Energy Characterization of the Stargate Sensor Network Gateway

Selim Gurun Chandra Krintz
Computer Science Department
University of California, Santa Barbara

Abstract

We present a new energy estimation model for sensor network intermediate nodes (i.e. the Crossbow XScale Stargate). Such devices are battery powered and resource constrained and commonly employed as communication, processing, and gateway elements within sensor networks. Understanding and accurately characterizing the energy behavior of such devices is key to conserving the battery life of sensor systems.

In this paper, we present a macro-model for estimating the energy consumption of the device as a whole that couples estimation techniques for computation and communication. We construct our model using empirical data that we collect via hardware performance monitors. Our contributions are two-fold. 1) We demonstrate that hardware performance monitors are effective in modeling the computational energy consumption of sensor network gateways, 2) we present an analysis of linear dependencies between various hardware performance events, discuss their effects on model stability, and show how we can use principal component analysis to remove the undesirable side effects of such dependencies.

1 Introduction

Wireless sensor networks have become increasingly popular as a result of their low cost, small size, and their potential for enabling transparent interconnection between the physical world and powerful information systems. Typical installations of these systems consist of a hierarchy of heterogeneous devices that range in capability but are both resource constrained and battery powered. Computationally simple, low-power, sensor elements make physical measurements, perform minor processing, and relay the collected data to more powerful devices. These more powerful devices, i.e., gateways or intermediate nodes, implement significantly more functionality, computational power, and battery capacity than the simpler sensor elements. To program and to facilitate efficient use of sensor systems, we must be able to characterize accurately the power and energy consumption of tasks that execute using these devices.

A significant body of research exists that models and estimates power and energy consumption for a range of microprocessors [3, 12, 26, 9, 4, 13], sensor network devices [15, 21, 22] and battery technologies [2, 19, 27, 14, 24]. However, a key missing piece in this prior work is effective and accurate estimation of *full-system energy consumption* for sensor gateways. In this paper, we take the first step toward providing a method that can do so for a typical sensor network gateway node, the Crossbow Stargate.

To enable accurate power modeling for complex Stargate tasks, we couple the use of hardware performance monitors (i.e. HPM) with operating system (OS) software performance monitors. We consider two primary Stargate activities: computation and wireless communication. In the integrated model, we consider the energy consumption of the device as a whole including CPU, memory, bus and other peripherals. Our approach is empirical in that we use program profile information from a range of programs to develop our model.

We study applicability of hardware performance monitors to power modeling in general. There is no standard implementation of hardware performance monitors across architectures, however, in all architectures with which we are familiar, hardware performance monitors are specifically designed for monitoring program performance behavior and not for estimating power consumption. Thus, important power consumption events are typically missing. For instance, in our target architecture, there are no events to monitor directly external memory accesses. Furthermore, there are dependencies between many hardware events. Such dependencies increase model variance and reduce the accuracy and usefulness of model parameter estimations. To mitigate this problem, we apply statistical techniques that reduce model variation and improve model quality.

We evaluate our model using a wide range of programs that perform a number of computational and communication activities typical of sensor network tasks for intermediate nodes. Our model produces an average error of 7% for computation bound tasks and of 13% for tasks that perform both computation and communication – significantly outperforming an extant but similar approach for estimation of power consumption for the CPU and memory in isolation.

We investigate techniques that enable these error rates while reducing the number of model parameters using the statistical correlation between them. We also investigate the use of principle component analysis to remove the multicollinearity in the model input data – an effect that is due to the dependence between HPM metrics. Removing multicollinearity from our model is important if we are to extract the relative impact on energy for each of the individual components in our model. Such information is vital for online energy estimation. We quantify the cost of obtaining such benefits in terms of model accuracy.

In summary, we contribute with this paper:

- A simple, full-system energy estimation model for the Stargate sensor network gateway device that considers both computation and communication;
- A set of techniques that reduce the number of model parameters and that enable extraction of useful information from the coefficients in our model;
- An empirical evaluation of our models using a wide range of real programs and a comparison with the best-performing extant model for estimating the energy consumption of the CPU and memory system in isolation.

In the next section, we present our approach to energy estimation in sensor network gateways. We then present our empirical evaluation (Section 3), related work (Section 4), and conclusions (Section 5).

2 A Macro-Model for Sensor Network Energy Consumption

The goal of our work is to accurately characterize the energy behavior of the Stargate sensor network intermediate node – at the system level. The Stargate implements a 400MHz Intel XScale CPU, short range 802.11 and long range WAN radio interfaces, and flash memory, among other devices. The Stargate resources are managed by the Linux operating system. Characterizing energy

