International Journal of Computer Networks and Communications Security

VOL. 3, NO. 7, JULY 2015, 298–304
Available online at: www.ijcncs.org
E-ISSN 2308-9830 (Online) / ISSN 2410-0595 (Print)

Dynamic Optimization of Power Consumption and Response Time in Multiprocessors by Turning Processors Off

SOMAYYEH J. JASSBI[1] and JAMAL RAHIMI KHORSAND[2]

[1] PhD. Department of Computer Engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran

[2] Msc. Department of Computer Engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran

E-mail: [1]sjassbi@srbiau.ac.ir, [2]jrkhorsand@srbiau.ac.ir

ABSTRACT

The importance of energy and ways of managing it, because of the cost of unnecessary usage, mobile technologies and battery lifetime sensitivity, is increasing daily. In the last decade most of systems use multiprocessor computer systems to increase the performance. Hence, managing power consumption under time limitations present an important challenge. Using any scheduling algorithm, there are some gaps, in which no process is assigned to a certain processor. Furthermore, if response time of a process is less than a maximum value, the response time is acceptable and there is no need for the response time to be least possible. In this condition, we can use some of processors instead of all of them to help reducing power consumption. As a hypothesis, turning processors with low processing load off can reduce the power consumption in such a way that response time of processes stay acceptable. In this paper, we discuss about a new policy to manage turning processors on/off during run time of system to examine the mentioned hypothesis. Simulations show that in most of the cases this policy optimizes the power consumption of system better than other usable methods.

Keywords: *Multiprocessor, Response Time, Power Consumption, Energy Consumption, Processing Power.*

1 INTRODUCTION

Computer systems are one of the most energy consuming systems in the world. As reported by Mark Mills, in 2013, Information-Communication-Technologies (ICT) ecosystem, in which a large number of computing systems are used, consume about 1500TWh of electricity. This amount of energy is equal to the amount of electricity generated in Japan and Germany combined [1]. In the other hand the lifetime of batteries is limited. In fact lowering the power consumption of a system is a live or death issue [2]. With increased need of applications, a single processor system is not capable of responding to all applications in short time. "Moore's law has not been repealed since each year there are more transistors placed in the same space but the clock speed has not been increased without overheating" [3]. It also means the Dennard's scaling law is broken down [4]. Even if we consider that the Moore's law [5] is being repealed yet, based on Moore's law and Shannon's law, the processing load and cost of service for each process, is increasing with a higher rate in comparison to increasing the processing power of processors. So the probability of the situation in which a processor being in idle mode or processes waiting in queue for the processor being limited, is decreasing. In other words, the number of processes in queue is getting divergent and the probability of starvation or bottleneck is increasing. Thus, the manufacturers started to use multiprocessor structures instead of single processor structures.

Multiprocessor architectures face a lot of challenges such as scheduling and power consumption. Scheduling with a load-balancing approach, can help increasing throughput, using maximum possible processing power and solving

starvation or bottleneck problems. But if the processes are not assigned to processors with a load-balancing approach, the response time cannot decrease well. Furthermore the more processors used to execute processes the more power consumed by the system.

Nowadays considering a trade-off between run-time and energy cost of executing a set of processes on a set a processors, is an example of challenges the computing systems face with.

In this paper, we present a new policy to schedule processes in a multiprocessor system with an approach to decrease number of in service processors as well as power consumption in a way that response time does not violate the maximum acceptable value.

2 HYPOTHESIS

As a hypothesis, if we turn off the processors with little amount of load assigned to them in each period of time and use other processors to execute the processes assigned to turned off processors, the power consumed by the system will decrease and the system will be more efficient. In this paper we also examine this theory, to see if this approach could be true.

3 BASIC CONCEPTS

3.1 Multiprocessor

Multiprocessor is a computer system containing a set of processors connected to each other using a network infrastructure. The network infrastructure could be as small as a mesh graph between cores in a chip, or as large as a WAN such as internet to communicate in a cloud system. Hence, a multiprocessor system could use processors as cores in a chip, chips on a board, or multiple computers connected by a computer network. Using multiprocessor systems has communication motives, such as fault tolerance, matching the application and most important lowering power consumption.

