Efficient Implementation of Thermal-aware Scheduler on a Quad-core Processor

Xiaojing Yin[1], Yongxin Zhu[1], Liang Xia[1], Jingwei Ye[1], Tian Huang[1], Yuzhuo Fu[1], Meikang Qiu[2]

1 School of Microelectronics, Shanghai Jiao Tong University, Shanghai, China

2 Dept. of Electrical and Computer Engineering, University of Kentucky, Lexington, KY, USA

yinxiaojing@ic.sjtu.edu.cn, zhuyongxin@sjtu.edu.cn, xialiang@ic.sjtu.edu.cn, yejingwei@ic.sjtu.edu.cn, huangtian@ic.sjtu.edu.cn, yzfu@sjtu.edu.cn, mqiu@engr.uky.edu

***Abstract*—Due to power wall and slow performance improvement in a single core micro-architecture, multiple even many cores based processors rose as the main stream processor. Nevertheless, thermal threats regarding reliability and lifetime of processors are still among the major concerns which received much attention in terms of algorithms and hardware design to reduce processor temperature and keep application performance in recent years. In this paper, we propose and implement a thermal-aware Round-Robin scheduling algorithm for process migration in the Linux environment on a quad-core processor. Bearing designer's goals in mind, such as performance, load-balancing, and reliability, we managed to achieve much bigger temperature fall than previous results of Round-Robin scheduler on a dual-core processor as well as baseline Linux scheduler on a quad-core processor. Moreover, the performance loss due to scheduling overhead is modest in our approach. Our results indicate that thermal-aware scheduling is a valid approach to tackling thermal issues on multi-core processors. There will be increasing demand for thermal-aware scheduling as the number of cores on a single processor increases.**

I. INTRODUCTION

With the booming development of semiconductor technology and constant innovation of computer architecture, multi-core processors have become state-of-the-art in processor architecture to meet tougher requirements for better performance, faster response, more detailed and higher efficiency. Multiple even many cores emerged as collaborative roles coupled tightly in a single chip to yield better overall performance than the equivalent single-core processors by exploring thread level parallelism. It is believed that more cores will be integrated into a chip in the near future.

Meanwhile, multi-core processors is encountering many new challenges, such as interconnect complexity, power consumption, and silicon process yield, among which power management is a major concern from users' perspective. In the course of ongoing push for higher processor performance, users require higher power consumption which further leads to higher temperature that not only decreases system reliability but also results in additional sub-threshold leakage power consumption [1]. In [2], it is explained that the lifetime of electrical circuits would be reduced by half when their temperature increases by 10-15 degrees Celsius. It is quite necessary to avoid the high temperature which turns out to be a major reason for permanent faults when processors run at high temperature due to high clock frequencies. In these years, a widely used technique called as Dynamic Thermal Management (DTM) [3] has been employed to limit peak temperatures through throttling performance when a preset temperature threshold is reached. A typical mechanism of DTM is dynamic voltage and frequency scaling (DVFS) in hardware, which down-scales the voltage and frequency of the chip to ensure a graceful performance degradation.

There are many methods to implement DVFS to reduce power-consumption and decrease processor temperature, such as clock gating [4], body bias [5], and gating level circuit optimization [6]. A large body of work on DVFS has been presented to control temperatures of CPU cores through process migration in these years. The global strategy and the distributed strategy were presented by James Donald [7] to lower the temperature. The global strategy can avert the tough communication problem among cross clock domain to cool all running cores uniformly. The distributed strategy, which has finer granularity, responds more individually to the needs of different applications running on each core while requiring more complex hardware. Some focuses on the leakage power consumption are proposed in [8] which uses the energy-efficient scheduler depending on temperature. Besides on single-core processors, a heuristic scheduling algorithm was presented by Yang et al. in [9], to run the process referring to the precedence and preventing the highest temperature from exceeding the threshold. On the other hand, Almeida et al. in [10] presented a Multi-processor SOC platform which is capable of load balancing at run-time to improve the performance of the system. Musoll et al. in [11] also studied two load balancing schemes to build a power-consumption model on their event-driven simulator to study the scheduler in SMT (Simultaneous Multi-Threading) and CMP (Chip Multiprocessor) to maximize the processor resource. Recently, deep nanometer semiconductor process brings some process variations which produce differences in performance and thermal behavior of multiple cores. Gupta et al. [12] developed a genetic algorithm based on the principles of evolution found in nature for finding an optimal solution and Zhou et al. [13] proposed an operating system (OS) level scheduling algorithm that performs thermal-aware task scheduling on a 3D chip. Furthermore,

