Transactions on Engineering and Computing Sciences - Vol. 12, No. 06
Publication Date: December 25, 2024
DOI:10.14738/tmlai.1206.18051.
Agrawal, M. & Mehta, D. A. (2024). Thermal Aware Process Scheduling for Multicore Processors. Transactions on Engineering and Computing Sciences, 12(06). 39-53.

Thermal Aware Process Scheduling for Multicore Processors

Mahima Agrawal
Computer Engineering Department,
S G S Institute of Technology and Science, Indore, M.P., India

D. A. Mehta
Computer Engineering Department,
S G S Institute of Technology and Science, Indore, M.P., India

ABSTRACT

Multi-core processors seem to be an alternative way to higher frequencies for increasing microprocessor performance, by handling more work in parallel at lower frequencies. The addition of multiple cores on the same chip results in an increase in power density on the chip which in turn generates large amount of heat. The increased temperature increases leakage current; and negatively affects chip's performance, reliability and life expectancy. In addition, they release greenhouse gases in the atmosphere and have negative impact on the environment. The study done so far reveals that the issue of high temperature can be solved by assigning and migrating the processes to a cooler core; but this increases migration cost and temporal temperature gradients thereby decreasing the performance. So, in this work the issue of high temperature along with temporal temperature gradients of the cores is addressed at the Operating System level via scheduling of processes. The experiments are performed on an Intel i5-3470 Linux machine. The experimental results reveal reduction in the peak temperature of the processor up to 11.36%, thermal swings in the processor up to 80% and turnaround time of the processes up to 17.24%. The idea of thermal aware scheduler presented in this paper can be applied for scheduling of jobs in data centers and high-performance computers to achieve performance while making computing environmentally friendly.

Keywords: Completely Fair Scheduler, Green Computing, Linux, Reliability, Scheduling, Thermal metric.

INTRODUCTION

Performance was the ultimate goal of the IT designers until there was no need for energy-aware computing. Earlier high performance was achieved by increasing the frequency of processors. Later, it was discovered that processors running at high frequency causes increased power consumption. This led to the invention of multi-core processors where two or more cores reside on the same physical chip. Though multi-core processors run at low frequency but increases the transistor count on the chip. With each successive generation, this increasing number of transistors increased the power consumption which in turn, with the decrease in chip size increased the power density (i.e., the power consumed per unit area of the chip). The power density increases approximately by a factor of k^2 every generation [1]. Also, Shekhar Borkar an Intel Fellow says, 'with leakage power dominating, power consumption is roughly proportional

to transistor count' [2]. Intel has already yielded two experimental chips of 48 and 80 cores through its Terascale Computing Research Program and is supposed to integrate 256 billion transistors on a feature size of 8nm through its Exascale Computing Research Program. Therefore, multi-core systems will continue to face high power densities. The exponential increase in power density will result in increase in temperature. Moreover, diversity of applications also contributes in increase in power and temperature as applications usually have different thermal intensity. Different applications running on a system may contribute differently to the overall temperature, depending on the way they utilize the processor resources. For instance, while doing typical word processing, email tasks, the processor is idle most of the time and will run very cool whereas while running complicated tasks like 3-D rendering, games, the processor is more active and heat up frequently.

Each device incorporates a processor to deliver performance whilst generating greater amount of heat thus increasing CO_2 emissions. These CO_2 emissions adversely affect health and contribute towards global climate changes. As the temperature of the processor increases, there is an increase in leakage current which in turn increases the temperature again leading to thermal runaway. Also, when the temperature of the integrated circuit in processor rises beyond a specified rate, the operating behavior of the circuit changes thus degrading performance. In addition, the likelihood of catastrophic failure (through mechanisms such as electro-migration, junction fatigue, gate oxide breakdown, thermal runaway, package meltdown, etc.) also increases exponentially thus negatively affecting reliability. The negative effect of high temperature is not only limited to chip's reliability and life expectancy but also to smaller mobile devices with limited battery capacity and to ICT (Information and Communication Technology). Further, the growing use of Internet, Web Applications, E-commerce and Cloud Computing to store and retrieve information digitally requires large number of servers and data centers. These servers and data centers have higher performance power demands which leads to increase in power density, high temperature and cooling mechanism; thereby increasing overall operational costs and raising environmental concerns like increasing CO_2 emissions. Moreover, reliability of hardware is an important aspect of green computing because it reduces the overall costs of energy associated with system failures and E-waste. Increasing the reliability of the IT infrastructure leads to significant energy savings while minimizing hazardous waste materials and their disposal. [3] With the growing computing needs, restrictions on energy supply and access, and global climate changes, it is the need of hour to manufacture, use, design and recycle the IT equipments efficiently and effectively with minimal or no impact on the environment. So, in the current scenario it is extremely important to design an optimal global solution to address the issues of saving energy, reducing cooling costs, increasing performance and reliability of the processor. However, technically the major temperature induced problems for multi-core processor are the absolute rise in temperature, the spatial temperature gradient and the temporal temperature gradient.

