i	Busy Power of core i	Const
PI_i	Idle Power of core i	Const
CP_i	Clock Period of core i	Const
CP_{MAX}	Maximum of Clock Periods	Const
FIT_i	Error Rate of core i	Const
TL_j	Error Tolerance of job j	Const
L_t	Length of task t in cycles	Const
S_{MAX}	$\leq$ Sum of task lengths	Const
EC	Communication Energy for one unit of communication	Const
TC	Communication Delay for one unit of communication	Const
$W_{t_1t_2}$	Communication weight of edge $t_1 \rightarrow t_2$	Const
P_{MAX}	Predetermined upper bound on pipelining	Const
D_j	Deadline for job j	Const
τ	Schedule Period	Var $Real$
σ_t	Start of Schedule for task t in cycles	Var $Int_{\geq 0}$
p_t	Data id of task t	Var $Int_{\geq 0}$
a_{ti}	Core Assignment of task t to core i	Var $Binary$
$s_{t_1t_2}$	Communication occurs from task t_1 to t_2	Var $Binary$
pv_k	k is larger than all stream ids ($k \leq P_{MAX}$)	Var $Binary$
$e_{t_1t_2}$	Task t_1 runs after task t_2	Var $Binary$

As shown in [15], the relation of faults in the hardware (eg. a bit flip in the floating point unit) to the change of the output of the application (eg. a pixel that changes color) strongly depends on the particular application. We observed this issue with our applications as well; thus we chose to keep this connection external to the framework and define the reliability constraint in terms of number of hardware level faults occurring in a `job`.

In Figure 1, an example dependency graph is shown for the `filterbank` StreamIt benchmark. This figure summarizes the input of the scheduler. Note that computationally inexpensive but memory intensive tasks are manually selected to have "0" error tolerance for their strong effect on the output whereas error tolerances for others are varied as explained in Section III.

The scheduler also needs the task runtimes. This is provided in terms of clock cycles since the system is heterogeneous, i.e., the frequencies of the different cores may be different. This is then multiplied by the clock period for the core that the task runs on to get the absolute time. Each unit of data must satisfy the data-dependencies in the dependency graph. Given the

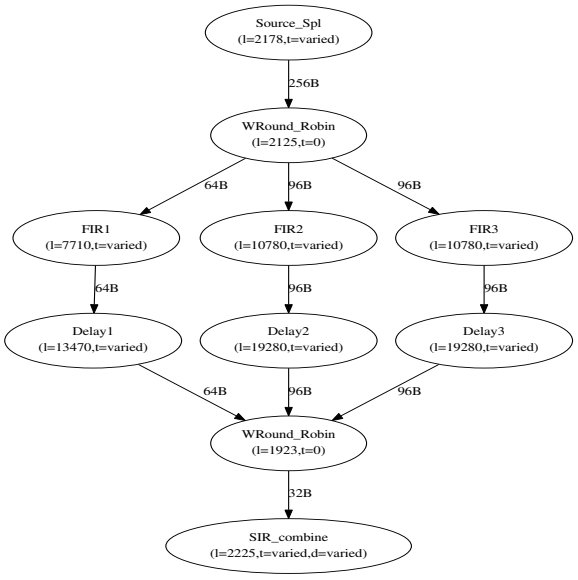

Fig. 1. Example input graph for benchmark filterbank (Task lengths (l), error tolerance (t), deadline (d) are shown in parentheses and communication between tasks is marked on edges)

streaming nature of the application, however, multiple units of data may be processed at the same time through pipelining.

Each edge in the dependency graph has an associated edge weight, which represents the communicated data bytes if the tasks at the head and tail of the edge are scheduled on different cores. To calculate the communication overhead, this weight is scaled with the per-unit core-to-core energy and performance communication overheads.

C. Objective Function and Constraints

The MILP formulation needs to handle traditional multiprocessor deadline scheduling issues as well as the energy and reliability aspects in the current context. Table II provides a summary of the constants, variables and sets that are used in the formulation provided in Figure 2.

The **Total Energy** consumed by the application has three parts; busy energy, idle energy and communication energy given by the summation terms of Equation 1. The first part, contribution from busy power, favors low-power cores whereas the second part, contribution from idle power, tries to maximize utilization. The final part, communication overhead, minimizes the inter-core communication. The scheduler uses these energies to find the minimum energy schedule for one iteration of the application.