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.147

MPSOC architecture based on FPGA hardware platform to simulate the thermal equilibrium scheduling algorithm referring to [14] and [15] was designed and implemented on process migration to balance the load as well as temperature. A system named Kappa [16] was implemented in Linux. It allows for transparent process migration between paired Linux computers to bridge the gap between high-performance computing and home users. Additionally, some work on solving load balancing problem at the operating system level was proposed to optimize for processes maintaining a massive amount of network connections in [17].

Other than most previous work on thermal-aware process schedulers with implementation on a simulator or using a numeric model, we shall present our thermal-aware thread scheduling experiments to evaluate our OS level scheduler for multiple threads on Intel Core 2 Quad Q8200 processor in this paper. We implemented process migration through Round Robin (RR) scheduling algorithm in the Linux 2.6 kernel which supports multiple cores because process migration is meaningful to research. We think that the main reason is the potentials offered by mobility as well as the attraction to hard problems. Process migration is generally regarded as a smart OS level management module which can improve the thermal conditions on chip and alleviate the pressure of performance loss of both workload latency and throughput. We take some representative benchmarks from SPEC2K to evaluate the implementation of our algorithm on multi-core processors, which demonstrates the efficiency of our experiment on reducing the temperatures of the processors and compensating the lack of thermal management of Linux baseline scheduler.

The remainder of this paper is organized as follows. In Section 2, we present the principle of some algorithms on process migration especially RR algorithm which we used in our experiment. The details about design of our thermal-aware RR scheduler will be shown in Section 3. Our experiment results will be presented in Section 4. In Section 5, we will analyze the results and then draw a conclusion.

II. Related Scheduling Algorithms

Due to continuous research efforts in scheduling, many new algorithms are proposed to take thermal relevant issues into account, e.g. power consumption and core temperature. Among those algorithms, Hottest Task Migration (HTM) is widely considered as an ideal algorithm to be adopted in process migration. As the name of algorithm implies, it migrates the hottest running task to the coolest core. The temperature of cores would be decreased and load balance would be alleviated too as explained in [18]. The disadvantage of this algorithm is that the hottest task is not always running on the hottest core because of the difference of technology process. As the similarity of HTM, Coldest Task Migration (CTM) is also applied to reduce core temperatures through migrating the coolest running task to the hottest core. However, it involves the same limitation that it is difficult to distinguish the heat characteristic of task as HTM has. Hottest CPU Migration (HCM) [19] that migrates all the tasks on the hottest CPU core to others could rapidly reduce the hottest core temperature. However, the costs brought out should not be ignored since all tasks on the hottest core are migrated away at the same time. This algorithm takes advantage of HTM and CTM in computing the characteristic of tasks while there is also a reduction causing certain resources to heat more quickly.

However, the heat generated on a single chip cannot be handled simply through threshold-based migration scheduling such as HTM or CTM. More proactive ways would be efficient to balance the heat among all cores. A practical algorithm used frequently is Round Robin (RR). Though RR appears to be one of the simplest algorithms applied to process migration, it shows the fairness to all tasks through scheduling them cyclically [21]. The goal of this algorithm is to minimize the imbalanced task distribution among cores. Further insights into scheduling algorithms on process migration were presented in [20] to improve load balancing and temperature control. In our experiment, we would like to take RR algorithm as our scheduling algorithm to balance the temperature among quad-cores. In terms of control of core temperatures, it is expected that RR performs better on quad-core or many core processors than dual-core processors.

III. Design and Implementation

In this section, we discuss the Linux scheduler mechanism and propose our temperature aware scheduling algorithm.

