Hard Periodic Real-time Task Scheduling on Heterogeneous Processor

Regular Submission

Aref Karimiafshar, Mohammad Ali Montazeri, Mahdi Kalbasi, Ali Fanian

Electrical & Computer Engineering Department
Isfahan University of Technology
{A.karimiafshar, M.kalbasi}@ec.iut.ac.ir, {Montazeri, A.fanian}@cc.iut.ac.ir

Abstract

High-performance heterogeneous processors are being implemented in embedded real-time systems because of the increasing computational requirements. A heterogeneous processor consists of cores that are asymmetric in performance and functionality. Such a design provides a cost-effective solution for processor manufacturers to continuously improve both single-thread performance and multi-thread throughput. These complex processors have a major drawback when they are used for real-time purposes. Their complexity difficults the calculation of the WCET (worst case execution time). This design, however, faces significant challenges in the operating system, which traditionally assumes only homogeneous hardware. The OS scheduler needs to be heterogeneity-aware, so it can match jobs to cores according to characteristics of both. In this paper, we make a case that a scheduler for heterogeneous multicore systems should target three objectives: optimal performance, minimum load and maximum satisfied deadline. We deal with this issue via optimal task-to-core assignment. The proposed scheduler enables performance improvements, reduction in load and satisfied deadline increase for range of applications. Different scheduling alternatives have been evaluated and experimental results show that the proposed algorithm provides, on average, improvement in our three objectives ranging from 5.34% to 8.75%.

Keywords*: embedded systems, real-time task, heterogeneous processors, OS scheduler.*

1 Introduction

Embedded systems are experiencing a growth of the number of functionalities they incorporate. Among others, they can act as phone cells, PDA's, car on-board systems, etc. Due to the performance requirements of these applications, high-end embedded processors nowadays are including complex mechanisms developed for high-performance architectures. Actually, advances in silicon technology have enabled processor manufacturers to integrate more and more cores on a chip. Most multi-core processors consist of identical cores, where each core implements sophisticated microarchitecture techniques, such as superscalar and out-of-order execution, to achieve high single-thread performance. This approach can incur high energy costs as the number of cores continues to grow and it is not desired in such devices. Alternatively, a processor can contain many simple, low-power cores, possibly with in-order execution. This approach, however, sacrifices single-thread performance and benefits only applications with

thread-level parallelism. Also, There is a combination of these two methods that make heterogeneous processor.

A heterogeneous processor integrates a mix of “big” and “small” cores, and thus can potentially achieve the benefits of both. In this way, there are tow main category in the point of view of instruction set architecture (ISA); heterogeneous processor with the cores of same ISA and heterogeneous processor including cores with different ISA or overlapped.

Single-ISA heterogeneous multicore processors, also known as *asymmetric* single-ISA (ASISA) [18], consist of cores exposing the same ISA, but delivering different performance. These cores differ in clock frequency, power consumption, and possibly in cache size and other microarchitectural features. Asymmetry may be built in by design [14][15], or may occur due to process variation [13] or explicit clock frequency scaling. In the other hand, some other Modern embedded systems combine different types of hardware accelerators or coprocessors with one or more general purpose processors. For example, a system may have general-purpose graphic processing units (GPGPUs) to encode or decode video with better throughput and lower latency than would be possible with a traditional microprocessor.

There are several usages motivate this design:
• *Parallel processing*: with a combination of a few big and many small cores, the processor can deliver higher performance at possibly the same or lower power than an ISO-area homogeneous design.
• *Power savings*: the processor uses small cores to save power. For example, it can operate in two modes: a highpower mode in which all cores are available and a lowpower mode in which applications only run on the small cores to save power at the cost of performance. *Accelerator*: unlike the previous models, where the big cores have higher performance and even more features, in this model, the small cores implement special instructions, such as vector processing, which are unavailable on the big cores. Thus, applications can use the small cores as accelerators for these operations. These usages are not disjoint. For example, in the parallel processing model, the small cores can also implement unique vector instructions and act as accelerators. There are typically more small cores than big cores in this model, but not necessarily in the others.

Power and area efficiencies of HMC systems have been demonstrated in numerous studies [3][14][15][16][18]. E.g, In a recent study[5], the benefit of HMC has been explored by emulating a heterogeneous processor with one big core and four small cores using a multiprocessor system, where the big core runs at 2.66 GHz with a 4 MB L2 cache and each small core has a lower frequency, fewer instruction execution units, and a 2 MB L2. Assuming one big core is of equal area to four small cores, they compared this design to a homogeneous one with two big cores only. Figure 1 shows the results. For the heterogeneous design, they also varied the small core frequency, resulting in four configurations as shown on the x-axis. As it is shown in figure 1, all of the benchmarks obtain higher performance from at least one heterogeneous configuration.