The **Error Tolerance** of an application is defined for each job. This constraint (first part of Equation 2) specifies the upper limit on the total number of errors that can occur during execution of the tasks of this job. The scheduler might want to pick lower-power cores, but may be limited on that since these cores have higher error rates. We also need to ensure that every task can be assigned to only one core and this is achieved by the second part of the equation.

Scheduling for **Communication** is formulated using MILP with a detailed communication model in [3]. Here, we use a simple model: Equation 3 forces switching variables to be 1 when the tasks at the head and tail of an edge switch cores and 0 when they do not. We use these variables in formulating energy and time overhead of communication in Equations 1 and 6.

We model **Pipelining** by assigning a data ID to each task that indicates which unit of streaming data (or iteration) the task is working on. This is a relative number, i.e., if t_1 has data ID k and t_2 has data ID l, t_2 is working on data that is $l - k$ units newer than t_1. Therefore, if they have the same data ID's then they are working on the same data (the dependency posed by the dependency graph should be preserved), otherwise they are working on different data. Using this method, we can represent infinitely many iterations as one schedule that will repeat itself periodically. Equation 4 ensures that a task cannot have a data ID larger than the task it follows. This means that a following task cannot work on a newer data set than its predecessor. To implement the deadline constraint in Equation 11, we use a binary vector, pv_k that represents a value greater than or equal to the highest data ID. (e.g. $highest\_data\_id = 3$ is represented by $pv_3 = 1, pv_{k\neq 3} = 0$). To define the binary vector, we use an upper bound on data ID's, P_{MAX} (length of the longest path in the dependency graph). Constraints in Equation 5 ensure that only one of these binary variables is 1 and that the value that the vector represents (an integer between 0 and P_{MAX}) is larger than or equal to all data ID's. Note that the value that the vector represents would be equal to the largest data ID for an optimal schedule.

Equation 6 assures the **Sequencing** constraint, i.e. that the resulting schedule satisfies the graph dependency ordering. This builds on the sequencing constraints in previous MILP or ILP schedulers [3, 4], by adding additional terms to support pipelining which the previous work has not considered. The schedule variable is in clock cycles; conversion to time is done using the constants CP_i. This constraint is written for all edges of the dependency graph, and all core assignments to the head and tail of these edges. There are, however, two cases where this constraint does not apply. First, when pipelining takes place, the dependency edge is no longer valid (this is how the scheduler extracts pipeline parallelism). Second, every task is assigned to only one core, so the other assignments should not take effect. Therefore, we introduce the constant S_{MAX} which is a value larger than sum of all task lengths. The term before this constant is negative, thus trivially satisfying the constraint when either of these cases happen and 0 otherwise. Similarly, the communication overhead is included when $s_{t_1t_2}$ is 1 (edge is across cores), and these two cases do not happen.

Tasks may use **locks/critical sections** and these cannot overlap even in the absence of any data dependency between them. This problem for unit-length tasks has been investigated in [2]. We model the general case as follows: Tasks that share a lock l belong to the set CS_l. For each pair in a set, the binary variable $m_{t_1t_2}$, is 1 exactly when t_1 is scheduled before t_2. Equation 7 prevents these tasks from running at the same time and Equation 8 ensures that symmetric variables are consistent.

Mutual Exclusion ensures that no task overlaps with another task on the same core. This constraint is found in any scheduler with an ILP formulation (e.g., [4, 17]). The binary variable $e_{t_1t_2}$ is 1 exactly when t_1 is scheduled before t_2 on the same core as shown in Equation 9. Equation 10 ensures symmetric consistency.

The **Deadline and Schedule period** are two different time intervals, which are related to latency and throughput. The schedule period is the period of every repetition of the schedule. The latency of a unit of data is the time interval that starts with the scheduling interval of the first task and ends at the end of the last task. The deadline should be larger than the latency; Equation 11 enforces this. To calculate the latency, we need the product $highest\_data\_id \times schedule\_period$. (Note how this deadline constraint puts a limit on pipelining.) As both sides

$$TotalEnergy: \sum_{j\in J,t\in T_j,i\in C} L_t.a_{ti}.PB_i.CP_i + \sum_{i\in C}(\tau - \sum_{j\in J,t\in T_j,i'\in C} L_t.a_{ti'}.CP_{i'}).PI_i + \sum_{j\in J,t_1\in T_j,t_2\in F_{t_1}} s_{t_1t_2}.EC.W_{t_1t_2} \quad (1)$$

