Comput Sci Res Dev (2012) 27:255–263
DOI 10.1007/s00450-011-0196-7

SPECIAL ISSUE PAPER

Determine energy-saving potential in wait-states of large-scale parallel programs

Michael Knobloch · Bernd Mohr · Timo Minartz

Published online: 31 August 2011

Abstract Energy consumption is one of the major topics in high performance computing (HPC) in the last years. However, little effort is put into energy analysis by developers of HPC applications.

We present our approach of combined performance and energy analysis using the performance analysis tool-set Scalasca. Scalascas parallel wait-state analysis is extended by a calculation of the energy-saving potential if a lower power-state can be used.

Keywords Power consumption · Energy efficiency · Energy · Performance · Analysis · Scalasca · MPI

1 Introduction

Energy efficiency has become a major topic in high performance computing (HPC) in the last couple of years, as today's leading systems in the Top500 list[1] consume several MW of power. Thus the operational costs of such a machine usually exceed the acquisition cost of the hardware. But power is also a limiting factor regarding future systems, especially when going towards Exascale computing. To reach the DARPA UHPC goal of 50 GFlop/Watt—which corresponds to 20 MW for an Exaflop system—energy efficiency has to be improved by a factor of 100. This is only possible when addressing this problem from multiple angles—the data center itself, hardware like CPU and memory but also system software, libraries or HPC applications.

On the application side however, energy is not yet a concern as only raw performance counts (and is accounted in terms of CPU time). Thus, comparatively large effort is put into performance analysis and tuning, but nearly none into energy analysis. Our approach is to combine these steps—doing the energy analysis at the same time as the performance analysis. For that, we extended the Scalasca (**Sc**alable **A**nalysis of **l**arge-**sc**ale **A**pplications) tool-set,[2] a well-known performance analysis tool-set that is able to identify wait-states in parallel programs. A lot of energy is wasted in wait-states as MPI usually uses busy waiting. We examine how much energy could be saved in the optimal case, i.e. the MPI library knows how long the waiting time is and can perform idle waiting. Also the energy is calculated in the more realistic case of busy waiting at a lower power-state in order to maintain reactivity. Further we examine which power-state could be used at which wait-state in order to maximize energy savings.

The paper is organized as follows. First we give an overview of other tools and related work. In Sect. 3 we give an overview of the Scalasca tool-set and present how it can be used to detect wait-states in large-scale parallel programs. Then we present our Scalasca extension to determine the energy-saving potential in wait-states, followed by an evaluation of our approach in Sect. 5. Section 6 concludes this paper and outlines future work.

[1] See http://www.top500.org.

[2] See http://www.scalasca.org.

This project is funded by the BMBF (German federal ministery for education and science) under grant 01—H08008E within the call: "HPC-Software für skalierbare Parallelrechner".

M. Knobloch (✉) · B. Mohr
Jülich Supercomputing Centre (JSC) Forschungszentrum Jülich, 52425 Jülich, Germany
e-mail: m.knobloch@fz-juelich.de

T. Minartz
Department of Informatics, University of Hamburg, 22527 Hamburg, Germany

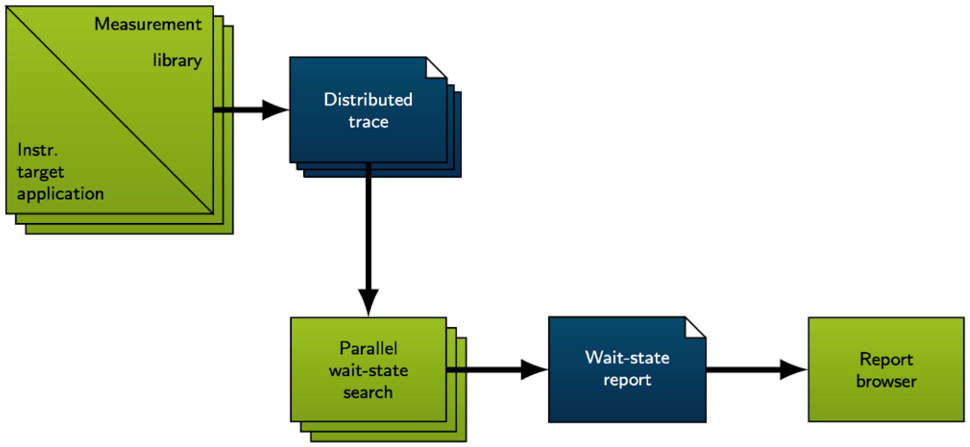


Fig. 1 The scalasca workflow—The *green rectangles* denote (parallel) programs, the *blue rectangles* with the kinked corner symbolize (multiple) files

2 Related work

In the last years several projects developed tools to measure and reduce the energy consumption of parallel applications by invoking hardware power-states, e.g. DVFS (Dynamic Voltage and Frequency Scaling) [4, 13].

DVFS is proved to be beneficial, i.e. energy savings could be achieved, when the CPU is not fully utilized, because the application is memory-bound [8, 9] or not effectively load-balanced [18]. Further projects analyzed the potential of inter-node slack [10] and modeled the energy-time trade-off of a large-scale machine [3].

Another area of research is power consumption profiling [2, 5], simulation [14] and prediction [19, 20].