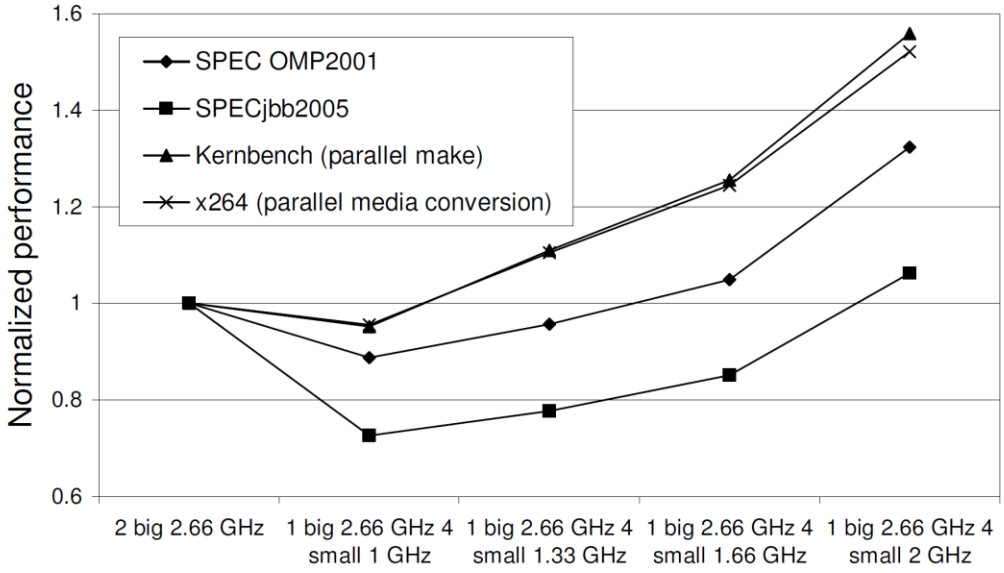


Figure1: Performance comparisons for two big cores only vs. one big core, and four small cores at different frequencies

Despite their benefits, heterogeneous architectures pose significant challenges to OS design, which has traditionally assumed homogeneous hardware. E.g, difficult the calculation of the WCET (Worst Case Execution Time) of real-time tasks that it makes the situation harder for the scheduler. Consider that Hard real-time tasks have critical deadlines that, if missed, may have catastrophic repercussions.

Operating system (OS) schedulers for conventional processors are tuned for response time fairness: they distribute CPU time among the jobs in some fair fashion and limit the delay associated with waiting for CPU [3,6]. The advent of heterogeneous multicore (HMC) processors motivates new performance considerations for scheduling algorithms. On an HMC system, performance can be improved by scheduling a job to run on the car that is best suited for that job – i.e., the job's *preferred* core [5]. This performance objective may conflict with the other objective of conventional schedulers such as satisfying the deadlines as the major concern in the real-time systems. Consider the following example: Suppose that at the time when the scheduler dispatches task *A* to run on a processor, *A*'s preferred core was occupied by another running task. The scheduler faces a dilemma: should it schedule *A* to run on its non-preferred core (that is available right away) or have it wait until its preferred core becomes available? Making *A* wait will potentially increase *A*'s response time and reduce its allocated fraction of CPU cycles (compared to a conventional, homogeneous system). On the other hand, scheduling A on a non-preferred core will produce sub-optimal instruction throughput for *A*. Making the right decision requires carefully weighing performance/deadline tradeoffs and making the choice that meets system's goals with respect to the objectives.

This paper studies scheduler for heterogeneous architectures in which cores have asymmetric performance and identical instruction sets.

The contributions of this paper are as follows:

(1) We consider a real trade-processor and config our experiments base on it.
(2) We evaluate our scheduler considering presence of the context switch penalty that many of the previous works did not care about it.
(3) We consider three objectives for evaluating the algorithm: optimal performance, minimum load, maximum satisfied deadline.
(4) We present an enhanced algorithm according to the characteristics of the heterogeneous processors.
The remainder of this paper is organized as follows. In Section 2, we discuss our architecture model. Section 3 describes our algorithm. Section 4 introduces our simulation framework and experimental results, also we present an enhanced algorithm base on the proposed algorithm in this section. Finally, we conclude in Section 5.

