Comput Sci Res Dev (2010) 25: 177–186
DOI 10.1007/s00450-010-0125-1

SPECIAL ISSUE PAPER

Adaptive estimation and prediction of power and performance in high performance computing

Reza Zamani · Ahmad Afsahi

Published online: 1 August 2010

Abstract Power consumption has become an increasingly important constraint in high-performance computing systems, shifting the focus from peak performance towards improving power efficiency. This has resulted in significant research on reducing and managing power consumption. To have an effective power management system in place, it is essential to model and estimate the runtime power of a computing system. Performance monitoring counters (PMCs) along with regression methods are commonly used in this regard to model and estimate the runtime power. However, architectural intuitions remain fundamental with regards to the current models that relate a computing system's power to its PMCs.

By employing an orthogonal approach, we examine the relationship between power and PMCs from a stochastic perspective. In this paper, we argue that autoregressive moving average (ARMA) models are excellent candidates for modeling various trends in performance and power. ARMA models focus on a time series perspective of events, and we adaptively update them through algorithms such as recursive-least-squares (RLS) filter, Kalman filter (KF), or multivariate normal regression (MVNR). We extend the notion of our model to predict near future power and PMC values. Our empirical results show that the system-level dynamic power is estimated with an average error of 8%, and dynamic runtime power and instructions per cycle can be predicted (65 time steps ahead) with an average error of less than 11.1% and 7%, respectively.

R. Zamani (✉) · A. Afsahi
Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada
e-mail: reza.zamani@queensu.ca

A. Afsahi
e-mail: ahmad.afsahi@queensu.ca

Keywords Power estimation · Power prediction · High performance computing · ARMA

1 Introduction

High-performance computing (HPC) is the cornerstone of the scientific community in tackling challenging problems in diverse fields such as energy, medicine, weather and climate, finance, defense, and data mining, to name a few. Traditionally, the performance of HPC systems has been the main focus. However, power consumption and cooling issues have become an increasingly important design constraints, resulting in high operational and maintenance costs. Modern HPC systems incur a substantial cost in their total cost of ownership due to the use of a very large number of power hungry cores. For instance, Jaguar, the current leading system on the Top500 list [5], recording a performance of 1.75 petaflops/s, has 224162 cores, requiring a 6.9 megawatts of power. Assuming a nominal cost of $0.10/kWh, this translates into an average annual cost of $6.9M, just for electricity bill, let alone the costs associated with the cooling systems and backup power generations. Such concerns gave rise to the Green500 list [1], effectively shifting the focus from peak performance to power efficiency. For example, the current leading system on the Green500 list (722 MFlops/Watt) is approximately three times more power efficient than Jaguar, ranking 44th on the list.

Substantial research has been done at different levels of abstraction to manage the power consumption of modern computing systems. Power management (PM) can target different objectives: keeping the instantaneous power consumption within acceptable thresholds, reducing the total energy consumption, keeping temperature metrics within acceptable thresholds, etc. Estimation and prediction of

power consumption and performance of a system is beneficial for many PM programs to address their objectives. Estimation of power is necessary due to the unavailability of embedded power measurement features and accurate power models for the current computing systems. Prediction allows us to abate the future penalty of present PM decisions for changing the power and performance states of a system. For example, choosing a power and performance state higher than the required computational needs leads to unnecessary excess power consumption, while choosing a lower state results in significant performance loss. However, the high complexity associated with a large number of interoperating hardware and software components makes it very difficult to derive an analytical model to accurately estimate or predict the system-level power consumption or performance of a computing system. External measurement devices partially alleviate the online power measurement challenge. However, they introduce a delay in sending the power measurements to the PM programs, exacerbating its effectiveness. Having such challenges in mind, a model that can accurately estimate and predict the power consumption becomes indispensable in PM.

Modeling the relationships of PMCs, workload, and power consumption of a system have been approached from the architectural point of view previously [8–10, 13, 16, 17]. In an orthogonal approach, we examine such relationships from a stochastic aspect. Although similar time series approaches have been used by researchers in other fields and applications [18–20], we propose using the ARMA models [12], as a promising candidate, for relating the power and performance metrics to each other.

Figure 1 depicts the block diagram of our approach. It consists of a power-performance model and an update algorithm. ARMA or moving average (MA) models relate the input and output values to each other via different coefficients. Finding and adaptively updating these coefficients can be done by different algorithms, such as RLS, KF, MVNR, etc. After each time step, the modeling error is provided to the *coefficient update* algorithm and the coefficients are adjusted according to the objectives of the chosen algorithm.

This work contributes by proposing the use of ARMA models jointly with the coefficient update algorithms, such as RLS, to model the relationship between power and performance in order to estimate and predict them. This stochastic approach utilizes "both" the current and the past PMC values in a model to estimate or predict the power consumption or PMC values. Furthermore, our model integrates feedback power measurements for more accuracy. The proposed method is both platform-independent and application-independent, and does not require tapping into the system's internal power supply lines.

We have conducted our experiments on a real multi-core system, and gathered PMC values and system power

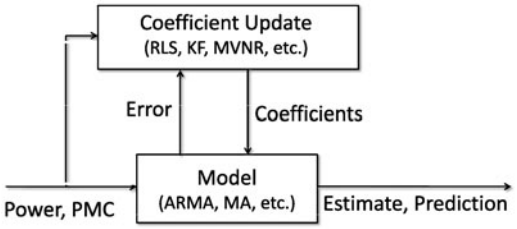


Fig. 1 Block diagram of the model and coefficient update algorithm

consumption measurements in real-time when running serial and OpenMP [4] applications from the NAS Parallel Benchmark (NPB) suite [3]. Using the proposed method, our model can estimate the "dynamic part" of the system-level power consumption for a set of serial and parallel benchmarks with an average error of 8%. The models studied in this paper have the potential to predict the future PMC and power consumption metrics based on their past values. We have evaluated the efficiency of this approach for different coefficient update algorithms and different prediction ranges. For example, on average, we are able to predict the (65 time steps ahead) dynamic part of the system-level power consumption of our benchmarks with an average error of less than 11.1%. Similarly, we can predict the instructions per cycle (IPC) of our benchmarks with less than 7% error.

