PACT: G: Multilayer Compute Resource Management with Robust Control Theory

Raghavendra Pradyumna Pothukuchi
Advisors: Josep Torrellas, Petros Voulgaris
University of Illinois at Urbana-Champaign

ABSTRACT

Computers increasingly operate in constrained environments, and are being equipped with controllers for resource management. However, the operation of modern computer systems is structured in multiple layers, such as the hardware, OS, and networking layers—each with its own resources. Managing such a system scalably and portably requires that we have a controller in each layer, and that the different controllers coordinate their operation. In addition, such controllers should not rely on heuristics, but be based on formal control theory.

This paper presents a new approach to build *coordinated multilayer formal controllers* for computers. The approach uses Structured Singular Value (SSV) controllers from Robust Control Theory. Such controllers are especially suited for multilayer computer system control. SSV controllers can read signals from other controllers to coordinate multilayer operation. They accept uncertainty guardbands, which incorporate the effects of interference between the controllers. We call this approach *Yukta*. We prototype a two-layer Yukta control system in an 8-core big.LITTLE board and demonstrate its modular design. Yukta reduces the *Energy*×*Delay* and the execution time of a set of applications by an average of 50% and 38%, respectively, over advanced heuristic-based coordinated controllers.

1. INTRODUCTION

Computing devices are ubiquitous and are increasingly operating in constrained environments where resources such as energy, power, or storage capacity are limited, and measures such as temperature, Quality of Service (QoS), or throughput need careful control. Computer systems use sophisticated controllers to change system parameters and meet resource management goals dynamically. [1, 2, 3, 4].

In the design of compute resource management systems, there is a tension between design *modularity* and *coordination*. As shown in Figure 1, modern computing systems are organized in multiple layers — e.g., the hardware, Operating System (OS), and application layers. Each layer is a complex subsystem designed independently by expert teams from possibly different companies. Each layer has its own resources, controllable parameters and partial information about the program execution that it uses to manage resources.

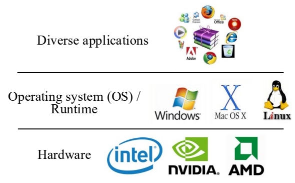


Figure 1: Multilayer organization of computer systems.

For scalability, portability and maintainability, it is vital that resource controllers in each layer are modularly designed. Modularity is also essential because designing a controllers for a layer requires expertise of that layer's internals, which may be proprietary and publicly unavailable.

At the same time, application execution interacts with multiple layers, and it is well known that resource efficiency is best achieved when these layers coordinate their resource management [5, 6, 7, 8]. However, it is difficult to coordinate controllers that are designed with partial system knowledge and operate with partial system view, creating a tension between the two design goals. These goals cannot be achieved simultaneously by either monolithic designs that use a single controller to manage all layers or decoupled designs that use an uncoordinated controller in each layer.

Instead, there is a need to use modular controllers in each layer of the system that collaborate through a mutually agreed interface. Industrial companies are working on coordinated hardware-software approaches where each layer performs its own resource management, and interacts with the other layers through well-defined interfaces [1, 2, 4].

Currently, most multilayer designs are based on ad hoc heuristics. It is not pragmatic to use ad hoc heuristics for building modular controllers. The costs to design, tune and verify the bulky heuristics needed to optimally manage even a single complex layer are prohibitively high. Prior research and commercial systems demonstrate many instances where even highly tuned heuristics fail unwittingly on application corner cases [9, 10]. The difficulty is higher when the controllers have only a partial system view. Moreover, heuristics are highly specific to particular choices, and may become unusable when a different hardware or software platform is used. The solution, then, is to use formal methodologies such as control theory, whose properties are well studied and promise robust operation [11].

In this work, we present a new approach to build coordinated multilayer formal controllers for computer systems. We consider Robust Control Theory, which focuses on uncertain environments, and pick the Structured Singular Value (SSV) controller [11] to be used for computers. This is a MIMO (Multiple Input Multiple Output) controller that can change many system inputs (i.e. parameters) to regulate many outputs (i.e. objectives).

