Metrics and Task Scheduling Policies for Energy Saving in Multicore Computers

J. Mair, K. Leung, Z. Huang
Department of Computer Science
University of Otago
Dunedin, New Zealand
Email: {jkmair;kcleung;hzy}@cs.otago.ac.nz

***Abstract*—In this paper, we have proposed three new metrics, Speedup per Watt (SPW), Power per Speedup (PPS) and Energy per Target (EPT), to guide task schedulers to select the best task schedules for energy saving in multicore computers. Based on these metrics, we have proposed the novel Sharing Policies, the Hare and the Tortoise Policies, which have taken into account parallelism and Dynamic Voltage Frequency Scaling (DVFS) in their schedules. Our experiments show that, on a modern multicore computer, the Hare Policy can save energy up to 72% in a system with low utilization. On a busier system the Sharing Policy can make a saving up to 20% of energy over standard scheduling policies.**

***Keywords*-Speedup per Watt (SPW), Power per Speedup (PPS), Energy per Target (EPT), Exclusive Policy, Sharing Policy, Hare Policy, Tortoise Policy, Performance per Watt (PPW), energy saving, power saving, Dynamic Voltage Frequency Scaling (DVFS), Green computing**

I. Introduction

The issue of energy consumption has long been an active area of research in portable consumer electronics such as laptops and smart phones, with the goal of extending the battery life of these devices. This has been achieved by dynamically adjusting operational states and by placing the devices in sleep and standby states. However, energy consumption is becoming a concern in large scale systems such as servers and clusters for grid and cloud computing due to their popular deployments. This is becoming more prevalent because new issues are arising due to their ever increasing scale, like the increasing demand on the energy grid, and general operational costs including cooling. The total cost of operation is growing so much that in some cases it is likely to start exceeding the initial capital costs for building such a cloud computing center, preventing some business from setting up such facilities.

In addition, consumers are concerned about environmental impact of energy consumption. In USA, 70% of electricity is generated from burning fossil fuels, and this is a big contributor of greenhouse gases and global warming [1]. As a result, many companies try to decrease their environmental footprint by focusing on energy efficiency of big energy consumers, including grid and cloud computing data centers. This is achieved through research in green computing, which aims at improving power and energy efficiency in computing facilities. In this paper, to avoid ambiguity, we use *power* and *energy* in strict terms, i.e. $energy = power \times time$.

Previous research in green computing had been focusing on power efficiency of high performance computing. The common metric used is Performance per Watt (PPW), which is often applied to computer hardware and systems, assuming they are always busily running applications. However, power efficiency is not equivalent to energy efficiency, since power efficient systems may still consume a substantial amount of energy when they are idle. Moreover, many real world systems have very low levels of utilization [2]. For example, many data centers only have the utilization of 10% to 50% and could be less than 5% for some computer nodes, as stated in [3]. Computers do not have good energy efficiency if they consume a lot of power at idle time, even though they may have good power efficiency during execution. This is due to energy been power multiplied by time. Therefore, eventually we should be aiming at energy saving, rather than power saving.

Furthermore, even on those busy computer nodes, running applications may have a relatively relaxed targeted performance. That is, they are not concerned about achieving the maximum performance that is possible on the computer system. For example, some computing centers may have a number of applications that need to run once a day, but for each of the applications there is no strict performance requirement, as long as it can be finished in a day. In these situations, new measures and policies could be introduced to save energy, as long as they do not come at the cost of the required performance target.

In this paper, we set out to explore new metrics and policies that could be used in multicore nodes of grid and cloud environments, in order to reduce energy consumption without impacting on the performance target of the applications. A unique aspect of this work is combining Dynamic Voltage Frequency Scaling (DVFS) and program parallelism to achieve energy saving on modern multicore computers. Our research uses the multicore computers because they are rapidly becoming the default in consumer electronics, as well as often having the architecture required to support DVFS. Also with such computers, parallelism is likely to become more commonplace in applications in order to take advantage of the multiple cores to increase performance.

The contributions of this paper are: first, we have proposed the new metrics, Speedup per Watt (SPW), Power per Speedup (PPS) and Energy per Target (EPT), which can guide task scheduling policies in various situations to achieve much better energy saving in multicore nodes; second, we have proposed the novel task scheduling policies, The Sharing, the Hare and the Tortoise Policies, to explore the opportunities for energy saving in busy and idle multicore nodes; third, we have applied our new metrics and policies in modern multicore nodes and clusters, where various experiments are designed and implemented to compare and evaluate their effect in terms of energy saving. Our experiments show that the best policies can save energy up to 72%.

