Throughput Regulation in Multicore Processors via IPA

N. Almoosa, W. Song, S. Yalamanchili, and Y. Wardi

Abstract— **This paper presents an online controller for regulating the throughput of instruction-sequences in multicore processors using dynamic voltage-frequency scaling. The proposed control law comprises an integral controller whose gain is adjusted online based on the derivative of the frequency-throughput relationship. This relationship is modeled as a stochastic DEDS having no analytic functional form, and hence its derivative is estimated by Infinitesimal Perturbation Analysis (IPA). However, the DEDS is multi-class and hence the IPA derivative is biased.**

Biasedness of IPA is a common problem in multi-class DEDS which has hindered the development of IPA as a general tool for practical applications. However, recently it has been suggested that as long as the relative bias has certain upper bounds, optimization algorithms and control laws can still converge to optimal or near-optimal parameters. The purpose of this paper is to demonstrate this point for the aforementioned problem of throughput regulation, thereby suggesting the potential emergence of a new class of effective control laws in computer architectures.

I. Introduction

Infinitesimal Perturbation Analysis (IPA) has been proposed as a sample-path sensitivity-analysis technique for stochastic Discrete Event Dynamic Systems (DEDS), and especially queueing networks [11], [3]. In particular, it computes the gradients (derivatives) of sample-performance functions with respect to a Euclidean variable. Let us denote this variable by $\theta \in R^n$, and let $J(\theta): R^n \rightarrow R$ be a sample performance function defined on a common probability space $(\Omega, \mathcal{F}, P)$; the IPA gradient is the sample gradient $\nabla J(\theta)$, whose dependence on the sample $\omega \in \Omega$ is suppressed in the notation used. The utility of $\nabla J(\theta)$ arguably can be had in situations where it is desirable to minimize the expected-value function $\zeta(\theta) := E\big[J(\theta)\big]$, $E\big[\,\cdot\,\big]$ denoting expectation in $(\Omega, \mathcal{F}, P)$, but $\nabla\zeta(\theta)$ lacks a closed-form expression and has to be estimated by the (sample) IPA gradient $\nabla J(\theta)$. This requires that $\nabla J(\theta)$ be an unbiased statistical estimator of $\nabla\zeta(\theta)$, namely that $E\big[\nabla J(\theta)\big] = \nabla\zeta(\theta)$.

IPA has the appealing property that its sample gradients often require low-complexity algorithms, not only for simple systems but also for networks of queues. However, soon after its inception it was discovered that IPA typically yields statistically-biased gradients for all but the simplest of systems [11], and this hindered its development and cast doubt about its eventual use in applications. Although various ways to circumvent this problem have been pursued, they typically resulted in highly-complicated estimators and hence deemed impractical (see [11], [3] and references therein). The search for low-complexity unbiased IPA gradients has not yet had an adequate solution suitable for a large class of practical optimization and control problems.

School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, nawaf@gatech.edu, wjhsong@ece.gatech.edu, sudha@ece.gatech.edu, ywardi@ece.gatech.edu.

Recently it has been suggested that the IPA gradients need not be unbiased in order for an algorithm to converge to a minimum value, but certain bounds on the bias would be sufficient. Thus, if an unbiased but complex sensitivity estimate can be replaced by a significantly – simpler IPA gradient with a bounded bias, then optimization or control algorithms could use it on practical problems. Experimental evidence was observed in [4], [5], [19], and theoretical justifications are currently under investigation. This observation is calling for a new approach to IPA by shifting its focus from unbiased gradient estimators to low-complexity estimators with bounded bias.

The purpose of this paper is to demonstrate this point on the problem of regulating throughput performance of multicore processors. For example, the need arises in multimedia applications where a fixed frame rate must be maintained to avoid choppy video or audio. Another application is in hard or soft real-time systems where constant throughput processors enable task and thread schedulers to effectively reason about the consequences of scheduling decisions and thereby provide tight performance bounds.