However, most of these tools concentrate on reducing the energy consumption by running loops or functions of application at a lower power-state, we instead examine how much energy could be saved by an energy-aware MPI library, which is, instead of busy waiting at the highest frequency, either idle waiting or busy waiting at a lower power-state. Another tool which considers the MPI library itself is the GreenBuildingBlocks (GBB) project [17] with the aim to provide a complete stack of energy-aware system-software. Analysis of the energy consumption in communication phases of the MPI program was done by Lim et al. [12] and Minartz et al. [15]. Dong et al. [1] further examined the power consumption of MPI collectives.

3 The Scalasca tool-set

Scalasca [6] is an open source performance analysis tool-set that automatically analyzes application traces to find performance problems, especially wait-states, i.e. one process has to wait for another process, as for example in an MPI Barrier. It is especially tailored for large-scale parallel programs written in C, C++ or Fortran using MPI, OpenMP or a combination of both. Two different analysis modes are offered by Scalasca: runtime summarization at call-path level (a.k.a. profiling) or in-depth analysis via event tracing. The workflow of the latter one is depicted in Fig. 1.

3.1 The Scalasca workflow

The application is instrumented, i.e. calls to the Scalasca measurement system are added, either automatically by the compiler or manually by the user.

The instrumented application is then executed and the Scalasca measurement library writes a process-local trace file. These trace files are automatically analyzed by the parallel trace analyzer called Scout. Scout has to be started with the same number of processes as the original application and performs a “parallel replay” of the application, however, at every send/receive operation and every synchronization point measurement data—like timestamps—is transmitted instead of real application data. Thus, the analyzer scales with the original application.

Wait-state detection

The parallel analyzer automatically searches for patterns in the trace files which indicate performance problems, of particular interest are so-called wait-states. An example, the so-called Late Sender pattern, is shown in Fig. 2. In this example with 4 processes, a message is sent from process A to process B and from process C to process D. The Send and Recv operation on A and B have been posted at the same time, so no waiting time occurs for these processes. The Send operation on process C on the other hand has been posted much later than the corresponding Recv operation on process D, thus this process has to wait till the sender is ready.

Scalasca is able to detect wait-states for most point-to-point and collective MPI operations, common patterns are:

- Late Sender, Late Receiver
- Wait at Barrier, Wait at $N \times N$
- Late Broadcast, Early Reduce, Early Scan.

Scalasca is further able to detect MPI one-sided (RMA) and OpenMP performance problems.

At each process the waiting times for each pattern on every call-path are aggregated, i.e. Scalasca has no knowledge about single events, and after finishing the analysis the results of all processes are merged into a final analysis report.

3.2 Result visualization with Cube3

The trace analysis is visualized with the Scalasca result browser Cube3, which is shown in Fig. 3.

Cube3 consist of a three-pane layout, the left pane shows the performance problem, the middle pane the distribution of this metric on the call-path and the right pane the distribution across the processes on the machine, where MPI topologies or special machine topologies like the Blue Gene torus are supported. For each pane the viewing mode can be adjusted, so it is possible to show absolute values, relative values and even relative values compared to other experiments to see the effect of performance tuning. A color bar indicates the severity of the problem.

4 Methodology

Current MPI implementations perform so-called busy waiting when a process waits for action of another process, e.g. the receiver of a message is waiting for the corresponding sender. Busy waiting means polling at the highest available frequency for a signal in order to be able to react instantly once the signal received. However, this consumes a lot of energy.

We performed some measurements [16] to show the effect of MPI on the power consumption of the application and visualized them with Vampir [11]. Figure 4 shows a well balanced test-case with two MPI collectives and two calculation phases. The power consumption in the MPI operations (1, $\approx$ 286 W) is considerably higher than in the calculation phase (2, $\approx$ 255 W) and when idling in the main routine (3, $\approx$ 160 W).

A constructed example of the Scalasca Wait at Barrier pattern is shown in Fig. 5. After an (unbalanced) computation phase the processes enter the MPI Barrier according to a sleep statement of different length for each process. We see that the more processes enter the Barrier, the higher the power consumption.

4.0.1 *Calculation of energy-saving potential*

To calculate the energy-saving potential in such situations, we assume that there exists a set of power-states *PS* for each core, this can be the processors P-States or other power-states, e.g. processor at a lower frequency and network or

