Structured Singular Value Control for Modular Resource Management in Multilayer Computers

Raghavendra Pradyumna Pothukuchi, Sweta Yamini Pothukuchi, Petros G. Voulgaris,[1] and Josep Torrellas
Department of Computer Science [1] Department of Aerospace Engineering
{pothuku2, seetham2, voulgari, torrella}@illinois.edu
University of Illinois at Urbana-Champaign, Urbana, Illinois, USA 61801

***Abstract*— Computer systems are operating in environments where applications are rapidly diversifying while resources like energy and storage are becoming severely limited. These environments demand that computers dynamically manage their resources efficiently to deliver the best performance and meet many goals. An important challenge in designing computer resource management systems is that computers are structured in multiple modular layers, such as hardware, operating system, and network. Each layer is complex and designed independently without full knowledge of the other layers. Therefore, computers must have modular resource controllers for each layer that are robust to modeling limitations and the uncertainty of influence from other layers. Existing designs either rely heavily on ad hoc heuristics or lack modularity. We present a design with multiple Structured Singular Value (SSV) controllers from robust control theory for systematic and efficient computer management. On a challenging computer, we build a two-layer SSV control system that significantly outperforms state-of-the-art heuristics.**

I. Introduction

Modern computer systems must efficiently manage several resources like energy or storage to carefully control measures like temperature and deliver the best Quality of Service (QoS) or throughput. Computers have controllers for dynamic resource management to achieve these goals [1], [2], [3], [4].

An important challenge in designing computer resource controllers is that computers have many functional layers. For example, as shown in Figure 1, the hardware layer consists of the processor and physical system circuitry. There is an Operating System (OS) that runs on the hardware to provide abstraction and schedule applications in the higher layer. The Application layer has programs that the user wants to run. There can be several software layers on top of these layers.

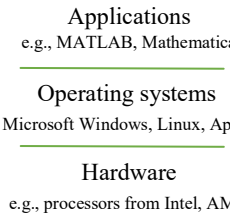


Fig. 1: Multilayer organization of computer systems.

Each layer is a complex subsystem designed independently by expert teams. It has its own resources, tunable parameters and partial information about the system status. Each layer uses the information it has to manage resources from its perspective, and the overall system efficiency is a function of all the layers.

Since each layer is is independently designed, it is necessary that there is a modular resource controller in each layer running simultaneously and the different controllers coordinate for overall efficiency. Centralized control can coordinate across layers but is difficult to develop because designers must understand the inner details of all the layers. This may be infeasible when the layers come from different companies as in Figure 1. Such a design also scales poorly as the number of layers and their complexity increases. The controller must be re-designed even if a single layer in the system changes [5], [6], [7]. At the same time, fully decoupled control misses the interaction between the layers and can be greatly suboptimal [6].

Since computers are complex, it is also important that they are managed with systematic methods like control theory. Unfortunately, due to the lack of effective systematic methods, most computer controllers are based on ad hoc heuristics (e.g. [1], [2], [6], [7]). Many studies demonstrate the design difficulties and unanticipated runtime failures with such heuristics when encountering situations slightly different from the training set [8], [9].

Recently, we presented an approach to design modular and coordinated multilayer controllers for computers using robust control theory, and demonstrated its effectiveness on a prototype system [10]. In this paper, we present the problem formulation, and the controller design and synthesis for the prototype computer from a control systems perspective.

Our prototype is a state-of-the-art computer with a processor developed by Samsung that runs a Linux based OS. We build a novel two-layer control system and show that it is very effective. To design our controllers, we consider Robust Control Theory, which focuses on uncertain environments, and pick the popular Structured Singular Value (SSV) controllers [11], [12]. The key idea is that modeling limitations and inter-layer interaction is considered uncertainty when designing modular controllers for a layer in a computer. Using SSV controllers helps us to guarantee robust performance of these modular controllers. Further, each controller reads signals from other layers to coordinate better

* This work was supported by NSF under grant CCF 16-49432.

under uncertainty. This design reduces the Energy × Delay[1] of a diverse set of applications by an average of 50% beyond what advanced heuristic-based coordinated controllers attain.

This is the first work to describe the use of Multiple Input Multiple Output (MIMO) SSV control for computer systems, and is an important new application of robust control theory. This work also opens opportunities for advanced solutions to the general computer resource management problem by presenting several design challenges that we overcome.

In this paper, Section 2 presents the related work; Section 3 describes our system; Section 4 describes the challenges in applying control theory for computers; Section 5 presents the controller and design decisions; and Section 6 describes our evaluation.

