Simulation of Cluster Power Consumption and Energy to Solution

Timo Minartz[1], Julian M. Kunkel[2], Thomas Ludwig[1,2]
minartz@informatik.uni-hamburg.de

1: Department Informatik, Universität Hamburg
2: Deutsches Klimarechenzentrum

Abstract

In recent years the power consumption of high-performance computing clusters has become the limiting factor of scalability. The high power consumption of clusters is a consequence of their design goal: high performance. In high performance computing a big amount of energy is needed to reach the peak performance up to 2 TFlops under load. But with low utilization, today's cluster hardware consumes nearly as much energy as when fully utilized. Theoretically, in low utilization phases cluster hardware can be turned off or switched to a low-power state without affecting time-to-solution, while decreasing energy-to-solution.

We designed a model to estimate power consumption of hardware based on utilization. Applications are instrumented to create utilization trace files for a simulator realizing this model. Each hardware component can be simulated using one of multiple estimation strategies. The optimal strategy determines an upper bound of energy savings for existing hardware without affecting the time-to-solution. Additionally, the simulator can estimate the power consumption of energy-proportional hardware. This way the minimal energy-to-solution can be determined for a given application. Naturally, this provides an upper bound for any power saving strategy. Experiments on a small cluster show the validity of this approach. The capability to monetarily evaluate concrete application traces with different power-saving strategies is evaluated for a Jacobi PDE solver.

Model of Cluster Power Consumption

In general, power consumption of a cluster environment is the sum of the power consumption of its components, especially nodes, switches and HVAC (Heating, Ventilating and Air Conditioning). The power consumption for the HVAC depends on the power consumption of the other components. Every watt needed by a component results in heated air, which has to be cooled down. Therefore, total power consumption can be estimated by multiplying power consumption of the cluster with the PUE (Power Usage Effectiveness) of the whole environment.
The estimation of the node's consumption is the main challenge, because power consumption of today's switches used in the HPC environment is nearly independent of network utilization. Most switches consume as much power when active as when inactive. Hence, the power consumption of a switch is a constant, which can be added to the node's power consumption.
The power consumption of a node is the sum of its components' power consumption. Some of these hardware components (like CPU or power supply) have a bigger effect on the power consumption than other components. And for most components the power consumption is based on the utilization.

Switching between two states is realized by sequentially incrementing or decrementing states until the desired state is reached.
Based on this model and the utilization from the trace file, the power consumption is estimated. For each fixed step (defined by the tracing periodicity of the utilization values), the power consumption for each component is calculated. Without usage of state changes this is the simplest strategy for power estimation and equates to the real power consumption, because the tested cluster hardware is not ACPI capable. However, with the model potential power savings can be calculated, when putting components to a higher state. To estimate the power consumption of a cluster four different strategies have been implemented -- each component in a node can use an individual strategy. The four strategies are named: *Simple*, *Optimal*, *Approach* and *MultipleState*. The first strategy estimates the power consumption without using any power saving mechanism. For the Simple strategy, every single timestep is evaluated independently. The other strategies are look-ahead strategies, which make decisions based on future utilization values. The *Optimal* strategy decides to put a component into sleep mode, if the future utilization is zero for a sufficient period of time, and wakes up the component in time. Because periods with zero utilization are not frequent for some components such as a CPU, the *Approach* strategy puts a low-utilized component into sleep and delays the utilization, e.g. below 5 %. To compensate for this utilization, at the end of a low utilization phase an equivalent phase with high utilization is created. The last strategy (*MultipleState*) is an extension of the *Optimal* strategy allowing multiple states with different power consumptions and wake up times (all ACPI Device Power States). This is useful for components like the main memory, if the main memory utilization is under 50 %, half of the banks can be theoretically switched off to conserve energy.