SSV controllers are particularly suited for multilayer computer control. First, they can read *External Signals*, which provide information that the controller cannot directly change, but can use to make better decisions; we use them to pass coordinating information between the controllers in different layers. Second, the design of SSV controllers accepts *uncertainty guardbands*, which are useful to incorporate the effects of interference between independently-designed controllers. Third, designers can specify the maximum bounds on the deviations of outputs from their goals, enabling accurate computer control. Finally, SSV controllers support systems with discretized values for inputs such as computers — unlike other controllers that assume inputs to have continuous unlimited values.

We call this approach of using multilayer SSV controllers for computer system control *Yukta*. With Yukta, controllers at different layers can be built with little interaction. To assess its effectiveness, we prototype it in an 8-core big.LITTLE system running Linux and build a two-layer control system. Yukta reduces the $E \times D$ (Energy $\times$ Delay) and the execution time of a set of applications by an average of 50% and 38%, respectively, beyond what advanced heuristic-based coordinated controllers attain. Our contributions are:

1. Applying MIMO SSV control from robust control theory for systematic computer resource efficiency.
2. *Yukta*, an approach for independent teams to design coordinated multilayer controllers with MIMO SSV.
3. A prototype of Yukta on a big.LITTLE multicore board and its evaluation.

This is the *first* work that uses MIMO SSV control to address the collaborative multilayer computer control problem.

2. BACKGROUND AND RELATED WORK

2.1 Related Work

There are many works on managing resources from different layers of a computer system (e.g., [5, 6, 7, 8, 12]). They emphasize the need for modular coordinated control. However, most of them rely heavily on heuristics. There is strong evidence from research and practice demonstrating the difficulty of developing heuristics for control, and the mishaps that occur when the system encounters an application not seen during the training set [9, 10].

Currently, there are no control theoretic methods to develop coordinated multilayer controllers for computers even though control theory has been used to design computer controllers (e.g., [1, 10, 13]). Most designs use PID (Proportional Integral Derivative) or similar Single Input Single Output (SISO) controllers [1, 13] that only monitor one goal and change one parameter. Some designs are Multiple Inputs and Single Output (MISO) or Multiple Inputs and Multiple Outputs (MIMO) [10]. The MIMO approach is the most applicable to computer systems, since multiple goals (performance, power) are typically coupled with each other. However, existing controllers are intended for standalone use, and do not have channels for coordination between multiple controllers. Some designs employ heuristics or controller ordering to make up for this deficit [8, 13], but this defeats the purpose of formal control methods. Some designs use a combination of heuristics and control theory [8] or heuristics and optimization [5].

Existing designs are not natively robust to the large uncertainty that appears in the presence of multiple controllers, each acting with partial system information.

2.2 Mathematical Theory of SSV Controllers

Computers are complex, and program behavior is determined by many factors. As a result, controlling computer environments intrinsically involves dealing with uncertain dynamics and approximate models. Robust Control Theory is a branch of control theory that considers variability and uncertainty of the system dynamics to be an integral part of the controller synthesis process.

Among the robust controller methodologies, one of the most mature and better understood, with standard packages and tools, is *Structured Singular Value (SSV)* control [11]. SSV design is automated with tools and designers only need to express high level specifications [14].

To obtain an SSV controller (K), the designer first specifies the model of the system (M). Then, there are real world inaccuracies denoted by Δ. One is due to the true system behavior deviating from the model (Δ_u) because of model limitations. This is the model uncertainty for which we specify guardbands. Another is due to the inputs taking only a discrete (or quantized) and limited (or saturated) set of allowed values (Δ_{in}). This is the input discretization. These specifications are pictorially represented as in Figure 2. The designer also gives the desired bounds B on the allowed deviations of outputs from their targets, and the relative weights/overheads W, to change the inputs.

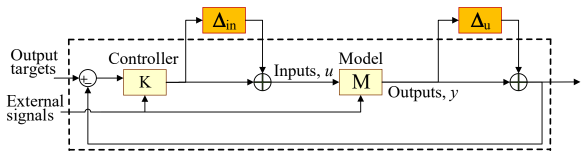


Figure 2: Internal representation in SSV synthesis.