However, there are several challenges precluding predictable throughput behavior in multicore processors. In general, the degree of concurrency in an application instruction stream, as measured by the instructions per cycle (IPC), is time-varying and most often data-dependent. Furthermore, instructions from multiple cores interfere in the caches, the on-chip network, and memory controller queues adding variability to instruction execution. This variability is amplified in asymmetric multicore processors where different types of cores exhibiting different degrees of instruction level parallelism and throughput capabilities are integrated on chip.

Our starting point is the control law that was proposed for power regulation in multicore systems [1]. The considered problem was to control the dissipated power by the clock frequency, so as to have it track a given reference value. In the feedback system shown in Figure 1, the reference power is represented by the input r, the power dissipated is the output y, and the input to the plant, u, is the clock frequency. Reference [13] proposes a proportional control law and carries out an analysis of stability margins and tracking-convergence rates, under the assumption of a linear, constant-gain plant. Reference [1] derives for the plant a detailed, accurate system-model based on physical principles, and proposes an integral controller with an adaptive gain.

The purpose of adjusting the gain continually is to ensure a wide stability range and a high convergence rate of the control algorithm, thereby having the control law adjust well to frequent changes in the program workload.

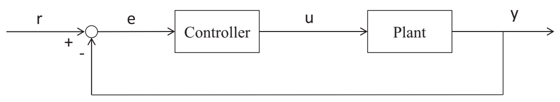


Fig. 1. Closed-loop system.

The gain is computed, in real time, in the following way. The plant is modeled as a nonlinear, (discrete) time-varying, memoryless system having the functional form $y_n = g_n(u_{n-1})$, the controller's transfer function is $G_c(z) = K_n \frac{z^{-1}}{1-z^{-1}}$, and the gain K_n is defined by $K_n = \left(g_n^{'}(u_{n-1})\right)^{-1}$, with "prime" denoting derivative with respect to u. We mention that any implementation of this control law requires the computation of the derivative term $g_n^{'}(u_{n-1})$ in real time, and we showed that this was possible for the considered problem.

Our objective in this paper is to apply a similar control law to regulate the throughput performance, but the problem is that the derivative of the frequency – throughput functional relation must be computable in real time. However this relation is based on a stochastic-DEDS model lacking closed-form expression for the performance, and hence the derivative term cannot be computed analytically, let alone in real time. For this reason we propose to use IPA for estimating the derivative, and despite its bias we are confident that the robustness of the control law, argued for in [1], will yield good convergence results.

Section II describes the computer-systems setting for our problem and surveys the main existing approaches to throughput-performance regulation. Section III describes our IPA approach to the problem, Section IV presents simulation results, and Section V concludes the paper.

II. Problem Setting and Established Approaches to Throughput Regulation

The target application domain is that of asymmetric multicore processors [12], [17], [10]. Cores can be classified as in-order (IO) cores where instructions are executed and retired in the order they are issued, and out-of-order (OOO) cores which employ aggressive pipelining and speculation to issue and execute instructions out of order. In this paper we consider OOO cores which tend to have higher throughput than IO cores.

The throughput of OOO cores may have a high degree of variability due to their inherent parallelism, dependencies between instructions in a given thread, and variable delays in the cache hierarchy. While recent work has shown that throttling the frequency of the cores can be effective in reducing interference in the network and memory system, relatively little progress has been made on robust throughput stabilization for the cores. Consequently, in this paper we focus on control schemes that adjust the voltage-frequency of a core to maintain a stable throughput rate. In principle, the operating point can be selected to minimize interference between cores in the network and memory hierarchy, or to achieve a balance between instructions throughput and power dissipation.

The architecture that we consider consists of a separate control loop for each core instead of having a single control applied to all of the cores in the processor. This clearly is advantageous since concurrent core-workloads may have different characteristics and throughput requirements. This implies that each core is in a separate voltage island, which is not uncommon; for example, the Intel 48 core single chip cloud computer has 8 voltage domains and 28 frequency domains [2].