2 System model

In this paper we consider a real trade-processor, SPARC64™ VI of Sunsystem, to practically deal with the context switch penalty. SPARC64™ VI has two cores and each core implement a hardware thread vertically. Table 1 contains the main characteristic of this processor. We explored through it's datasheet and calculate it's switch penalty to evaluate our algorithm more precisely. Generally we have two kinds of schedulers: global scheduling algorithms and partitioned scheduling algorithms. In global algorithms at each scheduling point the scheduler takes the all c.u. Into account. In the such fashions There is a global queue that all tasks enter into this queue and scheduler with taking care of all the tasks and all the c.u. Assign tasks but in partitioned scheduling algorithm the tasks classify in certain groups and then each group assign to proper c.u. And in each unit a uniprocessor scheduling algorithm is used. In global scheduling algorithms tasks are free to migrate between units but in a pure partitioned scheduling algorithm, it is not happening. Whenever a task migrates between units, at first take some cycles of processor to migrate and second it miss cache affinity. So we chose real processor to deal with this issue precisely. Although in new processor have been tried to reduce this penalty but in a real-time system it is considerable yet. We deal with hard periodic real-time tasks That can preempitibly run to gather and each task has certain release and execution time that are known precedent.

	SPARC64™ VI
CPU cores per chip	2
Threads per CPU core	2 (VMT)
Level1 cache	256KB with 2ways per core
Maximum Level2 cache	6MB with 12 ways at maximum (per CPU chip)

3 Scheduling Algorithm

Each time a task is added to the system , enter into two ordered lists. One of them has been ordered based on execution characteristics of the task and the other one has been ordered according to the period of that task. In this paper, we consider each task deadline within its period. We classify the tasks in these two ordered lists according their computational requirements With criterion (1) and (2) .

$$Eg(i) = \{\, t_i \mid E_{Eg}(low) \le e_i < E_{Eg}(Heigh)\} \tag{1}$$

$$Pg(i) = \{\, p_i \mid P_{Pg}(low) \le p_i < P_{Pg}(Heigh)\} \tag{2}$$

We define a new operaton as |xj| for each class of ej and pj. Equation (3) and (4) describe how |xj| operate on each list. We use the results of |xj| on both classes to define Z as the main factor for assigning the tasks to proportion c.u.

$$Eg_{C_j} = Index\big(List(|Eg(i)|)\big) + 1 \tag{3}$$

$$p_j = Index\big(List(|Pg(i)|)\big) + 1 \tag{4}$$

Equation 5 is used for calculation of the Z. For each task in the system Zi would calculate and if two Zi have same quantity then we must go through calculating A as the second factor.

$$Z_j = \frac{1}{(Eg_{C_j}{}^2 . p_j + Eg_{C_j} . p_j{}^2)} \tag{5}$$

$$A = \frac{Z}{P} \tag{6}$$

We order the tasks according to A ascending and assign them to c.u. That was ordered according to comoutional power Ascending too. Subsequent example, describe how the proposed algorithms work. In this example the set of tasks in table 2 is going to assign to four c.u. In a SPARC64™VI processor.

Tasks	Execution time	Period
Ctx0	30	70
Ctx1	78	133
Ctx2	129	238
Ctx3	189	390
Ctx4	26	70
Ctx5	19	65

Example, firs, classifying the tasks in table 2 into Eg and p_j Group according to equation (1) and (2), then calculating the $|Eg|$ and $|p_j|$.

Tasks	Eg	p_j
Ctx0	4	1
Ctx1	3	2
Ctx2	2	3
Ctx3	1	4
Ctx4	4	1
Ctx5	4	1

Step2: calculating the Z and A factor, assigning the tasks to c.u.

Tasks	A	Z	C.U.
Ctx0	0.05	0.05	C0
Ctx1	0.016667	0.0333	C1
Ctx2	0.011111	0.0333	C2
Ctx3	0.0125	0.05	C3
Ctx4	0.05	0.05	C0
Ctx5	0.05	0.05	C0

4 Simulation and experimental results

4-1 Simulation

We use a heterogeneous–aware simulation, multi2sim, to evaluate our proposed algorithm. Multi2sim (M2S) conventional is an emulator that provides heterogeneity through combining GPGUs with general purpose cores. We change the M2S to simulate hard periodic real-time task and check about the deadlines and evaluate different alternatives scheduling. Respect to deadlines and execution time of each task that are pre-known to the system. We used 13 mixes of WCET benchmarks with different CPU and memory load. Each time we run the simulation, continue it for 3 million cycles . Table 5 shows the WCET benchmarks and their characteristics and table 6 shows the 13 mixes that we chose.

