Fine-grained CPU Throttling to Reduce the Energy Footprint of Volunteer Computing

Technical Report

Peter Hanappe
Sony Computer Science Laboratory Paris
6 rue Amyot, 75005 Paris

January 2012

Dynamic Voltage Scaling (DVS) is a key CPU technology to reduce the energy footprint of volunteer computing networks. However, most operating systems do not give an application control over its runtime CPU frequency. We show that computation-intensive applications can regain some control over the DVS through fine-grained CPU throttling. We compare our technique with the coarse-grained throttling technique found in the BOINC framework [1]. Using either the fine or coarse grained throttling, we estimate that volunteer computing networks can reduce their energy consumption by more than 40%.

1 Introduction

Volunteer computing is a form of grid computing in which the nodes are the personal machines of volunteers. When the PCs are idle, a scientific application uses the machine's resources to run simulations or to process data. Most often, volunteers leave their machine activated during their absence to allow the scientific application to run. Our goal is to change this approach and to recommend volunteers to perform the computation in the background while they are actively using the machine. During extensive office work, the average CPU load rarely rises above 20% [3, 7] and the remaining processing capacity could be used advantageously for scientific computing.

To run the CPU intensive application in the background, the volunteer computing middleware will have to take precautions. In particular, the middleware must keep the computer in a low-performance state for two reasons: 1) to minimize the additional

energy requirements and 2) to avoid the heating up of the machine and the activation of the CPU fans.

Dynamic voltage scaling (DVS) allows a CPU to run at different voltages and clock frequencies. When a CPU executes at a lower frequency, the required voltage can be reduced, too. This yields an energy saving following the approximate equation $P = CV^2f$ with P the power consumption of the processor, C its capacitance, V the applied voltage, and f the clock frequency [2].

DVS has been proposed as a solution to reduce energy for multi-tasking machines [8] or for grid computing networks [5]. It has been widely adopted and has been standardized in the Advanced Configuration and Power Interface standard (APCI). ACPI defines a number of performance state, or P-states, ranging from P_0 (high performance) to P_N (low performance), with $N > 0$ hardware-dependent.

Popular operating systems have based their implementation of the DVS control system on interval-based algorithms [6]: an analysis of the CPU load in the previous time slices is used to estimate the CPU load of the next time slice and to set the CPU frequency accordingly.

The problem with interval-based approaches is that, without additional information, they will boost the CPU performance for the CPU intensive, volunteer computing application. There is unfortunately no application-centric interface in GNU/Linux, Windows 7, or Mac OS to prevent this.

We hypothesized that by artificially slowing down the scientific application, the OS's interval-based mechanism can be tricked into keeping the CPU in a low-power state.

The Berkeley Open Infrastructure for Network Computing (BOINC), a widely used framework for volunteer computing [1], also throttles the scientific computation by intermittently putting the application to sleep. However, the coarseness of the intervals, on the order of seconds, gives the computer's DVS system the time to adjust the P-state, which alternates between P_0 and P_N.

In this paper, we will test what happens when these work/sleep intervals become smaller than the update intervals used by the DVS system. We will see that the CPU no longer alternates between P_0 and P_N, and that it can remain in a low-power mode.

2 Methodology

To measure the energy consumption, we created a dummy scientific project in BOINC. The task consists of incrementing a counter for a fixed number of times. On our test machines, we also installed a small daemon application that provides the work units with performance information, including the machine's energy consumption, CPU load, and the current P-state. This data is made available to the work units through a shared-memory segment. The work units return this information to the BOINC server for evaluation. The system-wide energy consumption is measured using a WattsUp! power meter that is connected to the computer over USB. The instantaneous power usage is measured every second and integrated over time to estimate the total energy needs.

We compared the energy that is needed to execute a work unit in four different configurations:

1. *Dedicated*: The computer is dedicated to the scientific application and all its resources are available for the execution of the work units.
2. *BOINC's CPU throttling*: BOINC's user-space CPU throttling is used to limit the average CPU load to 20%.
3. *Fine-grained CPU throttling*: The computation is artificially slowed down use fine-grained throttling in order to trigger the OS's interval-based DVS scheme.
4. *Low-performance*: We manually configure the OS to keep the CPU in the lowest P-state.