The rest of the paper is organized as follows. In Sect. 2, we review the related work. In Sect. 3, we introduce the mathematical models and algorithms that we use in this paper. Our experimental platform is described in Sect. 4. In Sect. 5, we introduce a PMC-based power model, and discuss its sensitivity to feedback delay. Section 6 discusses our performance and power prediction methods. In Sect. 7 we investigate how the estimation model handles the extreme cases. Finally, in Sect. 8, we make concluding remarks and indicate some directions for future research.

2 Related work

Many researchers have used PMCs or a sort of access rate metric in various energy models of computer systems or some of its components [8, 10, 15, 17]. Contreras et al. [10] estimate the power consumption of CPU and memory using a first-order linear sum of PMCs. Joseph et al. [15] estimate the runtime power dissipation of different units of a Pentium Pro processor using PMCs combined with architectural information provided by an architectural processor power simulator. Rajamani et al. [17] have used PMCs to model instantaneous performance and power of an Intel Pentium-M processor. Their approach is platform-specific and needs training data set through micro-benchmarking. Some researchers [8] have found IPC metrics useful in power estimation techniques. Bircher et al. [8] have used linear regression models to estimate the power consumption of a

single-core processor (Pentium 4). They report that IPC related metrics show a strong correlation with CPU power, in particular the *uops fetched per cycle* metric. A downside of many previous work is the necessity to tap the supply points that power the processor to measure its power consumption.

Kalman filter [6] has been used in different ways to improve power or energy efficiency: as a workload estimator [7], as well as a resource management tool [14]. Jain et al. [14] have looked at stream resource management from a filtering point of view and have exploited KF in their solution. Bang et al. [7] have proposed a KF-based workload estimation method for dynamic voltage scaling, where they estimate the processing time of real-time multimedia workloads. The way we use KF in this paper is different from [7, 14] as we are not estimating a workload or an unknown value. We use KF as a coefficient update algorithm for an ARMA model.

Time series prediction methods are well known and widely used in financial and industrial problems, where ARMA and autoregressive (AR) models are commonly utilized along with system identification techniques. Some researchers have applied them to computing systems [18–20]. Xu et al. [19] have studied predictive closed-loop control techniques for systems management by comparing algorithms based on AR models, combined analysis of variance (ANOVA) and AR models, and a multi-pulse (MP) model. Zhu et al. [20] as well as Wang et al. [18] have discussed the strength of control theory approaches in computing systems, where they use an ARMA model along with system identification experiments to capture the dynamic relationship between the CPU allocation to a web server and its measured mean response time. To the best of our knowledge, an ARMA model has not been used in modeling power and performance relationships in computing systems.

Gurun et al. [11] have proposed a runtime feedback-based full system energy estimation model for embedded devices, where a moving average of order zero (i.e., MA(0) model) of two or three hardware and software performance counters is used to model communication or computation energy consumption. The advantage of their approach is in using *recursive least squares linear regression with exponential decay* (RLS-ED) for finding and updating the model parameters on-the-fly. Our work in this paper is substantially different than [11] in that we are using an ARMA model that is non-zero order for both the AR part and the MA part. The MA part of our model incorporates the past PMC values, while the autoregressive part of our model utilizes feedback power measurement.

The core contributions of this paper with respect to the previous work can be summarized as: (1) proposing the use of a non-zero order ARMA model that takes into account the previous power and PMC values in estimating and predicting the future values; (2) studying different coefficient update algorithms to find the best algorithm that enables the proposed time-series approach to adaptively adjust to the behavior of the application and system; (3) engaging feedback power measurements in our model without a necessity to intrusively tap into the internal power supply lines; and (4) evaluating the efficacy and efficiency of our model on a real multi-core system using HPC benchmarks.

3 Models and algorithms

In this section, we provide the mathematical background for the models and algorithms that are used in this work.

3.1 ARMA and MA

Our estimation and prediction methodology in this paper is based on autoregressive moving average models [12]. ARMA models are widely used in different fields, such as in economic time series prediction, hydrology, dendrochronology, etc. A general form of ARMA(p,q) represents a model with p autoregressive terms (i.e., AR(p)) and q moving average terms (i.e., MA(q)) and is shown in (1). The X_i, c, ε_i, ψ_i, and θ_i are scalar variables.

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \psi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{1}$$

In order to estimate the parameters of our ARMA models in this paper we use different algorithms, such as recursive least-squares filters and Kalman filters, in addition to standard multivariate normal regressions.

3.2 The discrete-time Kalman filter

The Kalman filter is a recursive filtering algorithm that enables us to estimate the state of a process [6]. It has been extensively studied and used in different fields of science and engineering. The Kalman filter estimation is done in a way that the mean of the squared error is minimized. The KF algorithm consists of a cycle of *prediction* and *correction*. In the prediction phase, the process state in the upcoming time step is estimated. In the correction phase, based on the observed measurement the algorithm is adjusted for a better prediction. The signal model consists of a *process equation* shown in (2) and a *measurement equation* shown in (3).

$$x_{k+1} = F_k x_k + w_k \tag{2}$$

$$z_k = H'_k x_k + v_k \tag{3}$$

The state of process is shown by $x \in \mathbb{R}^n$. The measurement of process is shown by $z \in \mathbb{R}^m$. The *process noise* is shown by the random variable w_k with normal distribution

$N(0, Q)$. The *measurement noise* is represented by the random variable v_k with normal distribution $N(0, R)$. The initial process state x_0 is $N(\bar{x}_0, P_0)$. The process noise $\{w_k\}$, measurement noise $\{v_k\}$, and the initial process state x_0 are jointly Gaussian and mutually independent. The $n \times n$ matrix F_k relates the current state x_k to the next state x_{k+1}, when there is no process noise. The $m \times n$ matrix H'_k relates the current state x_k to the current measurement z_k. The matrix H'_k is the transpose of matrix H_k. For conciseness, we do not provide the solution to the KF algorithm. The interested reader is referred to [6].