II. Related Work

Many works use heuristics to control computers (e.g., [1], [2], [7]). Among control-theoretic designs, most use Single Input Single Output (SISO) PID controllers [3], [13], [14], [15] while some use collections of SISO controllers [16], [17]. SISO designs are too limited to control even a single layer that has many goals. Decoupled SISO controllers cannot manage the interaction between the goals in tightly coupled systems like computers [9], [18]. Some designs employ heuristics to manage controller interaction [16], but this defeats the purpose of using control-theoretic methods. Some works use Multiple Input and Single Output (MISO) Model Predictive Control (MPC) [19], [20].

Researchers also proposed the use of MIMO designs with Linear Quadratic (LQG/LQR) or MPC controllers [9], [15], [18], [21]. Each computer layer has a MIMO system, and MIMO controllers are the most appropriate for computers. However, existing designs are intended for centralized use, and do not prioritize robustness to the uncertainty in multi-controller environments.

Our work focuses on the design of modular and coordinated formal controllers for computers. We describe how we can use modular Structured Singular Value (SSV) controllers in each layer of a challenging computer system, and achieve higher efficiency over the state of the art.

III. Prototype Computer System

Our prototype system is an ODROID XU3 computer board [22], where the processor is built by Samsung and the Operating System is based on Linux. The processor is a Samsung Exynos 5422 eight-core processor built using ARM big.LITTLE technology [23]. Figure 2 shows a picture of our experimental platform.

Figure 3 shows the hardware and operating system layers, with the inputs our controllers actuate on, and the outputs that our controllers sense. The hardware layer includes the processor, which is made of 8 processing units or cores. Four of these cores are high performance, high power units (called

[1]Delay here refers to the time taken by an application to complete its work. Energy×Delay is the product of the application's energy consumption and running time. It is a common metric to judge a computer's efficiency. Lower is better.

Fig. 2: The Odroid XU3 used for our prototype.

big cores) and are organized as one cluster, while the others are low performance, low power units (called little cores) that form another cluster. The processor runs the Ubuntu 15.04 Operating System based on Linux. The sampling interval in both layers is 0.5 s. On this system, applications can create multiple software tasks (called threads) that execute in parallel to speedup the total application duration. Hence, there can be many applications and many threads running simultaneously. We refer to each schedulable entity (an application or an application thread) as a task.

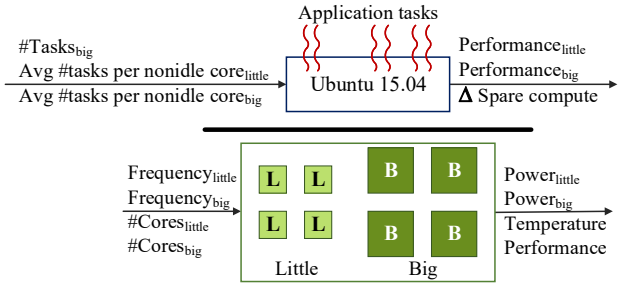


Fig. 3: Odroid XU3's hardware and operating system layers with the input and output signals we consider.

A. Inputs and Outputs in Each Layer

The inputs and outputs in each layer are shown in Figure 3. In the hardware layer, there are four inputs: operating frequency of the little and big clusters, and the number of active little and big cores. The number of active cores in each cluster can vary from 1 to 4. The big cluster frequency can vary from 0.2 to 2.0 GHz, and the little cluster frequency from 0.2 to 1.4 GHz, both in steps of 0.1 GHz. Changing the number of cores has nearly twice the overhead than changing the frequency.

We consider four controlled outputs for the hardware layer: the power of little and big clusters, temperature, and the application performance measured in billions of instructions committed per second (BIPS). Among these outputs, the power of both clusters and temperature are critical for system integrity.

The software or OS layer assigns the application tasks cores. Ignoring differences between tasks, one decision is to partition the tasks between the big and little clusters. The other is to assign the tasks in a cluster to only some cores, possibly leaving some other cores idle[2] so that the hardware controller can power down the idle cores. Therefore, we consider three inputs in this layer: the number of tasks assigned to the big cluster (leaving the rest for the little cluster), the average

[2]A core is non-idle if it is running at least one task.

number of tasks running on each non-idle big core, and the average number of tasks running on each non-idle little core.