A. The Linux Scheduler

The Linux scheduler has experienced two major changes since the kernel version 2.6. In Linux 2.4, all CPU cores share one global task *runqueue*(queue of tasks to run). The scheduler in Linux 2.4 is to pick the task which has the highest priority from *runqueue*. Since it takes significant CPU time to select tasks in single queue, the scheduler in Linux 2.6 changed scheduling mechanism to multiple priority *runqueues*. Tasks were selected from the *runqueue* with the highest priority, which could reduce the selection time. Later in Linux kernel version 2.6.23, a new scheduler called Completely Fair Scheduler is introduced to schedule tasks more fairly.

All the Linux schedulers aim to provide a possiblely cheap and fair scheduling algorithm, which usually focuses on system response time and throughput. Temperature distribution of CPU cores is usually ignored in the design of Linux schedulers. Thus, our goal is to add the ability to balance temperature through task scheduling by Linux schedulers.

B. Thermal-aware RR Scheduling in Quad-core Processor

The RR Algorithm is one of the simplest scheduling algorithms for process in an OS, which assigns time slices to each process for equal portions and circular order without priority. The key implementation of the RR Algorithm is to balance temperature among a group of processors to avoid the peak temperature at each interruption. The tasks in the runqueue of each processor would be re-scheduled when the

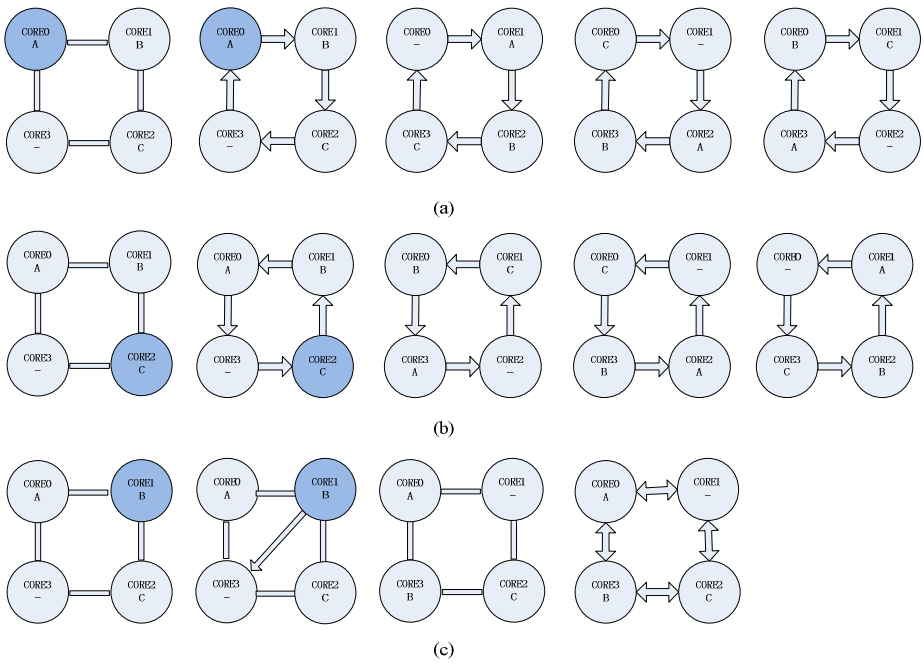

Figure 1. Three-task RR scheduling on a Quad-core Processor

temperature of one of the processor cores exceeds pre-defined threshold. Each task, hot, warm or cool, will have equal chance to dissipate almost the same amount of heat which keeps the temperature on any core not too high as well as the temperature difference among each core not too large. The balanced temperature distribution and reduced peak temperature will be very useful to save the lifetime of processor chip as well as energy cost.

In [21], a RR algorithm applied to the process migration between 2 cores plays an important role in tackling thermal aware. With the increased number of cores, more significant and meaningful reduction in processor temperatures is expected. Therefore we focus on RR for process migration among 4 cores in our experiment.