3.3 System identification using KF

In this part, we explain how KF can be used for finding the coefficients $a^{(1)}, \ldots, a^{(n+m)}$ of the scalar ARMA equation using the system input measurements $\{u_k\}$ and system output measurements $\{y_k\}$ as they become available overtime in (4).

$$y_k + \sum_{j=1}^{n} a^{(j)} y_{k-j} = \sum_{j=1}^{m} a^{(n+j)} u_{k-j} \tag{4}$$

Let (4) describe the behavior of a system with *constant* coefficients $a^{(i)}$, then with enough number of measurements and solving a set of linear equations one can find those coefficients. However, if the coefficients are varying or the equation is not fully modeling the system, in order to enable the model to follow the changes, the coefficients of the ARMA equation should be adaptively updated as new measurements become available. By assuming that the coefficients in (4) are changing overtime, we can rewrite the equation as (5). A Kalman filter approach, among others, can be used for finding and updating the coefficients. In short, the $(m + n)$ coefficients of the ARMA are assumed to be the state of a process as in (6). The F_k of the KF as defined in (2) is set as the identity matrix. The H' matrix is defined as a row vector in (7). This method is explained in detail in [6].

$$y_k + \sum_{j=1}^{n} a_k^{(j)} y_{k-j} = \sum_{j=1}^{m} a_k^{(n+j)} u_{k-j} + v_k \tag{5}$$

$$x_k^{(1)} = a_k^{(1)}, x_k^{(2)} = a_k^{(2)}, \ldots, x_k^{(n+m)} = a_k^{(n+m)} \tag{6}$$

$$H'_k = [-y_{k-1} \ \ldots \ -y_{k-n} \ u_{k-1} \ldots \ u_{k-m}] \tag{7}$$

3.4 Recursive least-squares filter

The RLS algorithm is used for finding the coefficients of adaptive filters, and it recursively produces the least squares of the error signal. Unlike many other adaptive filtering methods that try to reduce the *mean square error* and require statistical information about the input or the desired output signals, the RLS calculates a least squares error directly from the input and the desired output. This makes the RLS filters a signal-dependent algorithm. The RLS filter is computationally less intensive than the KF as it does not require any matrix inversion. It is noteworthy to mention that a RLS filter can be reformulated as a KF as in (8). Details can be found in [12].

$$x_{k+1} = \lambda^{-1/2} x_k, \qquad z_k = H'_k x_k + v_k \tag{8}$$

4 Experimental framework

All the experiments in this paper are conducted on a Dell PowerEdge R805 SMP server. The server has two quad-core 2.0 GHz AMD Opteron processors. The processors have 12 KB shared execution trace cache, and 16 KB L1 shared data cache on each core. The L2 cache available for each core is 512 KB. Each processor chip also has a shared 2 MB L3 cache. Our system has 8 GB DDR-2 SDRAM (667 MHz) memory.

Our measurement infrastructure consists of a Keithley 2701/7710 digital multi-meter (DMM), a shunt resistor, and the node under measurement that performs the profiling task. We measure the power consumption of the node by measuring the voltage of the shunt resistor placed between the wall power outlet and the node. Knowing the value of the resistor, we first calculate the current and then the power and energy consumption of the node. We read 3 AC-voltage samples per second. In the DMM, the signal first goes through an internal analog RMS-converter, where 1000/60 DC samples are read out and averaged for each AC sample. The power measurements are validated with another industry-made power meter, Wattsup, and the measurement error is less than 1%.

The operating system is Ubuntu Linux, running kernel version 2.6.28.9 patched with the `perfctr` library version 2.6.39 for PMC measurement purposes. In our tests, the measured events are: *Dispatch stalls*, *memory controller page access event*, *retired ×86 instructions*, and *cycles with no FPU ops retired*. Our event selection is roughly based on previous research on modeling power by PMCs. We run an application program on the node along with the PMC profiling code which is synchronized with the power measurement software. The overhead of the PMC profiling code and the power measurement software is shown to be minimal.

We use a number of serial and OpenMP applications from the NAS parallel benchmarks (NPB) [3] suite in our study. These applications consists of NPB-3.3-SER benchmark suite (*BT.A*, *BT.B*, *CG.B*, *EP.B*, *FT.B*, *LU.A*, *LU.B*, *SP.A*, *SP.B*, *UA.A*, and *UA.B*) and NPB-3.3-OMP (*BT.C*, *CG.C*, *LU.B*, *SP.C*, *UA.B*, and *BT.B*) running with eight threads. We

have chosen those applications in class B and C of NPB-3.3 that run for longer than 100 seconds on our system, in order to have sufficient samples to compare the algorithms used in this study. In the serial applications, we set the affinity of the application process to only one core.

As stated earlier, the methodology proposed in this paper consists of a model and an update algorithm. The six different combinations of models and algorithms that we study in this paper are as follows. An ARMA model of order $(4, 4)$ equipped with the RLS, KF, MVNR, and block MVNR update algorithms, which are referred to as RLS, KF, MVNR, and BMVNR, respectively. The BMVNR algorithm is an algorithm similar to MVNR, except that it only looks at a fixed size block of information, unlike MVNR that uses the whole trace. We also use a MA of order zero model equipped with the RLS update algorithm (similar to [11]), which we denote it as MA-0 in this paper. The combination of a zero-order MA model and a MVNR update algorithm, when applied to the entire signal profile in advance, is referred to as "Oracle" (similar to [8]). Oracle is a non-adaptive and a non-causal algorithm and is merely implemented here for comparison purposes. It should be mentioned that the estimation and prediction algorithms are performed offline in MATLAB. We are currently integrating them within the system.