The remainder of this paper is organized as follows. Section II introduces the metrics and policies to be investigated. Results for each policy and a general discussion are in Section III. Section IV discusses some related work in the area of power management, with Section V giving the final conclusions and future work.

II. Metrics and policies

The original intention of this work is to explore what the impact of parallelism and DVFS has on the level of power consumption on multicore nodes and how a task scheduler can manipulate them to achieve energy saving. For this, we have proposed three new metrics, Speedup per Watt (SPW), Power per Speedup (PPS) and Energy per Target (EPT), to measure the power and energy efficiency for a number of task scheduling policies in various situations. Also we have proposed three novel scheduling policies, the Sharing Policy, the Hare Policy, and the Tortoise Policy, to achieve energy saving in multicore nodes and clusters. The details are explained as follows.

A. Speedup per Watt (SPW)

Performance per Watt (PPW) is a common metric used for calculating energy efficiency [4]. Performance is often measured with FLOPS for computer systems. However, since many applications do not even have floating point calculations, FLOPS is not a suitable metric to measure performance of many applications. Another performance metric, the number of instructions per second (MIPS), is often used to measure performance of sequential applications, but it is not suitable for measuring the performance of parallel applications, since the extra instructions for synchronization and communication between processes do not contribute to the performance of parallel applications. On the contrary, excessive use of those instructions often results in low performance of parallel applications.

In order to measure the energy efficiency of parallel applications, we use speedup as the metric for their performance. This is very suitable because speedup has been the most important metric for measuring performance in parallel computing in the past decades. Therefore, based on speedup, we propose a new metric, Speedup per Watt (SPW), to measure power efficiency of parallel applications.

To calculate speedup, we use the execution time of the application using a single thread running at the fastest frequency, e.g. 2.5GHz, because the fastest frequency is usually the default in a multicore node. The equation is:

$$speedup = \frac{time_{fastest\_sequential}}{time_{current\_configuration}} \quad (1)$$

where $time_{current\_configuration}$ is the execution time of the application running with a particular number of cores at a particular CPU frequency. We then divide the speedup by the power (in Watts) measured over the execution with the power meter to get SPW.

$$SPW = \frac{speedup}{power} \quad (2)$$

B. Power per Speedup (PPS)

SPW is a useful metric to measure how parallel applications can use power efficiently to achieve high performance. However, many real world application programs, particularly those in real-time systems are not concerned about achieving the maximum performance that is possible on a given computer system. Instead, the applications are only required to meet their targeted performance. Therefore, we propose this new metric, Power per Speedup (PPS),

PPS is calculated with the following equation:

$$PPS = \frac{power}{speedup} \quad (3)$$

Even though PPS is just the reverse of SPW, it changes the focus from performance to power, and is more intuitive when energy saving is concerned, since the smallest PPS value suggests the least energy to be consumed. PPS tells us how much power is needed for each performance unit (i.e. speedup). In the situations where performance is not a concern, we should focus on how much power is needed to achieve the targeted performance.

In our research, we use PPS to choose the best hardware configuration that consumes minimal energy but still achieves the required performance of the application. For example, if we need to run an application on a quad-core CPU with four different operating frequencies, we can have 16 combinations of configurations with different numbers of cores and different frequencies. On one hand, DVFS can reduce the amount of power with lower frequencies but incurs degraded performance. On the other hand, parallelism can increase performance at the cost of using additional CPU cores. With the many possible configurations that can satisfy the performance target of the application, it is difficult to decide which configuration is most efficient. However, with PPS, we can simply choose the configuration with the minimal PPS value to save the maximum energy.

C. The Exclusive and the Sharing Policies

In the context of parallel computing, the task scheduler has more options for scheduling a number of parallel applications on a multicore node. For example, suppose there are four parallel applications to be executed on a quad-core CPU. Usually the task scheduler schedules them one by one according to their priority. However, in many real world situations, it does not matter in which order they should be executed, as long as their executions are finished in a time window (e.g. overnight). These situations allow the scheduler to schedule the applications in alternative ways. For instance, the scheduler can use two cores to execute one application and use the other two cores to execute another application. After either of them is finished, the scheduler can load another application onto the two available cores. Essentially the scheduler can now select between the inter-application parallelism and the intra-application parallelism, where the inter-application parallelism is the parallelism that multiple applications are executed in parallel and the intra-application parallelism is the parallelism inside the application.

For the above situations we can use two scheduling policies, the Exclusive Policy and the Sharing Policy that we have proposed. The Exclusive Policy allocates all resources (e.g. cores) to one application at a time as done in traditional scheduler, while the Sharing Policy allows multiple applications to share the resources simultaneously.