Benchmarks	CPU%	Mem%	Overlap%
Adcmp	61.5	27.9	0.3
BS	58.5	25.7	5.6
Cnt	64.2	23.8	4
Comperss	61.8	22.6	4
Crc	49.1	30.8	1.3
Edn	63.1	25.9	2
fdct	59.9	24.6	4.4

Fft1	63.2	24.3	4.1
Fir	57.2	27.6	4.3
Lms	70.4	25.6	1
Ludcmp	59.9	21.8	3.9
ns	61.8	28.6	2.2

Benchmarks							Mixes
crc	adpcm	lms	ludcmp	fdct	fft1	fir	MIX1
fir	fft1	fdct	ludcmp	lms	adcmp	crc	MIX2
fdct	fir	fft1	crc	adcmp	lms	ludcmp	MIX3
crc	fir	fdct	lms	fft1	ludcmp	adcmp	MIX4
lms	crc	ludcmp	fdct	fir	adcmp	fft1	MIX5
crc	lms	ns	comperss	cnt	edn	bs	MIX6
edn	comperss	crc	lms	cnt	ns	bs	MIX7
crc	fir	cnt	fdct	lms	fft1	edn	MIX8
adcmp	ludcmp	edn	fft1	fir	bs	ns	MIX9
ludcmp	crc	lms	fdct	edn	adcmp	fft1	MIX10
lms	comperss	cnt	edn	crc	bs	ns	MIX11
bs	lms	ns	comperss	cnt	edn	adcmp	MIX12
adcmp	ns	edn	fir	fft1	bs	ludcmp	MIX13

4-2 Evaluation metric

In this paper we focus on three objectives: 1- optimal performance, 2- minimum load, 3- maximum satistified deadlines. We measure performance by the number of cycles that each mixes needs to complete all the tasks, also we measure the load by keeping the number of the tasks that wait in each qeueu to receive the c.u. So an optimal perfect scheduler distributes the tasks in the way that none task waits for c.u. The most important parameter in the hard real-time systems is the deadline . So we show the ability of the algorithm to satistifing deadline by the percentage of tasks that cannot be completed before the predefined deadlines.

4-3 Enhanced Scheduling Algorithm

In a pure fashion partitioned scheduling algorithm, tasks classify into the number of c.u. And then in each group of task schedule locally and none task is permitted to migrate. In such algorithm we are neglecting the facilities that an HMC provides . Groups of tasks are scheduled on C.U. And run separately. Even one of the c.u. The queue is empty the task on congested c.u. Can not migrate to them. So we make a change in the pure fashion. Whenever a queue in higher order in the list of c.u. Face with congestion the tasks from the congested queue can migrate to one of the sparked qeueu in lower order in the list. This migration happens in only one direction and just from higher order queus to lower ones. So if one of the lower qeueu face with the congestion it is not permitted to send some tasks to higher qeues. There are several ways to detect congestion in each queue but most of them take plenty of time and may be hardware-requirements or significant cycles of the processor are wasted. So we set a default threshold,

whenever the number of the tasks in the high order queue goes over this threshold the recent task is sent to a lower qeueu. By setting threshold in some cases we experiences unnecessary migration and consequently their penalty due to migration but the order of the algorithm stay at o (n).

4-4 Experimental results

We compare our two proposed algorithms with three other algorithms that briefly describe follow. first, WCTE, this algorithm is kind of global ones and tries to assign the task to c.u. In the fairly way. The tasks enter to a global queue and then assing according the release time to the c.u. Passing one of below three criteria:

1- If the last c.u that was used by this task are free assigning to it.
2- If there is a non-occupied c.u. That was not used till now assign to it.
3- Find any free c.u. And assign to it.

Second,hhScc, this anotherimplementationn of the WCTE that distributes the task base on the execution time insteadd of thecachee affinityconsiderationn. Third, we called it r-Sc and is an implementation of the partitioned EDF. We call our first version of the porpoed algorithm Te and the enhanced version Sc. Figure 2 shows the performance comparison between these five algorithms.