A key issue in the design of our control law is to have it be applicable to a broad range of core types, and be independent of the specific application programs. In this case the design would be based not on extensive off-line analysis, but rather on a fundamental frequency-throughput relationship that is experienced across all application and core combinations. Such a general approach may eventually render control an integral part of a multicore design across a wide range of applications and core types. The rest of this section surveys the main existing approaches for this control problem.

Throughput stabilization has been proposed as a means to improve the predictability of real-time embedded systems. In this setting, a-priori guarantees on task completion times are required prior to deployment. Traditionally, computation of the Worst Case Execution Time (WCET) of a task is used to ensure predictability [20]. The significant drawback of this approach is that the WCET bounds are conservative – peak performance is significantly reduced while in practice these bounds may be rarely approached. Consequently, the use of WCET analysis has generally been limited to the application to in-order cores without caches. The successful application to high performance OOO cores is much more challenging. For example, as shown in [18], task execution-time uncertainty increases significantly in OOO processors due to the use of speculation in the control path compounded by variability in the memory hierarchy, and related works also underscore the difficulty of applying WCET-based methods [9], [21].

An early example of throughput stabilization is due to Zhu et. al. [22], who proposed an algorithm for hard real-time embedded systems using dynamic voltage-frequency scaling. Their approach is described in the context of energy-minimization. Tasks are executed on an in-order processor and are scheduled using the earliest deadline first (EDF) policy. To concurrently minimize energy, the authors proposed splitting each task into a fast subtask which is executed at the maximum frequency setting, and a subtask running at a lower frequency and therefore incurring a lower energy cost. An offline-tuned PID controller is proposed to tune the length of each subtask to ensure that the overall task meets its deadline. Recently, Suh et. al. [18] proposed stabilizing the throughput (measured in MIPS) of embedded OOO processors using feedback control. The proposed algorithm is a PID controller that adjusts the processor's voltage-frequency setting to track a MIPS setpoint. The parameters of the controller (values of the gain of the proportional, integral, and derivative components) are calculated offline using a task training set. The authors claim reasonable MIPS-tracking performance provided the workload does not vary significantly from the training set, limiting application to known workloads. Another approach to throughput stabilization for multithreaded processors was proposed by Lohn et.al [15]. The authors propose a statistical model of the relationship between the throughput of a thread and the time-slot in which the thread is scheduled to execute. The model is used to set the parameters of a proportional feedback controller that adjusts the time-slot allocation for a thread such that desired throughput for each thread is achieved.

The approach that we propose in the next section is based on an integral controller whose gain is adjusted on-line in a way that optimizes the tracking performance in a sense defined below. Since it computes the gain on-line, it is suitable to changing and unpredictable application workloads and programs. The gain's computation is based on the gradient (derivative) of the frequency-throughput relationship, which is derived from a fairly complicated queueing model and hence has no analytic (closed-form) formula. However, we use an IPA estimator which, though biased, is simple to compute and has a bounded error that yields good convergence results.

We close this section by mentioning that real-time scheduling regulation and control problems have been considered in the setting of soft real-time systems and web-server applications; see [6], [16] and references therein. The control laws proposed in these references are implemented at the software and operating-system levels; in contrast, our approach is aimed at a hardware-level controller that adjusts the processor frequency to regulate a throughput rate supplied by the scheduler. Consequently the proposed control laws for the hardware and software problems are different and could be complementary for eventual applications.

III. Control Law for Throughput Regulation

Consider the control system shown in Figure 1, where the output signal y denotes the instruction throughput, the reference input r is the target throughput, and the control variable u is the clock frequency, henceforth also denoted by ϕ. The system is assumed to evolve in discrete time, and hence we will use the notation y_n and $u_n = \phi_n$ for the instruction throughput and clock frequency, respectively, at time n.