Since multicore nodes offer many options to the task scheduler which could select a number of different schedules based on the idea of the above policies, it is important for the scheduler to know which policy is the best in terms of energy saving. In our experiments, we use real applications to evaluate the Exclusive Policy and the Sharing Policy with PPS. The experimental results are presented in Section III.

D. Energy per Target (EPT)

As mentioned before, many data centers only have a utilization of 10% to 50% [2, 3]. Additionally, those applications may have a relatively relaxed targeted performance. For example, some data centers only have a few applications that need to run once a day but take only a few hours to finish. It does not matter if they take longer to finish, as long as they can be finished overnight. In those situations, they could benefit from alternative scheduling policies that do not only focus on maximizing throughput but also take the period of idle time into consideration.

There are two opposing ideas that can be used in the above situations. Applications can either be running faster to sleep sooner or running slower and longer. For example, in the famous Tortoise-vs-Hare race [5], the hare runs quickly and then sleeps, but the tortoise crawls slowly but steadily. Regardless of the race result, let us consider the energy consumed during the whole race period. The power of the tortoise is low, but it takes a longer time to finish, while the power of the hare is high when it is running but it takes shorter time to finish, then quickly going to sleep after the race, during which it consumes very little energy. Now the question is, which of them consumes less energy over the whole race period.

The metric, Energy per Target (EPT) is useful to answer the above question. We define the target as a period of time in which an application has to finish execution. The EPT is calculated as:

$$EPT = power_{busy} \times time_{busy} + power_{idle} \times time_{idle} \quad (4)$$

where

$$time_{busy} + time_{idle} = time_{target} \quad (5)$$

E. The Hare and Tortoise Policies

With EPT, we can measure the energy efficiency of two extreme policies for task scheduling, the Tortoise Policy and the Hare Policy.

In a multicore node, the Hare Policy can be described as: use the highest CPU frequency and the maximum number of cores to execute the application as soon as possible, and then put the node into sleep mode once the execution finishes.

On the contrary, the Tortoise Policy is as follows: use the lowest CPU frequency and the smallest number of cores that can finish execution in the expected target time. Since there are many possible combinations of frequencies and the numbers of cores, the Tortoise Policy has to choose among the combinations the best one that consumes least energy.

Even though it seems the Hare Policy is likely to win most of the time in terms of energy saving, there are chances for the Tortoise Policy to win if the power of the sleep mode is close to the power of the slowest running mode. The condition when the Tortoise Policy wins is described by the following theorem.

Theorem 1: The Tortoise Hope

Suppose the power consumed by the fastest running mode is P_f, the power of the slowest running mode is P_s, and the power of the idle mode is P_i. The operating time of the fastest running mode is assumed T_f, the operating time of the slowest running mode is assumed T, which is also the target time. If $T_f > ((P_s - P_i)/(P_f - P_s)) * (T - T_f)$, the EPT of the Tortoise Policy is smaller than the EPT of the Hare Policy (The detailed proof of theorem is omitted).

In our following experiments, we will use a real application to run with both the Hare Policy and the Tortoise Policy. The EPTs of the policies will be calculated to show which policy is better in terms of energy saving in our multicore nodes.

III. Results

The experiments are run on a Dell PowerEdge R905 with four quad-core AMD Opteron 8380 processors (CPUs), with each core having its own FPU. This is organized in a NUMA

architecture with 16GB of RAM. The programs are compiled in OpenMP using gcc-4.4.1, using the optimization argument "-O3". OpenMP 3.0 [6] is used, running on a standard installation of Linux 2.6.33.

The power is measured with the use of a power strip and the Watts Up? PRO .net[1] power meter, which is a similar model to that used in [4]. The accuracy of the power meter is $\pm 1.5\% + 0.3$ watts [7].

For adjusting the operating frequency of the CPU, we use the simple mechanism of p-states which are a set of predefined voltage scales, which is the way of using DVFS in the AMD multicore node.

The benchmark programs used are Mandelbrot and Raytrace. Implementations of these programs used in this paper are adapted from [8].

The Mandelbrot algorithm is embarassingly-parallel. However, the workload of pixels is extremely uneven, and thus requires a load-balancing mechanism to prevent process starvation. Size of the screen is set to 1000 * 1000, the maximum number of iterations is set to 500 and each pixel is calculated 200,000 times.

Raytrace [9] takes *car.env* as input and casts 1000 antialiasing rays per pixel.

A. SPW of applications

The SPWs of two applications, Mandelbrot and Raytrace, are demonstrated in this section. We run them with different frequency and different number of cores. The power (in Watts) measured is for the whole system, with all unused cores placed into their lowest possible power state (i.e. 800 MHz), which is the standard practice in modern operating systems like Linux.