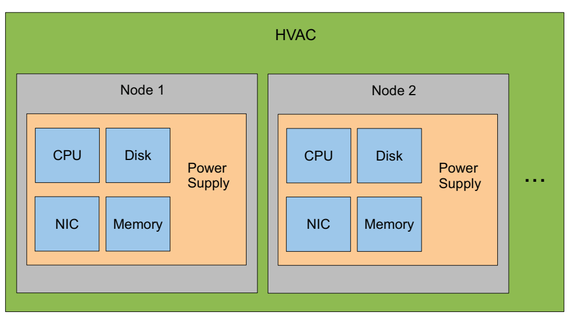


Figure 1: Modelled Components

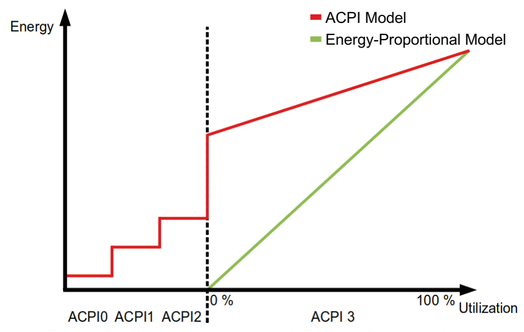


Figure 2: Model of Component Energy Consumption

Power Estimation Strategies

Optimal Strategy

The *Optimal* strategy calculates the minimal power consumption without reducing the performance, because components only switch to sleep mode if not used, and wake them up in time. The decision whether a component switches to sleep mode depends on multiple factors: the duration of the zero utilization phase, the power saving potential of the state change (P_{diff} – difference of power consumption between state 0 and state 3), the duration of the state change (t_{change} – sum of duration state 0 to state 3, and of duration state 3 to state 0) and the energy needed to perform the state changes (E_{change} – also the sum of both changes). With these values the minimal duration of a zero utilization phase $t_{optimal}$ can be calculated. For low-utilization phases longer than $t_{optimal}$ sleeping reduces the power consumption without reducing performance.

$$t_{optimal} = \frac{E_{change}}{P_{diff}} + t_{change}$$

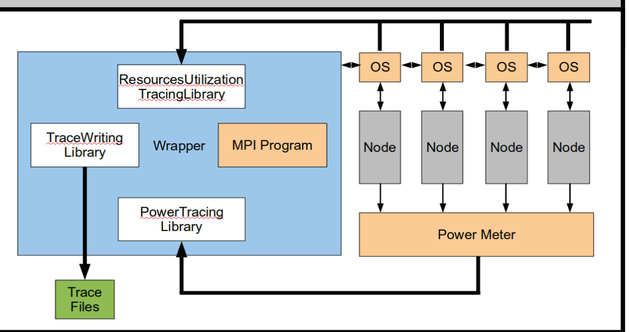
Approach Strategy

The intention of the *Approach* strategy is to simulate smart utilization of each component by rearranging (deferring) load. The rearrangement of load can be thought of reordering the code or imposing strategies like write-behind. However, in reality this might not be possible due to dependencies of the code. Consequently, the output produced by this strategy shows scopes where power saving could be scheduled.
Now, the calculation of the minimum time for an efficient state change depends on a tolerance value δ. The timesteps with a utilization lower than δ will be rearranged if the aggregated duration of timesteps under the specified tolerance is greater or equal $t_{apprcach}$. The rearrangement of load results in a phase with zero utilization and a utilization phase containing the equivalent load (t_{load}). The equivalent load is the aggregated utilization for the low-utilization phase.

$$t_{approach}(input) = t_{optimal} + t_{load}(input)$$

Methodology

The tracing is realized by an MPI wrapper library called *HDMPIwrapper* which intercepts the MPI library calls and traces multiple data sources: the *ResourcesUtilizationTracingLibrary* logs the utilization of various components of a node (CPU utilization in percent, memory usage in bytes, network usage in bytes, disk usage in blocks). The *PowerTracingLibrary* logs the amperage in A, the voltage in V and the resulting power consumption in W also on node level. To trace the power consumption an external power meter (LMG-450 of ZES ZIMMER Electronic Systems) is used. This power meter has a high accuracy and allows to trace the power consumption of four cluster nodes at once. For our experiments, the sampling interval is set to 100 ms. A graphical overview of the tracing environment is given in the figure.