Training Benchmark Set		Reference Benchmark Set	
Application	Description	Application	Description
basicmath	Math Test	gsmdecode	GSM decoder
dijkstra	Dijkstra shortest path	gsmencode	GSM encoder
matmult	Matrix multiplication	jpegdecode	JPEG decoder
stringsearch	String search	jpegencode	JPEG encoder
memri*	Memory read in-cache	mpegdecode	MPEG decoder
memro*	Memory read out-of-cache	mpegencode	MPEG encoder
memwi*	Memory write in-cache	em3d (Java)	Graph processing
memwo*	Memory write out-of-cache	bisort (Java)	Sorting
reg*	Register operations	treeadd (Java)	Recursive depth-first traversal
scps	secure file send	scps	secure file send
scpr	secure file receive	scpr	secure file receive
netpipe	network analyzer	game of life	MPI life game
		pvnx	MPI non-linear solver (small)
		pvkx	MPI non-linear solver (medium)
		pvkxb	MPI non-linear solver (large)

Table 1: Benchmarks. We use two benchmark sets: Training (left) to parameterize our model and Reference (right) to evaluate the accuracy of our model. We use the benchmarks above the line to model/evaluate computation; those below for communication. We have developed the applications with asterisks ourselves.

consumption of Stargate is critical for us to better understand its energy behavior and develop power aware software and optimizations. In the following subsections, we first describe our energy measurement methodology and our benchmarks. Next, we describe our model, the challenges, and our solutions.

2.1 Benchmarking Methodology

We employ two benchmark suites. The first suite, to which we refer to as the training set, we use to define our model. The second suite, to which we refer to as the reference set, we use for the empirical evaluation of the accuracy of our model. We present the suites and their brief description in Table 1. The left half of the table is the training set and the right is the reference set. We use the benchmarks above the line to model/evaluate computation and those below for communication. We execute all programs from RAM drive to minimize the effect of flash read/write latency. The wireless network card is on for all of our experiments regardless of whether we use it or not.

Our applications come from popular benchmark suites (e.g. MediaBench [16], Mibench [7], and Java-Olden [1]). Our communication benchmarks include the secure copy protocol (scp) receive and transmit and netpipe [18]. For scp, we transfer a 1.7 MB file. Netpipe is a network analyzer. We also include distributed (message passing interface (MPI)) applications: Game of Life [6], Pvnx, Pvkx and Pvkxb [23]. MPI is typically employed for distributed computing applications in larger systems. These MPI applications have moderate computation requirements that are within the limits of the Stargate.

The characteristics of the MPI applications are analogous to the requirements of high-performance sensor network applications. For example in Life, the first processor divides the problem space into

Event	Description
0x0	Instruction cache miss requires fetch from external memory.
0x1*	Instruction cache cannot deliver an instruction.
0x2*	Stall due to a data dependency.
0x3	Instruction TLB miss.
0x4	Data TLB Miss
0x5	Branch instruction executed, branch may or may not have changed program flow.
0x6	Branch mispredicted
0x7	Instructions executed
0x8*	Stall because the data cache buffers are full.
0x9	Stall because the data cache buffers are full.
0xa	Data cache access, not including Cache Operations.
0xb	Data cache miss, not including Cache Operations.
0xc	Data cache write-back. This event occurs once for each 1/2 line (four words) that are written back from the cache.
0xd	PC Modified

Table 2: HPM events in Intel PXA-255. The flagged events count the number of cycles the event condition persists.

subspaces and distributes them to the other processors. Once the other processors complete the execution, they return the results back to the first processor. Then the first processor combines the results, and reiterates the process if necessary. This mechanism is very similar to recent query processing and vehicle tracking architectures for sensor networks. For instance in [20], the nodes are organized in a tree structure. The root node distributes a query to the network. Each node partially processes the query and returns the results to the parent node. It is the parent node which combines the results. In [25], the remote sensor nodes collaborate with a central sensor node for airport security and tracking moving objects. The remote nodes do partial stream processing and filtering using computationally expensive algorithms, while they continuously exchange updates with a central node. The central node produces the results.

We collect performance data from these benchmarks in the form of CPU hardware performance monitors (HPMs). We collect data for each hardware event for five repetitions of the same program. HPMs provide efficient hardware support for profiling CPU-based activities. To monitor I/O activity, we use operating system counters. The HPMs that the Stargate supports are shown in Table 2.

We collect the power consumption data using a 2-channel Agilent 54621A oscilloscope and a National Instruments data acquisition board (DAB). We connect these measurement devices to the exposed terminals of the Stargate. We collect HPM data using a very light-weight device driver that we have developed. Our monitoring overhead is less than 2%. The HPM driver collects HPM counters every 10 million instructions and uses an external output pin to control the oscilloscope and the data acquisition board. Throughout this paper, we use the term interval to refer to a period of 10 million instructions.

2.2 Computation Model

Our energy estimation model consists of a computation and a communication component. We describe the former in this section and the latter in Section 2.4. We employ precise power measurements of the *entire device*, i.e., the full system, to develop both components. We collect these measurements using an Agilent 54621A oscilloscope and a light weight device driver that we have developed. The device driver collects HPM counters every 10 million instructions, which we refer as an interval throughout this paper. Our monitoring overhead is less than 2%. Prior techniques that employ a similar methodology to ours focus on the power consumption of the CPU alone [3] or the CPU and memory subsystem [4, 13], in isolation, whereas our focus is on (and thus, our energy measurements are for) the entire Stargate device.