Figure 1 and 2 have shown the SPWs when the two applications are running. They are designed to show the SPWs on the y axis and the number of cores on the x axis. The SPWs are calculated with the Equation 2. The power is measured during the execution of the applications and the measured results are very stable with very little fluctuation.

Both Figure 1 and 2 illustrate the typical behavior of diminishing returns when the number of cores is increasing. This is because the speedup gained per core from parallel execution decreases due to the increasing overhead of parallelism when the number of cores is getting larger. However, this impact is not so strong when the cores are run at the lowest frequency (800 MHz). The reason is that the shared memory system is less of a bottleneck when the cores are running slowly, so the applications scale better with lower frequencies.

The results in Figure 2 show when the largest number of cores is used, the SPW is maximized (at its highest point).

It can also be seen that when the frequency is lower, the SPW is much worse than when the frequency is higher. For example, when using 16 cores and reducing the frequency from 2.5GHz to 1.8 GHz, it causes a 4% reduction in SPW. Stepping down another frequency results in a further 12% loss and dropping down to the lowest possible frequency sacrifices a further 26% of SPW. This gives a total loss of 42% in SPW when changing from 2.5 GHz to 800 MHz.

[1]Product specifications and user manuals can be found at https://www.wattsupmeters.com

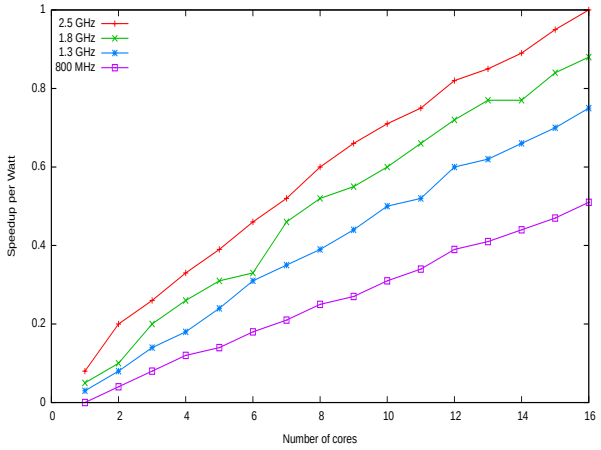

Figure 1. Speedup per Watt for mandelbrot

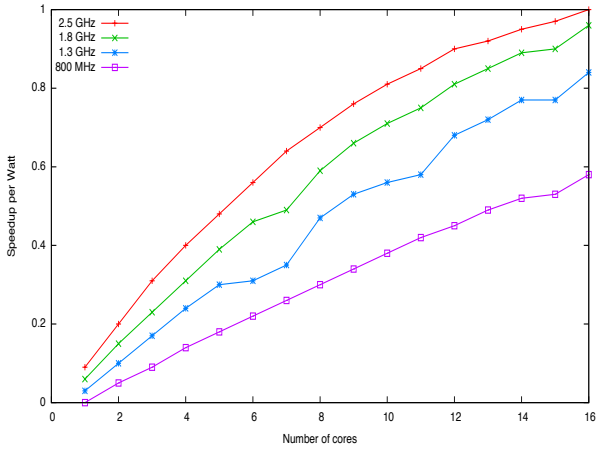

Figure 2. Speedup per Watt for raytrace

Each of these figures shows that peak SPW can be maximized by using all available cores and running at the highest frequency. This is the best possible value because it has the greatest speedup and also the lowest total energy despite having the highest power usage.

In summary, the above SPW results tell us that when we want to achieve the highest SPW, we should use the highest frequency with the maximum possible number of cores. Generally speaking, parallel execution is very important for using power more efficiently on multicore nodes. However, this is conditional on the scalability of the application, though scalability is so far not a problem for our applications running on up to 16 cores. If the performance gain becomes diminishing or even negative with more cores, there is no point using additional cores.

B. PPS of different schedules

In order to evaluate the Exclusive Policy and the Sharing Policy, we design an experiment where there are eight instances of the raytrace benchmark to be scheduled to run on our AMD 16-core machine. These eight instances can be scheduled using the Exclusive Policy. That is, they are executed one by one, each running with 16 cores. Alternatively, they can be scheduled using the Sharing Policy. In our case, they can be scheduled two by two, each running with 8 cores; or they can be scheduled four by four, each running with 4 cores; and so on. We can list more possible scheduling combinations with different CPU frequencies as well. However, as a case study, we restrict our experiment to four cases, which are labelled with *M-C*, where M represents the number of instances running in parallel (i.e. inter-application parallelism) and C is the number of cores allocated for each instance to explore intra-application parallelism. The four cases are represented with *1-16*, *2-8*, *4-4*, and *8-2*. In the experiment, we use the highest frequency (2.5GHz) of the cores.