The system inside the dotted line boundary in Figure 2 is called the nominal closed loop, N because it contains the components without imprecisions. Consolidating the individual Δ components into an overall Δ, the controller K is robust if it: (i) keeps N stable, (ii) generates optimal inputs according to designer-specified input weights W, and (iii) keeps all outputs within bounds B of the targets – for all possible model inaccuracies smaller than the specified Δ. Robust control theory uses the Structural Singular Value (SSV) defined as follows to assess a controller's robustness [11]:

$$SSV(N,\Delta,B,W) = \frac{1}{\min\{s \,|\, det(I - s \times N \times [\Delta; B^{-1}; W^{-1}]) = 0\}} \quad (1)$$

where $[\Delta; B^{-1}; W^{-1}]$ is a diagonal matrix of Δ, B^{-1} and W^{-1}; and I is the identity matrix. Physically, s is a scaling factor that multiplies the Δ, $1/B$, and $1/W$ given by the designer. The minimum scaling factor *min(s)* gives the worst-case values of Δ, B and W that the controller supports. If *min(s)* is larger than 1, it means that the controller can handle the Δ, B, and W requested by the designer. MATLAB runs an iterative search to generate such a controller. If the synthesis fails, the designer selects lower Δ, $1/B$, and $1/W$ values, and restarts.

3. Yukta: MULTILAYER SSV CONTROL

To address the challenge of controlling multiple layers, we propose using Collaborative MIMO SSV controllers. In this solution, there is preferably an SSV controller in each layer. Less desirably, there is an SSV controller at least in the layer that controls outputs requiring accurate control (e.g., temperature or power), and other types of controllers in the other layers. We utilize the properties of SSV controllers to create a modular coordinated computer control system.

First, SSV controllers can read an additional type of signals called *External Signals*. We use them to pass information from one layer's controller to the other at runtime. For example, an OS controller can pass the number of running threads as an external signal to a hardware power controller. The second controller can use the signals to make better decisions, although it cannot control them.

Second, SSV control designers can specify model *uncertainty guardbands*, typically expressed in percentages. For example, a 20% uncertainty means that, due to unanticipated effects, the values of the outputs can possibly be ±20% different than predicted by the model. In a multilayer system, one controller's actions indirectly affect the outputs that a second controller is supposed to control. This interference can be incorporated in the SSV controller design by increasing the uncertainty guardband of the second controller.

Third, designers can specify bounds on the allowed deviation of the outputs from their targets. This ability enables accurate computer control. Finally, robust controllers accept the the discreteness (Saturation and Quantization) in the values taken by the inputs. This is in contrast to non-robust controllers where each input is assumed to take continuous and unbounded values. This makes SSV controllers natively applicable to computing systems, which have discrete resources. For example, core frequency can only take a few discrete values. When these inputs and outputs are passed as an external signals to another layer's controller, the availability of precise bounds or discretization information helps the pair of controllers improve their coordination.

Often, computers need to optimize outputs (or combination thereof) subject to other outputs being within certain limits. An example is to minimize $E \times D$ (Energy$\times$Delay) subject to a power constraint. In this case, the controller needs to perform some search. Hence, we augment each SSV controller with an optimizer module. The Optimizer generates progressively better targets to the controller, which in turn find the best inputs to meet those targets. Eventually, the Optimizer converges to a desirable set of targets. We call our general approach *Yukta*. Figure 3 shows the envisioned Yukta control system for a two-layer system. Each controller takes external signals from the other.

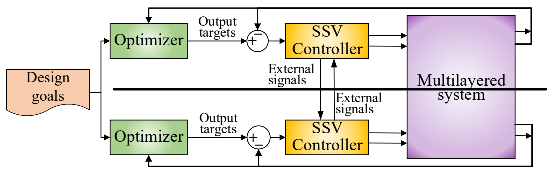


Figure 3: Yukta controller augmented with optimizers.