The rise in processor temperature can damage the processor components owing to thermal runaway, thereby reducing the reliability of the processor. To reduce thermal resistance and improve heat flow away from the chip, the hardware designers used solutions like heat sinks, air cooling, liquid-cooling, phase-change cooling, efficient floor planning. However, as the chip size continues to shrink, the integration of these cooling techniques becomes expensive. The

other hardware measures currently used to lessen the generation of heat on the chip are clock gating, fetch toggling, localized throttling, idling the CPU etc. All of these approaches have the objective to reduce overheating of cores when the temperature of the processor reaches beyond a threshold; but each method varies in the amount of performance degradation. The performance is degraded as these approaches delay the execution of instructions or reduce the frequency at which instructions are executed. Moreover, these techniques lead to temporal temperature gradient as the temperature falls abruptly. Therefore, such hardware measures should only be applied if really necessary.

The spatial temperature gradient, i.e. the temperature difference between two adjacent cores at a given time, leads to clock skew, negative bias temperature stability and hot carrier junction effects, resulting in performance and reliability degradation. The spatial temperature variation of the core is minimized by employing task allocation and task migration at Operating System (OS) level. However, these techniques lead to thermal swings giving rise to temporal temperature variation of the core as the migration of a hot task to a cold core results in an increase in temperature because the hot task rises the temperature.

The temporal temperature gradient, i.e. the temperature variation a core experiences within a short time interval, leads to thermal cycling (temporal temperature fluctuations/ thermal swing) that may cause acceleratedpackage fatigue and plastic deformations of materials, leading to permanent failures. The issue of temporal temperature gradient can be resolved by employing effective scheduling policy.

In order to have a balance between heat generation and performance, software based thermal management is required. In this work, the problem of temporal temperature variation of the core is tackled by adopting a software-based Dynamic Thermal Management (DTM) technique which selects the next process to run on the core on the basis of the execution profile of the process. To implement the thermal aware scheduling, the Completely Fair Scheduler (CFS) of Linux OS is considered as a case study. Linux OS is referred because it is an open source, widely used OS.

RELATED WORK

To attain high performance along with reduced temperature, the researchers conceived of task scheduling techniques at the Operating System level. Donald and Martonosi [4] classified thermal control techniques for multi-core processors by simulating Stop and Go technique; and DVFS technique both globally and distributed; with and without thread migration. Choi et al. [5] proposed heat balancing by the assignment of hot and cold tasks to each core in a thermally balanced manner. The other technique postpones the execution of hot job and allows the colder jobs to run, runs less number of hardware threads in SMT and to utilize a cool loop in a single threading environment. Yeo, Liu, and Kim [6] proposed a task migration scheme for multi-core processors where migration occurs when the predicted temperature exceeds the migration threshold. The future temperature is predicted based on the transient variations in application temperature, steady state temperature and workload. The core having the minimum value of future temperature is selected as the new core for migration. The thermal model used in [6] is a statistical workload dependent model and requires a training phase while Ayoub, and Rosing

[7] proposed a new thermal predictor for multi-core processors, based on the band limited property of the temperature spectrum which is workload independent. The authors suggested to migrate the most active thread from the hot core to the colder core if the coldest core is idle else to migrate the thread with the modest activity. Liu et al. [8] proposed a dynamic 0^{th} moment temperature based indicator to perform scheduling at the start of each execution cycle. The approach assigns the heavy loaded task to the core that has the lowest effective initial temperature. Gomaa, Powell, and Vijaykumar [9] proposed two methods: The Heat-and-run thread assignment (HRTA) and The Heat-and-run thread migration (HRTM) to balance the chip temperature online. The HRTA co-schedules threads that use complementary resources on the same core. The HRTA schedules the IPC of all threads across available SMT cores to distribute the power density evenly across all the cores. The HRTM migrate threads away from overheated cores to free SMT contexts on alternate cores using HRTA. Stavrou, and Transcoso [10] proposed the Coolest Core scheduling algorithm where the process is always executed on the coolest core. They also proposed the Maximum Scheduling Threshold heuristic where a process is not scheduled for execution if the temperature of the core is above a certain predefined threshold. Musoll [11] proposed a new load balancing technique for allocating a new task “waiting idle first” (WIF) for multi-core processors. The new task is allocated to the most recent core (if available) that has completed a task else follows a round-robin policy. Xia et al. [12] rotates all the tasks on a multi-core system from one core to the next core when the temperature on any core exceeds a predefined threshold. Liu, Fan and Quan [13] predicted the temperature using the thermal history of the local core and the neighbouring cores. Their migration policy considers the core with the smallest 'heat index' and 'heat index increasing factor' as a destination core for migration. Salami et al. [14] suggest co-scheduling of the complementary threads on individual SMT cores. One of the strategies is to reschedule the tasks between the core that is in critical condition and the predicted coolest core. If on rescheduling some of the core is in critical condition then they decrease the processor frequency, but when the performance degrades, they increase the global frequency. Salami et al. proposed the adaptive threshold temperature based on the total number of migrationsof a task. Coskun et al. [15] proposed the Adaptive Random policy where a probability value is assigned to the cores on the basis of thermal stress on the core. The task is assigned to the core having the least probability value.