A RR scheduler will choose the core that happens to have a thermal violation to trigger scheduling tasks clockwise or counter-clockwise. The process migration is achieved through the *pull_task* method that is involved in load balancing function in Linux Kernel code. The *pull_task* method implements to move a task from a remote *runqueue* to the local *runqueue*. Considering the case that the task to be migrated is exactly running on the core, the *pull_task* method cannot be simply called to migrate the task. A flag added into *runqueue* is set to trigger the context switch when the RR Scheduling algorithm starts. It also allows a special procedure to swap out current task and to swap in idle task on each core before the task being migrated to the other cores. As we all know, only a few active tasks are heavily consuming the resources of cores while the rest running concurrent tasks are almost idle. It is reasonable to assume some cores as being idle state with the ever increasing number of cores. The idle task proposed in our algorithm dose not waste the system capability presumptuously.

Without loss of generality, we focus on the situation of three tasks running on Quad Core in our experiment which is shown in Fig. 1. The procedure of RR scheduling can be regarded as a state machine. With the change of state variables represented as the number of tasks and cores which tasks are running on, the algorithm judges which task to be migrated in different case. The RR scheduling algorithm is initiated when the temperature of any core exceeds the threshold detected by sensors. The sensors are used to measure the temperature information of each core after we patch Linux OS kernel with a CoreTemp module. It is convenient for us to grab the CPU temperature synchronously in user mode.

In our design, there are three situations to be taken into account before migration begins. Fig. 1 describes three-tasks thermal-aware RR scheduling on Quad Core. The dark circles in the Fig. 1 denote the core which has a thermal violation compared with other cores at the beginning of process migration. If multiple cores happened to have a thermal violation at the same time, the scheduler would choose the core according to the number of cores in the algorithm. In Fig. 1(a), the temperature of Core 0 where task A running exceeds the threshold first. The clockwise RR scheduling will be executed to migrate the task onto the neighbor core, and reduce the temperature of Core 0. The similar operation is conducted when Core 2 where task C running happens to violate the temperature threshold through counter-clockwise RR scheduling described in Fig. 1(b). However, it cannot reduce the temperature quickly in this scheduling while the task to be migrated is running at

Pseudo Code of N-Task Quad-core RR Algorithm

```
1: Input: task number n=N
2: if T_i >= T_th then
3:   if run time – last migration end time >= 100ms then
4:     switch (n)
5:     case 0 : break;
6:     case 1 : Migrate A in a clockwise direction;
7:     case 2 :
8:       if two tasks are at opposite then
9:           Migrate tasks in a clockwise direction;
10:      else if two tasks are neighbor then
11:          if A is at the head then
12:              Migrate tasks in a clockwise direction;
13:          if A is at the tail then
14:              Migrate tasks in a counter-clockwise direction;
15:    end if
16:    case 3 :
17:      if Task A is at head of three tasks then
18:          Migrate  tasks in a clockwise direction;
19:      else if Task A is at tail of three tasks then
20:          Migrate tasks in a  counter-clockwise direction;
21:      else if Task A is at middle of three tasks then
22:          Migrate Task A to the idle core;
23:          goto 16;
24 :     end if
25:end if
```

Figure 2. RR Algorithm for N-Task on a Quad-core Processor

the middle core in the RR scheduling direction described in Fig. 1(c). One more step, which is to migrate the task at the middle core to the opposite idle core, is added in order to convert to the same state as Fig. 1(a) or Fig. 1(b) shows. Then the clockwise or counter-clockwise RR scheduling will take effect. After determining the task to be migrated, we would trigger context switch and then use *pull_task* function to achieve this operation.

The pseudo code of our Thermal-aware RR algorithm is described in Fig. 2. We supposed that Task A is running on the core of whose temperature exceeds the threshold firstly. Since Linux default scheduler assigns a time slice which is a default value, 100ms, to each task, we set a check point at every 20 ms in the OS scheduler tick. Besides, the new migration is allowed to start when the last migration is finished for least 100ms, whose guarantee of interval time reduces the cost brought by frequent migration.