For lowering the power consumption using multiprocessor one can lower the frequency of processors. We describe two parameters, throughput and power consumption. Throughput of the system is the total number of instructions which can be executed in a clock cycle. Equation 1 shows the relation between IPC (Instruction per clock) or throughput and number of processors (or cores) which is shown by C and the frequency which is shown by F [6].

$$IPC = C * F \quad (1)$$

Equation 2 shows the relationship between the power consumption of a system (P), number of processors (C), voltage of each processor (V) and the frequency (F) [6].

$$P = C * V^2 * F \quad (2)$$

In an ideal scenario, if we double the number of processors and decrease the frequency by half, the performance will not change. Equation 3 proves this fact [6].

$$IPC = (2 * C) * \left(\frac{F}{2}\right) = C * F \quad (3)$$

In the other hand, the system in case above will use only a quarter of energy it used before. Equation 4 proves this fact.

$$P = (2 * C) * \left(\frac{V}{2}\right)^2 * \left(\frac{F}{2}\right) = \left(\frac{1}{4}\right) * C * V^2 * F \quad (4)$$

Considering the facts mentioned above, if we use $k * C$ processors with clock cycle equal to $\frac{f}{k}$, instead of C processors with clock cycle equal to f , the process power will decrease as shown in Figure 1.

According to the figure 1, although more processors with less clock frequency have less power consumption, but if the k parameter is too high, the decrement of power consumption is not beneficial when we consider the product and maintenance costs.

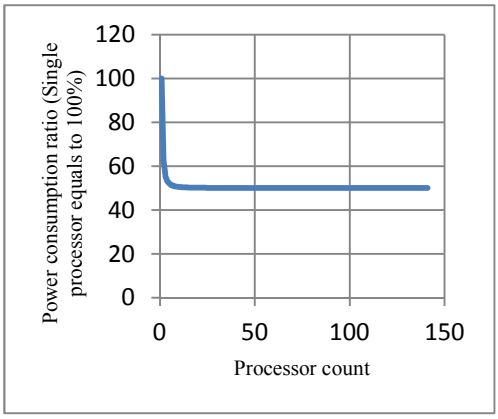

Fig. 1. Relationship between process power and processor count

The other problem is that we cannot always make all the processors busy all the time so the performance will be lower than the mentioned value.

Although the performance mentioned in equation 1 is not accessible in a real system, if the utilization of processors (U) could be considered equal in all processors, we could have a good performance. The performance of system in this situation could be estimated by equation 5.

$$IPC = U * C * F \quad (5)$$

If we want to increase the performance of system, we should increase the number of processors without changing the clock frequency of processors, and to increase the throughput, we should also increase the utilization of processors.

Thus a good scheduling method should help to maximize the utilizations while the number of processors is limited. Furthermore, if the processing load of processes is not too high, as mentioned as a hypothesis above, we should turn some of processors off.

3.2 Concurrent optimizing Two Parameters

For optimizing two contradicted parameters (x, y), there are three approaches. In first approach, after optimizing the parameter x, the optimized value for parameter y in set of solutions will be chosen [7].

The second approach considers a parameter f which is a function of parameters x and y. Optimizing parameter f means we compromised between parameters x and y [7].

The third approach considers a limit value for parameter x as the maximum acceptable value and then we try to optimize parameter y. The solution should not violate the maximum acceptable value rule [7].

To compromise the response time and consuming power of system, in our method, we use the third approach, which means after defining a maximum acceptable value for response time, we try to minimize the power consumption.

Using this approach is reasonable because of the fact that although response time is a key parameter for defining time characteristics of a system, if the response time is less than a limited value, neither system nor processes will not face any trouble time wise. Furthermore, the user will not notice slight changes in response time.

3.3 Background

There are many policies and algorithms presented for power management indicating that how important this issue is. There are two types of decision making for power management. Static power management (SPM) is the type in which, any processing load entering the system in future is predictable before the system starts working.

Oracle policy is an example for SPM. “It gives the lowest possible power consumption, as it transitions the device into sleep state with the perfect knowledge of the future. Oracle policy is computed off-line using a previously collected trace.” [8].