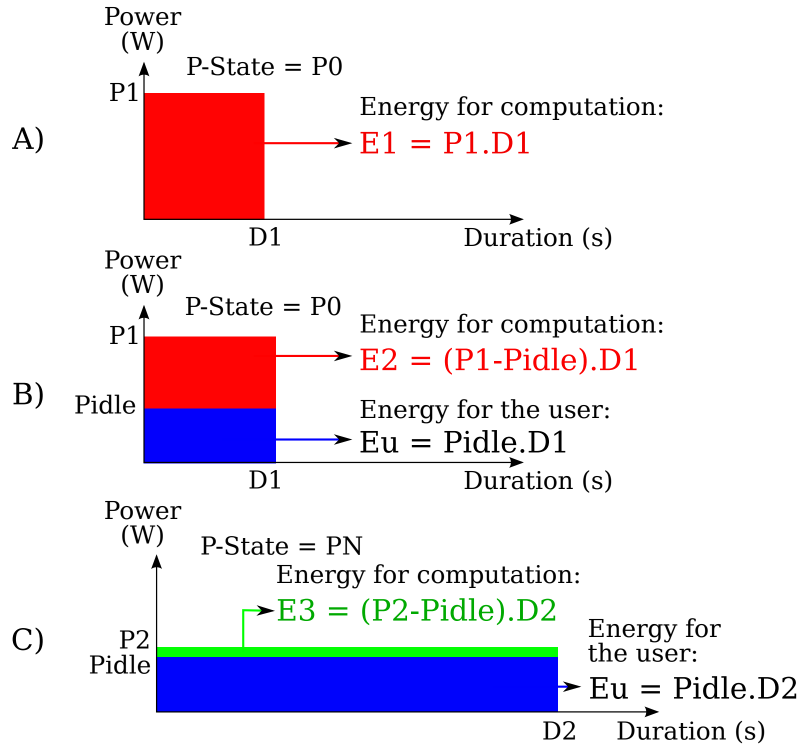

Figure 1: A) The energy requirements for the computation on a dedicated machine. B) The energy requirements for background computing, in high-performance mode. C) The energy requirements for background computing, in low-performance mode.

We made the following assumptions. First, we targeted personal computers only, not dedicated servers. We also supposed that the computation takes place in the background during active use of the machine and that the average CPU load of the owner's

	Machine 1
Model	VAIO VPC-F11S1E
CPU	Intel Core i7 Q720
Cores	4
Frequency	0.933 - 1.60 GHz
P-States	6
Idle energy	38.0 W (Linux)
	33.0 W (Windows 7)

Table 1: The hardware configuration of test machine 1.

	Machine 2
Model	VAIO VGN-SZ71VN
CPU	Intel Core 2 Duo T9300
Cores	2
Frequency	0.8 - 2.5 GHz
P-States	5
Idle energy	27.0 W (Linux)

Table 2: The hardware configuration of test machine 2.

applications is low [3, 7]. Furthermore, we did not take into account the power needed to keep the machine alive, except for the dedicated set-up. That means that we only considered the *additional* power consumption required to perform the computation. As can be seen in Fig. 1b, by running the computation in the background, a large part of the total energy consumption is put on the owner's account. When the computation is slowed down (Fig. 1c), it will take longer but require less additional energy. The last assumption we made is that the scientific application is CPU intensive and that it puts a long and continuous workload on the system.

We compared the results of two laptop computers, shown in Tables 1 and 2. The Linux tests were performed on machines 1 and 2. The Windows 7 tests were performed on machine 1, only. In all tests, the displays remained lit and the batteries were removed.

3 Results

The summary of the results is shown in Tables 3 to 5.

Dedicated set-up On a dedicated machine, we measure effectively that the CPU's load is around 98%. On our first testing machine, the application requires about 562.6 seconds to complete a work unit. During the execution, the machine has an average consumption of about 84.5 Watt which leads to an the energy per work unit of 11.88 kJ. On machine 2, the work units take 944.0s to complete while the laptop draws 49.46 W. With two concurrent tasks, this results in roughly 23.35 kJ to complete a work unit.