Evaluation

Hard- and Software Environment

Experiments were conduced on the reseach group's 10 node cluster. A cluster node has two Intel Xeon Northwood CPUs (single core) with 1024 MB main memory (two memory bars). Each node has a local P-ATA disk and GigE. The cluster nodes run *Ubuntu 8.04* (Kernel 2.6.30) via NFS. Applications use MPICH2 (version 1.0.8) and PVFS (version 2.8.1) as parallel filesystem.
As a benchmark the program *partdiff-par* (a parallel PDE solver) has been chosen due to the following features:
•The calculation-communication ratio is flexible – based on the interference function of the matrix.
•It uses parallel I/O to perform checkpointing and monitoring of the PDE convergence.
•The component utilization depends on the checkpointing frequency and the calculation-communication ratio.
•*partdiff-par* is a real application and no synthetic benchmark.

To determine the component power consumption the zero utilization power consumption is measured with different hardware configurations. To estimate the power consumption of the utilized components a micro benchmark is used. The power characteristics for the sleeping state (ACPI state 3) are typical values from desktop and mobile components, because the concrete hardware doesn't support multiple ACPI states.

	CPU	Memory	Disk	NIC
$P_{100\%}$	58 W	13.585 W	7.02 W	2 W
$P_{0\%}$	28.1 W	10.985 W	4.42 W	0.78 W
P_3	4 W	0.1 W	2 W	0.2 W
$E_{0\to3}$	$2.74 * 10^{-7}$ Wh	$1.1 * 10^{-8}$ Wh	0.001 Wh	10^{6} Wh
$E_{3\to0}$	$2.74 * 10^{-7}$ Wh	$1.1 * 10^{-8}$ Wh	0.026 Wh	0.001 Wh
$t_{0\to3}$	0.017 ms	0.006 ms	1000 ms	0.1 ms
$t_{0\to3}$	0.017 ms	0.006 ms	4000 ms	0.1 ms

Table 1: Component power characteristics for the experiments

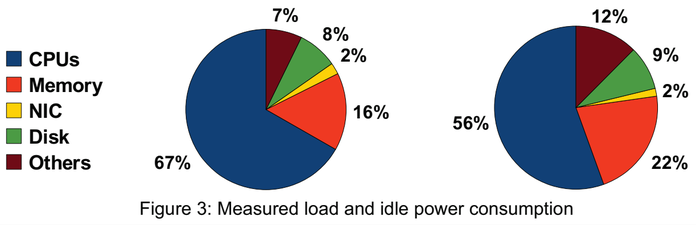


Figure 3: Measured load and idle power consumption

Energy Efficient Sleeping

In this experiment multiple program configurations without checkpointing and a varying count of MPI processes are traced. Based on these traces the ratio between calculating and sleeping devices is analyzed. The program *partdiff-par* is configured with a varying count of iterations, each of these was run with 4 and 8 MPI processes respectively.
The estimated energy consumption of each setup is shown in table 2 and visualized in figure 3.
For this experiment the configuration of four calculating CPUs and four sleeping CPUs can be more energy efficient than using eight CPUs for calculating. This is based on the parallelization overhead, which exceeds the performance growth. Using 100 and 200 iterations respectively, the run time is increased by a multiple of the run time with 5 iterations. Even with those setups the configuration with four calculating CPUs (and sleeping of the remaining four CPUs) is more power efficient than using eight CPUs for calculating, although the total power saving decreases.
With 500 iterations the performance growth exceeds the overhead of the parallelization and using eight CPUs for calculating decreases the total energy consumption in comparison to the setup with four CPUs.

Setup		Energy consumption in Wh			Savings in %		Time in sec
Iter	MPI proc	Simple	Optimal	Approach	Optimal	Approach	
5	4	2.153	1.554	1.432	27.8	33.5	6
5	8	2.141	1.610	1.598	24.8	25.4	4
100	4	18.663	13.186	14.121	29.4	24.3	105
100	8	15.262	13.983	14.105	8.4	8.6	67
200	4	36.142	26.429	27.467	26.9	24.0	209
200	8	29.048	27.007	27.101	7.0	6.7	133
500	4	90.615	71.330	69.193	21.3	24.6	536
500	8	70.592	6.253	66.218	6.1	6.2	334