There are three controlled outputs in the OS layer: performance of the little-cluster tasks (in BIPS), performance of the big-cluster tasks (in BIPS) and the difference in Spare Compute Capacity (SC) between the big and little clusters. At a high level, the higher the difference in SC is, the more tasks the controller will move from the little to the big cluster. We define a cluster's SC [10] as:

$$SC = \#\text{idle\_cores\_on} - (\#\text{tasks} - \#\text{cores\_on}) \quad (1)$$

B. Control Objectives

There are two types of goals in computer resource management: tracking a set of output references, and optimizing a combination of the measured outputs. The latter is more common, where designers want to minimize metrics like Energy×Delay of the whole application. A dynamic measure of this metric is obtained by considering Energy = Power×Delay, and Delay = $\frac{\text{Application instructions}}{\text{Application performance in BIPS}}$. Since the number of application instructions is fixed, Energy×Delay is inversely proportional to $\frac{(\text{Application performance in BIPS})^2}{Power}$, in which both quantities can be measured dynamically.

The controller in each layer (i.e., the hardware one (HW) and OS one), dynamically maximizes the same metric, $\frac{(\text{Application performance in BIPS})^2}{Power}$, where both performance and power are functions of time T. The hardware controller has additional constraints to ensure the physical integrity of the system, namely, keeping the power of the little cluster, the power of the big cluster, and the temperature below certain limits. If we call U_{HW} and U_{OS} the set of values taken by the inputs in the HW layer ($u_{HW}(T)$) and OS layer ($u_{OS}(T)$) repsectively, the goals of the controllers are:

$$\begin{gathered}\underset{u_{OS}(T)\in U_{OS}}{\text{maximize}} \ \frac{Performance^2(T)}{Power(T)} \\ \underset{u_{HW}(T)\in U_{HW}}{\text{maximize}} \ \frac{Performance^2(T)}{Power(T)} \\ Power_{little}(T) < Power_{little}^{limit} \\ Power_{big}(T) < Power_{big}^{limit} \\ Temperature(T) < Temperature^{limit}\end{gathered} \quad (2)$$

The controllers must meet these goals using imperfect models. In our board, the limit powers of the little cluster and big cluster, and the limit temperature are 0.33 W, 3.3 W, and 79 °C, respectively. Moreover, production systems make decisions at the order of 10 ms (0.01 s) [3] although the smallest sampling interval feasible in our system is 0.5 s. So, we require our controllers to compute decisions in a few ms to meet the production system requirements.

IV. Challenges in Applying Control Theory

There are several challenges in developing control-theoretic solutions for the computer resource management problem described above:

1) Fundamentally, it is infeasible to obtain accurate models of each layer or inter-layer interaction except in some limited contexts (e.g., see [21]). The layers are too complex to model from first principles, and the applications are numerous, exhibiting diverse behavior.
2) The metrics to optimize (such as Energy×Delay) have non-convex and non-linear relationship with the inputs. These metrics do not conform to usual signal norms that are commonly used in control theory.
3) Computer inputs such as processor frequency are finite and discrete-valued, instead of having the continuous values usually assumed in control theory.
4) The controllers are invoked at a millisecond granularity and must complete their decisions even faster. The storage and computation resources for the controllers, particularly in the hardware, should be small (e.g., a few kilobytes for storage, and a few hundreds of arithmetic operations per invocation).
5) Control design should be supported mostly by standard tools with intuitive tuning processes, to allow mainstream adoption by computer designers.

Since modeling is a fundamental problem, we follow black box system identification [24] for modeling. Then, we design controllers from robust control theory [12] to deal with model limitations and unknown inter-layer interaction. We use SSV controllers in every layer. The properties of these controllers are suitable for computer management for three reasons.

First, SSV design natively targets uncertain environments (unlike LQG, for example). We formulate model limitations, inter-layer interactions, and input discretization as uncertainty. As this uncertainty can include Non-Linear Time-Varying (NLTV) phenomena, we consider all NLTV system dynamics as uncertainty. We can use well-established linear design procedures to obtain controllers that attain robust performance under such uncertainty.

Second, SSV controllers can read disturbance signals from other modules for improved control. In computer systems, we call them External Signals, and use them to pass information from one layer to the controller of another layer. For example, the OS controller passes the number of tasks currently running as an external signal to the HW controller.

Finally, the design and tuning of SSV controllers is extensively supported by standard tools [25], [26]. These controllers do not require online solvers to compute decisions (unlike MPC, for example), which is essential for fast decision-making. In the next section, we describe how we design our control system to optimize $\frac{Performance^2(T)}{Power(T)}$.