The general idea of most of the previous work [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,16] is the assignment and migration of the hot task away from an overheated core to a cooler core. The number of migrations due to thermal imbalance can be either a single swap or equal to the number of cores as in [12]. A high number of migration results in too many context switches thereby affecting the performance. The migration of a hot task to a cold core not only degrades performance, but also causes temperature fluctuations resulting in thermal cycling effect. Hence, in this work scheduling of tasks assigned to each core is proposed in such a manner that thermal swings and high temperature of the core is reduced.

COMPLETELY FAIR SCHEDULER

The Linux Kernel uses the CFS that divides the scheduler in two main components: three Scheduling Classes, which implement the policy details, and a Scheduler Core, which handles the Scheduling Classes as objects, i.e., calling the appropriate Scheduling Classes methods for

any low-level operations (for example, selecting the next task to run or accounting for the time elapsed) [17]. Whenever the CFS has to select the next task for execution, it starts with the highest priority class and selects the highest priority process in that highest priority class. If the class has no runnable process available, then the Scheduler Core moves to the next scheduling class. This operation repeats until the Core Scheduler finds a runnable task to run on the CPU.

The CFS algorithm is based on the idea of an ideal multi-tasking processor by giving all running tasks a fair share of the processor. The share for a task changes with the total number of running tasks. In reality, only one task can be executed on the processor at once, so the concept of virtual runtime \textit{vruntime} was introduced. The virtual runtime of a task is the actual runtime normalized by the number of running tasks. A smaller value of process's virtual runtime means that the smaller amount of time a task has been permitted to access the processor, hence more it is in need for the processor. Thus, CFS selects the process with the smallest vruntime for execution. The CFS maintains a time-ordered red-black tree data structure where all runnable tasks are sorted by their vruntime. A red-black tree is a self-balancing binary search tree in which the left-most node is the node holding the smallest key value. Thus, the left-most node holds the task having the smallest vruntime. Whenever a task is to be executed CFS picks the left-most task in O(1) time by caching the left-most node. As the time passes, the vruntime of the running task increases at every timer interrupt (or scheduling event). When vruntime of the running task increases high enough such that another task becomes the leftmost task, the task is moved to the right side of the red-black tree, and the CFS scheduler selects another leftmost task to execute.

PROPOSED APPROACH

In this work, a thermal-aware scheduler for multi-core processors has been designed that alters the sequence of processes assigned to the core for execution on the basis of thermal metric. The thermal metric is a relative term for a process that gives the estimation of the probability of rise in temperature of the core on the basis of the process characteristics. The thought behind the approach of reordering the sequence of a core's processes for execution on the core is that: if the processes allocated to the individual cores are run in a thermally aware manner, then the temperature of the individual cores would not reach the temperature threshold and hence there would be no need to trigger hardware based DTM or migration.

The thermal aware scheduler selects the tasks in decreasing order of their thermal metric to execute on the processor. It is observed that in a given scheduling interval if the colder process is selected for execution in the last then the temperature of the core at the end of the interval is less as compared to the temperature of the core when the hotter process is selected for execution in the last. Thus, if the first process selected for execution is a hot process followed by colder processes then it will eventually lead to a lower temperature at the end of the scheduling interval. This also prevents thermal cycling [19].

The architecture of thermal aware process scheduler for multi-core processors shown in Fig. 1 is composed of two components for each core of a multi-core processor namely thermal metric calculator and thermal aware scheduler.