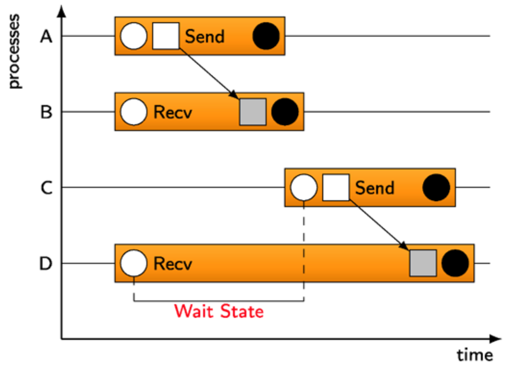
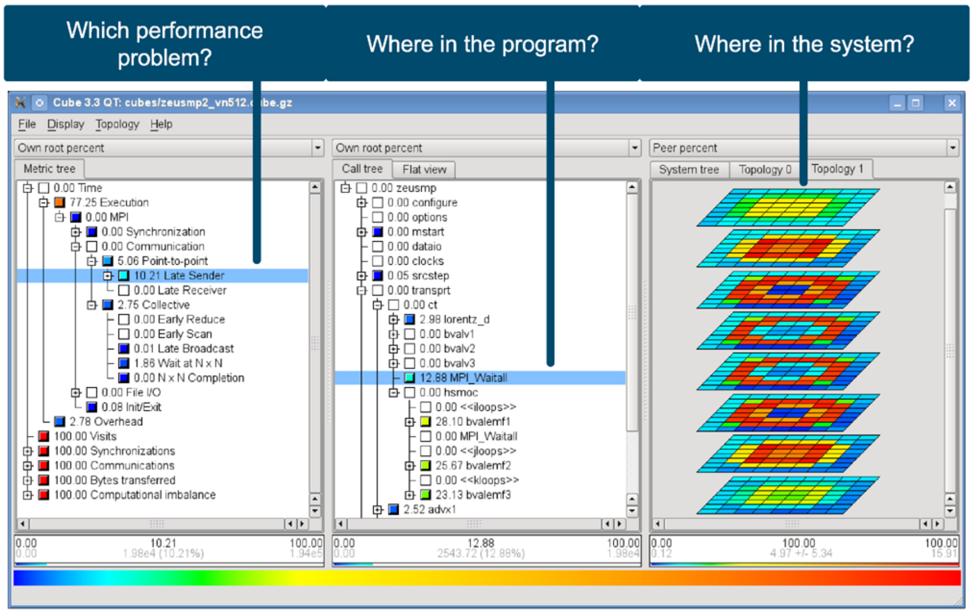


Fig. 2 Late Sender pattern

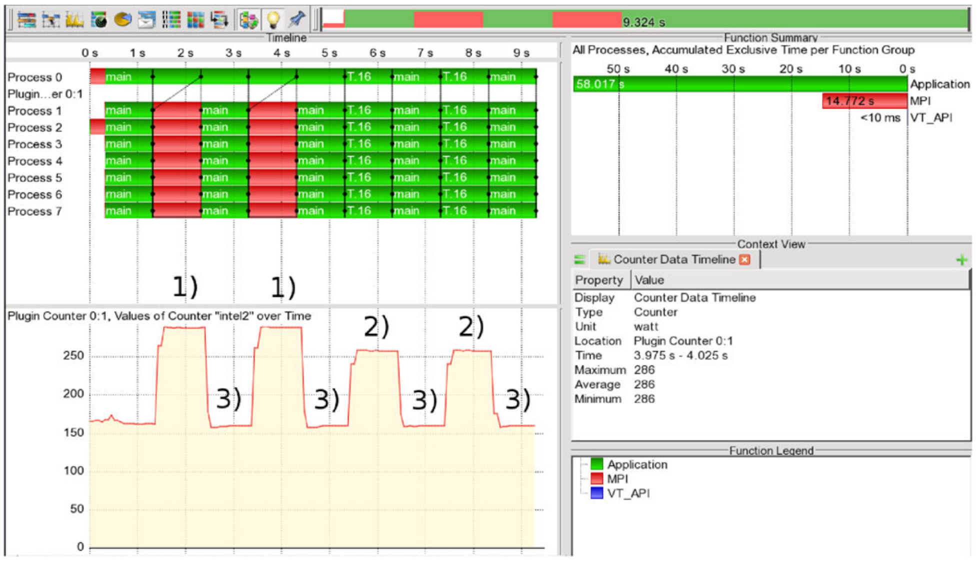
Fig. 3 The Cube3 report viewer



```
0.00 Time
  77.25 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        5.06 Point-to-point
          10.21 Late Sender
          0.00 Late Receiver
        2.75 Collective
          0.00 Early Reduce
          0.00 Early Scan
          0.01 Late Broadcast
          1.86 Wait at N x N
          0.00 N x N Completion
      0.00 File I/O
      0.08 Init/Exit
  2.78 Overhead
100.00 Visits
100.00 Synchronizations
100.00 Communications
100.00 Bytes transferred
100.00 Computational imbalance

0.00          10.21                100.00
0.00     1.98e4 (10.21%)           1.94e5
```