Another example of SPM algorithms is always-on policy, in which processors are always staying in active mode. Always-on policy is being used when the duration of time in which a processor is idle is so narrow that transitions between states could not be beneficial while talking about time cost [8].

The other type of power management is dynamic power management (DPM). Although DPM cannot save power and energy as well as oracle policy but it can support any unpredictable changes in system [8].

Dynamic Voltage Scaling (DVS) algorithm is another approach of power management where the power consumption will be configured by dynamically changing the speed and voltage of processors depending on the needs of the applications running [9][10].

Other examples of power management policies are TISMDP[11], Adaptive [12], Karlin’s [13], 30s timeout [8], DTMDP [8], 120s timeout [8].

4 METHODOLOGY

There are two important decisions to make when talking about dynamically turning a processer off and on again during run time of a multiprocessor system. First decision is when and which processor should be turned off. The second one decides if a turned off processor should be turned on again.

4.1 Turning processor off

4.2.1 Time of turning off

Obviously, changing system configuration, such as turning processors off, is only reasonable when the state of system is changed. The state of system changes when an event is occurred. In multiprocessor scheduling, the most important

events are entering of a process into system and finishing of a process execution.

If for each event occurred in the system, we decide about system configuration changes, the decision making overhead during run time is too high and it decreases the system performance. So we should define newly events based on main events. There are some defined events to trigger decision makings for turning a processor off including:

1. When minimum processor utilization is less than $MiPU_{off}$.
2. When maximum processor utilization is less than $MaPU_{off}$.
3. When difference of utilization of processors is more than DUP_{off}.
4. When average response time is less than ART_{off}% of maximum acceptable value.

$MiPU_{off}$, $MaPU_{off}$, DUP_{off} and ART_{off} are parameters which could be set based on the characteristics of system. Each of events above will be checked when a process enters the system or leaves it.

4.2.2 Which processor to be turned off

The problem is how to define maximum acceptable response time value. Let's think about an ideal scenario in which the processing load of system is shared between processors in a fully-balanced way, so the utilization of all processors is equal to other processors. Considering the context switch time being very low in comparison to the execution time of processes, we can think of context switch time be equal to zero.

Assuming the situation mention above, we can think of system as a modeled system in which there is only one processor with the processing power of total processing power of all processors in system. The modeled system also has only one process with load of total processing load of all processes. The response time of the process in modeled system is known as the minimum possible average response time. It can be estimated by equation below:

$$RT_{min} = \frac{\sum L_i}{\sum PP_j} \quad (6)$$

In equation above, L_i is the processing load of process i, and PP_j is the processing power of processor j.

Obviously, the overhead caused by context switch, lack of balance between utilizations and the wait time of queued processes, makes the exact average response time be more than Rt_{min} estimated above.

If the maximum acceptable average response time is RT_{lim}, maximum acceptable overhead is:

$$\alpha = \left(\frac{RT_{lim}}{RT_{min}}\right) \quad (7)$$

Hence, if we let the maximum acceptable overhead be equal to β, RT_{lim} which is maximum acceptable value for response time can be estimated by the equation 8:

$$RT_{lim} \leq \beta * RT_{min} \quad (8)$$

So for optimizing the power consumption, the solution will be chosen which has the average response time be less than RT_{lim}.

Let's consider the maximum acceptable average response time be RT_{lim} and the current scheduling as the basic solution and current average response time equal to RT_0.

Based on values of RT_0 and RT_{lim}, there are two possible situations. First, if RT_0 is less than RT_{lim}, it is possible that if we turn a processor off, the average response time remains less than RT_{lim} and the consuming power is getting lower.

To select the candidate processor to be turned off, we think of a system without one of processors and with same processes. In this system, the RT_{min} for ideal scenario will be estimated. This process will be repeated for each processor as the processor which is neglected. For each processor, if the estimated RT_{min} is less than RT_{lim}, the processor is candidate to be turned off.

After all candidates are defined, the candidate processor with minimum number of processes assigned to it will be turned off. This policy is being used to help reducing the migration cost of the processes. Before turning the processor off, all processes assigned to it will be rescheduled on remaining processors. If the migration cost of processes on two candidate processors is equal, the processor with lower processing power will be selected.