5 Power estimation using ARMA

In the previous sections, we discussed how an ARMA (or MA) model can be equipped with an update algorithm for adaptively tuning its coefficients. In this section, we put this idea into practice. We apply power and PMC traces of different benchmarks to different configurations of a model with an update algorithm to evaluate their effectiveness in modeling and relating the power and performance trends. In fact, we are examining if power consumption is linearly related to previous power measurements, as well as linearly related to the current and past PMCs. We model this relationship as an ARMA model in (10) where $\mathrm{P}[t]$ represents the power measurement of the system at time t.

$$\mathrm{P}[t] = \sum_{j=1}^{j_{\max}} \alpha_{0,j} c_j[t] = A_0 \mathrm{C}'[t] \tag{9}$$

$$\mathrm{P}[t] + \sum_{i=1}^{n} \beta_i \mathrm{P}[t-i] = \sum_{i=0}^{m} A_i \mathrm{C}'[t-i] \tag{10}$$

We define $\mathrm{C}[t]$ as a row matrix that contains the PMC values at time t and can be shown as $[c_1[t], \ldots, c_{j_{\max}}[t]]$. There are $j_{\max}$ PMCs used in the model ($j_{\max} = 4$). The jth performance monitoring counter measurement at time $t - i$ is linearly related to the current power consumption measurement $\mathrm{P}[t]$ via a coefficient $\alpha_{i,j}$. We define A_i as a row matrix that contains the $\alpha_{i,j}$ coefficients and can be shown as $[\alpha_{i,1}, \ldots, \alpha_{i,j_{\max}}]$. A past power measurement at time $t - i$ is related to the current power measurement via a coefficient β_i. The time window that (10) covers includes the current and m previous PMC measurements, as well as the past n power measurements.

We gather the power and PMC profile of different NPB benchmarks and estimate the power consumption using the six combinations of our models and algorithms explained in Sects. 3 and 4 in order to evaluate their effectiveness. The model parameters used in the following results are: $m = 4$, $n = 4$, and $j_{\max} = 4$ as mentioned in (10). The PMC events used in our tests are: dispatch stalls, memory controller page access event, retired $\times 86$ instructions, and cycles with no FPU ops retired. In case of the MA(0) model, the power and PMCs can be formulated as in (9).

5.1 Estimation results

Let power measurement P range between $\mathrm{P}_{\min}$ and $\mathrm{P}_{\max}$. One can split a power measurement into a dynamic part and a static part, $\mathrm{P} = \mathrm{P}_{\mathrm{dynamic}} + \mathrm{P}_{\mathrm{static}}$. If all the measurements are larger than $\mathrm{P}_{\min}$, then $\mathrm{P}_{\min}$ is a static part of the measurements; $\mathrm{P}_{\mathrm{dynamic}} = \mathrm{P} - \mathrm{P}_{\min}$. One can use the total signal to derive the percentage of the mean of absolute error of an estimation method, in contrast to using the dynamic signal. However, if the static part of the estimated signal is large, the difference between the efficiency of the various estimation methods will not be distinguished as much as when the dynamic signal is taken into consideration. For completeness, we report both the mean absolute error of dynamic signal (MAEDS) and the mean absolute error of total signal (MAETS).

As an example, power measurements between 250 W and 300 W and a mean absolute error of estimation of 10 W, yields a dynamic signal range of 50 W, static part of 250 W, MAETS of 3.3% (10/300), and MAEDS of 20% (10/50). Some estimation methods tend to have a "training" or "warm up" period to become efficient, thus to avoid unfair comparisons we calculate the MAEDS and MAETS metrics using the measurements and estimation after the first hundred samples. However, the whole signal profile has been used for the estimation phase. In this paper we use the term "overall average MAEDS (or MAETS)" that is calculated in two steps: (1) calculating the MAEDS (or MAETS) of each benchmark, (2) then calculating the average of the MAEDS (or MAETS) among all the benchmarks.

A part of the measured runtime power and its estimate for BT.C OpenMP application running with eight threads is shown in Fig. 2 for all the six mentioned methods. The block size for BMVNR is set arbitrarily to 75 samples in