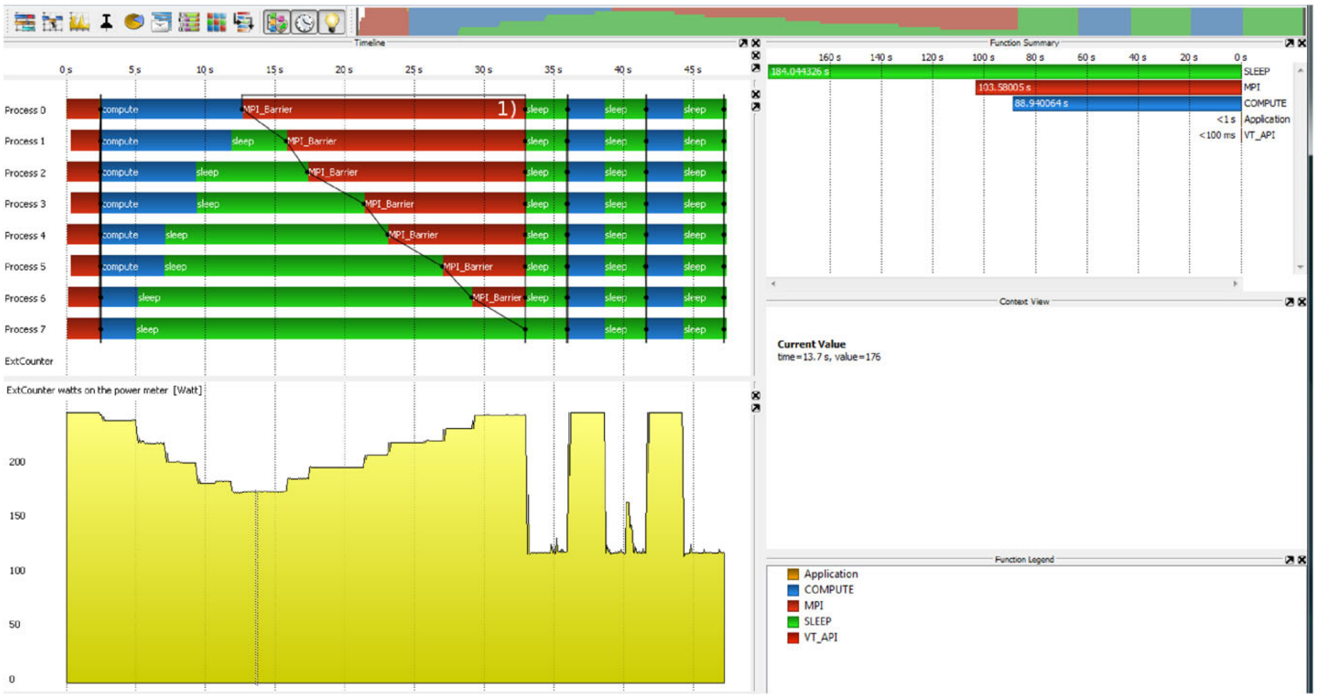
```
0.00 zeusmp
  0.00 configure
  0.00 options
  0.00 mstart
  0.00 dataio
  0.00 clocks
  0.05 srcstep
  0.00 transprt
    0.00 ct
      2.98 lorentz_d
      0.00 bvalv1
      0.00 bvalv2
      0.00 bvalv3
      12.88 MPI_Waitall
      0.00 hsmoc
        0.00 <<iloops>>
        28.10 bvalemf1
        0.00 MPI_Waitall
        0.00 <<jloops>>
        25.67 bvalemf2
        0.00 <<kloops>>
        23.13 bvalemf3
    2.52 advx1

0.00          12.88                100.00
0.00     2543.72 (12.88%)          1.98e4
```

```
0.00          100.00               100.00
0.12       4.97 +/- 5.34           15.91
```



Property	Value
Display	Counter Data Timeline
Type	Counter
Unit	watt
Location	Plugin Counter 0:1
Time	3.975 s - 4.025 s
Maximum	286
Average	286
Minimum	286

Fig. 4 Vampir screenshot—power consumption in MPI phases is considerably higher than in any other phase



Current Value
time=13.7 s, value=176

Fig. 5 Vampir screenshot—the more processes enter the MPI barrier (1), the higher the power consumption. This corresponds to the Scalasca Wait at Barrier pattern

disk turned off. Than we define for every power-state $p \in PS$ the active power consumption A_p, i.e. the power consumption under load and the idle power consumption I_p. For the transition between two power-states $p, q \in PS$, $t_{T_{p,q}}$ denotes the time and $E_{T_{p,q}}$ the energy needed to perform the transition in both directions.

Than the energy-saving potential (ESP) for every wait-state with waiting time t_w can be calculated as:

$$ESP = \max_{p \in PS}((t_w * A_{p_1}) - (t_w - t_{T_{p,p_1}}) * I_p + E_{T_{p,p_1}}) \quad (1)$$

This potential poses an upper limit, however, it could only be exploited by an MPI library with oracle capabilities, i.e. it must be known in advance how long the wait-state will last in order to maintain reactivity.

A more realistic case would be that MPI is busy waiting at a lower power-state, the energy-saving potential for this (ESP_BW) can be calculated as:

$$\text{ESP\_BW} = \max_{p \in PS}\left((t_w * A_{p_1}) - (t_w - t_{T_{p,p_1}}) * A_p + E_{T_{p,p_1}}\right) \quad (2)$$

We further investigate for each wait-state with an energy-saving potential greater zero which power-state $p \in PS$ leads to the greatest energy savings. In equation (1) this can be p_1, which is always the case when the waiting time is too short for power-state transitions to be effective. In the busy waiting case (2) on the other hand this is obviously not possible, so for very short wait-states no energy savings are possible.

Both calculations are done for every wait-state on every process and the potentials, as well as the possible power-states, are aggregated separately for idle and busy waiting.

5 Evaluation

To evaluate our approach we analyzed the plasma physics code PEPC (Pretty Efficient Parallel Coulomb-solver) [7] on two clusters, a cluster with power measurement capabilities at the Research Group Scientific Computing at the University of Hamburg and Juropa, a Intel Nehalem based Supercomputer at Jülich Supercomputing Center (JSC)[3] at Forschungszentrum Jülich[4].

5.1 Test systems

5.1.1 eeCluster

Our test cluster at University of Hamburg consists of five dual socket Intel Nehalem (Xeon X5560, 4 cores) and five dual socket AMD Magny-Cours (Opteron 6168, 12 cores) compute nodes which are connected to ZES LMG450 high precision power meters with an accuracy of 0.1%.