The purpose of the feedback law is to achieve asymptotic tracking of a given reference value r by the output signal y_n, $n = 1, \ldots$. Suppose that the plant is a nonlinear, time-varying, memoryless system represented by the functional relation $y_n = g_n(\phi_n)$, where $g_n : R \rightarrow R$ is a function that depends on time n (this assumption is justified in the sequel). The error signal, e_n, is defined by the difference term $e_n := r - y_n$. In order to achieve tracking we use an integral control, and hence the controller is defined by the following relation,

$$\phi_n \;=\; \phi_{n-1} + K_n e_{n-1}, \qquad (1)$$

where $K_n > 0$ is its time-dependent gain.

This paper concerns multicore computer systems where each core is controlled separately, and hence the system shown in Figure 1 pertains to an instruction-throughput regulator at each core individually. Accordingly, for a particular core, y_n represents the average throughput over a given number of instructions, say M, and r is the throughput-reference value that is to be tracked. The control variable ϕ_n is the core-clock frequency, and the relations between the error signal and the control signal are given by Equation (1). The plant in Figure 1 represents the functional relationship between the frequency ϕ_n and the throughput y_n during a period defined by M consecutive instructions. The challenge that we are facing is that the plant characteristics, defined via the function g_n, are changing in unpredictable ways and there are no closed-form functional expressions for them.

Reference [1] considered a similar problem where the output is the power dissipated in the core which, similarly to this paper, is controlled by the clock frequency. Thus, y_n represents the throughput during the nth observation period. In this case, the frequency-power relationship is given by an explicit equation, $y_n = g_n(\phi_n)$, where the function g_n was derived from basic physical principles. The controller's gain K_n was defined as

$$K_n \;=\; \frac{1}{g_n'(\phi_{n-1})}, \qquad (2)$$

where "prime" denotes derivative with respect to ϕ. Moreover, this gain was shown to be computable in real time, and hence could be used in the control system.

Asymptotic tracking of such systems was proved, in an abstract setting, under the assumption that the plant functions $g_n(\phi)$ are convex. The results include convergence rate, error analysis, and tracking robustness with respect to estimation errors of $g_n'(\phi_{n-1})$ and time-variability of the plant. Specifically, if the relative estimation error of $g_n'(\phi_{n-1})$ is bounded by any number $\alpha < 1$, and if $|g_{n-1}(\phi_{n-1}) - g_n(\phi_{n-1})| < \epsilon$ for a given $\epsilon > 0$, then the asymptotic tracking error is in the order of ϵ. As a special case, if the plant is time invariant and $g := g_n$ is known, then tracking is achieved despite relative error of under an upper bound that is less than 100%.

The situation in this paper is different principally in the fact that we have no analytic form for the function $g_n(\phi)$. Instead, this function is defined as the measured instruction throughput over M consecutive instructions (for a given M), and its derivative $g_n'(\phi)$ is computed by IPA from measurements taken in real time.[1] The IPA derivative, described below, is certainly biased, but we will argue that the relative

[1]g_n depends on the load-program and can be viewed as a random function. However, since we are concerned with control and IPA, we focus on its realizations along sample paths without having to specify their underlying probability laws. Furthermore, a large M reduces the dynamic effects of $g_{n-1}(\phi)$ on $g_n(\phi)$, and this justifies the view of the system as static.

bias is small, far-less than 100%, and we will show that this will not impede tracking in light of the aforementioned robustness result concerning the relative error. The variability of the plant, measured by $|g_n(\phi_{n-1}) - g_{n-1}(\phi_{n-1})|$, cannot be predicted, and any tracking algorithm would have to contend with the time-varying feature of the plant. It can be controlled to some degree by the choice of M, the number of instructions underscoring the plant function, which balances precision versus temporal properties of the control system. The convexity assumption, made in [1], cannot be ascertained, but the results in [1] regarding tracking also hold true when the plant functions $g_n(\phi)$ are concave. If these functions are neither convex nor concave then the closed-loop system may be unstable, but this problem can be practically overcome by imposing an upper bound on variations in the control variable ϕ in each iteration. In our case this was not necessary, and extensive simulation studies exhibited concavity of these functions.