	Total energy (kJ)	Add. energy (kJ)	Savings	Time (s)	P_0 state
Dedicated	11.88	11.88	-	562.6	98%
BOINC throttling	49.42	11.70	2%	3970.6	26%
Fine-grained throttling	21.63	7.01	41%	1538.9	10%
Low performance	15.53	5.65	52%	1040.5	1%

Table 3: The summary of the results on machine 1 running GNU/Linux.

	Total energy (kJ)	Add. energy (kJ)	Savings	Time (s)	P_0 state
Dedicated	23.35	23.35	-	944.0	98%
BOINC throttling	88.20	15.61	33%	5377.1	20
Fine-grained throttling	66.38	11.17	52%	4089.9	12%
Low performance	42.84	7.36	68%	2901.0	2%

Table 4: The summary of the results on machine 2 running GNU/Linux.

	Total energy (kJ)	Add. energy (kJ)	Savings	Time (s)	P_0 state
Dedicated	9.68	9.68	-	130.92	96.9%
BOINC throttling	26.88	4.76	51%	670.28	22.5%
Fine-grained throttling	19.01	5.94	39%	396.19	35.6%
Low performance	10.47	4.01	59%	195.64	0%

Table 5: The summary of the results on machine 1 running Windows 7.

Using BOINC's CPU throttling In the second series of tests, we configured BOINC to restrict the CPU usage of the scientific application to 20%. This value is somewhat arbitrary but it was chosen because we found that if we set the CPU usage to a value greater than 20%, the machine quickly becomes noisy and that the computation can no longer be performed in the background. For completeness, we included Figure 2, which shows the amount of additional energy and the computation time in function of the BOINC's CPU load setting. In a 20% configuration, BOINC will let the application run for 2 seconds and then put it to sleep for 8 seconds. During the intervals in which the application runs, the CPU load will be 100%. When the application sleeps, the CPU load will fall back to its idle level.

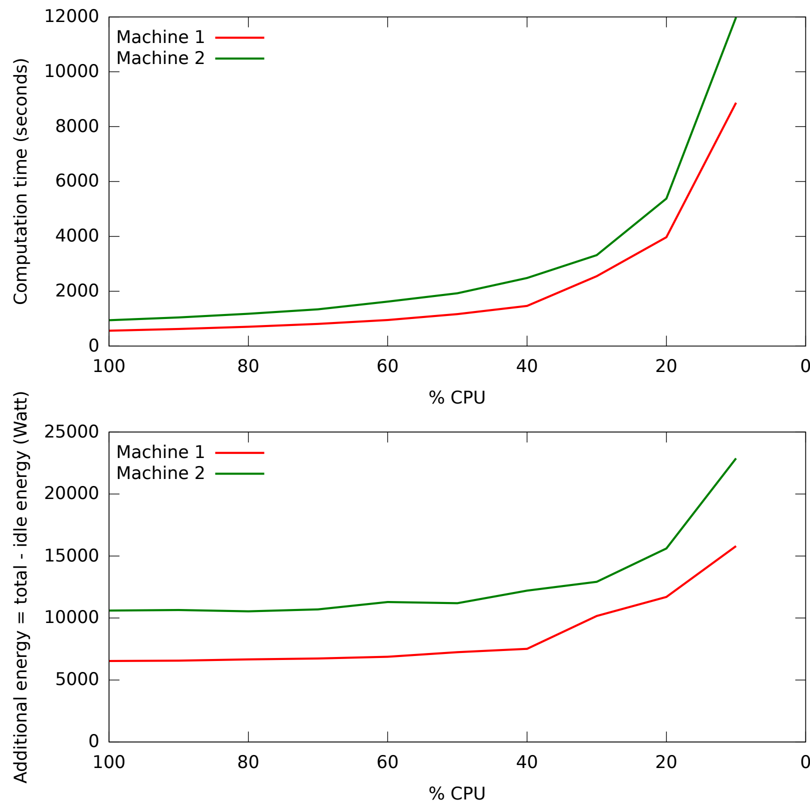

Figure 2: The computation time and the energy consumption as a function of the CPU time, as managed by the BOINC client. The graph shows the additional energy and does not include the "idle" energy consumption.