The measured power consumption of one node under load and idle is shown in Fig. 6 for the Opteron nodes and Fig. 7 for the Xeon nodes, respectively [15]. C-states are disabled on the Xeon nodes, as they are disabled on most production HPC systems.

Tables 1 and 2 present the power-states (per core) we derived from the values of Figs. 6 and 7. Unfortunately, the values for the transition time and energy are just an approximate, as we were unable to obtain the real values in the data-sheets.

The AMD Opteron provides 5 P-States while the Intel Xeon can operate at 10 different frequencies (11 if the Turbo Mode is considered too), of which we choose 5 for our experiments.

The Opteron core consumes significantly less power than a Xeon core and has a much better active/idle power ratio, but the Xeon core is much more powerful. With the power-states described in Tables 1 and 2 a maximal saving potential of 66.56 and 48.95% for idle waiting as well as 30.1 and 31.13% for busy waiting can be reached.

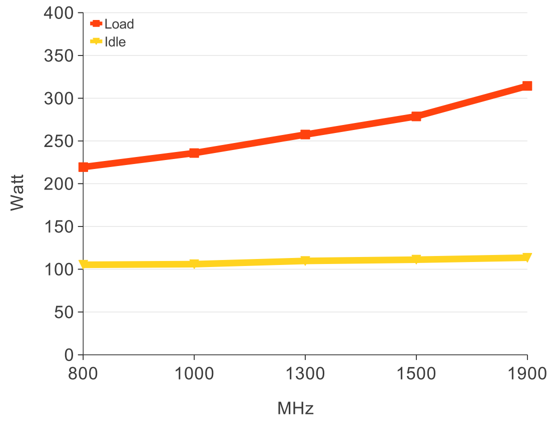


Fig. 6 Power consumption of Opteron nodes depending on P-State

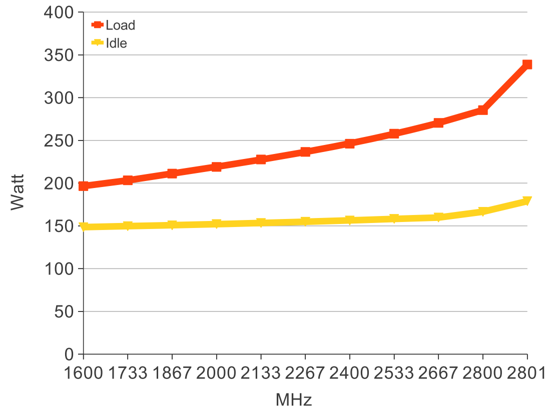


Fig. 7 Power consumption of Xeon nodes depending on P-State

Table 1 Power-states per core on AMD Opteron

P-State	A_p (W)	I_p (W)	$t_{T_{p,p_1}}$ (s)	$E_{T_{p,p_1}}$ (J)
1–1900 MHz	13.1	4.73	0	0
2–1500 MHz	11.61	4.63	0.00001	0.05
3–1300 MHz	10.73	4.57	0.00002	0.1
4–1000 MHz	9.82	4.41	0.00003	0.2
5–800 MHz	9.14	4.38	0.00004	0.3

Table 2 Power-states per core on Intel Xeon

P-State	A_p (W)	I_p (W)	$t_{T_{p,p_1}}$ (s)	$E_{T_{p,p_1}}$ (J)
1–2800 MHz	35.68	20.81	0	0
2–2533 MHz	32.24	19.77	0.00001	0.1
3–2267 MHz	29.56	19.36	0.00002	0.2
4–1867 MHz	26.4	18.83	0.00003	0.4
5–1600 MHz	24.57	18.57	0.00004	0.8

[3]See http://www.fz-juelich.de/ias/jsc/EN/Home/home_node.html.

[4]See http://www.fz-juelich.de.

Table 3 Power-states per core on Juropa

P-State	A_p (W)	I_p (W)	$t_{T_{p,p_1}}$ (s)	$E_{T_{p,p_1}}$ (J)
1	58.8	34.3	0	0
2	53.13	32.58	0.00001	0.1
3	48.72	31.91	0.00002	0.2
4	43.51	31.03	0.00003	0.4
5	40.48	30.6	0.00004	0.8

5.1.2 *Juropa*

Juropa is a 26304 core Intel Nehalem (Xeon X5570, 4 cores) based cluster at JSC ranked #23 at the November 2010 Top500 list[5] with a Linpack performance of 274800 GFlop and a power consumption of 1549 kW, which corresponds to 58.9 W/core running the Linpack benchmark. We take this value as a baseline for our measurements and—as no direct power measurements are possible on Juropa—estimate the other values corresponding to the values measured on the Xeon X5560. This yields to the power-states described in Table 3.

Since the power-states are derived from the power-states on the Xeon nodes of the cluster at DKRZ the relative maximum energy-saving potential is similar with 47.96 and 31.15%, respectively.

5.2 PEPC

PEPC (Pretty Efficient Parallel Coulomb-solver),[6] is a parallel tree-code for rapid computation of long-range Coulomb forces in N-body particle systems based on the original Barnes-Hut algorithm. The code uses successively larger multipole-groupings of distant particles to reduce the computational effort in the force calculation from the generally unaffordable $O(N^2)$ operations needed for brute-force summation, to a more amenable $O(N\log(N))$ complexity.