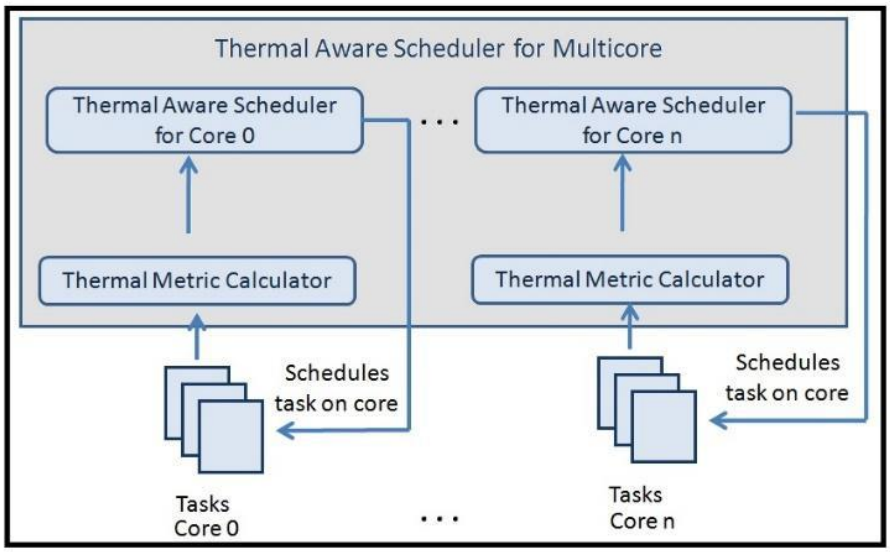

Figure 1: Architecture for Thermal Aware Scheduler for Multicore Processor

Thermal Metric Calculator

This component is responsible for computing the thermal contribution of each process. The thermal contribution of a process is determined by accounting the amount of heat generated by the process with respect to different hardware events. The thermal metric is the summation of n_i number of times the weighted i^{th} event occurs as in equation 1. A weight w_i is associated with each event i on the basis of relation between the event and the temperature of the core in execution.

$$Thermal\ Metric\ = \Sigma n_i \times w_i \qquad \text{(eq. 1.)}$$

The runtime thermal contribution of various hardware events can be determined using Hardware Monitoring Counters (HMC). The HMC are also known as performance monitoring counters and performance event counters. The calculation of thermal metric relies on HMC to track architectural characteristics. The pseudo-code for thermal metric calculator is given in Listing 1.

```
Input: process p_i
Output: thermal_metric_i of the p_i
Procedure:
BEGIN
   while (p_i in execution) do
    count hardware events using perf_event_counters
    thermal_metric_i = 4 * perf_event_counter1 + 3 *
    perf_event_counter2 + 2 * perf_event_counter3 +
    perf_event_counter4
end while
END
```

Thermal Aware Scheduler

The thermal aware scheduler selects the processes in decreasing order of their thermal contribution to the cores for execution on the core. The thermal metric of the processes from the Thermal Metric Calculator component is given as input to the Thermal Aware Scheduler component to select the sequence/order of processes for execution. The modified thermal aware scheduler inserts the processes in the red black tree on the basis of vruntime and thermal metric respectively as compared to the existing scheduler where the insertion of process solely depends on the vruntime of the process. Thus, the modified CFS schedules the processes in a thermally aware manner and at the same time maintains the fairness and interactivity of the existing CFS, as the structure of existing CFS is not altered.

IMPLEMENTATION AND EXPERIMENTS

To implement the proposed approach, the experiments are performed on a quad-core Intel i5-3470 processor having 22nm Ivy Bridge processor architecture with CentOS 6.4 having Kernel 2.6.32. All the experiments are performed using Phoronix-Test-Suite.

The foremost task for implementing the proposed approach includes the quest for selection of hardware events, total number of hardware events and the weights assigned to each event. For this purpose, profiling of different tests namely aio-stress, compress-gzip, dhrystone, encode-mp3, floating-arithmetic, integer-arithmetic, register-arithmetic and sunflow from the Phoronix-Test-Suite for various hardware events is conducted. To get the thermal statistics of the events, these tests are run, and the run-time system-wide statistics of the events are collected by the OProfile tool at an interval of 2 seconds. The OProfile tools gives the statistics about the counts of the listed events using *ocount* command. Simultaneously, the temperature of the cores is also recorded at every 2 seconds using a Kernel module that read the thermal sensors available on the cores. In order to monitor and control the above readings, a shell script is implemented. This imparts us the knowledge about how much count of an event can increase the temperature in 2s with respect to previous reading. This also helps to know the number of thermal cycles. Thus, the event which is supposed to increase the temperature is assigned the highest weight.

The examination of this experiment revealed that not only the type of instructions executed by the processor contribute to the rise in temperature but also, the different resource stalls contribute for increase in temperature; during these stalls the different units of processor are occupied and the hardware attempts to extract instruction-level parallelism from the instruction stream. For example, if a fetched instruction stalls, the out-of-order logic tries to find another to execute, examining reservation stations to check each new instruction's dependences and using more power resulting in more temperature. Moreover, the author in [18] says that whenever there is a cache miss, each external access requires many clock cycles and at least by two orders of magnitude more power than on-chip access. This asserts that the relation between a hardware event and the temperature of the core can be truly known using HMC. Also, the event count for the same process is found to vary from time to time as the behaviour of the execution of a process depends on the available resources.

The maximum number of MSRs available within the processor are four, so the total number of

hardware events that can be selected is restricted to four. In order to select the most suitable set of four events giving an optimal temperature performance balance, the modified CFS is experimented against four sets. Each set is unique in itself containing the events based on the type of instructions, resource stalls and memory misses, and the event temperature correlation. The first set comprises of the events based on types of instructions executed. The second set is a combination of events for executed instruction, retired instruction and stalls. The third set is selected on the basis of the events that showed highest correlation for increasing temperature. The fourth set comprises of the stalled events. These sets are listed in Table 1 where a weight is associated to each event.