V. Multilayer SSV Computer Control System

We separate the goals of Equation 2 into two sub-goals to design our control system. The first is to make the outputs track a given reference robustly and optimally according to a conventional cost function. The second is to search for the best references that maximize $\frac{Performance^2(T)}{Power(T)}$ under constraints. For the HW layer, these sub-goals are:

$$\begin{gathered}\min_{u\in U_{HW}} \|\Gamma_{zw}\|_\infty \\ w : \begin{pmatrix} y_{HW\circ} \\ d \end{pmatrix} \xrightarrow{\Gamma_{zw}} z : \begin{pmatrix} W_p(y_{HW\circ} - y_{HW}) \\ W_u u_{HW} \end{pmatrix}\end{gathered} \quad (3a)$$

$$\max_{y_{HW\circ}(T)\in Y_{HW}} \frac{Performance^2(T)}{Power(T)} \text{with constraints} \quad (3b)$$

where u_{HW} represents the hardware inputs, and $y_{HW\circ}(T)$ is the reference for the hardware outputs y_{HW}. $\|\Gamma_{zw}\|_\infty$ is the induced H_∞ norm that captures robustness to bounded disturbance d, the performance requirements W_p and input weights W_u.

We use this formulation so that the first sub-goal is cast as a mixed sensitivity robust control problem that ensures optimal tracking according to the designer requirements under uncertainty. We could use a standard Structured Singular Value (SSV) controller to achieve this sub-goal.

The second sub-goal requires searching for the best references for the outputs to meet the actual goal. This search is in the space of output references and is simpler than searching through the input space. For e.g., to optimize $\frac{Performance^2(T)}{Power(T)}$, the search module can progressively increase performance targets by larger amounts than the increase in power targets, or decrease performance targets by smaller amounts than the decrease in power targets. As a result, implementing the search in hardware consumes lesser resources, which is important for us. We design an Optimizer module for this search.

The SSV controller ensures that the overall control is effective under the uncertainty of unmodeled intra-layer and inter-layer conditions. The Optimizer does not have to explicitly deal with uncertainty and instead, relies on the SSV controller to generate inputs suitably. Since the SSV controller converges fast, the overall optimization is also fast.

Figure 4 shows our proposed multilayer control architecture with an SSV controller and the Optimizer in each layer. The SSV controllers also read the inputs in other layers as measured disturbances ("external signals") for improved decisions. This is a modular architecture in which each controller can be designed independently. The design teams need to exchange only the interface information for external signals or bounds for commonly monitored outputs. Next, we describe model identification, the design of SSV controllers, and the design of Optimizers.

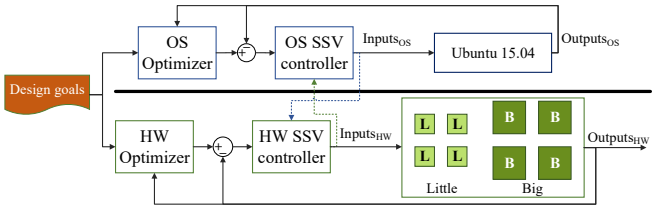


Fig. 4: Multilayer SSV controller system.

A. Black Box System Identification

Due to the complexity of computers, we find that empirical black-box model identification [24] combined with robust controller design is the best approach. Our identification experiments use two applications (swaptions and vips) from the PARSEC 2.1 application suite [27] and four applications (astar, perlbench, milc and namd) from the SPEC06 suite [28][3].

[3]These suites are standard to evaluate the performance of computers. From these suites, we pick applications for identification at random.

We use 2 tests with pseudorandom input sequences for each of the training benchmarks. The value of each input is chosen randomly and it is held unchanged for a duration randomly selected between 1 and 16 sampling intervals. The hardware model has 4 inputs, 3 disturbance inputs and 4 outputs; the OS model has 3 inputs, 4 disturbance inputs and 3 outputs.

We know that the output values can be related with prior values, and use a Box-Jenkins polynomial ($y(T) = \frac{B}{F}u(T) + \frac{C}{D}e(T)$) as the model structure. We get the coefficients of the model from the experimental data using MATLAB. We reduce the state dimension of the models to 15 and 26 using Hankel singular values before designing the controllers.

The models are nominally stable similar to the underlying system. A unique feature of the models is that the frequency response of the outputs is nearly flat. This is expected from the behavior of computers. For example, Figure 5 shows the magnitude of the frequency response (in dB) for the little cluster power (y1) from the external signals (u1 - u3) and the first two hardware inputs (u4 - u5). The frequency range of interest based on the sampling interval (0.5 s) is from 0.5 Hz to 10 Hz. Figure 5 also shows the confidence regions.