The parallel version is a pure MPI implementation of the Warren-Salmon ‘Hashed Oct Tree’ scheme, including several different variations of the tree traversal routine—the most challenging component in terms of scalability.

5.3 Results

Experiments have been performed on 4 Xeon and Opteron nodes, i.e. 32 and 96 processes, respectively and for scalability tests on 128 nodes of Juropa, which corresponds to 1024 processes. Table 4 shows for the three most severe wait-state patterns, in this case the Late Sender, Wait at Barrier and Wait at $N \times N$, the waiting time, the energy spent waiting as well as the energy-saving potential for both idle and busy waiting. On the Xeon and Opteron nodes we simulated 25600 and on Juropa 256000 particles with 50 timesteps in each case.

As the power-states from Table 1 indicate the Opteron has the highest saving potential for idle waiting, but slightly tails at the saving potential for busy waiting.

The dominant patterns in each execution derive from MPI collectives, in particular the Wait at $N \times N$ pattern whose distribution on the call-tree is displayed in Fig. 8 and the Wait at Barrier, shown in Fig. 10.

We observe that the optimal power-states differ significantly from idle to busy waiting. While for idle waiting the higher power-states are important, especially in the smaller experiments (see Figs. 9 and 10) dominate the lower power-states for busy waiting.

An interesting fact is that on both Xeon-based systems we observe a nearly uniform distribution of waiting time and thus energy-saving potential for the collective operations across the nodes. Contrary, on the Opteron nodes we see a huge variation in the distribution, e.g. waiting time and energy-saving potential for busy waiting at an MPI_ Barrier goes from 1290 s and 4840 J on AMD4 to 3529 s and 13600 J on AMD5 (see Fig. 10). We have to further investigate these differences and verify that behavior on another Opteron-based system.

6 Conclusion & future work

In this paper we presented an extension to the Scalasca toolset to determine the energy-saving potential in wait-states of parallel programs. We showed that MPI consumes lots of energy while busy waiting and a considerable amount of this energy could possibly be saved with an energy-aware MPI library, even if busy waiting in a lower power-state in order to maintain reactivity.

A lot of future work is still to be done. A next step is to build such an energy-aware MPI library which is able to use information of wait-states in order to reduce energy consumption.

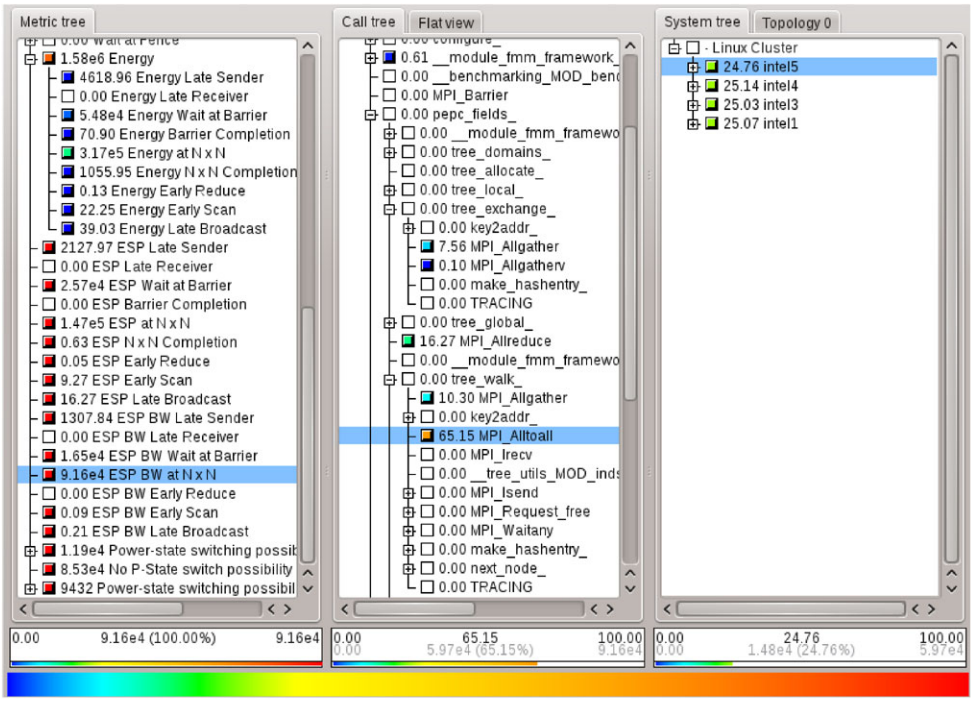
On the Scalasca side the next step is to analyze the energy-saving potential by reducing the voltage and frequency of processes not lying on a critical path, i.e. those with wait-states before global synchronization points, and compare those to the saving potential presented in this paper.

Further an analysis of hardware performance counters to automatically identify phases of low computation, where energy could be saved would be desirable.

Acknowledgements We would like to thank Daniel Molka from the Center for Information Services and High Performance Computing (ZIH) at TU Dresden, for the Vampir measurements.

[5]See http://top500.org/lists/2010/11.

[6]See https://trac.version.fz-juelich.de/pepc.