On Linux, we measure an additional energy requirement of 11.70 kJ for machine 1 and of 15.61 kJ for machine 2. So there is a reduction of 33% of energy requirements for machine 2, but almost none for machine 1 (2%). On Windows 7, the results are more satisfying, with 51% less energy used for machine 1.

```
start = now()
loop until work done
    do some work
    if (now() - start > T)
        sleep(S)
        start = now()
    end if
end loop
```

Figure 3: The pseudo-code of the time-based algorithm.

Using fine-grained throttling For the fine-grained throttling , we tested two approaches. The first one is best suited for Linux. Consider the pseudo-code in Figure 3. This small application repeatedly performs some work during the interval T and put itself to sleep for a period S. By varying the values of T and S, we should observe a change in the behavior of the operating system. For $S = 0$, we see the behavior of the dedicated system discussed above. No idle intervals are introduced and the CPU load will be 100%. When T and S are large compared to the OS's scheduling interval, we see a succession of high and low CPU loads, similar to what we discussed for BOINC's coarse-grained throttling.

On GNU/Linux, interesting behavior can be observed for small values of T and S. The average CPU load stays low and the operating system does not increase the CPU frequency on behalf of our application (Fig. 4 and 5). To measure the small time intervals, we used Intel's Time-Stamp Counter, and to put the application to sleep, we used the `clock_nanosleep` function.