The computation model estimates the energy consumption on average per instruction for tasks that do not have any significant persistent storage access or communication behavior. In other words, our computation model models the energy consumption of three most important unit; CPU, memory, and the memory bus. Our model employs 6 parameters: cycles per instruction (CPI) (x_1), instruction cache misses (x_2), instructions not delivered (x_3), data stalls (x_4), instruction TLB misses (x_5) and data TLB misses (x_6). These and similar events has been shown to be effective for power estimation of the CPU and memory [4, 10, 8].

We use a linear parametric function to characterize energy consumption. Linear models have been successfully used in previous studies to model energy consumption of various components including CPU and memory. Our model is as follows:

$$E(nanojoules) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_6 x_6 \quad (1)$$

where x's are the model input and the α's are the weights determined by the model. The model inputs are expected number of events for an average instruction, for instance x_2 gives the expected number of cache misses during the execution of an instruction. We compute this value by dividing the number of event counts in an interval by the interval length (10Million). The model outputs the estimated energy consumption of an instruction on average, in nanojoules. We estimate the parameter weights using least squares linear regression (i.e. LSQ). LSQ models are simple and robust and they do not require a priori knowledge of the distribution associated with the observations [5].

Since the XScale processor is only able to monitor two events at once, we execute the same program many times to monitor the different events. The measurement data can differ across runs (of the same program/input) as a result of hardware state or operating system events. To be able to better understand the extent of such perturbations, we monitor each event 5 times. However, *since averaging causes linear regression to look stronger than it really is* [5], we only use the 3^{rd} dataset for each event. We use the remaining 4 observations to evaluate the impact of such perturbations on model accuracy.

The benchmarks that we use are significantly different in their durations. Since we do not want any single benchmark to be represented more than its share in our model, we choose an equal number of observations from each benchmark. To extend the range of possible behaviors, we select the first, middle, and last 10 intervals from each profile.

We present the coefficients of our model and evaluate its fit on the training benchmark set in Table 3. The top portion of the table shows the coefficients for each of the HPMs. The bottom portion of the table shows the fit statistics. We evaluate the accuracy of our model for the reference set in Section 3.

Computation Energy Consumption Model			
	Description	Coefficient	t-statistic
α_0	Constant	-0.19	-0.73
α_1	CPI	7.06	38.78
α_2	Inst. Miss	678.07	0.55
α_3	Inst. Not Dlvrd	-4.28	-1.61
α_4	Data Stalls	-1.07	-5.99
α_5	Inst. TLB Miss	686.06	-0.02
α_6	Data TLB Miss	-593.39	-8.33
R^2		0.99	
Average Error		3.80%	

Table 3: Coefficient and fit statistics for the computation model.

	CPI	IMISS	INDLVR	DSTALL	ITLBMISS	DTLBMISS
CPI	1.00					
IMISS	-0.04	1.00				
INDLVR	-0.14	0.89	1.00			
DSTALL	0.71	-0.05	-0.15	1.00		
ITLBMISS	-0.04	0.97	0.84	-0.04	1.00	
DTLBMISS	0.74	-0.03	-0.11	0.05	-0.03	1.00

Figure 1: Correlation among model parameters. The darker entries show events that have strong correlations.

The coefficient of determination, i.e., the R^2 fit statistic, indicates the amount of variation that the model explains. Under most circumstances, it is a reliable indicator of model goodness. The R^2 varies between 0 and 1, and larger values are better. The high R^2 value of our model is a positive indicator of its high quality.

The average error statistic shows the absolute model estimation error. We compute this value using $1/n \times \sum (|\text{measured} - \text{estimated}|/\text{measured}) \times 100$, where n is the number of measurements. The model fits very well to the data with an average error of 3.8%. The final column shows the t-statistics for the model coefficients. The t-statistic values that are larger than 2 indicate that the corresponding coefficient is statistically significant (i.e. has a high probability that it is non-zero). The t-statistic values indicate that only three of the coefficients in our model, CPI, DSTALL, and DTLBMISS are statistically significant.

2.3 Reducing the Number of Model Parameters and Multicollinearity

We investigate two ways to improve our model. First, we identify statistically valid ways to reduce the number of parameters in our model. Our goal is to develop a model that is accurate but that employs a small number of parameters. If we can reduce the number of parameters used by our model, we can reduce the complexity and overhead of the model. However, we must be careful to do so without reducing accuracy of the model significantly.

To eliminate model parameters, we empirically analyze the statistical correlations between the events in our model. We show this data in Figure 1. For each event pair, the closer the number is to 1.0, the higher the correlation. Our goal is to eliminate events that are highly correlated since only one event is needed to represent the others.