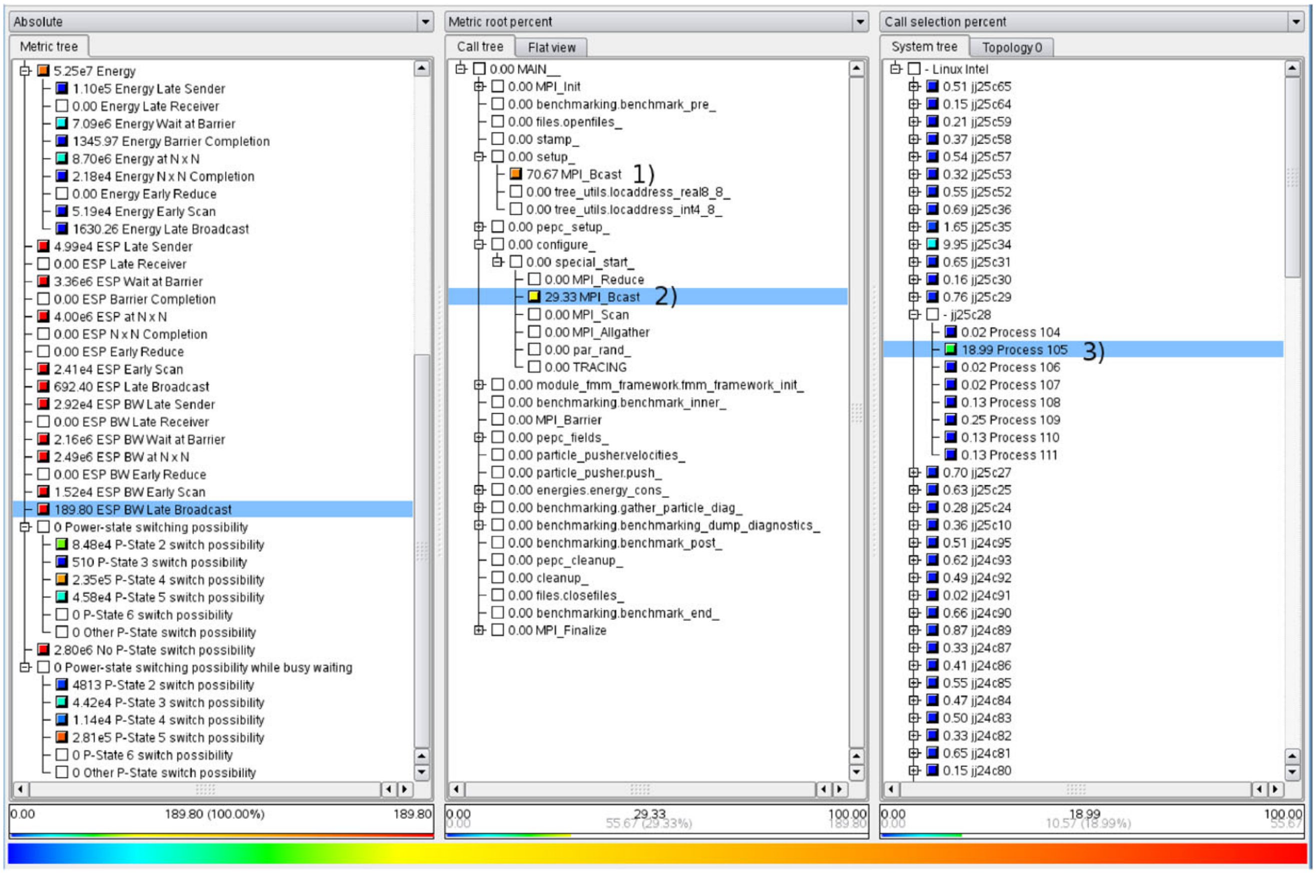


```
Metric tree
0.00 Wait at Fence
1.58e6 Energy
  4618.96 Energy Late Sender
  0.00 Energy Late Receiver
  5.48e4 Energy Wait at Barrier
  70.90 Energy Barrier Completion
  3.17e5 Energy at N x N
  1055.95 Energy N x N Completion
  0.13 Energy Early Reduce
  22.25 Energy Early Scan
  39.03 Energy Late Broadcast
2127.97 ESP Late Sender
0.00 ESP Late Receiver
2.57e4 ESP Wait at Barrier
0.00 ESP Barrier Completion
1.47e5 ESP at N x N
0.63 ESP N x N Completion
0.05 ESP Early Reduce
9.27 ESP Early Scan
16.27 ESP Late Broadcast
1307.84 ESP BW Late Sender
0.00 ESP BW Late Receiver
1.65e4 ESP BW Wait at Barrier
9.16e4 ESP BW at N x N
0.00 ESP BW Early Reduce
0.09 ESP BW Early Scan
0.21 ESP BW Late Broadcast
1.19e4 Power-state switching possib
8.53e4 No P-State switch possibility
9432 Power-state switching possibil
0.00        9.16e4 (100.00%)        9.16e4
```

```
Call tree    Flat view
0.00 configure_
0.61 __module_fmm_framework_
0.00 __benchmarking_MOD_benc
0.00 MPI_Barrier
0.00 pepc_fields_
  0.00 __module_fmm_framewo
  0.00 tree_domains_
  0.00 tree_allocate_
  0.00 tree_local_
  0.00 tree_exchange_
    0.00 key2addr_
    7.56 MPI_Allgather
    0.10 MPI_Allgatherv
    0.00 make_hashentry_
    0.00 TRACING
  0.00 tree_global_
  16.27 MPI_Allreduce
  0.00 __module_fmm_framewo
  0.00 tree_walk_
    10.30 MPI_Allgather
    0.00 key2addr_
    65.15 MPI_Alltoall
    0.00 MPI_Irecv
    0.00 __tree_utils_MOD_inds
    0.00 MPI_Isend
    0.00 MPI_Request_free
    0.00 MPI_Waitany
    0.00 make_hashentry_
    0.00 next_node_
    0.00 TRACING
0.00        65.15         100.00
0.00   5.97e4 (65.15%)    9.16e4
```

```
System tree    Topology 0
- Linux Cluster
  24.76 intel5
  25.14 intel4
  25.03 intel3
  25.07 intel1
0.00        24.76         100.00
0.00   1.48e4 (24.76%)    5.97e4
```