For each of the above cases, we use the power meter to measure the average power of the machine during execution. Since every core is busy and runs the same program, we evenly distribute the measured power to each core. That is, the power of every core $P_c = power_{measured}/16$.

Once the power of each core is known, we can calculate the PPS for each instance in the above cases. Generally, for each case *M-C*, $PPS = C \times P_c/S_{M\_C}$, where S_{M_C} is the speedup of the instance running with C cores. Since all instances execute the same program in our experiment, the PPS of each instance is the same in the same scheduling case. Table I shows the PPS of the instances in each scheduling case. The *speedup* column gives the speedup of each instance.

case	energy	speedup	PPS
1-16	85645	12.44	264.19
2-8	72811	7.32	224.49
4-4	69780	3.82	215.08
8-2	68276	1.95	210.67

Table I
PPS FOR RAYTRACE BENCHMARK AT 2.5 GHZ

From Table I, we know the Exclusive Policy (case *1-16*) has the largest PPS. Among the cases using the Sharing Policy, the case *8-2* has the smallest PPS (210.67), which means this scheduling case uses minimal energy. Assuming all the four cases can satisfy the performance requirement of the instances, the task scheduler should select the scheduling case *8-2*, which can save 20.3% energy compared with the case *1-16* using the Exclusive Policy.

The reason that the case *8-2* consumes much less energy is because it has better speedup per core using two cores than using 16 cores, and thus the total execution time is much shorter than executing the eight instances one by one with 16 cores.

This experiment has demonstrated the significant energy savings that could be achieved with the Sharing Policy in a computing center where performance is not demanding on individual applications.

C. EPTs of Hare and Tortoise

We carry out two experiments to evaluate the Hare Policy and the Tortoise Policy. Each experiment has a different level of energy saving in the appropriate sleep state. For the purposes of completeness, all possible core and frequency combinations are explored. The results for the Tortoise and Hare Policies will be highlighted as well. Both experiments use the raytrace application. The target time period is set 1020 seconds, which is the time for the Tortoise to finish.

A series of power measurements are taken of the raytrace application during execution on our 16-core node. These values are read from the power meter while connected to the entire node. A problem we encountered was that the system BIOS does not allow any form of sleep state to be entered for any of the components.

So in order to determine exactly how much power of a multicore node is used when some cores are idle and put into a sleep state, we need to know the power usage of each core at different frequencies. However, AMD only provides the power usage of the quad-core CPU at the highest frequency (2.5GHz).

To determine the power usage of each core at various frequencies, we have adopted the following method. We use a micro-benchmark running on every core simultaneously. This micro-benchmark consists of an infinite loop surrounding an assembly instruction to move values between registers. As a result, the load of the micro-benchmark is only placed on the CPUs but not on any other system component. The CPU's were run at the four frequencies, and once the micro-benchmark reached a stable power level, then the mean values were recorded.

The base power of the system was found by subtracting 75 Watts for each CPU from the above measured mean value for the frequency of 2.5 GHz, as the 75-Watt is the power usage of a CPU given in [10]. With this base power, we could find the power usage of the CPUs at all other frequencies using the formula $(power_{measured} - power_{base})/4$. The results are shown in table II. As for the power usage of each core, it is reasonable to assume that each core consumes a quarter of the power of the quad-core CPU.

Frequency	Watts
2.5 GHz	75
1.8 GHz	55.25
1.3 GHz	42.94
800 MHz	35.57

Table II
CPU POWER USAGE

1) Experiment 1 – Halt state: For this experiment, we assume an unused core is placed in a halt state (shallow sleep) when the core is idle. This means the power of the idle cores should not be counted in the power measurement. However, since we cannot change BIOS to physically put the idle cores into the halt state, the measured power has included the power consumed by the idle cores. Therefore, we use the following equation to subtract the power of the idle cores from the measured power of the entire system. Both $power_{busy}$ and $power_{idle}$ in the Equation 4 for EPT are calculated in the same way.

$$power = power_{measured} - C_{idle} * power_{core} \quad (6)$$

where C_{idle} is the number of idle cores.

Based on Equations 4 and 6, the EPTs are calculated for all possible configurations (16 cores x 4 frequencies) and are shown in Figure 3.