Coefficients of Principal Components					
1	2	3	4	5	6
0.17	0.68	0.00	0.10	0.02	-0.71
-0.57	0.18	-0.02	-0.25	-0.77	-0.02
-0.55	0.08	0.01	0.81	0.16	0.06
0.14	0.47	-0.72	0.02	0.00	0.48
-0.56	0.18	-0.02	-0.52	0.62	-0.02
0.12	0.50	0.69	-0.01	0.00	0.51
% of Variance Explained					
47.61	33.44	15.77	2.75	0.40	0.04

Figure 2: Coefficient of principal components and the amount of variance explained for our training model

The correlation matrix from this data shows that CPI, data stalls (DSTALL) and data tlb misses (DTLBMISS) have high correlation. Among those three, we retain the CPI, since it has the highest t-statistic, and discard the others. Furthermore, the instruction cache (IMISS, INDLVRD) and instruction TLB miss (ITLBMISS) events also have strong correlation. Using similar reasoning, we can use IMISS and discard ITLBMISS and INDLVRD events. We form a model from IMISS and CPI data that we refer to as *MMISS*. Furthermore, we also form a model, *MCPI*, which uses the CPI metric alone.

Our second method for improving our model focuses on enabling the extraction of meaningful information from the coefficients of the model. That is, we would like to be able to understand the relationship to overall power consumption that each component of the model contributes to as has been attempted in prior work [4]. Such characterization of model components is vital for enabling online energy estimations.

However, an interesting phenomenon in HPM data is the existence of *multicollinearity*. Multicollinearity indicates that some linear relationship exists between the model parameters. For example, data cache misses are related to data stalls. As the amount of linearity increases between metrics, the stability of the coefficient estimates decreases precluding us from extracting useful information from the coefficients in our model [5].

If the purpose of regression is only to estimate a variable y using a set of model inputs x_i, without assigning any particular meaning to the values of x_i, multicollinearity is not a significant problem and model predictions will still be accurate. However, if the goal is to understand how much each x_i effects y, then multicollinearity can lead to misleading results. Models that suffer from multicollinearity have much larger confidence intervals in parameter estimations, that is, the parameter estimations can shift substantially when there are small changes in input. Another side effect of such wide confidence intervals is that the t-statistic value of individual parameters cannot be used confidently to remove arbitrary parameters from the model.

Our approach for reducing multicollinearity is to apply principal component analysis [11] to transform the correlated variables into a smaller number of uncorrelated variables called as principal components. The first principal component captures as much of the variability in the data as possible, and the succeeding components capture the rest of the variability.

In the first step of PCA, we standardize the dataset, that is, we subtract sample mean from each observation, and then divide the result by standard deviation of each parameter.

Let the variables x_{ij} are model inputs,(i.e the explanatory variables), such that x_{ij} is equal to j^{th} observation of i_{th} variable. The variable $\bar{x_i}$ gives the observed mean of variable x_i that is

$\bar{x_i} = \sum_1^n x_{ij}/n$, and s_i is the sample standard deviation of x_i that is $s_i = \sqrt{\sum_1^n (x_{ij} - \bar{x_i})^2/(n-1)}$. In both formulas, n is the number of observations.

After standardization, our goal is to find the coefficients γ_i that best fit to the linear equation:

$$y_j = \gamma_0 + \gamma_1 x^s_{1j} + \gamma_2 x^s_{2j} + \gamma_3 x^s_{3j} + ... + \gamma_6 x^s_{6j} + \varepsilon_i \tag{2}$$

In the equation above, x^s_{ij} are the standardized observations of HPM counters. y_j is called the response variable or model output and here it gives the energy consumption per instruction. The ε is the error term (which we assume to be normally distributed). We can rewrite Equation 2 as

$$y = \gamma_0 + X^s\gamma + \varepsilon \tag{3}$$

where X^s is an $n \times 6$ matrix whose columns X^s_i are standardized HPM observations, $x_0, x_1, ..., x_i$ respectively. γ is a column vector of size 6 and its entries are coefficients $\gamma_0, \gamma_1, ..., \gamma_i$.

Our goal is to transform the model explanatory variables, X^s, into a new set of uncorrelated variables, which are the principal components of the correlation matrix $X^{s'}X^s$. We then remove the components that explain the least amount of variance. By doing so, we can decrease the effect of multicollinearity, and therefore narrow the confidence interval in our coeffient estimates.

As $X^{s\prime}X^s$ is a square matrix, we can decompose it into its eigenvectors V_i and eigenvalues λ_i such that $(X^{s\prime}X^s - \lambda_i I)V_i = 0$, where I is the identity matrix. The eigenvectors are orthonormal, that is $V_iV_j = 0$ for $i \neq j$. Let eigenvector matrix be $V = [V_1, V_2, ...V_6]$. Again, as eigenvectors are orthonormal, VV' gives the identity matrix I. Thus, we can rewrite Equation 3 as:

$$y = \gamma_0 + X^sVV'\gamma + \varepsilon$$

We can substitute X^sV with Z and $V'\gamma$ with θ:

$$y = \gamma_0 + Z\theta + \varepsilon \tag{4}$$

The normalized eigenvectors V_i of squared matrix $X^{s\prime}X^s$ give the coefficients of principal components. In other words, the principal components matrix Z is equal to the product of parameter observations X^s with eigenvector matrix V. Once we compute the principal components of the data, we can remove the multicollinearity by discarding the components that account only a fraction of the total variability. The penalty however is a decrease in model accuracy in exchange of an increase in confidence level of model parameter estimations. The eigenvalues show this variation. Let V_i is an eigenvector and λ_i is the corresponding eigenvector. The ratio $\lambda_i / \sum_1^6 \lambda_k$, gives the amount of variation that the corresponding principal component explains. Figure 2, shows the principal components of our data and the amount of variance that they explain (the largest first). As the figure suggests, the first three principal components account for more than 95% of model variability. Hence, we retain the first three components and discard the rest.