$$ErrorTolerance: \forall j\in J: \sum_{t\in T_j,i\in C} FIT_i.L_t.a_{ti}.CP_i \le TL_j\ , \qquad \forall j\in J, t\in T_j: \sum_{i\in C} a_{ti} = 1 \quad (2)$$

$$Communication: \forall j\in J, t_1\in T_j, t_2\in F_{t_1}, i\in C: \ \ s_{t_1t_2} \le 2 - a_{t_1i} - a_{t_2i}, \quad s_{t_1t_2} \ge a_{t_1i} - a_{t_2i} \quad (3)$$

$$Pipelining: \forall j\in J, t_1\in T_j, t_2\in F_{t_1}: p_{t_2} \le p_{t_1} \quad (4)$$

$$\sum_{k\in 0..P_{MAX}} pv_k = 1\ , \qquad \forall j\in J, t\in T_j: \sum_{k\in 0..P_{MAX}} k.pv_k \ge p_t \quad (5)$$

$$Sequencing: \forall j\in J, t_1\in T_j, t_2\in F_{t_2}, x\in C, y\in C: \sigma_{t_2}.CP_y \ge (\sigma_{t_1} + L_{t_1}).CP_x + (p_{t_2} - p_{t_1} + a_{t_1x} - 1 + a_{t_2y} - 1).S_{MAX}.CP_x + TC.W_{t_1t_2}.(s_{t_1t_2} + p_{t_2} - p_{t_1} + a_{t_1x} - 1 + a_{t_2y} - 1) \quad (6)$$

$$Locks: \forall l\in LK, t_1\in CS_l, t_2\in CS_l, x\in C, y\in C, t_1\ne t_2: \ \ \sigma_{t_1}.CP_x \ge (\sigma_{t_2} + L_{t_2}).CP_y + (a_{t_1x} - 1 + a_{t_2y} - 1 + m_{t_1t_2} - 1).S_{MAX}.CP_y \quad (7)$$

$$\forall l\in LK, t_1\in CS_l, t_2\in CS_l, t_1\ne t_2: m_{t_1t_2} + m_{t_2t_1} = 1 \quad (8)$$

$$MutualExclusion: \forall j_1\in J, j_2\in J, t_1\in T_{j_1}, t_2\in T_{j_2}, i\in C, t_1\ne t_2: \sigma_{t_1} \ge \sigma_{t_2} + L_{t_2} + (e_{t_1t_2} - 1 + a_{t_1i} - 1 + a_{t_2i} - 1).S_{MAX} \quad (9)$$

$$\forall j_1\in J, j_2\in J, t_1\in T_{j_1}, t_2\in T_{j_2}, i\in t_1\ne t_2: e_{t_1t_2} + e_{t_2t_1} = 1 \quad (10)$$

$$Deadline: \forall j\in J, t\in LT_j, i\in C, k\in 0..P_{MAX}: D_j \ge (\sigma_t + L_t).CP_i + k.\tau + (a_{ti} - 1 + pv_k - 1).(k+1).S_{MAX}.CP_{MAX} \quad (11)$$

$$SchedulingPeriod: \forall j\in J, t\in T_j, i\in C: \tau \ge (\sigma_t + L_t + (a_{ti} - 1).S_{MAX}).CP_i \quad (12)$$

Fig. 2. Objective function and constraints for the MILP formulation

of this product are variable values (the left depends on pipeline depth and right depends on the schedule), we write this equation for all possible values of data ID's and rule out the ones that are not actually the highest. Here, we again make use of S_{MAX} to automatically satisfy the equation when k is not the highest data ID. Although we do not actually use the highest data ID, we use a value that is higher than all data ID's (see Equation 5). When deadline is a limiting factor, this number is minimized to be equal to the highest data ID. Even if deadline is not a limiting factor, we use a value larger than the highest, so that the schedule is still feasible. We use Equation 12 to ensure that τ represents the scheduling period. Similarly, τ is larger than all the end times of the tasks. Since τ is in the objective function, the scheduler minimizes it and it becomes the schedule period. Although this formulation does not include it, an upper limit on τ can be added as a constraint to achieve a minimum throughput.