We next turn to describe the plant model and derive its IPA derivative. Consider a sequence of instructions, I_i, $i = 1, 2, \ldots$, that have to be executed by a core. When an instruction I_i arrives it is directed to a *reservation station*, where it is stored until all of its operands become available. At the same time an entry is made for it in the *reorder buffer*, which guarantees that the instructions are dequeued (depart, or committed) in the order of their arrivals. The reorder buffer is called the queue, and the arrival time of the instruction to it is called the enqueue time. Once all of the operands of the instruction become available, it is issued to an *execution unit* for processing. Following completion of execution it remains stored until the instruction that had arrived before it, I_{i-1}, is dequeued, at which time I_i can depart (dequeued) from the system as well. The variables computed by I_i can become available as operands to other instructions once I_i is executed, and not when it is dequeued (which may happen later). This sequence of events describes architectures that allow for *out-of-order execution* while maintaining the order of departure of instructions (dequeueing) according to their arrival (enqueueing) order. This principle is described in [8] and is currently implemented in many core architectures; see, e.g., [7].

Let a_i denote the arrival time (enqueue time) of I_i at the reorder buffer, and let α_i denote the time it is directed to its designated execution unit once all of its operands become available. We assume that the execution units have enough buffer to start execution of instructions without queueing delays. Let δ_i denote the completion time of execution of I_i, and at that time, all the variables computed by it become available as operands to other instructions. The dequeueing time of I_i from the system is denoted by d_i.

All of these epochs can be expressed in terms of the clock cycle time, denoted by θ, as follows. Let $\ell(i)$ denote the clock-cycle count of the enqueue time a_i, and hence

$$a_i = \ell(i)\theta. \tag{3}$$

Furthermore, let $k(i)$ denote the index of the instruction computing the last operand required for execution of I_i; then,

$$\alpha_i \;=\; \max\big\{a_i, \delta_{k(i)}\big\} + \theta. \tag{4}$$

Next, consider the execution (processing) time of instruction I_i at its execution unit. We call instructions that are not memory fetches *synchronous*, and their execution times can be approximated well by $n(i)\theta$, where $n(i)$ is an integer constant, typically under 10, that depends on the execution unit where the instruction is processed. For memory accesses to the cache, we have a similar formula where the range of $n(i)$ depends on the level of cache ($L1$, $L2$, or $L3$), and is typically under 100. For memory fetches from other storage devices such as RAM, the major part of the latency can be approximated by a term T_{mem}, that typically is in the order of hundreds of clock cycles or larger. We assume that this term does not depend on θ because such memory systems use a different clock than the one used in the core.[2] Thus, the following is an approximate formula for δ_i:

$$\delta_i \;=\; \begin{cases} \alpha_i + n(i)\theta, & \text{synchronous instruction or cache} \\ & \text{memory fetch} \\ \alpha_i + T_{mem}, & \text{other memory fetches.} \end{cases} \tag{5}$$

Finally, the dequeueing time of I_i is given by the following formula,

$$d_i \;=\; \max\big\{\delta_i + \theta, d_{i-1}\big\} + \theta. \tag{6}$$

Let $y := M/d_M$ denote the instruction throughput for a given integer $M > 0$. We note that in the control law described in the sequel the throughput will be observed from the system rather than computed by Equations (3) - (6), and the purpose of these equations is to compute the IPA derivative $y^{'}(\theta)$ that will play a role in the control algorithm via Equation (2).

To compute the IPA derivative $y^{'}(\theta)$ we first present a recursive algorithm for the terms $d_i^{'}(\theta)$, $i = 1, \ldots, M$. To this end we define the following two quantities, $\nu(i)$ and $m(i)$, as follows:

$$\nu(i) \;:=\; \begin{cases} 0, & \text{if } I_i \text{ is a memory fetch that is not} \\ & \text{from cache} \\ n(i), & \text{otherwise,} \end{cases}$$

and

$$m(i) \;:=\; \max\big\{m \le i \;:\; I_m \text{ did not stall following} \\ \text{its execution}\big\}.$$

The following proposition yields the terms $d_i^{'}(\theta)$. Its proof follows from Equations (3) - (6) after some algebra, and will be omitted.