Our algorithm is to reduce core temperature as quickly as possible through making the core which has a temperature violation be idle in next clock tick. It is achieved by migrating the tasks according to the relative positions of each core. If the number of tasks is one, we just use RR scheduler to migrate the task in the clockwise direction. If the number of tasks is two, it is the same as the situation above when these two tasks are at the opposite positions. It is slightly complicated when these two tasks are at neighbor positions. The task on the core of which the temperature reaches the threshold is migrated in the clockwise direction if it is ahead of the other, and vice versa. While the number of tasks is three, another case we should consider is that the task to be migrated is at the middle of three tasks. Neither of the clockwise and counter-clockwise direction to migrate is able to make the core idle. So one more step is added if the core of which the temperature exceeds the threshold happens to be the middle core in the direction, the task on the core will be first migrated to the opposite idle core, and then the RR scheduler takes effect. In short, we regard the tasks on the cores as a linked list which is shifted left or right rotatable in terms of the demand for RR scheduler algorithm.

IV. Experiment and Results

A. Experimenl Setup

To make use of the kernel scheduler, we chose Intel Core 2 Quad-core Q8200 processor to implement our algorithms. The experiment environment is Ubuntu 8.04, whose kernel version is Linux 2.6.28.7. We choose 8 SPEC CPU2000 benchmarks in our experiment to evaluate our design.

B. Benchmark Classification

SPEC CPU2000 is an suite which has set up CPU benchmark search program designed to encourage and recognize achievements of the academic and industrial community in providing and developing application code. We divide these benchmarks into different grades based on the temperature of the core being run on. We use Linux command “taskset” to bind them to a certain core and sample the temperature every 10 seconds after the temperature is not bounced dramatically. Fig. 3 depicts the temperature profiles of the benchmarks we select to test in

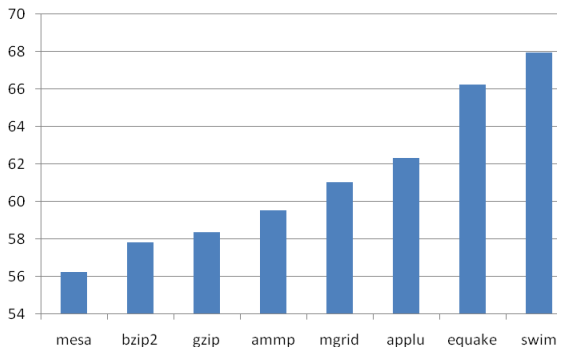

Figure 3. Temperature profiles of SPEC2000 benchmarks

TABLE I. CLASSIFICATIONS OF TASKS

Benchmark	T(°C)	Thermal Group
mesa	56.21	Cool
bzip2	57.81	Warm-Cool
gzip	58.36	
ammp	59.53	Warm-Hot
mgrid	61.03	
applu	62.29	Hot
equake	66.22	
swim	67.93	

TABLE II. COMBINATIONS OF TASKS

Temperature Balancing	
H-H-H	swim + equake+applu
H-Wh-Wh	swim + mgrid +ammp
H-Wh-Wc	swim + mgrid + gzip
H-Wc-C	swim + bzip2 + mesa
Wh-Wc-C	mgrid + bzip2 + mesa
Wc-Wc-C	gzip + bzip2 + mesa
Latency	
H-H-H	swim + swim + swim
Wc-Wc-Wc	gzip + gzip + gzip
C-C-C	mesa + mesa + mesa

our experiment. According to the sorting shown in Fig. 3, we can classify them into different groups, Hot process, Warm-Hot process, Warm-Cool process and Cool process, which are described in Table I.

Moreover, we select the benchmarks in different heat characteristic to combine into different process load for evaluating the scheduler performance. It is difficult for us to evaluate both the performance and time-consumption of our thermal-aware scheduler due to the different characteristic of these benchmarks. So we also combine some tasks for the time measurement according to the temperature and heat characteristic. Table II just shows the combination of tasks which we used in our experiment.

C. Experimental Results

We implemented our Thermal-aware RR scheduler and Linux build-in scheduler into the Linux kernel whose version is 2.6.28.7 on Intel Core 2 Quad Q8200 and set a flag to trigger each of them for temperature monitoring. Our experiments are studied in the environment where the room temperature is 26 ℃. Each set of data we got is measured at an interval of about half an hour in order to keep them as actual as possible. We not only focus on the temperature balancing but also latency in our experiment.