Table 2: Estimated ETS on four cluster nodes

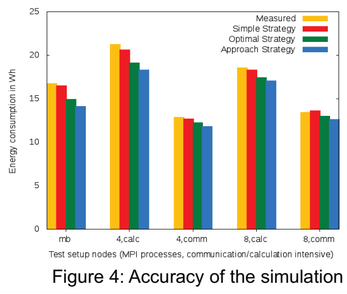


Figure 4: Accuracy of the simulation

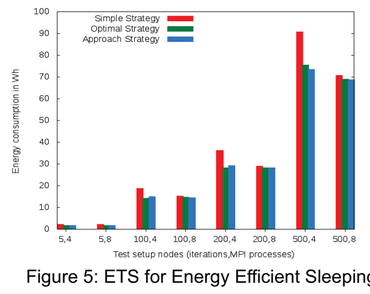


Figure 5: ETS for Energy Efficient Sleeping

Energy-Proportional Devices

The simulation allows to change the specifications of the underlying devices to compare different hardware power characteristics for a specific trace.
This test set characterizes each node device as energy-proportional device. As already discussed the power consumption for zero utilization is relatively high for each device. When this overhead can be technically reduced by the hardware vendors, power efficiency can be increased. The CPU with the optimal specification would consume zero watt if not utilized, the mentioned overhead would be omitted.
For the micro benchmark trace savings of about 30 % are estimated for the energy-proportional devices, the minimal saving of the traces with checkpointing is about 23 %. In these setups the savings are mainly dependent on the hardware utilization, the setups with four MPI processes have a greater energy saving potential, because the other four CPUs do not consume any energy when idle.
For the trace without checkpointing and better utilization of the hardware (using eight MPI processes on four nodes) potential savings of only 9 % have been estimated. For the trace with many idle times (using one PVFS server and four MPI processes on four nodes with checkpointing) the estimated saving with energy-proportional devices is about 63 %.
Comparing the power estimation with *Optimal* strategy and the estimation with energy-proportional devices, it becomes clear that the energy saving with energy-proportional devices are greater than the savings with the *Optimal* strategy. For these specific test cases the additional savings compared to *Optimal* strategy are between 7 % and 51 %. The lower this additional saving, the lower is the overhead for the hardware for this specific application.
Of course the savings (and the hardware energy overhead) for well-utilized hardware are much smaller than for idle hardware. Hence, these values are heavily dependent on the traced program and the configuration.

In the current configuration the calculation includes a percentaged overhead of the power supply of 35 %. It is possible to reduce this overhead, for example by using a switched-mode power supply (SMPS). When using this power efficient supply the percentaged overhead can be reduced to about 5 % (device specific). Hence, the resulting power consumption can be simply calculated for specific power supplies.
The figure on the right visualizes the energy estimations with the strategies *Simple* and *Optimal*, with energy-proportional devices and with energy-proportional devices and SMPS with 95 % efficiency.

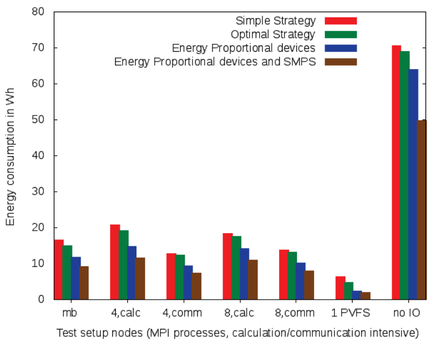


Figure 6: ETS for Energy-Proportional Devices

Summary & Conclusion

The existing work in the fields of Green IT in HPC has already shown that switching the hardware to power saving states is a suitable approach to decrease power consumption with small impact on time-to-solution.In this work an analytical model for evaluating power-saving strategies is presented. Component utilizations are traced on each node and a simulator replays the traces and implements the model.
Developed look-ahead strategies allow us to evaluate the benefit of these strategies to energy-to-solution. Besides, with an energy-proportional hardware configuration an upper bound for all possible energy saving strategies can be computed. Further the simulation of energy-proportional hardware offers the capability to determine the total energy consumed by a specific application without the hardware overhead. The capability to show application chunks where rearrangement of load would increase energy efficiency can be used to localize energy inefficient code. In the experiments the overhead caused by hardware is not surprising, but the capability to monetarily evaluate concrete application traces shows the potential of this approach.