[2]This is an approximation. T_{mem} has components that depend on θ but these are sublinear and hard to model. However, as we shall see, our controller works well with this approximation.

Proposition 1: The following Equations, (7) and (8), are in force for all $i = 1, \dots, M$.

$$\alpha_i^{'}(\theta) = \begin{cases} \alpha_{k(i)}^{'}(\theta) + \nu(k(i)) + 1, & \text{if } I_i \text{ stalls upon arrival} \\ \ell(i) + 1, & \text{if } I_i \text{ does not stall upon arrival.} \end{cases} \quad (7)$$

and

$$d_i^{'}(\theta) = \alpha_{m(i)}^{'}(\theta) + \nu(m(i)) + i - m(i) + 2. \quad (8)$$

$\square$

Now the IPA derivative $y^{'}(\theta)$ is given as

$$y^{'}(\phi) = \frac{1}{M}\Big(\frac{y}{\phi}\Big)^2 d_M^{'}(\theta). \quad (9)$$

The control law that we use is based on Equations (1) in the following manner. The duration of an application program is divided into a sequence of observation periods consisting each of a given number of M instructions. During the nth observation period the control parameter, namely the clock frequency ϕ_n, is fixed, the throughput y_n is measured, and its sample derivative $y_n^{'}(\phi_n)$ is computed, online, via (8) and (9). At the end of the period the error $e_n := r - y_n$ is computed, the gain K_{n+1} is computed by the formula $K_{n+1} := 1/y_n^{'}(\phi_n)$, and the clock frequency is updated via the equation $\phi_{n+1} = \phi_n + K_{n+1}e_n$. Notice that this is the integral controller described by Equations (1) and (2), with $g_n^{'}(\phi_n)$ replaced by the term $y_{n-1}^{'}(\phi_{n-1})$.

IV. Simulation Results

The IPA controller, derived in the last section, is tested and evaluated on an example using a detailed x86 microprocessor simulator [14], Zesto. Figure 2 illustrates the functional data flow within the core microarchitecture, and we consider a system consisting of four cores connected by a ring network and sharing an $L2$ cache. Our evaluations are based on the SPEC2006 benchmark suite executing in multi-programmed mode; each core is assigned a distinct application.

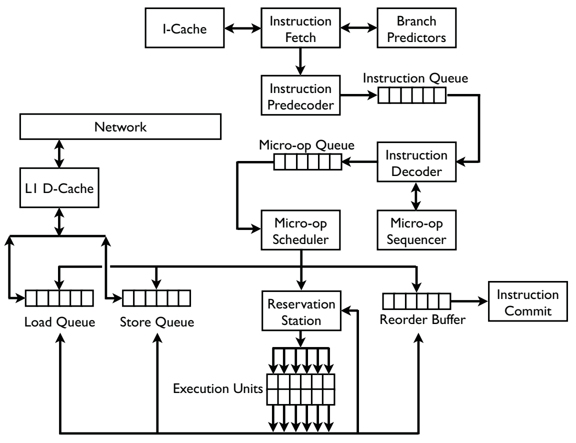


Fig. 2. Functional data flow in the out-of-order execution core.

Each core model implements the IPA-based throughput controller, defined via Equations (7)-(9). The voltage and frequency were adjusted approximately every 100,000 instructions to regulate the throughput. The period of 100,000 instructions was empirically chosen to be long enough to mask local, inconsequential high frequency variations in throughput yet long enough to be effective in tracking aggregate throughput. This period is refined dynamically to ensure that there are no instruction dependencies that exist from one interval to the next, and hence the exact sampling interval (observation period) may vary by a few tens of instructions.