Table 1: Performance monitoring events and their weights chosen for building the Kernel

Weight	Set 1	Set 2	Set 3	Set 4
4	uops_executed	idq_uops_not_delivered	idq_uops_not_delivered	l2rqsts:0x20
3	branch_inst_exec:0x81	branch_inst_retired	resource_stalls	ild_stall:0x04
2	branch_inst_exec:0xff	resource_stalls	cpu_clk_unhalted	resource_stalls
1	arith	cpu_clk_unhalted	branch_inst_exec:0x81	icache:0x02

The next task is to implement the thermal metric calculator in the Linux Kernel so that the count of the events can be maintained at the OS level without using OProfile tool. For this purpose, a header file is added to the Kernel source tree to perform the operations on the Model Specific Registers(MSR) like reading from and writing to the hardware counters. Since the events are to be counted for each process, therefore the counters are started and stopped during process switching. When the process arrives for the first time, the thermal metric is zero and the counters are started to count the number of times the event specified in the MSR occurs; while the process is executing on the core. When there are multiple processes to run, then OS performs scheduling of the processes and context switching. Whenever a context switch occurs, the hardware context of the old process is stored and replaced by the hardware context of the new process. So, here the counters are stopped to read the values and evaluate the thermal metric for the old process using equation 1; and then started to count for the new process. The calculated thermal metric value is used to schedule the process in the next scheduling epoch. The Fig. 2 shows the flowchart for the thermal aware process scheduler (modified CFS).