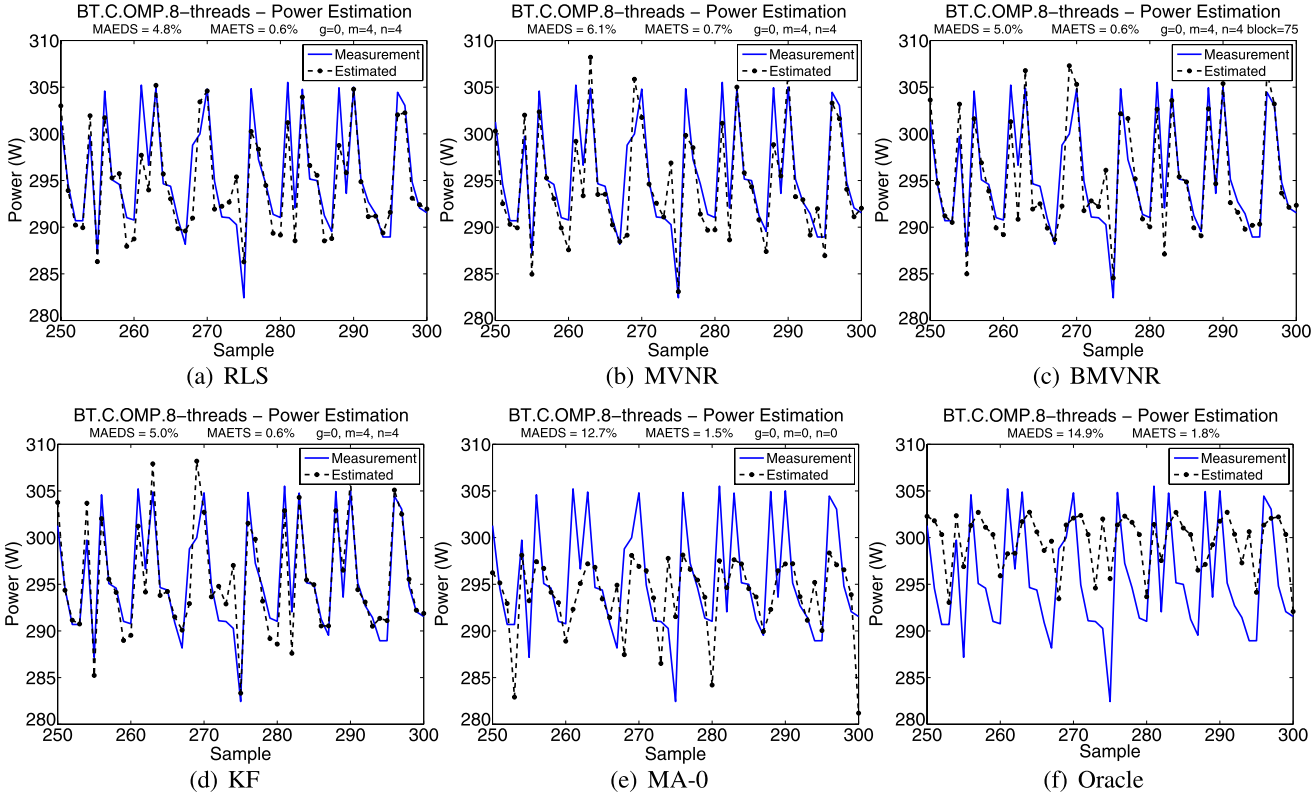


Fig. 2 Comparing power estimation of BT.C running with 8 threads using RLS, MVNR, BMVNR, KF, MA-0, and Oracle models

this paper. The MAEDS of BT.C.OMP.8 shown in Fig. 2 for RLS, MVNR, BMVNR, KF, MA-0, and Oracle methods is 4.8%, 6.1%, 5.0%, 5.0%, 12.7%, and 14.9%, respectively. For most of the application traces, RLS is the best method in terms of estimation error. The overall average of MAEDS for all the applications studied in this paper for RLS, MVNR, BMVNR, KF, MA-0, and Oracle methods is 8.0%, 8.1%, 8.2%, 8.5%, 15.6%, and 19.5%, respectively. Similarly, the overall average of MAETS among all the applications for RLS, MVNR, BMVNR, KF, MA-0, and Oracle models is 0.62%, 0.67%, 0.70%, 0.68%, 1.26%, and 1.46%, respectively.

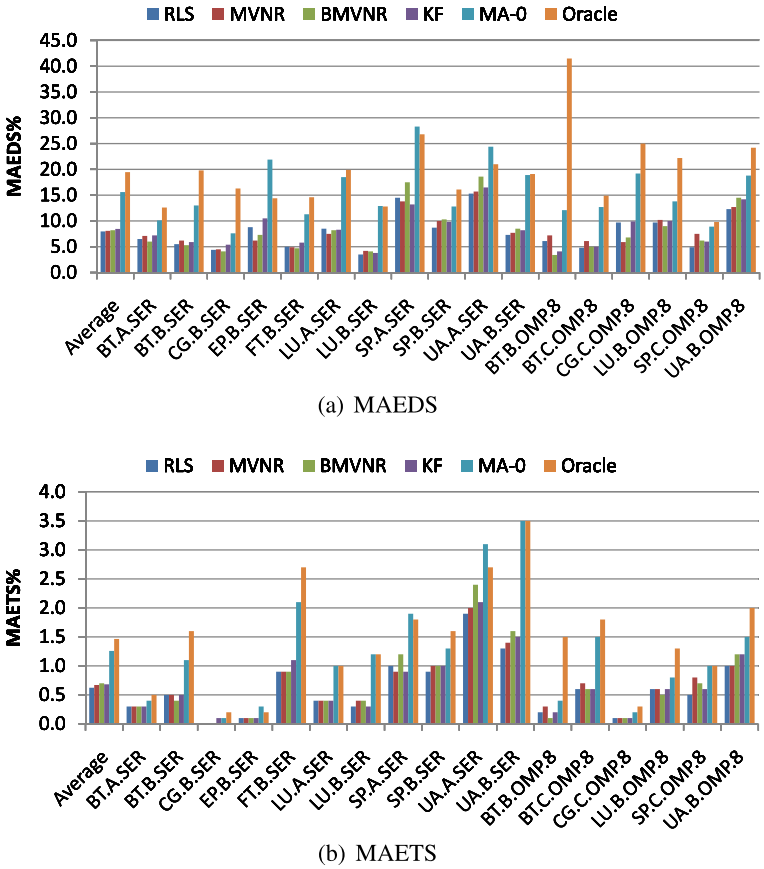
We present the MAEDS and MAETS metrics for all the applications and the overall average in Fig. 3. The minimum and maximum error when using MAEDS among all the applications are 3.4% and 41.5%, respectively. One can notice the excellent efficiency of the ARMA $(4, 4)$ model when combined with RLS, for estimating the power consumption. The order of the ARMA model has an impact on the error levels and computation time. For ARMA models of an order larger than four, we did not observe a significant improvement in efficiency. One can conclude that ARMA $(4, 4)$ is a good candidate for estimating power.