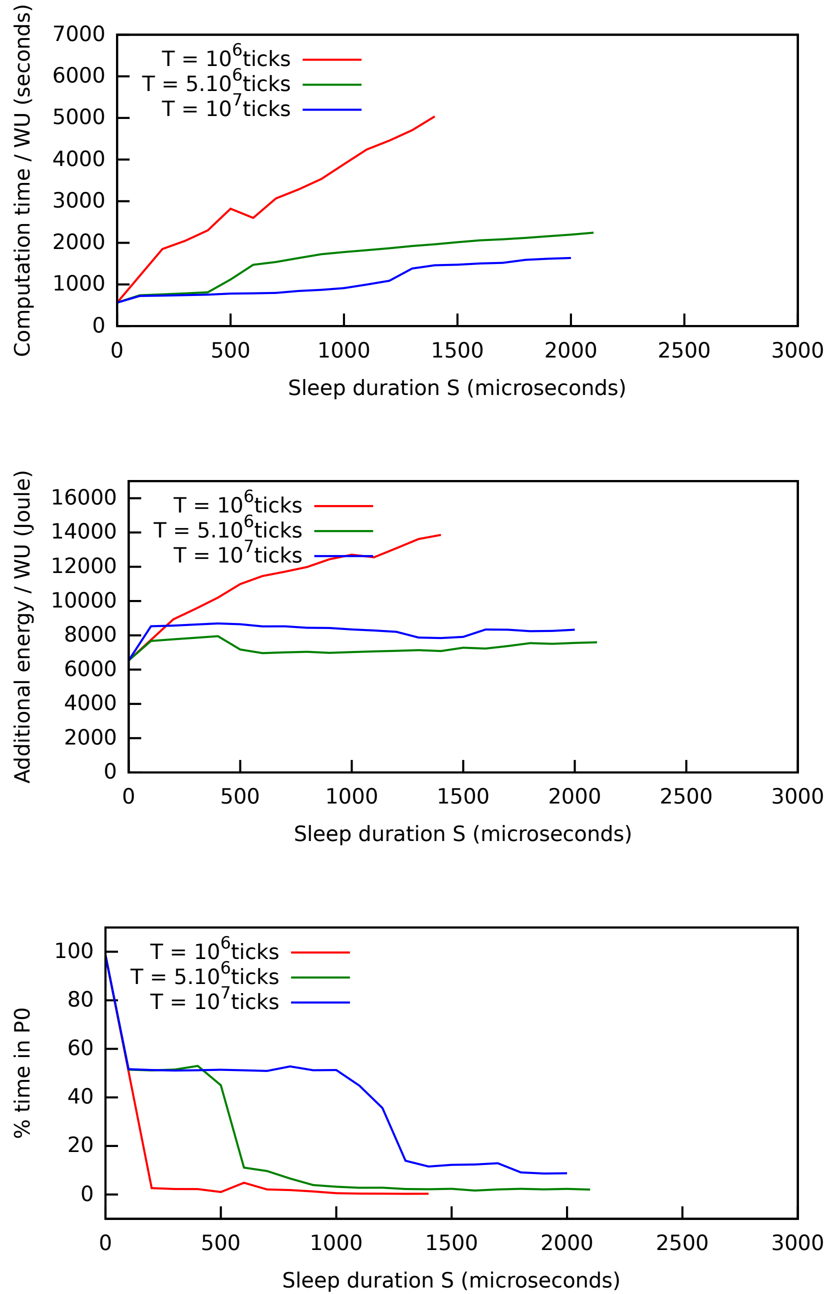

Figure 4: For machine 1, running Linux: The computation time per work unit (top), the additional energy per work unit (middle), and the percentage of time spent in P-state P_0 (bottom) in function of the sleep duration S and for three different values of T.