3.1 Designing SSV Controllers

Figure 4 shows the process of designing a Yukta multilayer SSV controller. In each layer, a team initiates the design of the layer's controller by selecting the input signals and their discretized values, the output signals and their deviation bounds, and the external signals that the controller takes. Then, the teams exchange *Interface* information. This is meta-information about external signals and common outputs. Specifically, for outputs common to both controllers, the teams exchange their layer's deviation bounds; for an external signal to a controller from a second layer, the second layer team passes the allowed discrete values if the signal is an input in the second layer, or the deviation bounds if it is an output in the second layer.

After this communication step, each team develops a model of the system according to their layer's perspective , sets its controller's uncertainty guardband, designs the SSV controller using MATLAB synthesis routines [14], and validates it. Finally, the designs of all the layers are combined, validated as a group, and deployed. This process can work across companies. Alternatively, a team can do without any extra information for their external signals. In this case, the team should increase their uncertainty guardband. This works because SSV controllers withstand inaccurate assumptions.

4. PROTOTYPING Yukta

We prototype a multilayer Yukta system in a challenging environment: an ODROID XU3 board [15], which has an 8-core Samsung Exynos 5422 processor running Ubuntu 15.04. The system has the HMP (Heterogeneous Multi-Processing) task scheduler built for ARM big.LITTLE technology [16]. It has a cluster of four little cores (in-order, low power Cortex A7), and a cluster of four big cores (out-of-order, high performance Cortex A15). Figure 5 shows our experimental platform. The number of active cores in either 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.

The prototyped two-layer SSV controllers are shown in Figure 6. One controller controls hardware parameters (hardware controller), and another controls thread scheduling parameters (software/OS controller). Tables 1 and 2 show the inputs and outputs for the hardware and software controllers respectively. Our goal for the hardware controller is to minimize $E \times D$ while keeping power and temperature below certain limits. Our goal for the software controller is to simply minimize $E \times D$. Since our goals involve minimizing $E{\times}D$, we also design optimizer modules for each controller.

First, we follow the System Identification methodology [17] and use experimental data to obtain models of each layer. Our models have a dimension of four — i.e., they predict the value of an output using the past 4 values of outputs, the current and past 3 values of inputs.

Table 1: Parameters of the hardware SSV controller.

Goal	**Inputs**		**Outputs**	
	Signals	**Weights**	**Signals**	**Bounds**
Minimize $E \times D$ subject to	#big cores	1	Performance	±20%
$\text{Power}_{big} < \text{Power}_{big}^{max}$,	#little cores	1	Power_{big}	±10%
$\text{Power}_{little} < \text{Power}_{little}^{max}$,	frequency_{big}	1	Power_{little}	±10%
and $\text{Temp} < \text{Temp}^{max}$	$\text{frequency}_{little}$	1	Temp	±10%

Table 2: Parameters of the software SSV controller.

Inputs		**Outputs**	
Signals	**Weights**	**Signals**	**Bounds**
$\#\text{threads}_{big}$	2	$\text{Performance}_{little}$	±20%
Avg #threads per non-idle core in cluster_{big}	2	Performance_{big}	±20%
Avg #threads per non-idle core in cluster_{little}	2	$\Delta\ \text{SpareCompute}_{big-little}$	±20%

Since the overhead of changing hardware inputs are comparable, we set all the input weights to be 1. Similarly, for the OS controller we assign the same weight to all inputs. However, we want the software controller to react more conservatively to output changes than the hardware controller. This is because applications change the number of threads dynamically in an unpredictable manner for the controller — e.g., some threads block on I/O. We do not want the controller to react immediately and cause oscillations. Consequently, we set the weight of all inputs to 2 (Table 2), which happens to be twice the weight of the hardware controller's inputs.

Among the hardware outputs, the power of both clusters and the temperature are critical for the integrity of the board.