We next do our regression using the principal components Z and compute the coefficients of principal components, that is the column vector θ. The estimated regression coefficients are free from the correlation (and thus, are more stable). Using θ, we compute the coefficients of standardized HPM counters, γ:

$$\gamma = V\theta$$

However, the γ gives the coefficients of the standardized HPM counters. Finally, as described in [17], we transform the coefficients back to the original HPM variables. We do this using:

$$\alpha_i = \frac{\gamma_i}{s_i}, 1 \leq i \leq 6$$

Computation Energy Consumption Models			
Description	MPCA Coefficients	MMISS Coefficients	MCPI coefficients
Constant	6.6	1.90	1.90
CPI	2.8	5.74	5.74
Inst. Miss	2814.5	180.0	$--$
Inst. Not Dlvrd	-18.6	$--$	$--$
Data Stalls	3.05	$--$	$--$
Inst. TLB Miss	83039.4	$--$	$--$
Data TLB Miss	1055.8	$--$	$--$
R^2	0.99	0.99	0.99
Average Error	12.9%	5.81%	5.88%

Table 4: Coefficient and fit statistics for the computation model.

$$\alpha_0 = \gamma_0 - \sum_{i=1}^{6} \frac{\gamma_i \bar{x_i}}{s_i}$$

We call this model *MPCA*.

We develop MMISS and MCPI using the same method that we applied previously in Section 2.2. Table 4 summarizes our new models and their coefficients. As we expect, our removal of three principal components reduces the accuracy of *MPCA* model, and consequently its error rate increases to 12.9%. However, we note that this model enables us to extract information about the impact of each component in the model. The MPCA coefficients provide significant insight into memory and CPU energy consumption. We find that, (1) an instruction miss is approximately 1000 times more expensive than a clock cycle, (2) a data stall cycle (a clock cycle where pipeline stalls and waits for data) is slightly more expensive than an instruction execution cycle (probably due to memory access activity), and (3) an instruction TLB miss is 75 times more expensive than a data TLB miss. However, the most interesting coefficient is instructions not delivered, which has a negative coefficient. A negative coefficient here does not mean a flaw in our model, but it indicates that the CPU (and memory) energy consumption drops below average level, which is the sum of constant factor, CPU clock counter and other events that our model includes. This indicates that the CPU applies several techniques such as clock gating to reduce its energy consumption once pipeline is stalled.

The MCPI and MMISS model perform similarly and both are more accurate than MCPI. However, the t-statistic for the IMISS event is only 0.49 which indicates that this parameter is not significant. We therefore, remove MMISS from our model set.

Prior work [3, 4] on models for energy consumption of the CPU alone and CPU and memory subsystem in isolation also show that CPI is a significant metric in the estimation of energy consumption. Interestingly, we find that a CPI based one-input model is very effective and performs best for estimating full-system computational energy consumption. In Section 3, we will evaluate our results on a wide set of different applications from our reference set.

2.4 Communication Model

We next introduce our model for wireless communication. Our model is independent from our computation model to enable portability, i.e., we can swap the model for others for comparison or to improve accuracy. We combine the models via arithmetic addition of the two estimates. Modeling the network interface is more challenging than modeling the processing unit because the

Communication Energy Consumption Model			
Coef.	Description	Bytes+Packets	Bytes
TXB	TX bytes	2.40×10^{-6}	6.29×10^{-6}
RXB	RX bytes	-4.78×10^{-7}	-1.69×10^{-6}
TXP	TX packets	-2.90×10^{-3}	
RXP	RX packets	5.50×10^{-3}	
K	Constant	2.37×10^{-2}	2.00×10^{-1}
R^2		0.972	0.796
Average		26.9%	208%
95^{th} Percentile		59.6%	470%
Wireless Idle Power		0.78 ± 0.06 Watts	

Table 5: Communication Energy Model. We model the energy consumption of wireless card as a function of transferred bytes and packets.

network interface is significantly impacted by external effects such as RF interference, network congestion, asymmetric links due to badly calibrated hardware, etc. We have not tested our radio model with all of these conditions and we do not expect it to perform well in extreme conditions. Furthermore, we assume an 11Mb/s communication rate setting for the purpose of this paper. We plan to incorporate other supported rates into our model as part of future work.

As we did for the computation model, we employ a wide range of empirical observations from benchmarks to develop our communication model. Our wireless network includes a set of 6 hosts, including PDAs and laptop computers. The network load varies from idle to a few megabits/second and is susceptible to interference from two separate wireless networks. Furthermore, we assume an 11Mb/s communication rate setting for the purpose of this paper. We plan to incorporate other supported rates into our model as part of future work. Thus, we believe our model captures a wide variety of common situations.