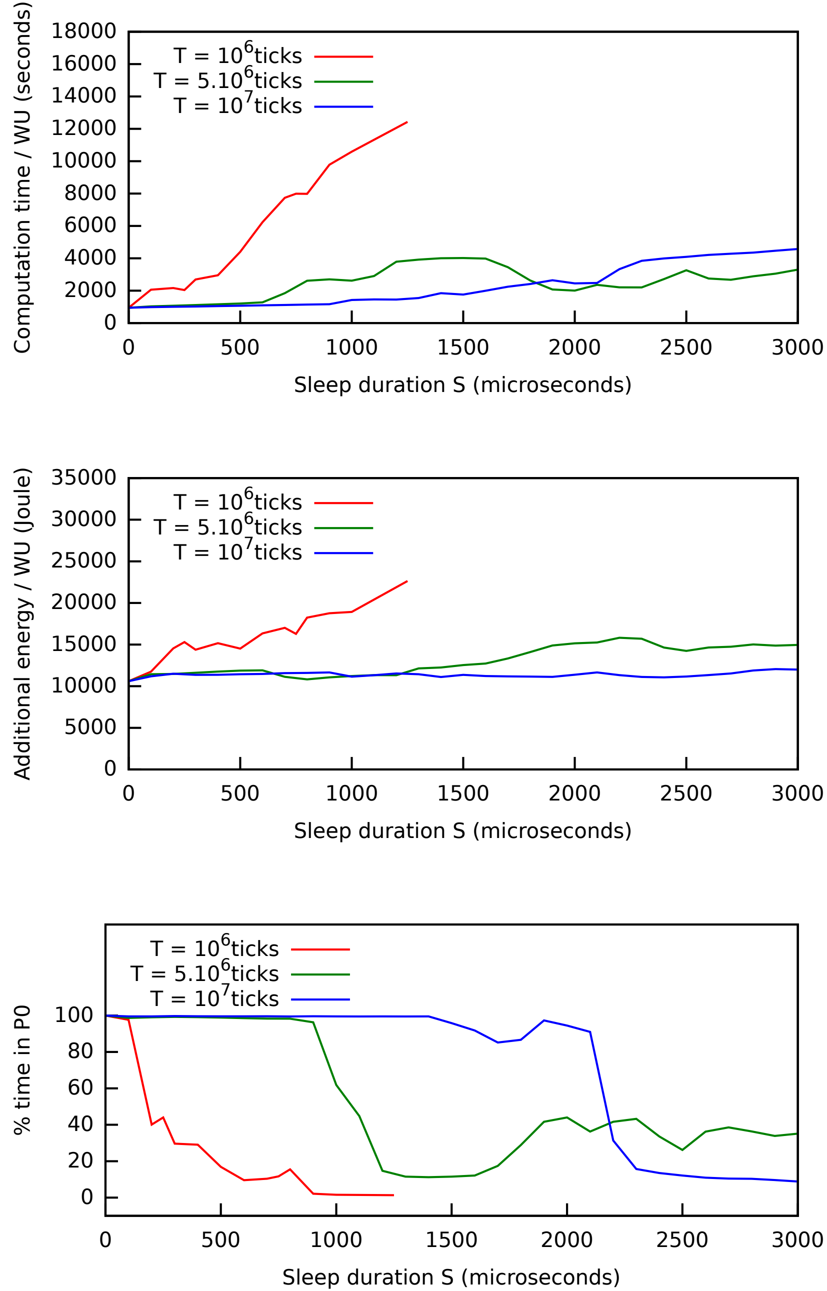

Figure 5: For machine, 2 running Linux: The computation time per work unit (top), the additional energy per work unit (middle), and the percentage of time spent in P-state P_0 (bottom) in function of the sleep duration S and for three different values of T.

With a value of $S = 700\mu s$ and $T = 5.10^6$ CPU clock ticks, machine 1 completes a work unit using 41% less energy (7.01 kJ of additional energy). On machine 2, the

energy reduction is 52% (11.17 kJ of additional energy) for $S = 2500\mu s$ and $T = 10^7$ CPU clock ticks. The technique is reasonably successful in keeping the machine in a low P-state: only 10 to 12% of the time is spent in P_0.

On machine 1, running Windows 7, this method produced erratic results and we therefore adopted a slightly different approach: the application goes to sleep for an interval S as soon as finds that CPU P-state is in high-performance mode (Fig. 6). The results can be seen in Fig. 7. For, S equal to 10 ms, machine 1 consumes 5.94 kJ of additional energy, or 39% less than the dedicated case. The method is not as effective as on Linux in maintaining a low P-state. One third of the time, the CPU is in state P_0.

```
start = now()
loop until work done
    do some work
    if (P-state == P0)
        sleep(S)
    end if
end loop
```

Figure 6: The pseudo-code of the P-state based algorithm.