5.2 Computation-time overhead

In order to have a real-time power estimation method integrated in a system, such as the ones mentioned above, the estimation method requires to perform computation much faster than the measurement sampling rate. An estimation method, such as MVNR, that uses an increasing window size of inputs from the beginning until the estimation time, becomes slower over time, and therefore not suitable for integration in a system. To alleviate this problem, one can use a fixed-size block for MVNR (e.g., BMVNR) to lower the computational time of the MVNR method. The KF method in principle should have a fixed computation size as it has only a fixed number of matrix operations for each time step. However, depending on the parameters of the ARMA, the matrix inversion operations can be costly for the KF approach. In our simulation, the MVNR, BMVNR, and KF methods ran 321, 37, and 117 times longer than the RLS method for BT.C.OMP-8 shown in Fig. 2. The actual implementation of RLS takes approximately 710 microseconds per sample on our experimental system described in Sect. 4.

5.3 Sensitivity to measurement update delay

In this section, we investigate the impact of presence of a delay in receiving some of the measurements in our ARMA



(a) MAEDS

(b) MAETS

Fig. 3 Mean absolute error of power estimation (dynamic and total signal) for different applications using different coefficient update algorithms

model (e.g. delay in receiving measurements from an external DMM). To include a time gap g between the current samples and the previous samples, we rewrite the ARMA equation of (10) as in (11), where Δ_{ij} is defined as $1-\delta_{ij}$, and δ_{ij} is the Kronecker delta, which by definition its value is 1 for $i=j$; 0, otherwise. In other words, $\Delta_{ij}=1$ for $i\neq j$; 0, otherwise.

$$\mathrm{P}[t]+\sum_{i=1}^{n}\beta_i\mathrm{P}[t-i-g]=\sum_{i=0}^{m}A_i\mathrm{C}'[t-i-g\Delta_{i0}] \qquad (11)$$

In Fig. 4 we present the impact of increasing the gap from 0 to 56 samples, with a step size of 8 samples, on overall MAEDS percentage of our applications. The increase in overall MAEDS is under 4% for RLS, MVNR, and KF approaches, which shows the excellent delay tolerance of our ARMA model.

6 Performance and power prediction using ARMA

In this section, we extend the notion of using an ARMA equation as in (11) to model the performance behavior of an application. In particular, we try to predict the number of *retired* $\times 86$ *instructions* per cycle in the upcoming time steps for an application by using the past values of the same metric, as well as other PMCs (i.e., *dispatch stalls*, *memory controller page access event*, and *cycles with no FPU ops retired*). For this purpose, we formulate this problem in the form of an ARMA equations shown in (12) (note that both sums start at $i=1$). The PMC that we try to predict at time step t is noted as $\mathrm{M}[t]$. The matrix definitions here are similar to the ones used in the previous sections.

$$\mathrm{M}[t]+\sum_{i=1}^{n}\beta_i\mathrm{M}[t-i-g]=\sum_{i=1}^{m}A_i\mathrm{C}'[t-i-g] \qquad (12)$$

A part of the predicted IPC at 65 steps ahead for *SP.C* running with eight threads is shown in Fig. 5(a). For conciseness, we only show the average MAEDS across all the applications (except *cg* that has a very large error compared to all other applications that we believe is due to its small variance in its PMC values) in Fig. 6. The overall MAEDS for our applications when predicting one time step ahead is 5.9%, 6.5%, and 5.6% for RLS, KF, and MVNR, respectively. As we increase the gap between the prediction time and the predicted sample we expect the error increases. The proposed model shows a very good behavior in terms of error growth. For example, the overall MAEDS is under 7.0% when predicting 65 samples ahead. This means that ARMA is a great candidate for predicting PMC values.

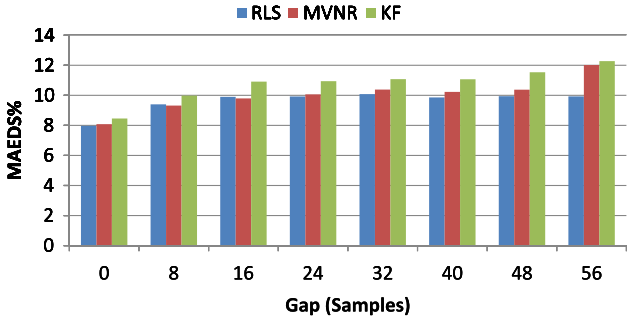


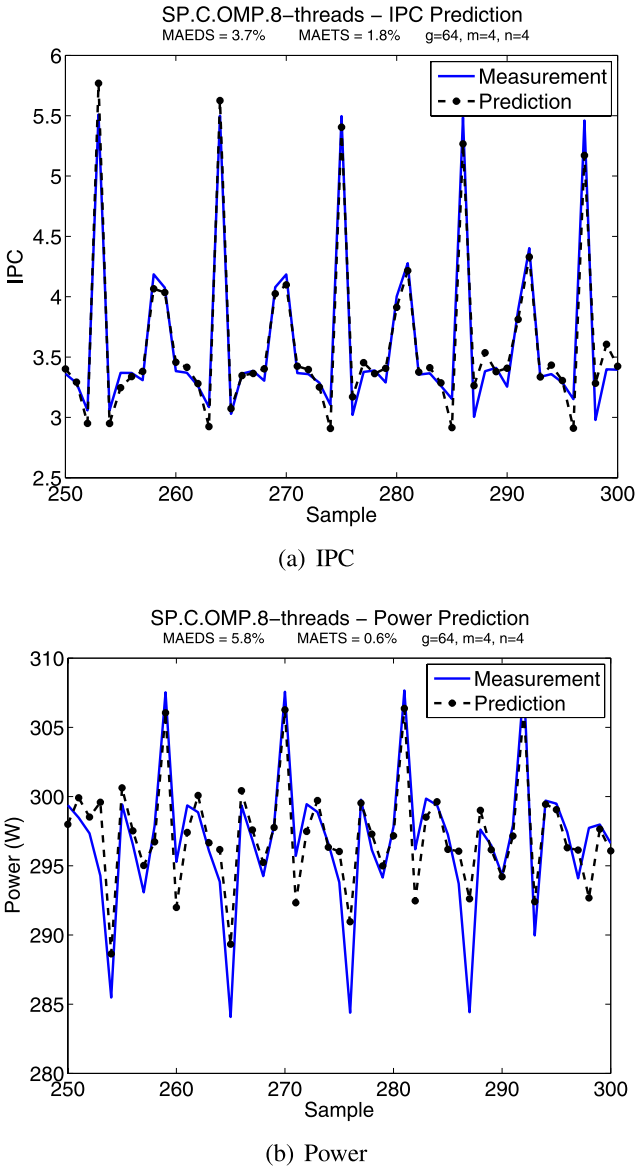
Fig. 4 Delay effect on mean absolute error using different coefficient update algorithms