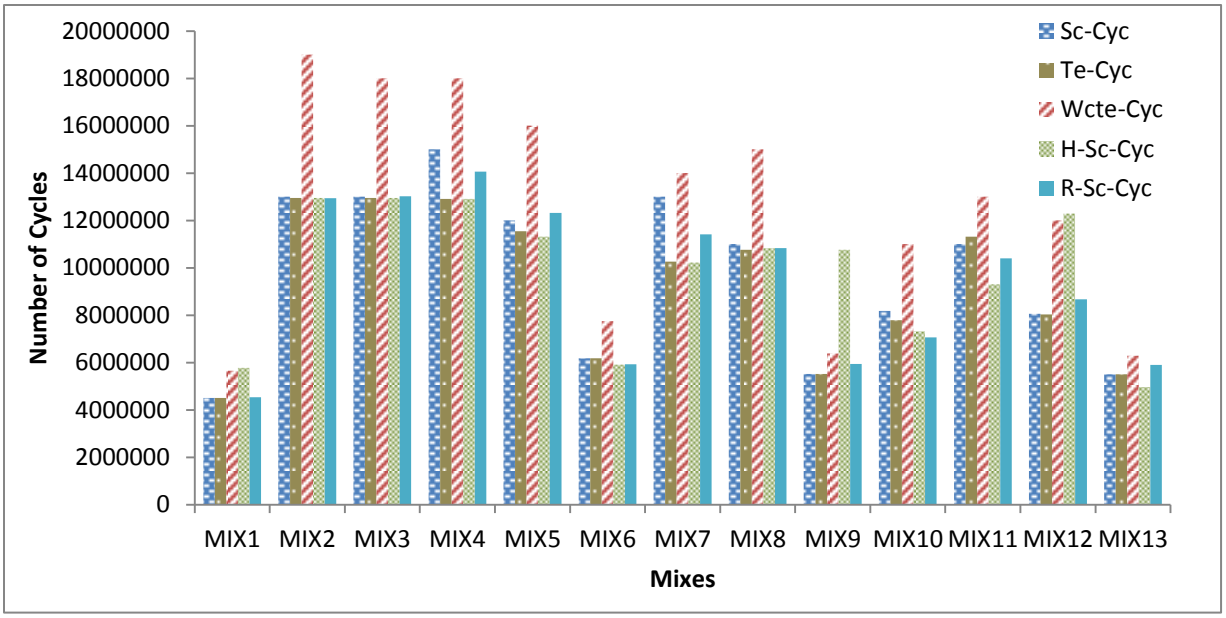
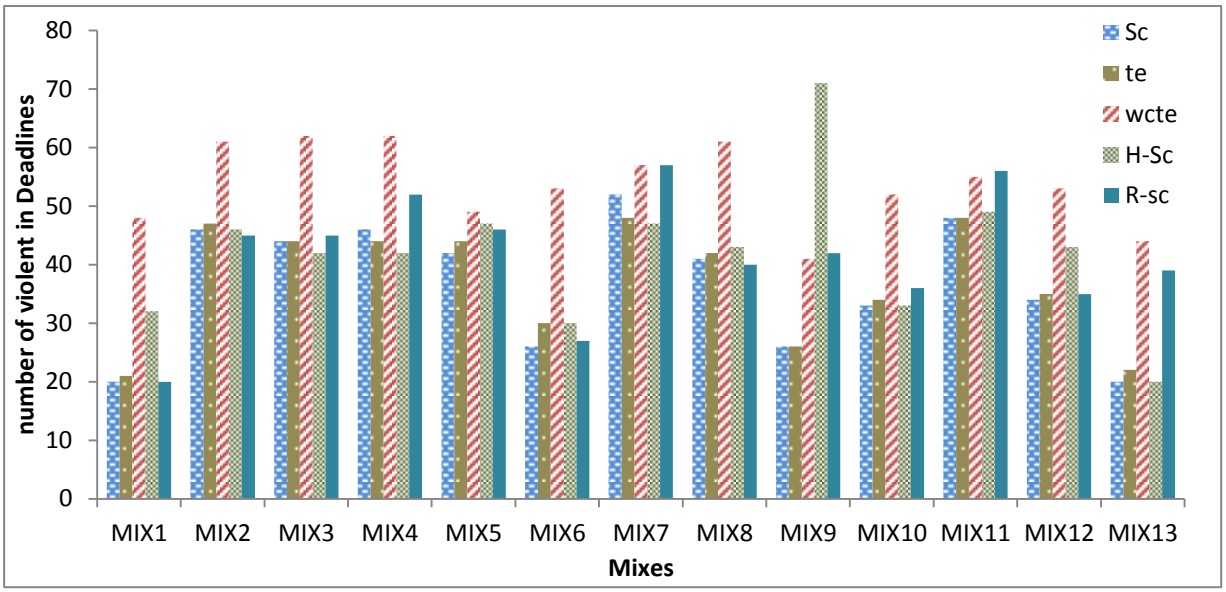
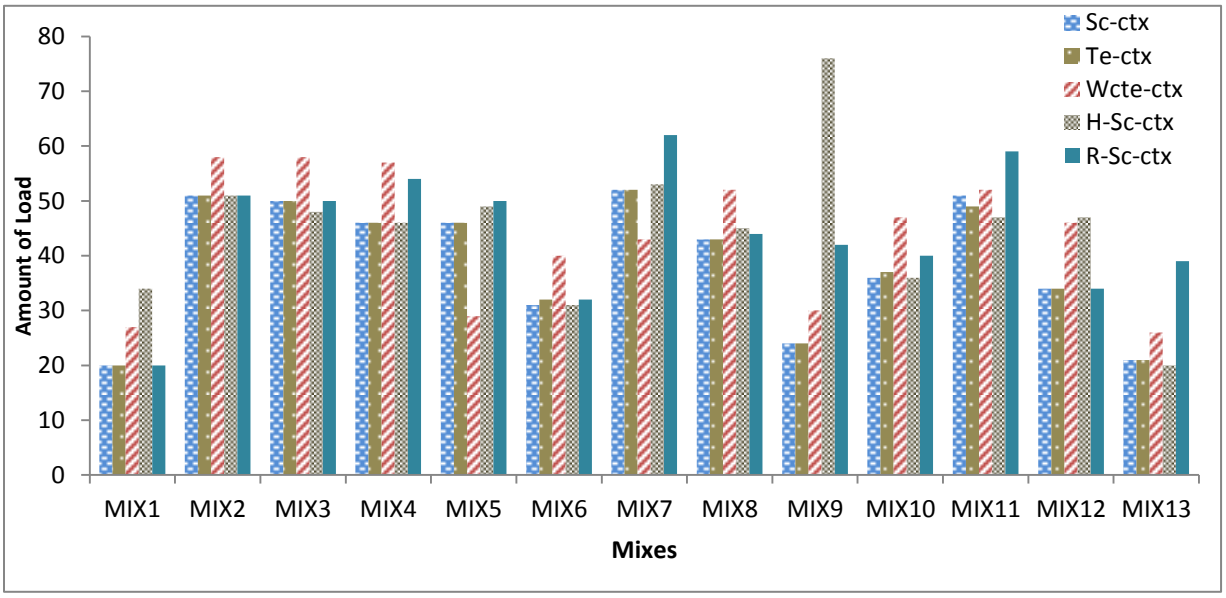