Our model is a linear parametric function like the computation model, and has four parameters: transmit bytes (TXB), receive bytes (TXB), transmit packets (TXP), and receive packets (RXP). The model uses these parameters as follows:

$$E_n(Joules) = TXB\beta_1 + RXB\beta_2 + TXP\beta_3 + RXP\beta_4 + K$$

Our training benchmark suite for our communication model considers three different scenarios: (i) upload heavy communications (ii) download heavy communications and (iii) almost symmetrical, mesh type communications. For the first two scenarios, we use the scp benchmark. To collect behavior from the symmetric communications, we use the netpipe benchmark to generate network load. Typically, netpipe transfers are ping-pong like, it transfers one packet to a server and receives another packet before continuing. This forces the network to transmit every single packet, without opportunity to stream multiple small packets together. Netpipe also exposes idiosyncrasies that result from the internal hardware buffer, by re-evaluating each packet size using a constant perturbation factor. We consider for transfer size categories: (1) small: < 100 bytes; (2) medium: 100 to 1000 bytes; (3) large: 1000 to 4000 bytes; and (4) very large: 4000 bytes to 200KB. We repeat each transfer 100 times for the first tree categories and 10 times for the last category.

We consider two different models, one that considers both bytes and packets transfered and one that only considers bytes transferred. We refer to the former as (Bytes+Packets) and the latter as (Bytes). We present the LSQ coefficients for both models as well as the fit statistics for the training data set for the selected intervals in Table 5.

Both models exhibit much higher error rates than those from the computation model. The error is due to the difficulty of capturing external effects. However, these results are for energy consumption of the wireless card only. In our evaluation section, we consider benchmarks that perform both computation and communication. The error for the latter will impact overall estimation depending on the amount of communication performed by the application.

Interestingly, these results show that it is very important to consider both packet count and bytes transfered to produce an accurate model. By extending the byte model to include the packet counts, we improve the error rate of the model by almost an order of magnitude. The low accuracy of byte model reflects the non-linearity between transfer sizes and packet sizes. Small packets have a disproportionately large overhead due to protocol headers.

3 Evaluation

In this section, we empirically evaluate the efficacy of our techniques. In the subsections that follow, we compare our model to the one described in prior work [4]; we refer to this second model as MCPUMEM in our results.

MCPUMEM was developed for the same Intel XScale CPU and memory configuration as our own and uses similar HPM-based techniques. One primary difference between our model and MCPUMEM is that the latter combines HPM and *energy measurements of the CPU* to model and estimate *CPU* and *memory* power consumption. In our work, we are interested in modeling and estimating the energy consumption of the *full system*. Our model combines HPM and energy measurements of the entire device and employs PCA to ensure statistical soundness of our model. Moreover, as we evaluate in the next section, we combine this model with a communication model to estimate full system power consumption for applications that perform both computation and communication.

To implement a similar experimental methodology as the prior work, we (1) add the idle power consumption of wireless card to MCPUMEM (a constant factor), and (2) report MCPUMEM's mean error and its deviation. Deviation indicates how much the mean prediction error varies from one benchmark to another. As mean error can be affected by the idle energy consumption of hardware components, the deviation of the estimations better describes the quality of model. In the subsections that follow, we evaluate our computation and communication model.

3.1 Accuracy of the Computation Model

Figure 3 shows the error rate for the reference benchmark set for the models that we describe in previous section. These models include MLARGE – the original model without the removal of multicollinearity; MPCA – the original model that uses principle component analysis for multicollinearity removal; MCPI – the original model with HPM metrics CPI (our second approach to multicollinearity removal); MCPUMEM – the CPU-only HPM-based model from prior work [4]. MCPI and MLARGE perform similarly with an average error of 7%. The error is slightly lower for the MediaBench benchmarks than for the Java benchmarks, this is because the behavior of the Java benchmarks is much more variable due to benchmark activity, garbage collection, class loading, interpretation, and other factors.

MPCA produces an average error of 22% making it unsuitable for energy estimation of computation-bound tasks. As we explained previously, removing some principal components can reduce model accuracy while improving the parameter estimates. On the other hand, MPCA error rate is partic-

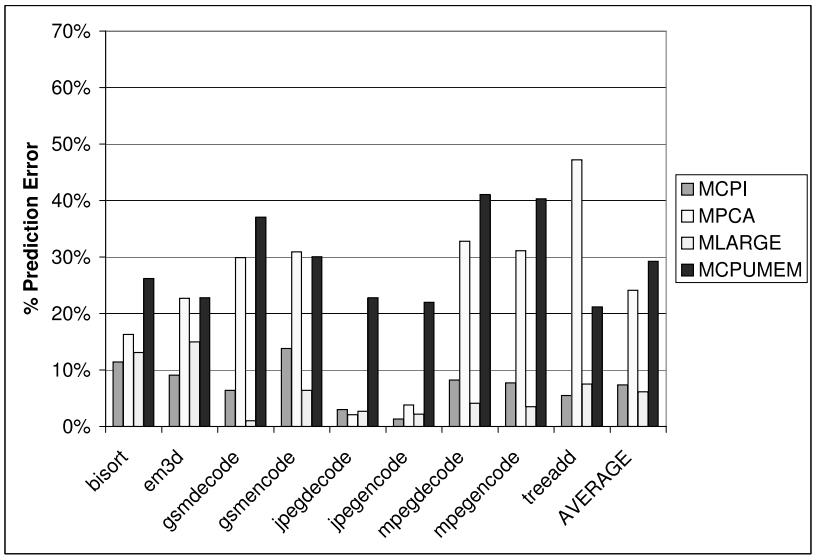

Figure 3: Error rate for the computation model.