The results show that the difference between all possible configurations is 2%. Due to their being very little difference between the EPT values of the configurations, it can be concluded that the tortoise (using one core running with the slowest frequency 800 MHz) and the hare (using 16 cores running with the highest frequency) would finish the race in a draw, with neither coming out a clear winner in terms of energy saving during the race period. This result prompts the next experiment which looks into the results when a sleep state is used which provides much greater savings in power between execution and idle periods.

2) Experiment 2 – Sleep state: To give a larger power difference, the entire system should be put into sleep state (deep sleep) during the period when all cores are idle, where deep sleep means the whole multicore node is put into sleep. This also makes more sense because there is not much point in refreshing memory when no operations are to be performed.

Multicore nodes are able to be put into a sleep state in such a way that they have very little overhead for going to sleep or waking up and will use power close to that of a shutdown node while sleeping, which is 19 watts according to our measurement on the AMD multicore node.

Therefore, in the original Equation 4, the power for the idle time is assumed 19 watts when the multicore node is assumed to be put into a deep sleep during the idle time.

Figure 4 shows the EPT values with the same target time 1020 seconds, within which the application is able to finish with every possible configuration.

In this figure, the tortoise uses the configuration which has the longest running period, but still satisfies the finishing target. The tortoise (the Tortoise Policy), with one core running at 800 MHz, has the largest EPT (106868) by far out of all configurations.

The hare (the Hare Policy), on the other hand, finishes in the shortest possible time using sixteen cores at 2.5 GHz, with an EPT of 29351.6. This is only 27.46% of the tortoise EPT and makes for a significant (72.56%) saving compared with the tortoise.

Looking at the figure, it can also be seen that the EPT values for high levels of parallelism (e.g. 16 cores) are not significantly affected by the differences of CPU frequencies, which indicates increasing CPU frequencies is not an effective measure to reduce energy consumption for parallel applications. From the figure, we can also see that parallelism can be used to provide sizeable EPT differences. Even when running at 2.5 GHz, an energy saving of 44.99% can be made by running on sixteen cores instead of running on a single core.

D. Energy saving strategies in multicore clusters

In this section, we will demonstrate how our new scheduling policies can save energy in multicore clusters when they are applied in a cluster in combination with a global task migration manager similar to that used in [11, 12].

In the experiment, we assume a cluster of four AMD 16-core nodes is used to run 10 instances of the raytrace application. The number of instances queued on each node can be seen on the first row of data in Table III. From the table, Nodes 1-3 each have two instances and Node 4 has four instances in their task queues.

The goal of this scheduling scenario is to finish the execution of all instances with the least energy (i.e. EPT) within a target time period, which is set to 210 seconds. This time period is just over a second longer than the time taken to run eight instances of the application one by one using 16 cores at 2.5 GHz. This time constraint means that no node can have more than eight instances in its queue; otherwise, some instances cannot be finished within the target time.

The experiment looks at four different strategies arising from the use of four different policy combinations. All scheduling strategies begin with the same scheduling scenario described above.

- N — This is a non power-aware (N) scheduling strategy. Each node just executes their instances one by one with 16 cores. When its instances are finished, the node remains idle operating with the lowest CPU frequency (800 MHz), which is the current standard practice in operating systems.
- HG — This is the strategy using a global task migration manager (G) to migrate the tasks from Node 1 and 2 to Node 4, but executing the 8 tasks (instances) on Node 4 one by one with 16 cores, which is usually done by such migration managers. Also Node 1 and 2 are assumed to be in a deep sleep state, consuming only 19 Watts in our experiment. We also assume Node 3 and 4 use the Hare Policy (H) which put the nodes into a deep sleep after their instances are finished.