III. Experimental Results

A. Methodology

The scheduler was experimentally validated on six streaming benchmarks from the StreamIt suite [16] using different configurations of an eight-core CMP: `audio-beamformer`, `fft`, `filterbank`, `fmradio`, `matrixmult`, and `mp3decoder`. These benchmarks have different distributions of task lengths as well as parallelism in the dependency graphs. The ILP formulation was done in AMPL and solved using IBM's CPLEX solver running on a Quad-Core Intel Xeon processor with 16GB memory and Red Hat Enterprise Linux 5.4. The run time for each schedule was limited to 10 minutes. 68% of the schedules gave an exact solution within this time. For instances that do not complete in this time, the average optimality gap was found to be 6.4% where optimality gap is output by the CPLEX solver and denotes the difference between the proven lower bound (value corresponding to the solution of the linear programming relaxation of the remaining search space) and the best integer solution found in the given time [8].

Possible characteristics of the heterogeneous 8-core CMP target are taken from Table I, and different configurations are achieved as combinations of these settings. For example, 2-4-2 indicates that there are two perfect, four moderate-error and two high-error cores in the CMP. In addition to these configurations, we experimented with different idle power values (60%, 30%, and 0%) on configuration 2-4-2. These parameters are chosen to span a wide range of different hardware configurations and the experiments show how the relation between hardware parameters and application characteristics determine the resulting energy. The benchmarks were compiled for our target hardware, Intel i7 2.66GHz, using the StreamIt compiler [16]. The compiler produces the dependency graphs and multi-threaded C++ code. By instrumenting the output code with the RDTSC timer instruction [9], which reads a high-resolution timer in Intel CPUs, we profiled the length of each task through 100k iterations. As the energy and delay values for communication between the cores were not available, we assumed an ideal communication medium with coefficients EC and TC set to zero.

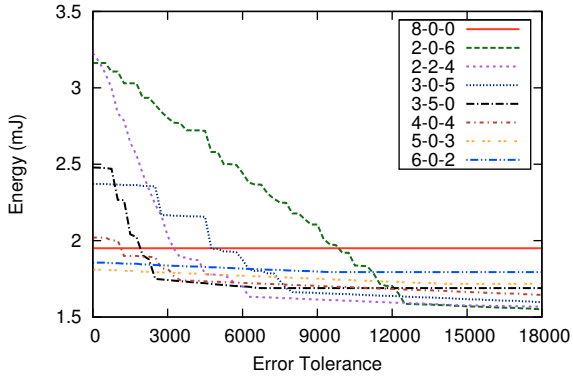

Fig. 3. Energy vs Reliability for `filterbank` benchmark running on different hardware configurations with idle power set to 60% of busy power. Each configuration is represented with a count triple of "perfect - moderate error - high error" cores

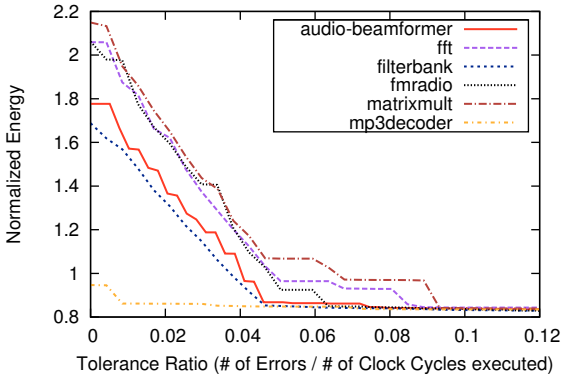

Fig. 5. Energy vs Reliability graph for all benchmarks. 2-4-2 configuration with idle power at 60% of busy power. (Energy consumption of each curve is normalized to the energy consumption running on 8-0-0.)

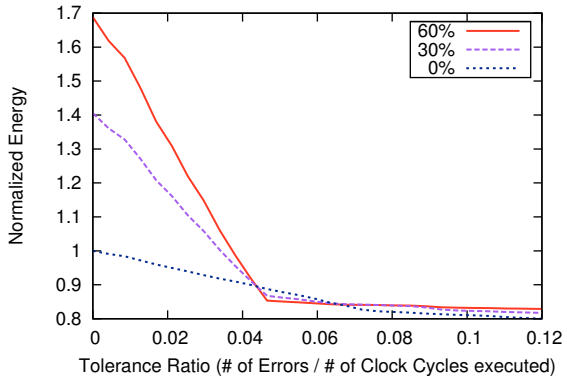

Fig. 4. Energy vs Reliability for `filterbank` benchmark running on configuration 2-4-2 for different idle power/busy power ratios. (Energy consumption of each curve is normalized to the case when running on 8-0-0.)