Figure 3 and 4 show the load and a deadlines satistifing comparison between these five algorithms.





5 Conclusion

In this paper we focus on the scheduler for real-time embedded systems that target three objectives: 1- optimal performance, 2- minimum load, 3- maximum satistified deadlines. We consider a real trade-processor to deal with the context switch penalty practically. We present an enhanced algorithm to achieve semi-dynamically partitioning. The experimental results show the proposed algorithm provides, on average, improvement in our three objectives ranging from 5.34% to 8.75%.

References

[1] Wierman, A., and Harchol-Balter, M., "Classifying scheduling policies with respect to unfairness in an M/GI/1." *ACM SIGMETRICS Performance Evaluation Review* 31, no. 1, pp. 238-249, 2003.

[2] Davis, R.I., and Burns, A., "A survey of hard real-time scheduling for multiprocessor systems", *ACM Computing Surveys (CSUR)* 43, no. 4, p 35, 2011.

[3] Liu, Chung Laung, and Layland, J.W.,"Scheduling algorithms for multiprogramming in a hard-real-time environment." *Journal of the ACM (JACM)*20, no. 1, pp. 46-61,1973.

[4] Dellinger, Matthew A. *An experimental evaluation of the scalability of real-time scheduling algorithms on large-scale multicore platforms*. Diss. Virginia Polytechnic Institute and State University, 2011.

[5] Jooya, A., Baniasadi, A., " History-aware, Resource-based Dynamic Scheduling for Heterogeneous Multi-core Processors", *IET Comput. Digit. Tech., Vol. 5, Iss. 4*, PP. 254–262, 2011.

[6] Li, T., Brett, Q., Knauerhase, R., Koufaty, D., Reddy, D., Hahn, S., " Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures", *IEEE International Conference on Networking, Architecture, and Storage*, pp 1-12, 2010.

[7] Hao, S., Liu, P., "Processes Scheduling on Heterogeneous Multi-core Architecture with Hardware Support ", *Sixth IEEE International Conference on Networking, Architecture, and Storage*, PP 236-241, 2011.

[8] Fedorova, A., Vengerov, D. and Doucette, D., "Operating system scheduling on heterogeneous core systems" In *Proc. of OSHMA workshop, 16th PACT*., 2007.