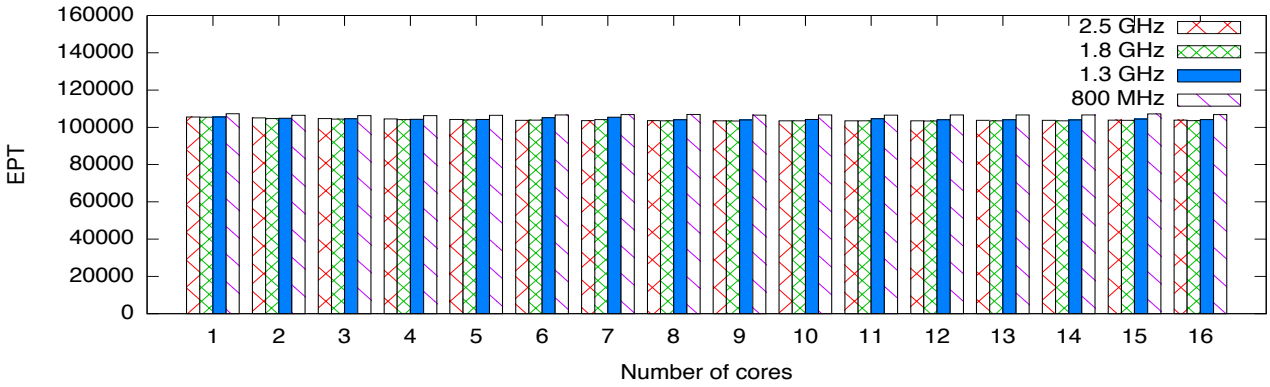

Figure 3. EPT of Raytrace using halt state

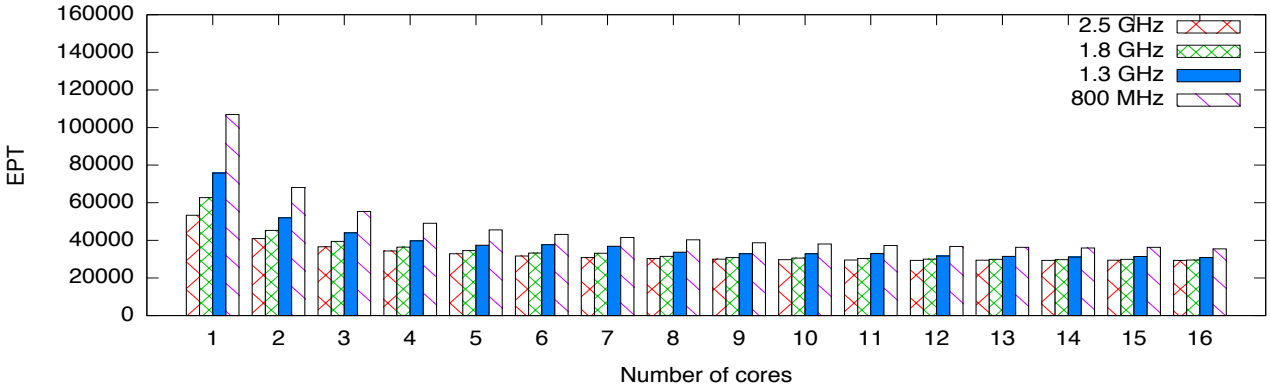

Figure 4. EPT of Raytrace using sleep state

SH This strategy applies both the Sharing (S) and the Hare Policies (H) in each node, but without task migration. The Sharing Policy selects the schedule with the least PPS value.

SHG This strategy uses the global task migration manager (G) to migrate the tasks as in the strategy HG, but executes the tasks on Node 3 and 4 with the Sharing and the Hare Policies (SH). The Sharing Policy selects the schedule with the least PPS value.

The results are shown in Table III for the scenario of a small 4-node multicore cluster. We can see that when both the Hare and the Sharing Policies are applied to the individual nodes (case SH), it can save energy up to 57.43%, compared with the non-power-aware strategy. When the new policies are combined with task migration (case SHG), the saving is up to 58.63% compared with the non-power-aware strategy. This percentage of saving can result in significant amount of energy saving for a large scale multicore cluster.

	Node 1		Node 2		Node 3		Node 4		energy(J)
policies	I	C	I	C	I	C	I	C	
-	2	-	2	-	2	-	4	-	-
N	2	16	2	16	2	16	4	16	233294.18
HG	8	16	2	16	-	-	-	-	115675.98
SH	2	8	2	8	2	8	4	4	99321.95
SHG	8	2	2	8	-	-	-	-	96509.55

Table III
EXPLORATION OF MULTIPLE SCHEDULING POLICIES, N = NON POWER-AWARE, G = GLOBAL TASK MIGRATION, S = SHARING POLICY, H = HARE POLICY

IV. RELATED WORK

There are some power efficiency metrics such as ED^n [13] and GCPI [14] proposed recently. Similar to Performance per Watt, they stress more on performance. In contrast, our metrics like PPS and EPT stress more on power and energy.

There has not been much work done on energy saving through task scheduling. One such policy [15] uses DVFS to reduce power consumption in real-time systems by lowering the CPUs operating frequency to the smallest possible value that will not violate any deadlines. However, it does not take parallelism into consideration. In [16], an idea similar to our Hare Policy was mentioned but not explored.

In [17], the proposed scheduling policies used profiling (online and offline) to choose the most suitable cores to run applications based on the measure of Instructions per Cycle (IPC). A round-robin policy was used to gather the performance for each core type, before a scheduling decision

was made.