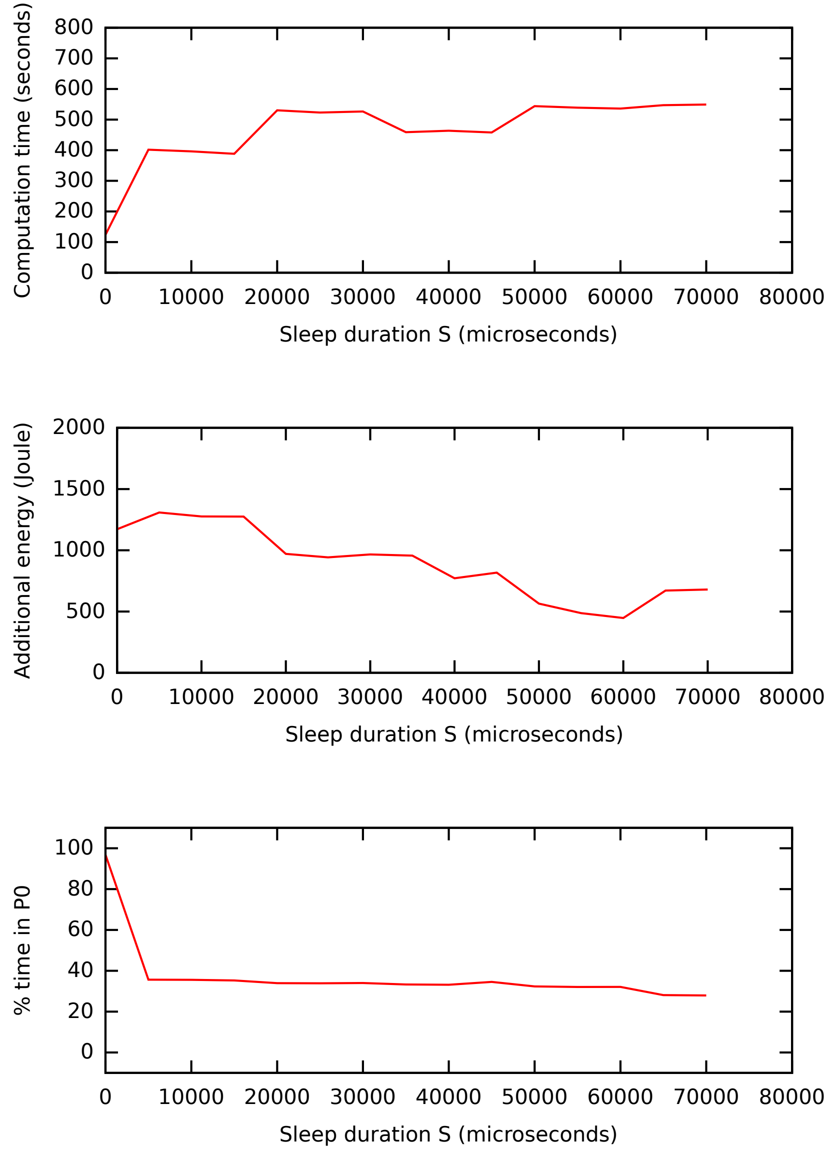

Figure 7: For machine 1, running Windows 7: The computation time per work unit (top), the additional energy per work unit (middle), and the percentage of time spent in P-state P_0 (bottom) in function of the sleep duration S.

Low-performance mode To measure how much energy could be saved if application-level support for DVS were available, we manually fixed the CPU in the low-performance state. On Linux, we used the `ignore_nice_load` flag exported by the *ondemand* kernel module. When this flag is set, the kernel module will not include low priority ("nice") processes in its evaluation of the CPU load. Both test machines stay in the lowest P-state for more than 98% of the time. On machine 1, we find that a work unit requires 5.65

kJ of additional energy to complete. This is less than 50% of the energy consumption when the work units run on a dedicated machine. On machine 2, the additional energy per work unit is 7.36 kJ, or 68% less than on a dedicated machine.

On Windows 7, we limited the maximum CPU frequency of the whole system by changing the maximum frequency in the CPU Performance Management settings [4]. The slow background computing required 59% less energy than the dedicated case.

4 Discussion and conclusion

In this paper, we evaluated whether is is possible to programmatically slow down an application and to trick the operating system's DVS control to reduce the CPU frequency.

On Linux, the fine-grained throttling gave very encouraging results. The method kept the CPU in a low-performance state for more than 88% of the time. As a result, the computation of a work unit can be performed in the background and requires 40% less energy than on a dedicated machine. Additional work is needed to automatically detect the appropriate values of S and T and to test the robustness of the method under a variable application mix.

On the tested Windows 7 system, the fine-grained throttling reduced the energy requirements by 39% but was less successful to maintain the CPU in low P-state. For this configuration, BOINC's coarse-grained CPU throttling is a more effective solution.

In this text, we focused mostly on maintaining a low P-state and reducing the energy consumption. Obviously, slowing down the computation, either through CPU throttling or by manually fixing the P-state, also extends the time needed to complete a work unit. In our test, the computation took between 1.5 and 7 times longer. In a volunteer computing network, this could be counterbalanced by having more volunteers to participate.

The numbers we have obtained depend on an estimation of the idle power of the machines, which is neither clearly defined nor easy to measure. We also tested an artificial benchmark application on a small set of machines. Further work is therefore required to validate the presented method using real applications on a wider range of hardware platforms.

References

[1] D. P. Anderson, C. Christensen, and B. Allen. Designing a runtime system for volunteer computing. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Supercomputing '06, New York, NY, USA, 2006. ACM.

[2] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, pages 288–297. IEEE, 1995.

[3] P. Domingues, P. Marques, and L. Silva. Resource usage of windows computer laboratories. In *Proceedings of the International Conference on Parallel Processing Workshops ICPPW05*, pages 469–476. IEEE, 2005.

[4] Processor Power Management in Windows 7 and Windows Server 2008 R2. Technical report, Microsoft Corporation, January 2010.

[5] B. Schott and A. Emmen. Green desktop-grids: Scientific impact, carbon footprint, power usage efficiency. *Scalable Computing: Practice and Experience*, 12(2):257–264, 2011.

[6] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37:195–237, September 2005.

[7] J. Wang, Y. Sun, and J. Fan. Analysis on resource utilization patterns of office computer. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, pages 626–631. IASTED/ACTA Press, November 2005.

[8] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.