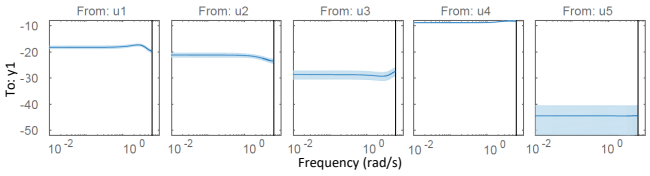


Fig. 5: Bode magnitude of the hardware output ($Power_{little}$).

B. Design and Synthesis of the SSV Controllers

We use the structure in Figure 6 to design the SSV controllers. $P_\circ$ is the identified nominal model. We consider two forms of uncertainty. One (Δ_{op}) is the output multiplicative uncertainty to account for intra-layer and inter-layer modeling limitations. This is bounded by W_{op}. Another (Δ_{nl}) is the additive uncertainty used to model input nonlinearity. The additive uncertainty setup lies inside a complex disk of radius 0.5 centered on the real axis at 0.5. We use W_p for the tracking error bounds of the outputs and W_u for the input weights.

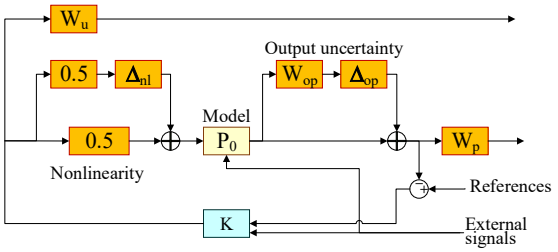


Fig. 6: Closed loop structure for each layer.

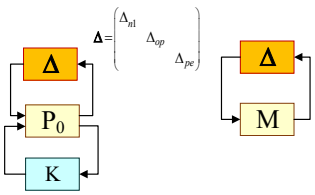


Fig. 7: LFT representation.

The structure in Figure 6 can be reorganized as a Linear Fractional Transformation (LFT) shown on the left side of Figure 7 by pulling out the uncertain elements. $\mathbf{\Delta}$ consists of block diagonal uncertain elements, including the fictitious Δ_{pe} used for enforcing robust performance (i.e. W_p and W_u). The right side of Figure 7 shows the nominal closed loop $M = F_l(P_\circ, K)$, i.e., the lower LFT of $P_\circ$ and K. The Structured Singular Value (SSV, μ, or $\|F_l(P_\circ, K)\|_\mu$) is defined as:

$$\mu_\Delta(M) = \frac{1}{min\{\|\Delta\| : det(I - M\Delta) = 0, \Delta \in \mathbf{\Delta}\}} \quad (4)$$

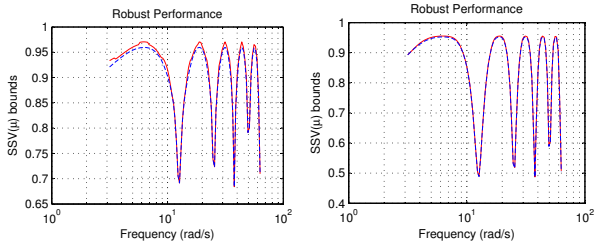
By the structured small gain theorem [11], the system is internally stable and meets performance if and only if $\sup_{w\in R} \mu_\Delta(M(j\omega)) \leq 1$. The controller K is obtained by solving $\min_{K-stab} \|F_l(P_\circ, K)\|_\mu$ using the DK iteration of MATLAB's Robust Control Toolbox [26].

The weights we use in our structure are given in Table I and are of the form $\frac{k(s+a)}{s+b}$. The weights to bound uncertainty (W_{op}) are set from model validation and confidence estimates on the identified model given by MATLAB. The OS layer has higher uncertainty as it is closer to applications, which are unpredictable. For the tracking error bounds of the outputs (W_p), power and temperature have tighter bounds as they are vital for system integrity. Our weights emphasize high frequency performance over constant reference tracking, as the common case is to track changing references to optimize a metric. The input weights (W_u) are set based on the relative overheads of changing the inputs (Section III). W_u for OS are more conservative as the OS is closer to application unpredictability.

TABLE I: Weight specification.

Weight	Hardware	OS
$\mathbf{W_{op}}$	$\frac{0.8(s+10)}{s+20}$ for all outputs	$\frac{(s+10)}{s+20}$ for all outputs
$\mathbf{W_p}$	$\frac{5(s+20)}{s+10}$ for $\text{Power}_{\text{little}}$, $\text{Power}_{\text{big}}$, and Temperature $\frac{2(s+20)}{s+10}$ for performance	$\frac{2(s+20)}{s+10}$ for all outputs
$\mathbf{W_u}$	0.5 for $\text{frequency}_{\text{little}}$ and $\text{frequency}_{\text{big}}$ 1 for $\#\text{cores}_{\text{little}}$ and $\#\text{cores}_{\text{big}}$	2 for all inputs

Figure 8 shows the SSV bounds for the closed loop system in each layer. The controllers provide robust stability and performance as $\max SSV(j\omega)$ is < 1.