1) Temperature Balancing: The selected combined tasks would be run cyclically and it is also guaranteed that there are three different benchmarks running on the cores from the beginning of task scheduling. Firstly, we check the initial temperature of each core and set the threshold temperature. In our experiment, the threshold temperature is set to 48℃. The RR algorithm kicks off once one of the cores' temperature reaches the threshold temperature. The temperature of cores which we observed after scheduling for a certain time is shown in Table III. TS, namely Thermal-aware Scheduler, illustrates that the scheduling is based on our thermal-aware RR algorithm in the table while NO-TS means the Linux build-in scheduling algorithm.

It is clearly that the scheduler based on our thermal-aware RR algorithm is better in balancing the temperature of cores than Linux build-in one in Table III. Each temperature measured in TS in any cores is lower than that measured in NO-TS. For example, RR scheduler lowered the temperature on Core 2 by 5℃ in the group of H-Wc-C. Similar result can be seen in other groups as well. Moreover, the temperature of each core is heavily dependent on the original position of tasks based on Linux build-in scheduler. It causes that the temperature gap is considerable. Besides, as we all know, the temperature gap implies the adept of balancing the temperature of each core. The scheduler would be preferred if it has effect on the reducing temperature gap among processors. It illustrates that the thermal-aware RR algorithm scheduler is able to narrow the temperature gap effectively in Table IV. The temperature gap in Linux build-in scheduler is nearly unchanged while it is reduced over a quarter in all combinations of tasks through our RR scheduler.

We can know that our thermal-aware scheduler takes more effect than Linux build-in scheduler to reduce the temperature as well as the the average temperature

TABLE III. CORE TEMPERATURE EXPECTATION USING VERSUS NOT USING TEMPERATURE-AWARE SCHEDULER

Combination of Tasks	Core0(℃)		Core1(℃)		Core2(℃)		Core3(℃)	
	TS	NO-TS	TS	NO-TS	TS	NO-TS	TS	NO-TS
swim+equake+applu	64.6	66.13	62.17	64.96	68.28	72.65	66.86	70.31
swim + mgrid +ammp	66.15	70.08	64.21	69.53	69.58	78.46	66.75	79.15
swim + mgrid + gzip	69.97	73.25	66.14	72.47	71.93	80.02	68.73	81.48
swim + bzip2 + mesa	67.20	71.25	65.33	70.48	71.02	79.70	67.95	80.15
mgrid + bzip2 + mesa	68.58	69.70	66.23	67.18	71.47	75.05	69.08	77.18
gzip + bzip2 + mesa	68.88	69.60	66.05	69.79	71.68	76.33	68.90	78.48

TABLE IV. TEMPERATURE GAPS IN DIFFERENT COMBINATION

Combination of Tasks	TS(℃)	NO-TS(℃)	Proportion
swim+equake+applu	5.58	7.69	27.43%
swim+mgrid+ammp	5.37	9.62	44.18%
swim + mgrid + gzip	5.765	9.01	36.90%
swim + bzip2 + mesa	5.69	9.67	41.58%
mgrid + bzip2 +mesa	5.24	10.04	47.80%
gzip + bzip2 + mesa	6.63	8.88	25.34%

TABLE V. AVERAGE TEMPERATURE EXPECTATION OF DIFFERENT COMBINATIONS

Combination of Tasks	TS (℃)	NO-TS (℃)	Proportion
swim + equake+applu	65.47	68.48	4.38%
swim + mgrid +ammp	66.67	74.33	10.30%
swim + mgrid + gzip	69.19	76.81	9.92%
swim + bzip2 + mesa	67.87	75.40	9.98%
mgrid + bzip2 + mesa	68.84	72.28	4.76%
gzip + bzip2 + mesa	68.87	73.6	6.4%