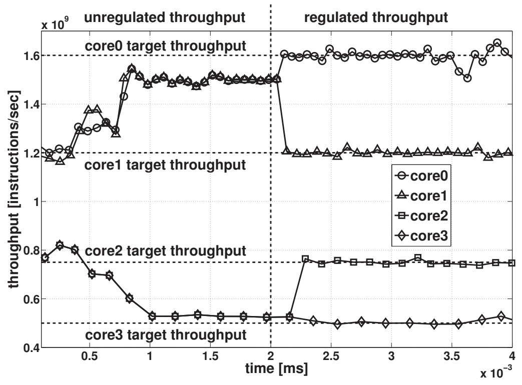
The subject of our simulation experiments is a four-core system executing two SPEC2006 benchmarks, *milc* and *GemsFDTD*, that can typically achieve two distinct levels of instruction throughput. Core0 and core1 executed the *milc* benchmark with target throughput of 1.6×10^9 and 1.2×10^9 instructions per second, respectively, and core2 and core3 executed the *GemsFDTD* benchmark with target throughput of 0.8×10^9 and 0.5×10^9 instructions per second, respectively. In all experiments, throughput remains unregulated for the first 2ms during which time each core executes at a constant frequency of 3.0GHz. At time $t = 2ms$ each core was assigned its target throughput, the controls were activated (the loops were closed), and the core throughput was regulated. The comparison was made against small fixed gain ($K_n = 0.5$) and large fixed gain ($K_n = 5.0$) controllers.

In Figure 3, we observe that the adaptive-gain IPA controller regulates the throughput and achieves tracking quite rapidly. In contrast, the small fixed-gain controller is sluggish in regulating the throughput especially with the lower throughput benchmarks, while the large-gain controller overshoots especially with the high throughput benchmark.

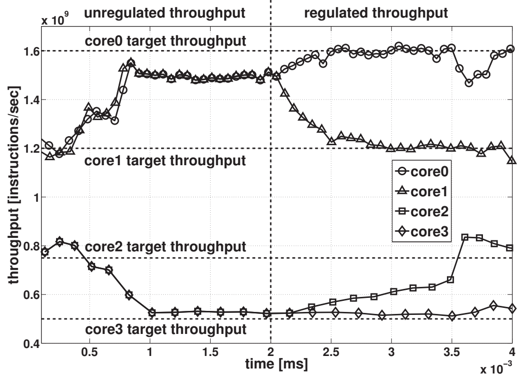
We conclude this section with comments about the relative bias of the IPA derivative estimators. Generally biasedness of IPA is associated with discontinuities of the sample performance functions [11], [3], and in the setting of this paper, these tend to arise primarily from memory-fetch instructions. To gauge the relative bias of the IPA derivatives, we compared finite-difference terms, obtained from simulations with a common seed at various values of the variable parameter, to their approximations that are derived from IPA via linear interpolation. Extensive simulations on 15 programs from the SPEC2006 benchmark suite yielded relative errors (between the two terms) of 30% in one case, 21% in another case, and under 6% in all other cases. As mentioned earlier, we expect the control algorithm to work well under such error conditions, and this was verified by simulation experiments.

V. Conclusions

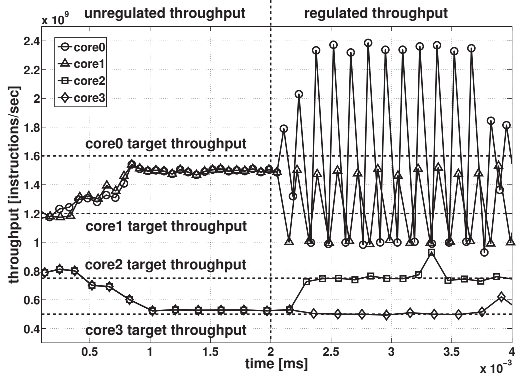
This paper presents an online controller for regulating the throughput of individual cores in a multicore processor using dynamic voltage-frequency scaling. The proposed control law is comprised of an integral controller whose gain is adjusted online based on the derivative of the frequency-throughput relationship. This derivative is estimated by IPA. The performance of the controller is demonstrated on a



(a) IPA-based dynamic gain controller



(b) Small fixed gain controller ($K_n = 0.5$)



(c) Large fixed gain controller ($K_n = 5.0$)

Fig. 3. Tracking analysis of throughput regulation.

cycle-level x86 multicore simulator executing SPEC2006 benchmarks, and rapid tracking and stable throughput were noted for each core.