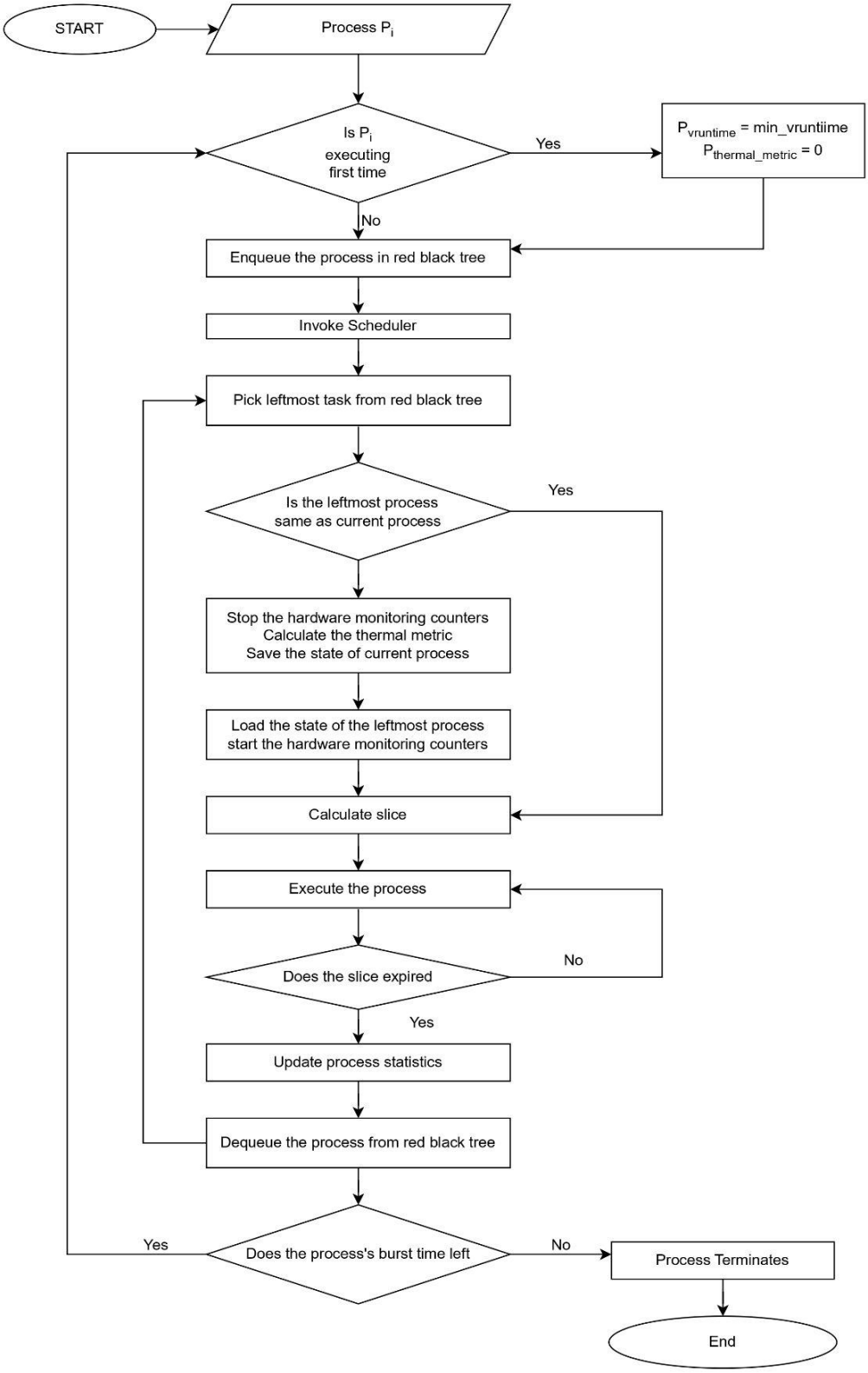

Figure 2: Flowchart for Thermal Aware Process Scheduler

The last task is to perform the experiments in order to compare the heat generated by the existing and the proposed approach. The modified scheduler is configured separately for each of these four sets. Each scheduler is then evaluated for single and multiple workloads to assess peak temperature of the cores, temporal temperature gradient and turnaround time. The single workload scenario executes only a single PTS test on the core whereas the multiple workloads scenario executes four PTS tests simultaneously on the core. The PTS tests are run on core0 using *taskset* command in order to avoid PTS task migration and evaluate the modified CFS. As the experiments are performed on a real machine at real time, so along with these tests the daemon processes continue to execute simultaneously on all the cores.

Single Workload

The existing and modified CFS are tested for the web server workloads, scientific computing, racing game and filesystem. The Phoronix-test-suite names are namely Ebizzy, Java SciMark, Corebreach and Compile Bench.

Multiple Workloads

The existing and modified CFS are tested for multiple workloads to reflect real time scenario. Here, 10 scenarios are created which consists of a combination of four tests. The different scenarios with their test combination are given in Table 2.

Table 2: Testing Scenarios for multiple workload

Scenario	Tests
Scenario 1	aio-stress, register-arithmetic, sunflow, compress-gzip
Scenario 2	integer-arithmetic, encode-mp3, compress-gzip, sunflow
Scenario 3	encode-mp3, aio-stress, floating-point arithmetic, compress-gzip
Scenario 4	dhrystone, register-arithmetic, integer-arithmetic, floating-point
Scenario 5	encode-mp3, sunflow, register-artithmetic, dhrytone2
Scenario 6	dbench, n-queens, crafty, hmmer
Scenario 7	blogbench, floating-point arithmrtic, tscp, systester
Scenario 8	openSSL, ffte, hmmer, sudokut
Scenario 9	c-ray, fhourstones, encode-mp3, mrbayes
Scenario 10	systester, compress-gzip, crafty, ffte

RESULTS AND DISCUSSION

The schedulers are evaluated for the change in peak temperature of the processors, to know the temporal temperature gradient and the time required to execute the scenarios. The temperature and time are recorded for a real machine at real time and thus include all the overhead incurred at runtime.

Reduction in Peak Temperature

The reduction in peak temperature is found to be up to 11.36%. The results also conveyed that the scenarios (1-5) consisting of the Phoronix-test suite tests that were used to gather the thermal statistics of the events showed better results than the other scenarios. The detailed

statistics regarding the percentage change in the peak temperature of the processor is in table 3.

Reduction in Thermal Cycling

The temporal temperature gradient for the core can be evaluated by the number of occurrences of high thermal swings/fluctuations over a time period (thermal cycling). The amplitude of thermal swings for the scenarios 1, 3 and 4 is considered to be greater than 15℃ while for the other scenarios the amplitude of thermal swing is 10℃. The reduction in thermal cycles over the entire execution of the testing scenarios for the existing CFS and the modified CFS for set1, set2, set3 and set4 is found to be 8%, 40%, 80% and 68% respectively. The detailed statistics regarding the number of occurrences of thermal cycling is in table 4.

Table 3: Percentage change in Peak temperature of the processor

Scenario		% change in Peak Temperature			
		Set 1	Set 2	Set 3	Set 4
	Idle	-6.20	3.12	0.00	-6.20
Single Workload	Ebizzy	2.43	4.87	9.75	7.31
	Java-SciMark2	2.22	0.0	6.66	4.44
	Compilebench	7.89	2.63	0.00	7.89
	Corebreach	-2.32	0.00	0.00	0.00
Multiple Workload	Scenario 1	2.17	2.17	6.52	4.34
	Scenario 2	6.52	4.34	8.69	8.69
	Scenario 3	8.88	6.66	8.88	11.11
	Scenario 4	11.36	4.5	9.09	9.09
	Scenario 5	8.88	8.88	11.11	8.88
	Scenario 6	2.22	4.44	6.66	-2.22
	Scenario 7	9.09	2.27	2.27	4.54
	Scenario 8	4.44	4.44	6.66	4.44
	Scenario 9	0.00	-2.32	2.32	6.97
	Scenario 10	6.52	6.52	6.52	2.17