B. Different Configurations for filterbank

Figure 3 shows the scheduler results for the benchmark `filterbank` scheduled on a range of CMP configurations (omitting 2-4-2, which is inspected in detail in later sections, to avoid redundancy) with idle power pessimistically set to 60% of busy power. The y-axis is the energy that is consumed in a schedule period (energy for one data set). The x-axis is the error tolerance of the application that is varied from 0 to 25k errors allowed over the roughly 90K schedule period.

The 8-0-0 configuration represents perfect hardware with all 0-error cores and serves as the baseline configuration. The results focus on two main aspects, core utilization and core power. The scheduler may choose to achieve maximum parallelism to increase core utilization and reduce Idle energy consumption. On the other hand, the scheduler may prefer less-reliable lower-power cores to decrease the Busy energy consumption. The choice between these two depends first on the error tolerance of the application. In addition, the ratio of Idle and Busy Power plays a role. When Idle Power is high, utilization has a higher weight and vice versa.

Consider the configuration 5-0-3; this curve starts at a lower value than the baseline at 0-tolerance. The reason for that is that 5 cores are enough to achieve sufficient parallelism, and increasing the number of perfect cores (high power) increases power more than it increases performance, thus increasing the total energy consumption. Higher error tolerance makes the configuration 5-0-3 even more advantageous with energy gains diminishing at around error tolerance = 18000.

Configuration 2-0-6 starts from a higher energy value, crosses the baseline, and goes to a lower value. At 0-tolerance, the application runtime increases substantially and 6 idle cores cause the total energy to be higher than the baseline. However, with increasing error tolerance, the energy gains of this configuration can be as high as 34% making it the most advantageous configuration.

Finally, Figure 4 shows the effect of idle power on the outcome. As idle power decreases, the scheduler's optimization leverage gets larger. Further, the energy dissipated at 0-tolerance decreases and equals that of the 8-0-0 configuration when idle power is 0%. Since utilization no longer affects the outcome, the scheduling result is always better than the baseline.

C. All Benchmarks on Configuration 2-4-2

Figure 5 shows results for all six benchmarks scheduled on the 2-4-2 configuration with idle power at 60% of busy power. The energy is normalized to the 8-0-0 baseline. Most of the benchmarks start from a value that is higher than the baseline, reach 1 at some point and then flatten out at a low value. The value at the starting point depends on the utilization when the benchmark is scheduled on 8 perfect cores. This depends strongly on the distribution of lengths of tasks. For example, mp3decoder starts from a lower value than the baseline. The reason for that is, mp3decoder has two equal but very large tasks, one medium task and several very small tasks. Therefore, the reference configuration has 8 cores but only two are well-utilized. When we turn this into a configuration that has 2 perfect cores with other cores having lower power, it actually starts from a lower value than the baseline configuration even though it can only utilize 2 cores. With increasing error tolerance, the energy drops as tasks can start using low-power cores.

IV. Related Work

Energy-Resiliency Tradeoffs: Recent work has explored reducing power consumption by sacrificing resiliency in several ways (e.g., [1, 5, 7, 10, 11]). In RAZOR [5], the voltage

is aggressively over-scaled resulting in incorrect values being latched. However, these errors are subsequently fixed using circuit and system techniques which incurs additional cost. In contrast, by judiciously exploiting application error tolerance, our work does not incur error-correcting overheads. The work on Error Resilient Systems Architectures (ERSA) [11] has similar goals in exploiting perfect and imperfect cores for implementing error resilient applications. However, it does not consider a systematic method to match the error-tolerance, performance targets and energy consumption of parts of the application with heterogeneous core characteristics. In EnerJ [14], authors provide a type-checking system to safely execute applications on a faulty system and show energy gains between 10% and 50%. In their work, the processor provides little or no effect on the overall energy gain while our work focuses on the energy consumption of the processors. Furthermore, the authors have only focused on the correctness instead of quality assurance, which is one of the main constraints in our formulation. Finally, dynamic voltage scaling [12] provides for energy-performance tradeoffs, but does not consider reliability, which is the added focus of our work.