REFERENCES

[1] N. Almoosa, W. Song, S. Yalamanchili, and Y. Wardi, "A Power Capping Controller for Multicore Processors," *Proc. ACC*, Montreal, Canada, June 27-29, 2012.

[2] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Ripen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Det Wijngaart, "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling", *J. of Solid-State Circuits*, Vol. 46, pp. 173-183, 2011.

[3] C.G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, Boston, Massachusetts, 1999.

[4] C.G. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C.G. Panayiotou, "Perturbation Analysis for On-Line Control and Optimization of Stochastic Fluid Models," *IEEE Transactions on Automatic Control*, Vol. AC-47, No. 8, pp. 1234-1248, 2002.

[5] C.G. Cassandras, "Stochastic Flow Systems: Modeling and Sensitivity Analysis," in *Stochastic Hybrid Systems: Recent Developments and Research Trends*, Eds. C.G. Cassandras and J. Lygeros, CRC Press, New York, New York, pp. 139-167, 2006.

[6] T. Cucinotta, L. Laopoli, and L. Marzario, "Stochastic feedback-based cpntrol of QoS in soft real-time systems", in *Proc. 43rd CDC*, Atlantis, Bahamas, December 2004.

[7] A. Fog, "The Microarchitecture of Intel, AMD, and VIA CPUs," *www.agner.org/optimize/microarchitecture.pdf*, 2012.

[8] J. Hennessey and D. Patterson, "Computer Architecture: A Quantitative Approach," *Morgan Kaufmann (pubs.)*, 2012.

[9] A. Hergenhan, and W. Rosenstiel, "Static timing analysis of embedded software on advanced processor architectures," *Design Automation and Test in Europe*, 2000.

[10] M. Hill and M. Marty, "Amdahls law in the multicore era," *IEEE Computer*, 41(7), 2008.

[11] Y.C. Ho and X.R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*, Kluwer Academic Publishers, Boston, Massachusetts, 1991.

[12] R. Kumar et al. "Heterogeneous chip multiprocessor," *IEEE Computer*, 38(11), 2005.

[13] C. Lefurgy, X. Wang, and M. Ware, "Power capping: A prelude to power shifting," *Cluster Computing,* vol. 11, no. 2, June 2008.

[14] G. H. Loh, S. Subramaniam, and X. Yuejian, "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *Proc. ISPASS*, Apr. 2009.

[15] F. Lohn, M. Pacher, and U. Brinkschulte, "A Generalized Model to Control the Throughput in a Processor for Real-Time Applications," *14th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, March 2011.

[16] C. Lu, Y. Lu, T.F. Abdelzaher, J.A. Stankovic, and S.H. Son, "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers", *IEEE Trans. Parallel and Distributed Systems*, Vol. 17, pp. 1014-1027, 2006.

[17] T.Y. Morad, U.C. Weiser, A. Kolodnyt, M. Valero, and E. Ayguade, "Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors," *Computer Architecture Letters*, Vol. 5, pp. 14-17, 2006.

[18] J. Suh, and M. Dubois, "Dynamic MIPS rate stabilization in out-of-order processors," *SIGARCH Computer Architecture News*, vol. 37, no. 3, June 2009.

[19] Y. Wardi and G.F. Riley, "Infinitesimal Perturbation Analysis in Networks of Stochastic Flow Models: General Framework and Case Study of Tandem Networks with Flow Control," *Discrete-Event Dynamic Systems: Theory and Applications,* Vol. 20, No. 2, pp. 275-305, 2010.

[20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, May 2008.

[21] T. Yudong and V. J. Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," *Design, Automation and Test in Europe*, 2004.

[22] Y. Zhu, and F. Mueller, "Exploiting synchronous and asynchronous DVS for feedback EDF scheduling on an embedded platform," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 1, 2007.