We use our ARMA model in (12) to predict the power consumption of a system, with the difference that M[t] represents power in this case. A part of the predicted power at 65 steps ahead for *SP.C* running with eight threads is shown in Fig. 5(b). One time step ahead prediction for all of our applications (including *cg*) has an overall MAEDS value of 8.4%, 8.6%, 9.5% for RLS, KF, and MVNR, respectively. The RLS method can predict power consumption with overall MAEDS of under 11.1% for prediction time distance of 1 to 65. One can notice in Fig. 6 that this prediction method is able to keep a good efficiency for a large range of prediction time gap (shown up to 64).

In summary, an ARMA model, when combined with update algorithms, proves to be a promising candidate not only for estimating power, but also capable of predicting future power and PMC values. We find the ARMA-RLS combination as the best combination in our study, from both performance and computation efficiency aspects.

7 Further investigation

Many HPC applications comprise of control structures that make their behavior repetitive for some periods of their execution time (e.g. loops). Such repetitive behavior of an application makes the task of estimation or prediction easier. Therefore, it is necessary to evaluate the efficiency of an estimation or prediction model under extreme cases. For instance, a model can be challenged by the periods of time that the behavior of the application varies significantly or the system is in idle periods. For this purpose, we run multiple benchmarks (ft.B, sp.B, cg.C, and lu.B) consecutively, with a sleep period (three seconds) between every two benchmarks. A part of the power estimation of ARMA-RLS and its error for this test case is shown in Fig. 7. This allows us to just focus on the extreme part of the trace. One can notice that the estimation error increases as soon as the cg.C benchmark ends. However, the model feedback adjusts the increase in error to follow the power trend more accurately. Although during the idle period the activity of the system



(a) IPC

(b) Power

Fig. 5 IPC and power prediction for SP.C.OMP with 8 threads using RLS, gap = 64 samples (i.e. 65 steps ahead)

is minimal and the PMCs are not changing significantly, the power estimation follows the power measurements. When the next application starts (i.e. lu.B), the PMCs are suddenly changed and the estimates follow the new power trend.

The overshoot or undershoot in power estimation varies based on the "experience" of the coefficient update algorithm through the different parts of a trace. The RLS algorithm, and some other algorithms, are data dependant. The update algorithm changes the coefficients significantly if a significant error is observed and each chosen set of the coefficients almost result in a unique behavior of the estimator. The presented extreme case result in Fig. 7 is just an example scenario, keeping in mind that other traces result in different worst case estimation errors (larger or smaller er-

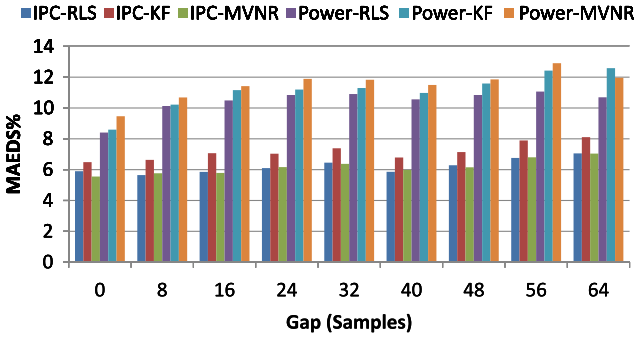


Fig. 6 Overall average of mean absolute error for different gap sizes when predicting IPC and power

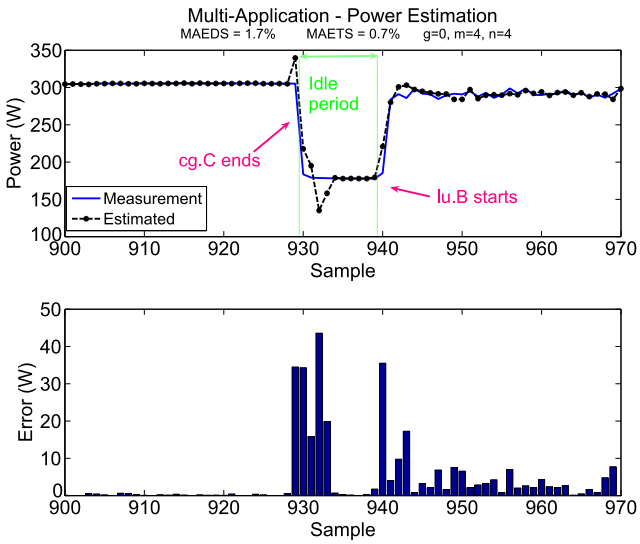


Fig. 7 Runtime power estimation of multiple applications (extreme cases: idle period, and start/end of a benchmark)

rors). We are currently investigating the different aspects of such extreme case handling by our proposed models.