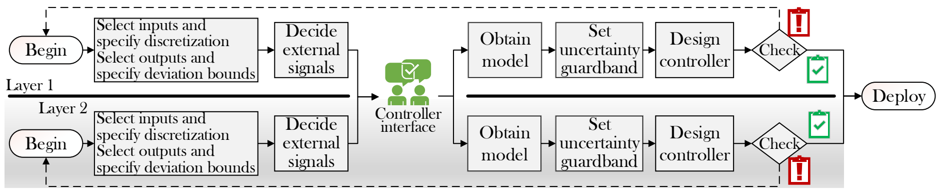

Begin
Select inputs and specify discretization Select outputs and specify deviation bounds
Decide external signals
Layer 1
Layer 2
Begin
Select inputs and specify discretization Select outputs and specify deviation bounds
Decide external signals
Controller interface
Obtain model
Set uncertainty guardband
Design controller
Check
Obtain model
Set uncertainty guardband
Design controller
Check
Deploy

Figure 4: Process to design a Yukta multilayer SSV controller.

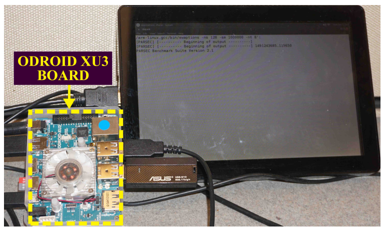


Figure 5: Prototype ODROID XU3.

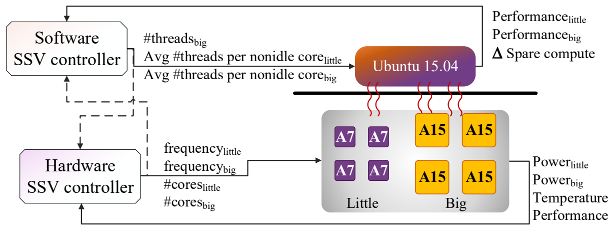
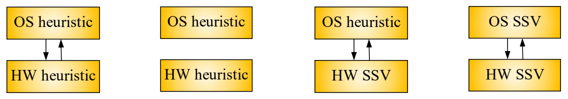


Figure 6: Prototyped controllers on the ODROID XU3.



(a) Coordinated heuristic (b) Decoupled heuristic (c) Yukta:HW SSV+ OS heuristic (d) Yukta:HW SSV+ OS SSV

Figure 7: Two-level controllers evaluated.

Hence, we assign them a bounds range that is ±10% of their maximum range. Since performance(measured in total committed Billions of Instructions Per Second) is less critical, we set its bounds as ±20%.

The OS controller monitors three outputs: performance of the big and little cluster threads, and difference in Spare Compute Capacity (SC) between the two clusters. Briefly, with a higher difference in SC, the controller will move more threads from the little to the big cluster We define a cluster's SC as, SC = $\#idle\_cores\_on - (\#threads - \#cores\_on)$. As all outputs have similar importance, we set their deviation bounds to ±20% of their maximum range — like the non-critical hardware outputs.

Each controller reads all the inputs actuated by the other as external signals. Finally, we provide the uncertainty guardbands. Recall that uncertainty is the result of limitations in how the model describes the real system, and of unpredictability in the system. An example of the latter is aspects of the HMP scheduler, which sometimes packs multiple threads on a core while leaving another idle. Based on the model training data, we pick a guardband of ±40% for the hardware controller. The uncertainty guardband used for the software controller should be higher because thread assignment is directly affected by an unpredictable event: dynamic changes in the number of application threads. Therefore, we set its guardband to ±50%.

Industry-grade heuristic controllers, in contrast, have an order of magnitude more parameters. For example, to control the same outputs the Samsung Exynos 5422 uses several tens of interdependent settings that require tuning. Our approach eliminates the need for this extensive tuning.

5. EXPERIMENTAL SETUP

We implement four two-level controllers shown in Figure 7. In *Coordinated heuristic*, the OS controller is similar to the HMP task scheduler from ARM, Linaro and Samsung [16], except that it is modified to optimize E×D. This OS-hardware scheme is representative of industry-standard controllers in big.LITTLE systems, and we use it as a baseline. The *Decoupled heuristic* scheme takes uncoordinated decisions at each layer. Its hardware controller is similar to the *Performance* power governor in Linux.

We design two schemes based on our proposed Yukta methodology. The first, *Yukta: HW SSV+OS heuristic*, uses an SSV hardware controller from Section 4 and a heuristic-based OS controller like the one in Coordinated heuristic. The second, *Yukta: HW SSV+OS SSV*, uses SSV controllers in hardware and software, as designed in Section 4.