(a) Hardware subsystem (b) OS subsystem

Fig. 8: Structured Singular Value (μ) bounds.

C. Designing the Optimizers

Algorithm 1 describes the hardware optimizer. It is based on the algorithm in [9], modified to support search constraints and additional outputs. In Algorithm 1, a small increase in a quantity is denoted by a single $+$ and an increase twice that amount is denoted by $++$. The same holds for a small reduction ($-$) and a larger reduction ($--$).

Algorithm 1: Operation of the Hardware Optimizer.

Input : Output and input measurements, output limits, convergence bounds ϵ, and restart probability δ
Output : output references

dir $\leftarrow$ Up, prevMetric $\leftarrow$ 0, stopSearch $\leftarrow$ False
initReferences()
// $Output_{agg}$ is output($\neq$Performance) most above limit
// $Output_{laz}$ is output most below its reference
// $Output_{lag}$ is output($\neq$Performance) most below limit
// $Output_{lead}$ is output ($\neq$Performance) closest to limit
Loop
  if *any output exceeds limits* **then**
    $Output_{agg\circ} \leftarrow Output^{limit}_{agg}-$, $Output_{laz\circ}-$
  else
    metric$\leftarrow \frac{Performance^2}{Power}$
    $\Delta metric \leftarrow abs(\frac{metric-prevMetric}{prevMetric})$
    if $\Delta metric \leq \epsilon$ AND *rand()* $> \delta$ **then**
      stopSearch $\leftarrow$ True
    else
      if *dir = Up* **then**
        if *metric>prevMetric* **then**
          Performance$_\circ++$, $Output_{lag\circ}+$
        else
          dir $\leftarrow$ Down
          Performance$_\circ-$, $Output_{lag\circ}--$
        end
      else
        if *metric>prevMetric* **then**
          Performance$_\circ-$, $Output_{lead\circ}--$
        else
          dir $\leftarrow$ Up
          Performance$_\circ+$, $Output_{lead\circ}++$
        end
      end
    end
  end
EndLoop

The optimizer provides increasingly better references for the outputs to maximize the metric $\frac{Performance^2}{Power}$. It first checks if any constraints are violated. If so, it reduces the reference of the output whose violation is most serious, and of the output that is most below its reference. The references are too high for these outputs.

When all constraints are met, it searches in one of the two possible directions: Up, by increasing performance and power or $Down$, by reducing both. Initially, the direction is Up, and it remains so until the metric stops improving. When

the metric no longer improves in one direction or if there is no room to continue, the direction is reversed.

In Up, it increases the reference for performance and $Output_{lag}$ that has the largest room below the limit. Performance reference is increased by a larger amount. In $Down$, it decreases the reference for performance and $Output_{lead}$ that has the largest room to decrease its reference. Performance reference is decreased by a smaller amount. The search does not cycle through the same points.

The search stops when the relative change in metric is small (below ϵ). Even after convergence, the algorithm can restart with a small probability ($\leq \delta$) because applications can suddenly change their characteristics.

The algorithm for the OS optimizer differs only slightly and is not shown. The OS Optimizer also has two search directions: Big-side (where the big cores contribute more to performance) and Little-side (where the little cores contribute more to performance). The search process finds the best operating point from the available search space.

VI. Evaluation

The implementation overheads for the SSV controller in each layer are given in Table II. We can keep the controller dimension small by reducing it using Hankel singular values. The number of operations include the number of 32-bit fixed-point additions and multiplications. We measured the power consumed for the computation on an ARM Little core. These values are small because the SSV controllers only need to perform matrix-vector calculations to generate decisions and do not need complex solvers. The overheads are low enough to be used in computers.

TABLE II: Implementation overheads of SSV controllers.

Parameter	HW SSV	OS SSV
Dimension	20	16
Required storage	2.6 KB	2.1 KB
Number of operations	≈700	≈600
Computation time	≈28μs	≈25μs
Power consumption	≈20-25mW	≈20-25mW

A. Overall Comparison with State-of-the-art

We compare our multilayer SSV design (called *Multilayer SSV*) with a heuristics-based control system (called *Heuristics*), representative of the state-of-the-art used in industry for our computer. We evaluate them in minimizing Energy×Delay. Figure 9 shows the Energy× Delay of the applications with *Multilayer SSV* and *Heuristics*. The bars from left to right correspond to SPEC applications, average of the SPEC applications (*SAv*), PARSEC applications, average of the PARSEC applications (*PAv*), and the average across all applications (*Avg*). For each application, the bars are normalized to *Heuristics*.