To reschedule the processes we should follow some steps:

1. All queued processes should be rescheduled as newly entered processes.

2. If the running process is finishing or the cost of restarting it is high, we let the process to be executed on current processor, but if restarting it is not cost-critical, we suspend execution of process and reschedule it.

3. When there is no process assigned to the selected processor, we turn it off.

The other situation is when the RT_0 is greater than RT_{lim}. In this situation, we cannot turn processor off since the average response time could not be in acceptable range. What we do is checking if utilization of processors is balanced. For rebalancing the load of processors, we find the processor with most execution time and call it the critical processor. The process with lowest load assigned to critical processor will be rescheduled on other processors. Then we check if new RT_0 is getting lower or not. We repeat this process while the time rescheduling processes on critical processor is reducing RT_0 and RT_0 is higher than RT_{lim}. Note that in each step the critical processor will be changed. If there is a condition in which reducing the load of critical processor cannot reduce the RT_0 and RT_0 is still greater than RT_{lim}, we should check if the maximum acceptable overhead and as a result RT_{lim} is selected properly.

The other reason for RT_0 being greater than RT_{lim} is that there might be a processor which has such a low processing power that it will rarely be assigned to a process. In this situation, this processor which is known as weak processor should be turned off and RT_{lim} should be estimated again.

By rebalancing the load of processors, although no processor is being turned off but the maximum execution time of processors will be decreased so all processors will be turned off sooner, if turning system off is an option. As an example, event based systems get new events all the time so there is always a new chain of processes to be executed on this systems. Furthermore, although there are exceptional events such as stack overflow or divide by zero which occur just once in a while, but most of events such as getting report from a sensor are time based. So these kinds of events finishes only when the time is finished which means they never end.

Note that the number of processors is limited so the algorithm overhead is not high.

There is also another reason for RT_0 being higher than RT_{lim}. If processes are entering the system in burst mode or the context switch time is not low enough, the policy of turning processors off and all calculations done before are useless. In this situation we should use idle mode to reduce power consumption.

4.2 Turning processor on

4.2.1 Time of turning on

Like the policy for turning a processor off, there are some defined events to decide if a processor should be turned on again. Turning a processor on will be occurred to reduce energy consumption. Defined events are:

1. When minimum processor utilization is more than $MiPU_{on}$.

2. When maximum processor utilization is more than $MaPU_{on}$.

3. When difference of utilization of processors is less than DUP_{on}.

4. When average response time is more than ART_{on}% of maximum acceptable value.

$MiPU_{on}$, $MaPU_{on}$, DUP_{on} and ART_{on} are parameters which could be set based on the characteristics of system. Each of events above will be checked when a process enters the system or leaves it.

4.2.2 Which processor to be turned on

When a processor is turned off it should be turned on again in a situation in which the processing load of system is high. Although the consuming power is important but the consuming energy is also an important parameter. In other words, reducing power consumption is related to energy consumption.

The consuming energy when consuming power is constant as in a multiprocessor system, is the multiply of power and time.

$$E = P * T \tag{9}$$

Consider the power consumption of a system when the selected processor is turned off is P_{on}. The power consumption of the selected processor is P_0. Then after turning the selected processor on the power consumption of system is equal to the sum of P_{on} and P_0.

The minimum possible execution time of system when the selected processor is turned off can be estimated as follow:

$$T = \frac{RT_{min} * n}{m} \quad (10)$$

In the above equation, n is the number of processes and m is the number of processors. So if we estimate the overhead of consuming power when the selected processor is turned off, we have:

$$\frac{E'}{E} = \frac{P_{on} * \frac{RT_{min} * n}{m}}{(P_{on} + P_0) * \frac{RT'_{min} * n}{m+1}} = \left(\frac{P_{on}}{P_{on} + P_0}\right) * \left(\frac{m+1}{m}\right) * \left(\frac{RT_{min}}{RT'_{\min}}\right) = \alpha \quad (11)$$

If the maximum acceptable overhead is β, the selected processor will be turned on again when the energy overhead is more than β.

To select which one of processors turned off before should be turned on, the overhead will be estimated for all turned off processors. The processor with maximum overhead will be considered as selected processor and it is the processor we turn on.