ularly low for two benchmarks, jpegencode and jpegdecode. These two benchmarks are less data intensive and more processor bound than the other benchmarks. Our HPM data indicates that jpeg executes one instruction per data stall. The MPCA is particularly successful in modeling processor than the memory – as there is no direct memory access counter in XScale, we can measure memory access cost only indirectly, as a combination of other counters. However, since we remove some principal components from our model to improve model coefficient estimations, MPCA lose significant information about memory access cost.

MCPUMEM produces an average error rate of 30%. Moreover, its error rates vary from 20% to 40% across benchmarks, with a standard deviation that is twice that of the other models. The reason for this is that MCPUMEM is designed to estimate the power consumption of the CPU alone and not the full system. In particular, MCPUMEM does not account for memory bus and device access. Our models consider the consumption of the entire device at the instruction level and thus, are able to account for energy consumption of the system more accurately. This result emphasizes the importance of taking a global view of energy estimation – which has not yet been done in prior work, to our knowledge.

A significant result of our model is the success of MCPI model. The MCPI has an error rate close to the MLARGE model, using a single model input -the clock cycles per instruction. On XScale, we can gather this information accurately by reading the two hardware performance monitors; the clock cycle counter and the instruction counter. One of our short term goal is to implement a runtime energy estimator that can predict energy consumption online, as the program executes, and verify it across a large set of benchmarks.

3.2 Computation and Communication

We next integrate the computation and communication model and evaluate the accuracy of the combined model. We estimate the energy consumption using $E_t = E_l + E_n$ where E_t is the total energy consumption, E_l is the computation model output and E_n is the network model output. We use the names of the computation models that we discussed in the prior section to identify the

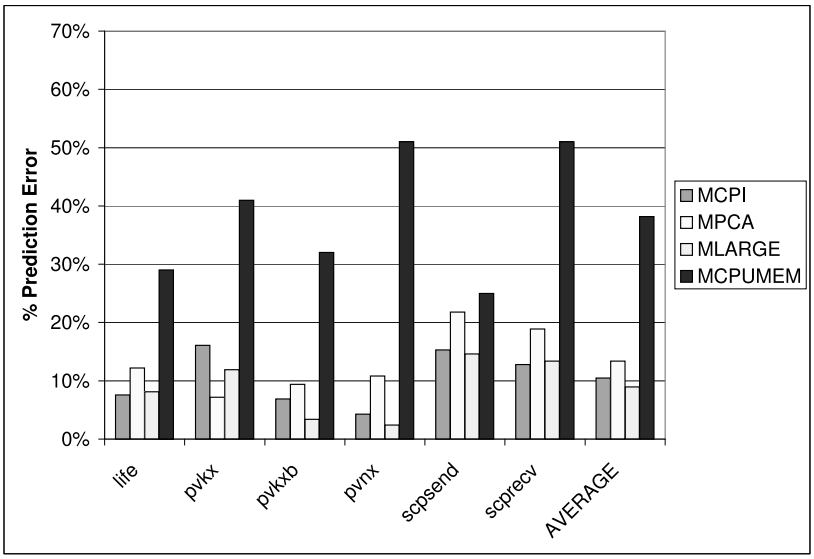

Figure 4: Error rate for the communication model.

integrated models in this section. MCPUMEM does not consider communication in its model.

We evaluate our model using the COMM reference benchmarks. The benchmarks exhibit (i) upload heavy, (ii) download heavy, and (iii) mesh like scenarios. We have selected these scenarios since they represent a wide set of sensor network applications. In terms of network activity, the total amount of data transfer is 2MB for pvkx, 550KB for life, 2.1MB for pvnx, 1.8MB for pvkxb and 1.7MB for scpsend.

Figure 4 shows the percent error for all our models. MCPI and MLARGE perform similarly with an average error rate of 11%. The error rate is 38% on average for MCPUMEM. MCPUMEM again does not account for other system activities that our models are better able to capture.

Interestingly however, in communication dataset the MPCA is as accurate as than the other models, with an average error of 13%. The prediction error of MPCA is below 25% for all programs. MPCA is significantly different than all other models. As we show in Table 4, MPCA gives much higher weight to instruction pipeline events (TLB miss, instruction not delivered, etc) than the other models. In our communication dataset, due to frequent I/O between wireless card and CPU, such events play a significant role in describing programs energy consumption. Hence, we believe MPCA is important as it addresses the deficiencies in other models.