[9] Shelepov, Daniel, Carlos Saez Alcaide, J., Jeffery, J., Fedorova, A., Perez, N., Huang, Z.F, Blagodurov, S., and Kumar, V., "HASS: a scheduler for heterogeneous multicore systems." *ACM SIGOPS Operating Systems Review* 43, no. 2, 66-75, 2009.

[10] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai., "The Impact of Performance Asymmetry in Emerging Multicore Architectures". *Proceedings of the 32nd Annual International Symposium on Computer Architecture (Madison, Wisconsin USA, June 04–08, 2005). ISCA '05. IEEE Computer Society, Washington*, DC, USA, pp 506–517, 2005.

[11] M. Becchi and P. Crowley. "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures", *Proceedings of the 3rd Conference on Computing Frontiers (Ischia, Italy, May 02–05,2006)*, Computing Frontiers '06. ACM, New York, NY, USA, 29–40, 2006.

[12] R. Kumar et al. "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance", *Proceedings of the 31st Annual International Symposium on Computer Architecture (München, Germany, June 19–23, 2004)*, ISCA '04. IEEE Computer Society, Washington, DC, USA, 64, 2004.

[13] T. Li, D. Baumberger, D. A. Koufaty, and Scott Hahn. "Efficient Operating System Scheduling for Performance-Asymmetric Multi-Core Architectures", *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (Reno, Nevada, USA, November 10–16, 2007)*, SC '07. ACM, New York, NY, USA, No. 53, 2007.

[14] J. Mogul et al. "Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems", *IEEE Micro, 28, 3, IEEE Computer Society Press, Los Alamitos*, CA, USA, pp 26–41, 2008.

[15] R. Teodorescu and Torrellas, J., "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors", *Proceedings of the 35th International Symposium on Computer Architecture (Beijing, China, June 21–25, 2008). ISCA '08. IEEE Computer Society*, Washington, DC, USA, pp 363–374, 2008.

[16] Li, T., Baumberger, D., Koufaty, D. A., and Hahn, S.,"Efficient operating system scheduling for performance-asymmetric multi-core architectures", *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, p. 53. ACM, 2007.

[17] Matschulat, D., Marcon, C.A, and Hessel, F., "A Reservation Scheduler for Real-Time Operating Systems", 2008.

[18] Anderson, J. H., Bud, V., & Devi, U. C.,"An EDF-based scheduling algorithm for multiprocessor soft real-time systems." In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pp. 199-208, IEEE, 2005.

[19] Nelissen, G., Berten, V., Nélis, V., Goossens, J., & Milojevic, D., "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks." In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pp. 13-23. IEEE, 2012.

[20] Bautista, D., Sahuquillo, J., Hassan, H., Petit, S., Duato, J., "A Simple Power-Aware Scheduling for Multicore Systems when Running Real Time Applications", *Parallel and Distributed Processing, IPDPS. IEEE International Symposium*, pp. 1-7, 2008.

[21] Krishnamurthy, V., A Novel Thread Scheduler Design for Polymorphic Embedded Systems, Master Thesis, Department of Electrical and Computer Engineering, Iowa State University, 2010.

[22] Tang, H., Ramanathan, P., " Combining Hard Periodic and Soft Aperiodic Real-Time Task Scheduling on Heterogeneous Compute Resources", *International Conference on Parallel Processing* , PP 753-762, 2011.

[23] Cho, M., Lee, C.,"A Low-Power Real-time Operating Systems For ARC(Actual Remote Control)Wearable Device", *Consumer Electronics, IEEE Transactions* , V. 56,PP 1602-1609, 2010.

[24] Tan, S., Nguyen Bao Anh, T., "Real-Time Operating System (RTOS) for Small (16-bit) Microcontroller", *Consumer Electronics, IEEE 13th International Symposium* , PP 1007-1011, 2009.

[25] Hui, C., Shiping, Y., "Reaearch on Ultra-Dependable Embedded Real Time Operating System", *Green Computing and Communicatin (GreenCom), 2011 IEEE/ACM International Conference* , PP 144-151, 2011.

[26] Mera, D.E, Santigo, N.G., "Low Power Software Techniques for Embedded Sytems Running Real Time Operating Systems", *Circuts and Systems (MWSCAS),2010 53rd IEEE Internatipnal Midwest Symposium* , PP 1061- 1064, 2010.

[27] Calandrino, J. M., Anderson, J. H., and Baumberger, D.P., "A hybrid real-time scheduling approach for large-scale multicore platforms." *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pp. 247-258. IEEE, 2007.