Scheduling using MILP: Mathematical programming formulations of various scheduling problems have been very well studied, and several of these are relevant to our work. In [4], the multiprocessor scheduling problem is formulated as an MILP problem for minimizing the deadline using as few variables as possible and for different classes of dependency graphs. In contrast, our context and resulting formulation adds and integrates several non-trivial extensions to the general scheduling problem. Specifically we add (i) a reliability constraint, (ii) communication overhead between tasks scheduled on different cores, (iii) pipelining, and (iv) mutual exclusion due to locks/critical sections. Our communication part differs from the previous communication model that uses bus-based resource limited communication [3] by only using data transfer time and energy. Our mutual exclusion approach differs from prior work in scheduling with locks [2] as that is not based on ILP. The scheduler in [17] also refers to energy-performance-resiliency tradeoffs, however in a different context. They use duplication to increase the reliability of an application and define a task to be reliable when it is duplicated and non-reliable when it is not. The scheduler in [18] considers pipelining but cannot be applied to infinitely many iterations. Overall, our framework provides a comprehensive modeling of constraints for the context considered in this paper.

V. Conclusions and Future Work

Emerging CMPs will likely have heterogeneous per-core power, performance and reliability characteristics, and many applications can exploit this by trading off limited computation errors for lower energy. Our research provides an optimization framework for this by developing an exact MILP formulation of the static scheduling problem for such applications on these processors. We demonstrate the practical applicability of this work for streaming media applications.

Experimental results show how the benefit varies with different benchmarks and hardware configurations with up to 34% energy gains possible for the StreamIt benchmarks. The variation across benchmarks and configurations demonstrates the value of automated search space navigation using our MILP formulation. Going forward, this framework can be further refined to consider different application-level reliability metrics (e.g. signal-to-noise ratio) and detailed power and communication models. Furthermore, scheduling heuristics can be developed using our formulation for scalability and efficiency.

References

[1] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge. Opportunities and challenges for better than worst-case design. In *Asia and South Pac. Design Automation Conf.*, 2005.

[2] B. S. Baker and J. Edward G. Coffman. Mutual exclusion scheduling. *Theoretical Computer Science*, 162, 1996.

[3] A. Bender. MILP based task mapping for heterogeneous multiprocessor systems. In *European Design Automation Conf.* IEEE, 1996.

[4] P. E. Coll, C. C. Ribeiro, and C. C. de Souza. Multiprocessor scheduling under precedence constraints: polyhedral results. *Discrete Appl. Math.*, 154(5), 2006.

[5] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *IEEE Micro Conf.*, 2003.

[6] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Design Automation Conf.*, 2008.

[7] R. Hegde and N. R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Intl. Symp. on Low Power Electronics and Design*, 1999.

[8] ILOG. ILOG CPLEX 11.0 User's manual. 2008.

[9] Intel. Using the RDTSC instruction for performance monitoring. Technical report, Intel Corporation, 1997.

[10] A. Kahng, S. Kang, R. Kumar, and J. Sartori. Designing processors from the ground up to allow voltage/reliability tradeoffs. In *IEEE Intl. Symp. on High-Performance Computer Architecture*, Jan 2010.

[11] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. Error resilient system architecture (ERSA) for probabilistic applications. In *Design, Automation and Test in Europe*, 2010.

[12] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron. Procrastinating voltage scheduling with discrete frequency sets. In *Design, Automation and Test in Europe*, 2006.

[13] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. Measuring architectural vulnerability factors. *IEEE Micro Conf.*, 23, 2003.

[14] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. 2011.

[15] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Characterizing the impact of soft errors on iterative methods in scientific computing. In *Int. Conf. on Supercomputing*, 2011.

[16] W. Thies, M. Karczmarek, and S. P. Amarasinghe. Streamit: A language for streaming applications. In *Intl. Conf. on Compiler Construction*. Springer-Verlag, 2002.

[17] S. Tosun, N. Mansouri, M. Kandemir, and O. Ozturk. An ILP formulation for task scheduling on heterogeneous chip multiprocessors. In *Intl. Symp. on Computer and Information Sciences*, 2006.

[18] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan. An ILP formulation for task mapping and scheduling on multi-core architectures. In *Design Automation and Test in Europe*, 2009.