As part of future work, we plan to investigate ways of improving our network model by considering multiple transfer rates, noise, and multiple wireless cards. Moreover, we plan to integrate a model for flash memory access into our full system energy estimation model. Finally, we plan to employ this resulting model for power-aware optimizations of the system; we plan to investigate ways of employing MCPI for computationally bound tasks and MPCA for communicating tasks to achieve the best of both models for energy estimation for a wide array of programs.

4 Related Work

In our work, we model full system energy consumption for sensor network gateways. Our model uses hardware and operating system monitors to estimate the power consumption of a sensor network gateway. The work most related to our own is on HPM-based models for CPU and memory energy estimation [3, 12, 26, 13, 9, 4]. In recent work, Bircher et al. [3] presents a power model for

the Pentium-IV class of processors. They develop their power model using least squares regression (LSQ) [5] and show that two hardware counters are enough to model energy consumption on their target architecture. However, they do not consider memory, and memory bus in their study.

In embedded systems, the most similar study to our own is by Contreras et al. [4]. In this work, the authors use LSQ to develop a power model for an Intel XScale processor attached to a development board. Using this model, the authors are able estimate CPU energy consumption with a 4% error rate. However, their efforts to construct a memory power model did not perform as well due to the lack of hardware counters in the CPU that count memory events. In our work, we extend (and compare) this HPM model to estimate full system energy consumption by employing software counters. Our model considers a larger set of components including memory and memory bus and demonstrates a lower error rate.

5 Conclusions and Future Work

In this paper, we present a system for estimating the full-system power consumption of the Crossbow Stargate sensor network device. Our system couples statistical techniques that employ empirical data from hardware and software performance monitors to model the computation and communication of executing tasks. Our results indicate that the CPI hardware metric is sufficient for accurate estimation full-system power consumption using our methodology. Moreover, we find that principle component analysis can reduce multicollinearity in the model data to enable us to extract useful information about the contribution of the model components; however, doing so comes at a cost in accuracy.

References

[1] B.D.Cahoon. *Effective Compile-Time Analysis for Data Prefetching In java*. PhD thesis, University of Massachusetts at Amherst, 2002.

[2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. In *In Proceedings of Design, Automation and Test in Europe*, 2000.

[3] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 275–280, New York, NY, USA, 2005. ACM Press.

[4] G. Contreras and M. Martonosi. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 221–226, 2005.

[5] R. Freund and P. Minton. *Regression Methods, A tool for Data Analysis*, volume 30. Marcel Dekker, INC, 1979.

[6] Game of Life – http://daugerresearch.com/vault/parallellife.shtml.

[7] M. R. Guthaus, J.S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, dec 2001.

[8] C. Isci and M. Martonosi. Identifying program power phase behavior using power vectors. In *WWC '03: Proceedings of the Sixth International Workshop on Workload Characterization*, 2003.

[9] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO '03: Proceedings of the 36th ACM/IEEE International Symposium on Microarchitecture*, 2003.

[10] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA '06: Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, 2006.

[11] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[12] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2001. ACM Press.

[13] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 28–31, New York, NY, USA, 2001. ACM Press.

[14] C. Krintz, Y. Wen, and R. Wolski. Application-level Prediction of Battery Dissipation. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, August 2004.

[15] O. Landsiedel, K. Wehrle, and S. Gtz. Accurate Prediction of Power Consumption in Sensor Networks. In *In Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

[16] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communication systems. In *Proceedings of the 30th ACM/IEEE International Symposium on Microarchitecture*, pages 330–335, 1997.

[17] R. Myers. *Classical and Modern Regression with Applications*. PWS-KENT Publishing Company, 1989.

[18] Q.O.Snell, A.Mikler, and J.L.Gustafson. Netpipe: A network protocol independent performance evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[19] D. Rakhmatov and S. Vrudhula. Time-to-failure estimation for batteries in portable electronic systems. In *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2001.

[20] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Tina: a scheme for temporal coherency-aware in-network aggregation. In *MobiDe '03: Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, pages 69–76, New York, NY, USA, 2003. ACM Press.

[21] V. Shnayder, M. Hempstead, B. Chen, and M. Welsh. PowerTOSSIM: Efficient Power Simulation for TinyOS Applications. In *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004.

[22] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, November 2004.

[23] Sundials: Suite of nonlinear and differential algebraic equation solvers – http://www.llnl.gov/CASC/sundials/main.html.

[24] K. C. Syracuse and W. Clark. A statistical approach to domain performance modeling for oxyhalide primary lithium batteries. In *In Proceedings of Annual Battery Conference on Applications and Advances*, January 1997.

[25] Y.F. Wang, E. Y. Chang, and K. P. Cheng. A video analysis framework for soft biometry security surveillance. In *VSSN '05: Proceedings of the third ACM international workshop on Video surveillance & sensor networks*, pages 71–78, New York, NY, USA, 2005. ACM Press.

[26] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, New York, NY, USA, 2002. ACM Press.

[27] Y. Wen, R. Wolski, and C. Krintz. History-based, Online, Battery Lifetime Prediction for Embedded and Mobile Devices. In *Workshop on Power-Aware Computer Systems (PACS)*, April 2003.