8 Conclusions and future work

In this paper, we proposed the use of ARMA models for modeling power and PMC relationships when running serial and OpenMP applications. We used the RLS, MVNR, BMVNR, KF, MA-0, and Oracle algorithms for adaptively changing the coefficients of the ARMA, in order to overcome the shortcomings of the linear modeling in capturing the behavior of time-variant and non-linear HPC applications and systems. Our results showed that ARMA is an excellent model to estimate the current power consumption of a system, and that it is not significantly sensitive to the delay in receiving feedback measurement updates. The percentage of the mean absolute error for the dynamic signal (MAEDS) for all the NPB applications that we studied in this paper is as low as 8.0% (for RLS). In fact, the combination of ARMA and RLS provides the smallest errors and is the fastest in terms of computation time.

Furthermore, we investigated how ARMA combined with RLS, MVNR, or KF performs with respect to predicting the future IPC or power values. We find that this approach is an excellent candidate for this purpose as we achieved an overall MAEDS value of less than 7% and 11.1% for predicting IPC and power, respectively.

We extended our investigation to understand the efficiency of ARMA-RLS in estimating power for some extreme scenarios, such as the start or end of a benchmark or the idle periods of the system. The results for such cases vary based on the trace of the applications. We have realized that the ARMA-RLS model is capable of maintaining its efficiency in many cases. However, fine tuning is needed during some extreme cases. We are currently studying different aspects on how to effectively handle such cases.

Our work in this paper is the first step in providing the evidence that ARMA models combined with update algorithms are excellent candidates for the estimation and prediction of power and performance in HPC. We believe that our estimation and prediction techniques can help alleviate the challenges the PM systems face when using the dynamic voltage and frequency scaling technique. We are in the process of integrating our model in real-time for dynamic power management decisions. We plan to extend this work by incorporating the CPU frequency variable into the proposed model. We also intend to experiment with MPI [2] applications running on a cluster.

Acknowledgements This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant RGPIN/238964-2005, Canada Foundation for Innovation and Ontario Innovation Trust Grant 7154.

References

1. The Green500 list. www.green500.org
2. The message passing interface forum. www.mpi-forum.org
3. The NAS parallel benchmarks. www.nas.nasa.gov
4. The OpenMP api. www.openmp.org
5. The Top500 supercomputing site. www.top500.org
6. Anderson BDO (1979) Optimal filtering. Prentice Hall, New York
7. Bang SY, Bang K, Yoon S, Chung EY (2009) Run-time adaptive workload estimation for dynamic voltage scaling. Trans Comput Aided Des Integr Circuits Syst 28(9):1334–1347
8. Bircher WL, Valluri M, Law J, John LK (2005) Runtime identification of microprocessor energy saving opportunities. In: International symposium on low power electronics and design, pp 275–280
9. Cho Y, Kim Y, Park S, Chang N (2008) System-level power estimation using an on-chip bus performance monitoring unit. In: ICCAD '08. Proceedings of the 2008 IEEE/ACM international conference on computer-aided design, pp 149–154
10. Contreras G, Martonosi M (2005) 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, pp 221–226. ACM

11. Gurun S, Krintz C (2006) A run-time, feedback-based energy estimation model for embedded devices. In: CODES+ISSS '06. Proceedings of the 4th international conference on Hardware/software codesign and system synthesis. ACM, New York, pp 28–33
12. Hayes MH (1996) Statistical digital signal processing and modeling. Wiley, New York
13. Isci C, Martonosi M (2003) Runtime power monitoring in high-end processors: Methodology and empirical data. In: MICRO 36. Proceedings of the 36th annual IEEE/ACM international symposium on microarchitecture. IEEE Comput. Soc., Los Alamitos, p 93
14. Jain A, Chang EY, Wang YF (2004) Adaptive stream resource management using Kalman filters. In: SIGMOD '04. Proceedings of the 2004 ACM SIGMOD international conference on management of data. ACM, New York, pp 11–22
15. Joseph R, Martonosi M (2001) Run-time power estimation in high performance microprocessors. In: ISLPED '01. Proceedings of the 2001 international symposium on low power electronics and design. ACM, New York, pp 135–140
16. Li T, John LK (2003) Run-time modeling and estimation of operating system power consumption. SIGMETRICS Perform Eval Rev 31(1):160–171
17. Rajamani K, Hanson H, Rubio JC, Ghiasi S, Rawson FL (2006) Online power and performance estimation for dynamic power management. Tech Report
18. Wang Z, Zhu X, Singhal S, Packard H (2005) Utilization and slo-based control for dynamic sizing of resource partitions. In: IFIP/IEEE distributed systems: operations and management, pp 24–26
19. Xu W, Zhu X, Singhal S, Wang Z (2006) Predictive control for dynamic resource allocation in enterprise data centers. In: Network operations and management symposium, pp 115 –126
20. Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Padala P, Shin K (2009) What does control theory bring to systems research? SIGOPS Oper Syst Rev 43(1):62–69

Reza Zamani is currently a Ph.D. Candidate at the Department of Electrical and Computer Engineering, Queen's University, Canada. He received his BSc in Electrical Engineering from Sharif University of Technology, Iran, and MSc from Queen's University, Canada. His current research focuses on power-aware high-performance computing, and high performance communications.

Ahmad Afsahi is currently an Associate Professor in the Department of Electrical and Computer Engineering at Queen's University in Kingston, Ontario. He received his Ph.D. in Electrical Engineering from the University of Victoria, British Columbia, in 2000, and his MSc and BSc in Computer Engineering from the Sharif University of Technology and Shiraz University, Iran, respectively. His research interests include parallel and distributed processing, high-performance computing, high-performance networking, power-aware computing and parallel computer architecture. His research has earned him a Canada Foundation for Innovation Award and an Ontario Innovation Trust Award.