When the selected processor is turned on, the processes should be rescheduled to balance load between all processors. To reschedule processes, we use the critical processor algorithm mentioned before.

5 EVALUATION

To evaluate the performance of the provided method, we compare this method with other algorithms. Once we are talking about power management in process scheduling, we should have a scheduling algorithm as basic solution. The basic scheduling algorithms used for this comparison are FCFS and Genetic Algorithm (GA).

To evaluate the method, it is simulated using C# programming language. The input data is five different test benches. The results provided are the averages of the results of these test benches.

The events for turning a processor off or on, were parametric. To simulate the method, we should set values to the parameters. The values set are as follow:

1. $MiPU_{off}$ is 15%.
2. $MAPU_{off}$ is 70%.
3. DUP_{off} is 50%.
4. ART_{off} is 70%.
5. $MiPU_{on}$ is 60%.
6. $MaPU_{on}$ is 85%.
7. DUP_{on} is 5%.
8. ART_{on} is 90%

Table 1: Total Run time average for testbenches

Algorithm	Without power management	Proposed method	DVS
FCFS	22.4464	20.2354	20.6058
GA	15.90603	14.8721	15.0598

Table 1 shows the total run time of all processors. If we consider the characteristics of all processors be relatively equal, this time is the key factor to estimate energy consumption and also average power consumption. As shown in Table 1, while talking about FCFS scheduling algorithm, the provided method has 9.85% improvement in comparison to the condition in which FCFS is implemented without power management. It also improves results of implementing DVS on FCFS by 1.85%. Although the improvement in comparison to DVS algorithm is not high but the proposed method is simpler in implementation and simplicity is a key factor while talking about implementation and run time overhead.

It also improves the results of GA approach by 6.5%. The improvement in comparison to GA with DVS power management approach is 1.24%.

6 CONCLUSION

In this paper, we presented a new approach for dynamic power management. According to hypothesis mentioned before, our approach is based on two different tasks:

1. Decide whether a processor should be turned off to reduce power consumption.
2. Decide whether a processor should be turned on again to reduce energy consumption.

We tested our algorithm by simulation of a multiprocessor system with many processes with different processing load. Results show that using our algorithm by considering decision time and power saving as important parameters in comparison to other policies is feasible and reasonable.

7 REFERENCES

[1] Mills. M. “Cload begins with coal”, Digital Power Group, 2013. Online at: http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf

[2] Vakilian Zand. H. “Scheduling in Multicore Systems Considering Energy Efficiency Using an Evolutionary Algorithm”. Amir Kabir University. 2014.

[3] Herlihy. M, Shavit. N. “The art of multiprocessor programming”. Morgan Kaufmann Publishers. 2008.

[4] Macmenamin. A. “The end of Dennard scaling”, Cartesian product, 2013. Online at: https://cartesianproduct.wordpress.com/2013/04/15/the-end-of-dennard-scaling/

[5] Moore, G. “Cramming more components onto integrated circuit”. IEEE proceedings, 1998.

[6] Minjoong, K. et al. “A Simple Model for Estimating Power Consumption of a Multicore Server System”, International Journal of Multimedia and Ubiquitous Engineering, Vol. 9, 2014.

[7] Bessai. K. et al. “Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments”, IEEE Fifth International Conference on Cloud Computing. 2012

[8] Simunic. T. "Dynamic Management of Power Consumption," Power Aware Computing, 2002, pp. 101-125.

[9] Geppert, L, Perry T. “Transmeta’s magic show,” IEEE Spectrum, vol. 37, 2000.

[10] Simunic T. et al. “Dynamic Voltage Scaling and Power Management for Portable Systems”, Proceedings of the 38th annual Design Automation Conference. 2001. Pp.524-529.

[11] Simunic T. et al. “Dynamic Power Management for Portable Systems”, the 6th International Conference on Mobile Computing and Networking, 2000. pp. 22–32.

[12] Chung. E. et al. “Dynamic Power Management for non-stationary service requests”, Design, Automation and Test in Europe. 1999. pp. 77–81.

[13] Karlin. A. et al. “Competitive Randomized Algorithms for Nonuniform Problems”, Algorithmica. 1994. pp. 542–571.