TABLE VI. PERFORMANCE LOSS OF SCHEDULERS

Combination of Tasks	TS(s)	NO-TS(s)	Proportion
mesa + mesa + mesa	903	837	7.88%
gzip + gzip + gzip	481	437	10.01%
swim + swim + swim	554	512	8.2%

expectation shown in Table V. The average temperature in our thermal-aware RR scheduler on Quad-core is generally lower than that in Linux build-in scheduler from5% to 10%. Besides, the effect of reducing temperature is more pronounced than that in [21]which implemented the RR scheduler on a dual-core processor. In [21], the authors also chose SPEC2000 benchmark to evaluate their design results. The peak temperature (PT) drops after implementing the RR scheduler in their expriment are between 1 to 3 degrees Celsius. Our implementation of RR scheduler on the quad-core processor lowers the temperature by more than 5 degrees Celsius. The result demonstrates intuitively that our scheduler is much more effective in dominating the temperature to improve the reliability and extend lifetime of processors.

Besides, as shown in Table III, we find an interesting phenomenon that the highest temperature does not come from the hottest combination task. Instead, the temperature of this hottest combination task is the lowest in all combination tasks. It seems not reasonably to match our common sense. We believe that the reason is complementarity in computing characteristic of each process. The heat different calculation units give out is more than that the same calculation unit running repeatedly gives out while CPU is suffering the high-speed operation.

2) Latency: Consider that the time-consumption is a significant measure in process migration, so we also calculate the workload latency which is impacted by scheduling since it enforces context switch through running the same benchmark on the cores. The value of research on scheduling algorithm would be lowed enormously if the performance loss cannot be ignored because the performance not the temperature is accessible to users. We take three groups which involve the same tasks to run in a certain period of time. The longer it spends on running tasks, the larger the performance loss is.

Table VI shows the performance loss of schedulers in our thermal-aware RR algorithm and Linux build-in scheduler algorithm. We can see that the three combinations of tasks are all lost about 10% in time-consumption. Considering that the significance of temperature decrement, it is acceptable for the performance loss while using our scheduler to balance the temperature of cores.

V. CONCLUSION

We presented design and implementation of thermal-aware RR scheduler instances on a quad-core

processor. In our experiments with 8 SPEC2000 benchmarks and their combinations, we demonstrated the efficiency of thermal-aware RR scheduler in balancing the temperature on a physical processor in real time. Other than most previous works on thermal-aware process schedulers with implementation on a simulator or using a numeric model, we implemented our scheduler on a real multi-core processor which is more convincing and valuable for practical applications. The results we obtained show that our scheduler brings up to 47% more temperature falls compared with baseline Linux scheduler while the performance of applications is not compromised much. In addition, compared with temperature drop of 1-3 degrees Celsius by previous implementation of RR on a dual-core processor, our implementation of RR on the quad-core processor brings temperature drop by 5 degrees Celsius to outperform previous implementation with a little performance decreased. We believe that more reduction of e temperature would be observed if our thermal-aware RR Scheduler is extended to apply on many-core processors. We also suggest users and peer researchers to choose proper combinations of benchmarks and reasonable configuration of time slices in our thermal-aware RR scheduler if our experience is adopted in their future work.

ACKNOWLEDGMENT

This paper is partially sponsored by the National High-Technology Research and Development Program of China (863 Program) (No. 2009AA012201) and the Shanghai International Science and Technology Collaboration Program (09540701900) as well as NSFC 61071061 and the University of Kentucky Start Up Fund.

REFERENCES

[1] David Brooks, Robert P. Dick, Russ Joseph, Li Shang, Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors, IEEE Micro, vol. 27, no. 3, pp. 49-62, May/June 2007.

[2] J. Choi, C-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware task scheduling at the system software level. In International Symposium on Low Power Electronics and Design, pages 213–218, 2007.

[3] David Brooks,Margaret Martonosi.Dynamic Thermal Management for High-Performance Microprocessors.hpca, pp.0171, Seventh International Symposium on High-Performance Computer Architecture (HPCA'01), 2001.

[4] Inchoon Yeo, Chih Chun Liu, Eun Jung Kim. Predictive dynamic thermal management for multicore systems. DAC '08 Proceedings of the 45th annual Design Automation Conference.