Table 4: Number of occurrences of thermal cycling in the processor

Scenario		Number of occurrences of thermal swings				
		Existing	Set 1	Set 2	Set 3	Set 4
Single Workload	Ebizzy	0	0	0	0	1
	Java-SciMark2	2	4	1	2	1
	Compilebench	1	0	0	0	0
	Corebreach	3	5	6	1	2
	Total	6	9	7	3	4
Multiple Workload	Scenario 1	6	0	0	0	0
	Scenario 2	7	2	1	0	0
	Scenario 3	2	1	0	0	0
	Scenario 4	2	2	1	0	1
	Scenario 5	0	6	3	0	0
	Scenario 6	1	0	0	0	0

	Scenario 7	1	1	3	2	1
	Scenario 8	0	2	0	0	2
	Scenario 9	0	0	0	0	0
	Scenario 10	0	0	0	0	0
	Total	19	14	8	2	4

Reduction in Turnaround Time

The average turnaround time for the existing CFS and the modified CFS for set 1, set 2, set 3 and set 4 is found to be 23:12, 22:37, 22:09, 19:12, and 20:52 respectively. Thus, the reduction in turnaround time for the modified CFS for set 1, set 2, set 3 and set 4 with respect to the existing scheduler is 2.51%, 4.52%, 17.24% and 10.05% respectively. The detailed statistics regarding the turnaround time is in Table 5.

The experimental results show that the modified CFS is able to reduce the peak temperature of the processor up to 11.36% as compared to the existing CFS. It is found that the modified CFS also reduces the thermal swings. All the sets for modified CFS are able to reduce the effect of thermal swings, hence ensuring reliability. The reduction in thermal swing is up to 80%. The experiments also reveal that there exists a relation between thermal swings and turnaround time for a pair of specific scenario and event set. The turnaround time is found to be less when there is less number of thermal swings. The scenarios 4,7,9 and 10 showed better results for the modified CFS for set 1 as these scenarios consist of different type of computation intensive tests. Also an improvement in peak temperature and thermal cycling is seen as the number of processes increased. Among all the sets, the modified CFS for set 3 shows the best results for improvements in peak temperature, temporal temperature gradient and turnaround time. However, the execution of parallel applications on multiple nodes remain as a future work.

Table 5: Turnaround time to execute the scenarios

Scenario		**Turnaround time (in min)**				
		Existing	**Set 1**	**Set 2**	**Set 3**	**Set 4**
Single Workload	Ebizzy	02:10	01:18	01:08	01:08	01:50
	Java-SciMark2	04:12	05:04	03:02	02:48	02:10
	Compilebench	15:38	15:02	15:02	14:58	15:02
	Corebreach	17:08	19:24	18:26	17:42	17:04
	Average	09:47	10:12	09:24	09:09	09:01
Multiple Workload	Scenario 1	26:32	24:28	24:22	24:08	14:48
	Scenario 2	13:06	11:22	12:22	11:16	11:38
	Scenario 3	24:52	21:44	25:06	24:04	24:38
	Scenario 4	13:46	12:36	14:10	12:16	13:26
	Scenario 5	10:14	15:06	13:24	06:52	12:34
	Scenario 6	01:18:36	01:16:08	51:38	41:32	01:13:14
	Scenario 7	35:54	36:08	40:02	25:54	39:38
	Scenario 8	11:48	11:54	13:10	11:54	12:38
	Scenario 9	46:00	46:20	46:06	46:22	36:04
	Scenario 10	24:52	20:10	32:14	27:58	17:20
	Average	28:34	27:36	27:15	23:14	25:36

The experiments inferred the following:

1. The rise in temperature of the cores along with the temporal temperature gradient can be restrained by scheduling the processes on the basis of their thermal contribution to the cores thereby mitigating the need of processor throttling.
2. The improvements in the peak temperature for the scenarios consisting of the tests used to select the hardware events show better results as compared to other scenarios.
3. The increase in number of process helps the scheduler to schedule the processes in the intended manner.
4. The hardware events having the highest correlated values for the temperature of the core results in optimal temperature performance balance.
5. The resource stalls and misses also contribute to the rise in temperature of the cores.
6. The temperature of the core is a result of the run-time statistics of different events, i.e. at any instant the temperature of the core is the result of the instructions executed by the core, the architectural state of the required resource and the neighbouring cores in a multi-core processor.

The thermal scheduling algorithm proposed in this work entirely depends on the HMC available on the processors. These counters are processor specific and hence the method adapted to determine the thermal metric for a process is not scalable and is limited by the number of events and counters available.

CONCLUSION

To ease our lives, we are moving towards automation. However, automation comes at the expense of huge amount of carbon footprint. This work ensures that computing needs along with healthier, safer and greener environment can be accomplished by designing a thermal aware scheduler. In this work, a thermal aware scheduler for multi-core processor is presented that takes into consideration the thermal characteristic of individual tasks. The idea is implemented for four different sets of hardware events on Linux Kernel 2.6.32. Each time the Kernel is built and evaluated for various real time scenarios. The thermal aware scheduler is able to reduce peak temperature and thermal swings for multi-core processor upto 11.36% and 80% respectively which in turn increases the reliability of hardware and maximizes the life-span. This thermal aware scheduler can be applied to efficiently schedule jobs in a large-scale data center which hosts tens of thousands of machines to gain an increased performance using available resources whilst cutting down the operational costs and carbon footprint. As OS continues to serve as a critical component of future exascale computing paradigm, so the designing of a thermal aware scheduler at OS level will continue to facilitate an optimal balance among performance, energy consumption, hardware reliability, life expectancy and carbon footprint.