Our evaluation uses 8-threaded PARSEC [18] programs (blackscholes, bodytrack, facesim, fluidanimate, raytrace, x264, canneal, streamcluster) and 8 copies of SPEC06 [19] programs (h264ref, mcf, omnetpp, gamess, gromacs, dealII). For training, we use: swaptions and vips from PARSEC, astar and perlbench, milc and namd from SPEC.

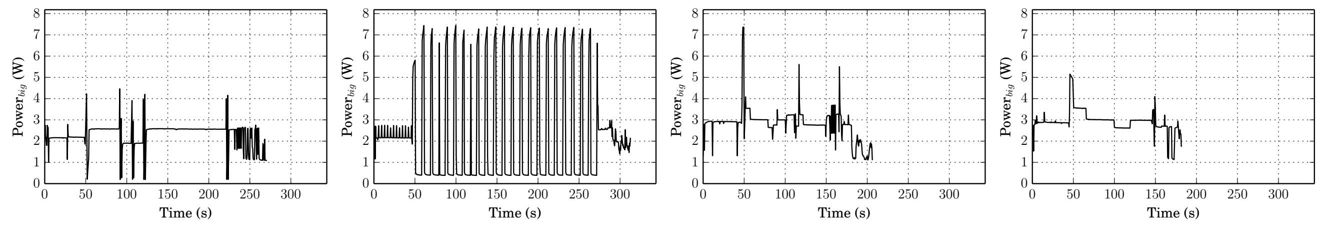
6. EVALUATION

Figure 9 compares our four two-layer controller schemes in minimizing $E \times D$ running our applications. The bars from left to right correspond to individual SPEC applications, the average of the SPEC applications (*SAv*), individual PARSEC applications, the average of the PARSEC applications (*PAv*), and the average of all the applications (*Avg*). Each application has a bar for each of the four controller schemes. The bars are normalized to *Coordinated heuristic*.

Figure 9 shows that *Decoupled heuristic* has higher $E \times D$ than *Coordinated heuristic*. On average, decoupling the controllers results in a 52% higher $E \times D$. On the other hand, using Yukta causes $E \times D$ to decrease. On average, *Yukta: HW SSV+OS heuristic* has a 37% lower $E \times D$ than *Coordinated heuristic*. Furthermore, having both SSV controllers as in *Yukta: HW SSV+OS SSV* results in an average $E \times D$ that is 50% lower than *Coordinated heuristic*. Thus, SSV controllers offer substantial improvements over existing systems.

For execution times (not shown), the trends are similar. *Decoupled heuristic* increases the execution time by 30% on average. On the other hand, *Yukta: HW SSV+OS SSV* reduces the time by 29% on average, and *Yukta: HW SSV+OS SSV* by even more, namely a substantial 38% on average.

To analyze the impact of the Yukta controllers, we focus on the execution of the blackscholes application (labeled *bla* in Figure 9). This application begins with a single thread and later executes 8 parallel threads. Figure 8 shows the power consumed by the big cluster in blackscholes as a function of time, for the four controller schemes. Recall that the limit in



(a) *Coordinated heuristic* (b) *Decoupled heuristic* (c) *Yukta: HW SSV+OS heuristic* (d) *Yukta: HW SSV+OS SSV*

Figure 8: Power consumed by the big cluster in blackscholes as a function of time for the four controller schemes.

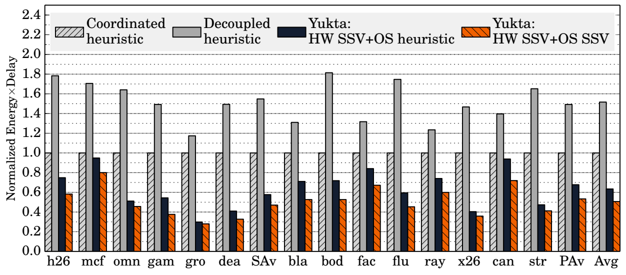


Figure 9: Normalized Energy×Delay (lower is better).

sustained power is 3.3W.