[28] Seo, E., Jeong, J., Park, S., and Lee, J., "Energy efficient scheduling of real-time tasks on multicore processors." *Parallel and Distributed Systems, IEEE Transactions on* 19, no. 11, pp. 1540-1552, 2008.

[29] Ubal, R., Jang, B., Mistry, P., Schaa, D., and Kaeli, D., "Multi2Sim: a simulation framework for CPU-GPU computing" *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 335-344. ACM, 2012.

[30] R. Ubal, J. Sahuquillo, S. Petit, and P. López. A simulation framework to evaluate multicore-multithreaded processors, Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing, 2007.

[31] Weaver, David L., and Gremond, t., *The SPARC architecture manual*. PTR Prentice Hall, 1994.

[32] Horel, T., and Lauterbach, G., "UltraSPARC-III: Designing third-generation 64-bit performance", *Micro, IEEE* 19, no. 3, pp. 73-85, 1999

[33] Maruyama, T., "SPARC64 VI: Fujitsu's next generation processor", *Microprocessor Forum 2003*. 2003.

[34] Parulkar, I., Ziaja, T., Pendurkar, R., D'Souza, A., and Majumdar, A.,"A scalable, low cost design-for-test architecture for UltraSPARC™ chip multi-processors." In *Test Conference, 2002. Proceedings. International*, pp. 726-735. IEEE, 2002.

[35] Tsafrir, D., "The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops)", In *Proceedings of the 2007 workshop on Experimental computer science*, p. 4. ACM, 2007.

[36] Li, C., Ding, C., and Shen, K., "Quantifying the cost of context switch", *Proceedings of the 2007 workshop on Experimental computer science*, p. 2. ACM, 2007.

[37] Mogul, J. C., and Borg, A., *The effect of context switches on cache performance*. Vol. 26, no. 4. ACM, 1991.

[38] M. R. time Research Center. Wcet analysis project. Wcet benchmark programs. Technical report, [Online]. Available: http:/www.mrtc.mdh.se/projects/wcet/benchmarks.html, 2006.

MIXes	MIX1	MIX2	MIX3	MIX4	MIX5	MIX6	MIX7	MIX8	MIX9	MIX10	MIX11	MIX12	MIX13
Sc-ipc	1.365	1.784	1.784	1.543	1.702	1.584	1.445	1.817	1.294	1.604	1.485	1.948	1.269
Sc	20	46	44	46	42	26	52	41	26	33	48	34	20
te-ipc	1.362	1.784	1.784	1.777	1.704	1.586	1.797	1.816	1.292	1.686	1.488	1.955	1.267
te	21	47	44	44	44	30	48	42	26	34	48	35	22
wcte-ipc	1.088	1.243	1.271	1.256	1.234	1.265	1.332	1.342	1.118	1.184	1.315	1.274	1.111
wcte	48	61	62	62	49	53	57	61	41	52	55	53	44
H-Sc-ipc	1.063	1.785	1.784	1.777	1.738	1.658	1.802	1.805	0.6623	1.795	1.809	1.277	1.409
H-Sc	32	46	42	42	47	30	47	43	71	33	49	43	20
R-sc-ipc	1.348	1.785	1.773	1.639	1.604	1.649	1.613	1.803	1.197	1.857	1.619	1.808	1.18
R-sc	20	45	45	52	46	27	57	40	42	36	56	35	39
Sc-ctx	20	51	50	46	46	31	52	43	24	36	51	34	21
Te-ctx	20	51	50	46	46	32	52	43	24	37	49	34	21
Wcte-ctx	27	58	58	57	29	40	43	52	30	47	52	46	26
H-Sc-ctx	34	51	48	46	49	31	53	45	76	36	47	47	20
R-Sc-ctx	20	51	50	54	50	32	62	44	42	40	59	34	39
Sc-Cyc	4493856	12944101	12945417	14856791	11552596	6179817	12748116	10746979	5505288	8172547	11333716	8051548	5495845
Te-Cyc	4497151	12944074	12947210	12902284	11538793	6171436	10249847	10750818	5511920	7775907	11314292	8023729	5504443
Wcte-Cyc	5641523	18691735	18180066	18365312	16010916	7739500	13827571	14559309	6370480	11079520	12800844	12312574	6279131
H-Sc-Cyc	5764435	12941715	12946989	12903179	11314600	5902219	10221690	10819840	10755265	7304165	9302895	12282463	4950977
R-Sc-Cyc	4543465	12940967	13028154	14073489	12324893	5936168	11417262	10831637	5950809	7062024	10398658	8675804	5909117