References

[1]. Elakkiya N, Meenakshi S. Supply Insensitivity Temperature Sensor for Microprocessor Thermal Monitoring Using Adc-

[2]. Sar. IOSR Journal of Electrical and Electronics Engineering, Vol. 5(3), pp. 1-7, 2013.

[3]. History of processor performance, 2012. url: http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf

[4]. Carinhas P. Green Computing Guide. Fortuitous Technologies, 2009.

[5]. Donald J, Martonosi M. Techniques for Multicore Thermal Management: Classification and New Exploration. Proc. 33rd International Symposium on Computer Architecture (ISCA'06) IEEE, 2006, pp. 78-88.

[6]. Choi J, Cher CY, Franke H, Hamann H, Weger A, Bose P. Thermal-aware task scheduling at the system software level. Proc. 2007 international symposium on Low power electronics and design (ISLPED'07), 2007, pp. 213-218.

[7]. Yeo I, Liu CC, Kim EJ. Predictive dynamic thermal management for multicore systems. Proc. 45th Annual Design Automation Conference, 2008, pp. 734-739.

[8]. Ayoub RZ, Rosing TS. Predict and act: Dynamic thermal management for multi-core processors. Proc. 14th ACM/IEEE international symposium on Low power electronics and design, 2009, pp. 99-104.

[9]. Liu Z, Xu T, Tan SXD, Wang H. Dynamic thermal management for multi-core microprocessors considering transient thermal effects. Proc. 18th Asia and South Pacific Design Automation Conference, 2013, pp. 473-478.

[10]. Gomaa M, Powell MD, Vijaykumar TN. Heat-and-run: Leveraging smt and cmp to manage power density through the operating system. Proc. International Conference on Architectural Support for Programming Languages and Operating Systems, 2004, pp. 260-270.

[11]. Stavrou K, Trancoso P. Thermal-aware scheduling: A solution for future chip multiprocessors thermal problems. Proc. of the 9th EUROMICRO conference on Digital System Design, 2006, pp. 123-126.

[12]. Musoll E. A thermal-friendly load-balancing technique for multi-core processors. Proc. 9th International Symposium on Quality Electronic Design, 2008, pp. 549-552.

[13]. Xia L, Zhu Y, Yang J, Ye J, Gu Z. Implementing a thermal-aware scheduler in linux kernel on a multicore processor. The Computer Journal, vol. 53, pp. 895-903, Jan. 2018.

[14]. Liu G, Fan M, Quan G. Neighbor-aware dynamic thermal management for multi-core platform. Proc. Design, Automation Test in Europe Conference Exhibition, 2012.

[15]. Salami B, Baharani M, Noori H, Mehdipour F. Physical-aware task migration algorithm for dynamic thermal management of smt multi-core processors. Proc. 19th Asia and South Pacific Design Automation Conference, 2014, pp. 292-297.

[16]. Coskun AK, Rosing TV, Whisnant KA, Gross KC. Static and dynamic temperature-aware scheduling for multiprocessor socs. IEEE Transactions on Very Large Scale Integration Systems, vol. 16, pp. 1127-1140, Sep. 2008.

[17]. Salamy H, Aslan S, Methukumalli D. Task scheduling on multicores under energy and power constraints. Proc. 26th IEEE Canadian Conference of Electrical and Computer Engineering, 2013, pp. 1-4.

[18]. Boneti C, Gioiosa R, Cazorla FJ, Valero M. A dynamic scheduler for balancing hpc applications. Proc. IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2008, pp. 1-12.

[19]. Piguet C, Low power processors and system on chips. CRC Press, 2005.

[20]. Stavrou K, Trancoso P. Thermal-aware scheduling for Future Chip Multiprocessors. EURASIP Journal on Embedded Systems, vol. 2007, pp. 1-15, 2007.

[21]. Safari S, Khdr H, Gohari-Nazari P, Ansari M, Shaahin Hessabi, Henkel J. TherMa-MiCs: Thermal-Aware Scheduling for Fault-Tolerant Mixed- Criticality Systems. IEEE Transactions on Parallel and Distributed Systems, Oct. 2021, pp.1678-1694

[22]. Farnaz Niknia, Vesal Hakami, Kiamehr Rezaee. An SMDP-based approach to thermal-aware task scheduling in NoC-based MPSoC platforms. Journal of Parallel and Distributed Computing, July 2022, pp. 79-106