In *Decoupled heuristic* (Figure 8(b)), there are many oscillations. In this scheme, the hardware controller increases the number of cores and their frequency to the maximum, while the OS controller assigns threads round-robin. This causes the power to go over the limit and trigger the emergency system that reduces the frequency and shuts off some cores. The power then drops to low values, and the hardware controller again increases the number of cores and their frequency to the maximum. The result is continuous power oscillation.

Coordinated heuristic (Figure 8(a)) drastically reduces the amplitude and number of these variations. This is thanks to the coordination between the two controllers: the hardware controller knows the distribution of the active threads, and the OS controller knows the number, type, and frequency of the active cores.

As we move to *Yukta: HW SSV+OS heuristic* (Figure 8(c)) and, especially, *Yukta: HW SSV+OS SSV* (Figure 8(d)), the number of peaks and valleys decreases. Moreover, the power during the steady-state periods gets closer to 3.3W. The Yukta designs control power much better due to their robust design.

The differences in power control translate directly into performance.With *Decoupled heuristic*, the application takes nearly 320 seconds to complete. With *Coordinated heuristic*, the application completes in 270 seconds. Finally, in *Yukta: HW SSV+OS heuristic* and *Yukta: HW SSV+OS SSV*, the steady-state performance keeps increasing, and the application completes sooner, in 205 and 180 seconds, respectively.

These results show that the Yukta approach achieves much higher resource efficiency over state-of-the-art for multilayer computers, while being modular in design.

7. CONCLUSION

To address the challenge of computer resource management in increasingly constrained environments, this paper presented a new approach to build coordinated multilayer formal controllers for computer systems. The approach uses SSV controllers from robust control theory. These controllers can read External Signals from other controllers to coordinate multilayer operation. They accept Uncertainty Guardbands, which incorporate the effects of interference between the different controllers. We called this approach *Yukta*. It is the *first* work to use robust control theory for multilayer compute resource management. To assess its effectiveness, we prototyped it in an 8-core big.LITTLE board. On average, Yukta reduced the $E \times D$ and the execution time of applications by 50% and 38%, respectively, over state-of-the-art.

8. REFERENCES

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

[2] M. Broyles *et al.*, "IBM EnergyScale for POWER8 Processor-Based Systems," IBM, Tech. Rep., Nov. 2015.

[3] Terry Myerson, "Windows 10 Embracing Silicon Innovation," https://blogs.windows.com/windowsexperience/2016/01/15/windows-10-embracing-silicon-innovation/, 2016, Windows Blog.

[4] I. Rickards and A. Kucheria, "Energy Aware Scheduling (EAS) progress update," http://www.linaro.org/blog/core-dump/energy-aware-scheduling-eas-progress-update/.

[5] V. Vardhan *et al.*, "GRACE-2: Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy," *Intl. J. Embed. Sys.*, vol. 4, no. 2, pp. 152–169, 2009.

[6] B. D. Noble *et al.*, "Agile Application-aware Adaptation for Mobility," in *SOSP*, 1997.

[7] H. Zhang and H. Hoffmann, "Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques," in *ASPLOS*, 2016.

[8] R. Raghavendra *et al.*, "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center," in *ASPLOS*, 2008.

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

[10] R. P. Pothukuchi *et al.*, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," in *ISCA*, 2016.

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

[12] S. Mohapatra *et al.*, "DYNAMO: A Cross-Layer Framework for End-to-End QoS and Energy Optimization in Mobile Handheld Devices," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 4, pp. 722–737, May 2007.

[13] S. Shevtsov and D. Weyns, "Keep It SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-based Self-adaptive Systems," in *FSE*, 2016, pp. 229–241.

[14] *MATLAB and Robust Control Toolbox Release 2016a*. Natick, Massachusetts: The MathWorks Inc., 2016.

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

[16] H. Chung *et al.*, "Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM$^{®}$ big.LITTLE^{TM} Technology," https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf, 2013, White Paper.

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

[18] C. Bienia *et al.*, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, Oct. 2008.

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