Other papers such as [18] have explored the benefits of Asymmetric Multicore Processors (AMP) on parallel applications. Simple policies are used in the paper, where parallel applications are scheduled on the slower cores, while sequentially executing programs are given the fast cores because they are seen to gain the most benefit from the higher frequency. Since these policies do not use metrics like EPT or PPS, the proposed policies seem to be empirical.

Other research which is complementary to our new policies such as the Sharing Policy is [11], which looked into the use of virtualization in a cloud infrastructure, allowing for workloads to be migrated between nodes. This meant that unused nodes were able to be shutdown, without impacting task completion.

Similar work to this has also being carried out in [12] which looks at using virtualization to allow the workloads to be moved around without interruption, as well as using DVFS policies, to reduce the operating frequency of the nodes with low utilization.

V. Conclusions and Future Work

In this paper we have proposed three new metrics (SPW, PPS and EPT) which are used for evaluating the energy efficiency of task schedulers in multicore nodes. Three new scheduling polices (Sharing, Hare and Tortoise) have been proposed and evaluated with the above metrics in our experiments. The Sharing Policy can save up to 20% energy and the Hare Policy saves up to 72% energy according to our experiments.

Our results show that we should pay more attention to energy efficiency rather than power efficiency, especially when the utilization of the multicore nodes is low or the performance of applications is not the greatest concern.

We have also found that parallelism can play an important role in energy saving, especially when the applications scale well with the number of cores.

Our experiments suggest that, to enable the Hare Policy to save more energy than the Tortoise Policy, multicore computers should have a more feasible power management interface that can put into sleep and wake up idle nodes quickly with little overhead.

In the near future, we will further test both the Hare and Sharing Policies on individual nodes with benchmark suites such as those used by GCPI [14] and SPEC Power and Performance [19]. Following on from this we would like to incorporate them into a global task manager for use on multicore clusters.

References

[1] B. Nordman and K. Christensen, "Greener PCs for the enterprise," *IT Professional*, vol. 11, no. 4, pp. 28–37, 2009.

[2] A.-C. Orgerie, L. Lefèvre, and J.-P. Gelas, "Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems," in *ICPADS 2008: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, 2008, pp. 171–178.

[3] L. A. Barroso and U. Hoelzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[4] R. Ge, X. Feng, H. Pyla, K. Cameron, and W. Feng. (2010, Mar.) Power measurement tutorial for the Green500 list. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.6243&rep=rep1&type=pdf

[5] G. F. Townsend, *Three Hundred Aesop's Fables: Literally Translated from the Greek*. London: George Routledge and Sons, 1867.

[6] O. A. R. Board, *OpenMP Application Program Interface Version 3.0*, May 2008.

[7] (2010, May) Watts up? operators manual. [Online]. Available: https://www.wattsupmeters.com/secure/downloads/manual_rev_9UO0812.pdf

[8] J. Zhang, Z. Huang, W. Chen, Q. Huang, and W. Zheng, "Maotai: View-oriented parallel programming on CMT processors," in *The 37th International Conference on Parallel Processing (ICPP08)*, 2008.

[9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24–36.

[10] AMD. (2010, May) Quad-core amd opteron processor with direct connect architecture. [Online]. Available: http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Quad-Core_AMD_Opteron_processor_2P_power_consumption_comparison.pdf

[11] L. Lefèvre and A.-C. Orgerie, "Designing and evaluating an energy efficient cloud," *The Journal of Supercomputing*, vol. 51, no. 3, pp. 352–373, 2010.

[12] F. Hermenier, N. Loriant, and J.-M. Menaud, "Power management in grid computing with Xen," *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, vol. 4331, pp. 407–416, 2006.

[13] S. Sharma, C.-H. Hsu, and W. chun Feng, "Making a case for a Green500 list." in *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006)/ Workshop on High Performance - Power Aware Computing*, 2006.

[14] W.-C. Feng and T. Scogland, "The Green500 List: Year One," in *5th IEEE Workshop on High-Performance, Power-Aware Computing (in conjunction with the 23rd International Parallel and Distributed Processing Symposium)*, Rome, Italy, May 2009.

[15] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design & Test*, vol. 18, no. 2, pp. 20–30, 2001.

[16] A. Grover, "Modern system power management," *Queue*, vol. 1, no. 7, pp. 66–72, 2003.

[17] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*, 2006, pp. 29–40.

[18] A. Fedorova, J. C. Saez, D. Shelepov, and M. Prieto, "Maximizing power efficiency with asymmetric multicore systems," *Communications of the ACM*, vol. 52, no. 12, pp. 48–57, 2009.

[19] *SPEC Power and Performance Benchmark Methodology V1.1.1*, Standard Performance Evaluation Corporation.