Multilayer SSV reduces Energy×Delay by 50%. This is due to systematic and efficient resource control. The execution times and energy consumption (not shown) are reduced by 38% and 20%, respectively. In *Multilayer SSV*, the costs and overheads are explicitly considered to design the controllers that perform robustly under uncertainty. *Heuristics* incorporates them implicitly, and has no stability or robustness properties. Hence, SSV controllers result in a substantial advancement over existing systems.

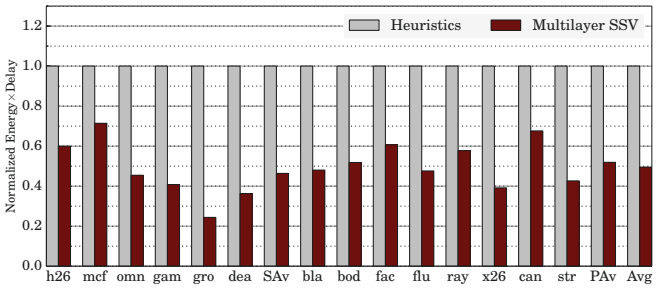
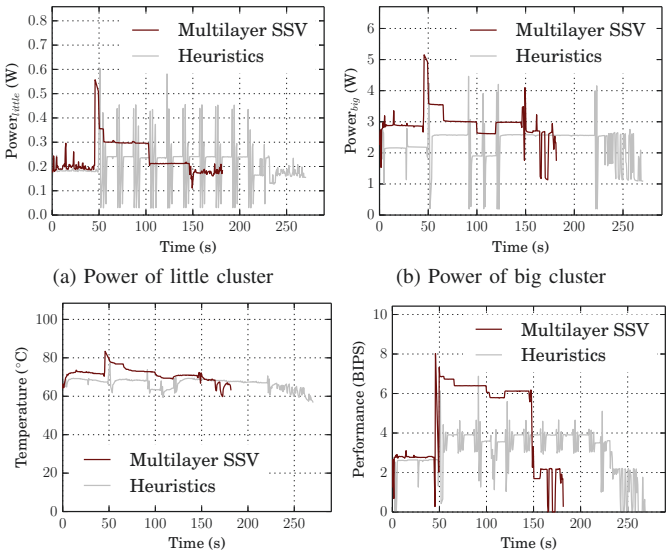


Fig. 9: Comparing Energy×Delay (lower is better).

B. Analysis of a Specific Case

We present how the two control systems differ by focusing on the blackscholes application (labeled *bla* in Figure 9). This application begins with a single task and later launches 8 parallel tasks. The work in the parallel phase does not have large variations. Finally, the parallel tasks complete their work and the application terminates. Figure 10 shows how the two control systems behave over time.



(a) Power of little cluster

(b) Power of big cluster

(c) Temperature

(d) Application performance

Fig. 10: Regulating blackscholes with the two control systems.

For *Heuristics*, there are many oscillations in the power of both clusters and performance. Each layer's controller measures the inputs from the other layer for improved coordination but this coordination is ad hoc. From the instant the application begins its parallel tasks at around 50s, *Heuristics* struggles to keep power and performance steady. The application takes 270 seconds to complete.

Multilayer SSV has a significantly smoother behavior. The outputs are kept within limits and the application has higher performance. Even when the application suddenly changes the number of parallel tasks (from 1 to 8 at 50s), the controllers quickly bring the outputs below the limits. In the parallel phase too, the search for the best references proceeds smoothly. The application completes in 180 seconds, much faster than with *Heuristics*.

C. Evaluating Heterogeneous Application Combinations

We evaluate four heterogeneous workloads. Each workload has a 4-task PARSEC application plus four copies of a single-task SPEC application. The workloads are: *blmc* (blackscholes+mcf), *stga* (streamcluster+gamess), *blst* (blackscholes+streamcluster), and *mcga* (mcf+gamess). The models were not trained under such conditions. Figure 11 shows the Energy×Delay with *Heuristics* and *Multilayer SSV*, normalized to *Heuristics*. The results are similar to those obtained earlier, with on average 47% lower Energy×Delay using *Multilayer SSV*. This demonstrates the robustness of *Multilayer SSV*.

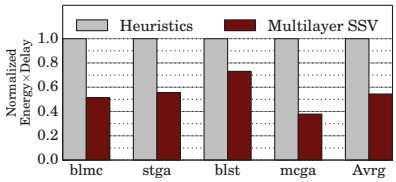


Fig. 11: Comparing Energy×Delay for heterogeneous loads.