[5] Jungsoo Kim, Younghoon Lee, Sungjoo Yoo, Chong-Min Kyung. An analytical dynamic scaling of supply voltage and body bias exploiting memory stall time variation. ASPDAC '10 Proceedings of the 2010 Asia and South Pacific Design Automation Conference.

[6] Pramod Kumar Meher, Shen-Fu Hsiao, Chia-Sheng Wen, Ming-Yu Tsai, "Low-Cost Design of Serial-Parallel Multipliers Over GF(2^m) Using Hybrid Pass-Transistor Logic (PTL) and CMOS Logic," ised, pp.131-134, 2010 International Symposium on Electronic System Design, 2010

[7] James Donald, Margaret Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. isca, pp.78-88, 33rd International Symposium on Computer Architecture (ISCA'06), 2006.

[8] Chuan-Yue Yang, Jian-Jia Chen, Lothar Thiele, Tei-Wei Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. European Design and Automation Association 3001 Leuven, Belgium, Belgium 2010.

[9] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, Lingling Jin. Dynamic Thermal Management through Task Scheduling. ispass, pp.191-201, ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software, 2008.

[10] Gabriel Marchesan Almeida, Sameer Varyani, Rémi Busseuil, Gilles Sassatelli, Pascal Benoit, Lionel Torres, Everton Alceu Carara, Fernando Gehm Moraes. Evaluating the impact of task migration in multi-processor systems-on-chip. SBCCI '10, the 23rd symposium on Integrated circuits and system design.ACM New York, NY, USA 2010.

[11] Enric Musoll. A Thermal-Friendly Load-Balancing Technique for Multi-Core Processors. isqed, pp.549-552, 9th International Symposium on Quality Electronic Design (isqed 2008), 2008.

[12] Sachi Gupta, Vikas Kumar, Gaurav Agarwal. Task Scheduling in Multiprocessor System Using Genetic Algorithm. icmlc, pp.267-271, 2010 Second International Conference on Machine Learning and Computing, 2010.

[13] Xiuyi Zhou, Jun Yang, Yi Xu, Youtao Zhang, Jianhua Zhao. Thermal-Aware Task Scheduling for 3D Multicore Processors. IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 1, pp. 60-71, Jan. 2010.

[14] Fabrizio Mulas, Michele Pittau, Marco Buttu, Salvatore Carta, Andrea Acquaviva, Luca Benini, David Atienza, Giovanni De Micheli. Thermal Balancing Policy for Streaming Computing on Multiprocessor Architectures. date, pp.734-739, 2008 Design, Automation and Test in Europe, 2008.

[15] Abhishek Bhattacharjee,Gilberto Contreras,Margaret Martonosi. Full-system chip multiprocessor power evaluations using FPGA-based emulation. islped, pp.335-340, Proceeding of the 13th international symposium on Low power electronics and design (ISLPED '08), 2008.

[16] Gaurav Mogre, Avinash H, Alwyn R Pais. Kappa: A system for Linux P2P Load Balancing and Transparent Process Migration, HiPC-2010, Student Research Symposium.

[17] Balazs Gerofi, Hajime Fujita, Yutaka Ishikawa. An Efficient Process Live Migration Mechanism for Load Balanced Distributed Virtual Environments. cluster, pp.197-206, 2010 IEEE International Conference on Cluster Computing, 2010.

[18] A. Merkel, F. Bellosa.Balancing power consumption in multiprocessor systems. SIGOPS Oper. Syst. Rev., 40(4):403–414, 2006.

[19] Kevin Skadron, Tarek Abdelzaher, Mircea R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. hpca, pp.0017, Eighth International Symposium on High-Performance Computer Architecture (HPCA'02), 2002.

[20] Nalini Vasudevan, Prasanna Venkatesh. Design and Implementation of a Process Migration System for the Linux Environment. 3rd International Conference on Neural, Parallel and Scientific Computations August 9-12, 2006.

[21] Liang Xia, Yongxin Zhu, Jun Yang, Jingwei Ye, and Zonghua Gu. Implementing a Thermal-aware Scheduler in Linux Kernel on a Multi-core Processor. The Computer Journal Oxford University Press on behalf of The British Computer Society (2010) Volume: 53, Issue: 7, Pages: 895-903