VII. Future Work

It is not sufficient to use a single SSV controller in each layer because there are many heterogeneous components that constitute a layer. We plan to design a framework of several communicating SSV controllers to manage a heterogeneous layer, that communicates with the control framework of another layer for overall efficiency. Another limitation of our current design is that control decisions do not distinguish between each task in a heterogeneous workload. We know that system control can be more efficient by considering the distinct requirements of different tasks. We are working on augmenting our control system with this feature.

VIII. Conclusion

This paper presented a novel control system to attain high resource efficiency in computers. It is based on robust control theory, and provides modular coordinated control for modern multilayer computers. Our scheme considers inter-layer interactions as uncertainty, and relies on modular SSV controllers to be robust to this uncertainty. The controllers can be designed independently and are guaranteed to work in coordination. On a representative computer, our two-layer control system reduced the Energy×Delay of a set of programs by 50% on average beyond the state of the art.

This paper also described the difficulties of applying control theory to computers that our design overcomes. It is hoped that these insights will enable other work on building formal and practical controllers for computers.

References

[1] M. Broyles, C. J. Cain, T. Rosedahl, and G. J. Silva, "IBM EnergyScale for POWER8 Processor-Based Systems," IBM, Tech. Rep., Nov. 2015.

[2] S. Jahagirdar, V. George, I. Sodhi, and R. Wells, "Power Management of the Third Generation Intel Core Micro Architecture formerly Codenamed Ivy Bridge," in *Hot Chips*, Aug. 2012.

[3] E. Rotem, "Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency," Intel Developer Forum, Aug. 2015.

[4] Microsoft, "Processor power management in Windows 7 and Windows Server 2008 R2," https://msdn.microsoft.com/en-us/library/windows/hardware/dn613983(v=vs.85).aspx, 2012, Microsoft Developer Network.

[5] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2008.

[6] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile Application-aware Adaptation for Mobility," in *ACM Symposium on Operating Systems Principles*, Oct. 1997.

[7] H. Zhang and H. Hoffmann, "Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2016.

[8] "CPU throttling broken for Atom BayTrail CPUs under Windows 10," https://communities.intel.com/thread/78086, 2015, Intel Support Community.

[9] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," in *International Symposium on Computer Architecture*, Jun. 2016.

[10] R. P. Pothukuchi, S. Y. Pothukuchi, P. Voulgaris, and J. Torrellas, "Yukta: Multilayer Resource Controllers to Maximize Efficiency," in *International Symposium on Computer Architecture*, Jun. 2018.

[11] J. C. Doyle, J. E. Wall, and G. Stein, "Performance and Robustness Analysis for Structured Uncertainty," in *IEEE Conference on Decision and Control*, Dec. 1982.

[12] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

[13] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[14] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable Power Control for Many-core Architectures Running Multi-threaded Applications," in *International Symposium on Computer Architecture*, Jun. 2011.

[15] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[16] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era," in *Design Automation Conference*, Jun. 2013.

[17] A. Filieri, H. Hoffmann, and M. Maggio, "Automated Multi-objective Control for Self-adaptive Software Design," in *Joint Meeting on Foundations of Software Engineering*, Sep. 2015.

[18] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Automated Control of Multiple Software Goals Using Multiple Actuators," in *Joint Meeting on Foundations of Software Engineering*, Sep. 2017.

[19] F. Zanini, C. Jones, D. Atienza, and G. De Micheli, "Multicore Thermal Management using Approximate Explicit Model Predictive Control," in *International Symposium on Circuits and Systems*, May 2010.

[20] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "A Distributed and Self-calibrating Model-Predictive Controller for Energy and Thermal Management of High-Performance Multicores," in *Design, Automation and Test in Europe*, Mar. 2011.

[21] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, *Introduction to Control Theory And Its Application to Computing Systems*. Boston, MA: Springer US, 2008, pp. 185–215.

[22] HardKernel, "ODROID-XU3," http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127.

[23] ARM®, "big.LITTLE Technology: The Future of Mobile," https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf, 2013, White Paper.

[24] L. Ljung, *System Identification : Theory for the User*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[25] *MATLAB and System Identification Toolbox Release 2015a*. Natick, Massachusetts: The MathWorks Inc., 2015.

[26] *MATLAB and Robust Control Toolbox Release 2015a*. Natick, Massachusetts: The MathWorks Inc., 2015.

[27] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques*, Oct. 2008.

[28] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.