Fig. 8 Cube3 Screenshot of PEPC run on Xeon nodes displaying the distribution of the ESP BW Wait at $N \times N$. The MPI_Alltoall in the tree_walk routine has with 65% the largest proportion which is uniformly distributed among the nodes



```
Absolute
Metric tree
5.25e7 Energy
  1.10e5 Energy Late Sender
  0.00 Energy Late Receiver
  7.09e6 Energy Wait at Barrier
  1345.97 Energy Barrier Completion
  8.70e6 Energy at N x N
  2.18e4 Energy N x N Completion
  0.00 Energy Early Reduce
  5.19e4 Energy Early Scan
  1630.26 Energy Late Broadcast
4.99e4 ESP Late Sender
0.00 ESP Late Receiver
3.36e6 ESP Wait at Barrier
0.00 ESP Barrier Completion
4.00e6 ESP at N x N
0.00 ESP N x N Completion
0.00 ESP Early Reduce
2.41e4 ESP Early Scan
692.40 ESP Late Broadcast
2.92e4 ESP BW Late Sender
0.00 ESP BW Late Receiver
2.16e6 ESP BW Wait at Barrier
2.49e6 ESP BW at N x N
0.00 ESP BW Early Reduce
1.52e4 ESP BW Early Scan
189.80 ESP BW Late Broadcast
0 Power-state switching possibility
  8.48e4 P-State 2 switch possibility
  510 P-State 3 switch possibility
  2.35e5 P-State 4 switch possibility
  4.58e4 P-State 5 switch possibility
  0 P-State 6 switch possibility
  0 Other P-State switch possibility
2.80e6 No P-State switch possibility
0 Power-state switching possibility while busy waiting
  4813 P-State 2 switch possibility
  4.42e4 P-State 3 switch possibility
  1.14e4 P-State 4 switch possibility
  2.81e5 P-State 5 switch possibility
  0 P-State 6 switch possibility
  0 Other P-State switch possibility
0.00        189.80 (100.00%)        189.80
```

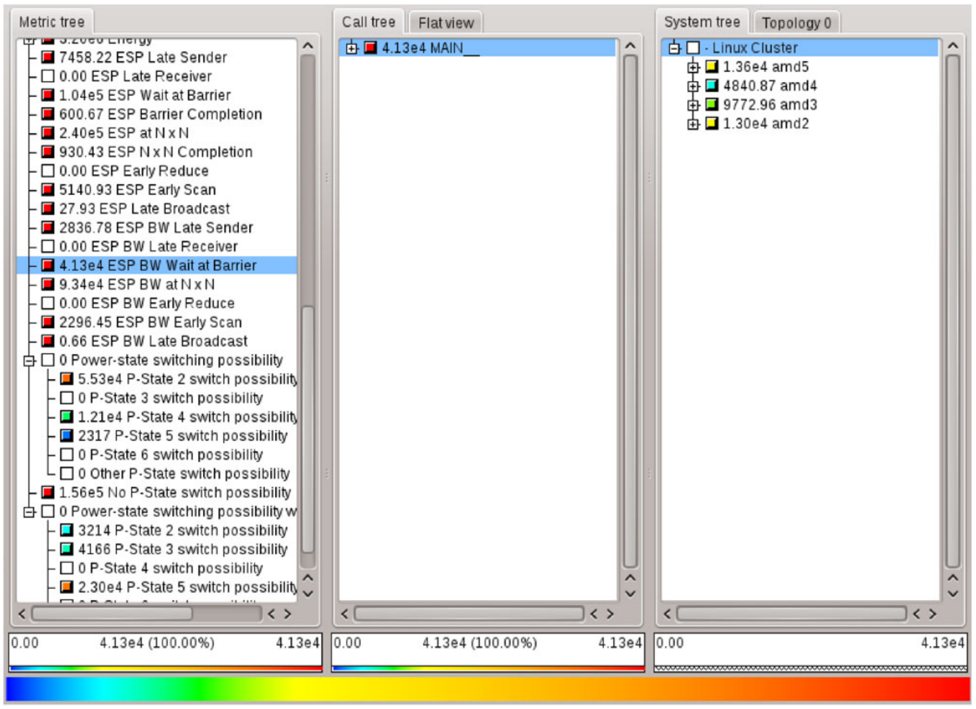
```
Metric root percent
Call tree    Flat view
0.00 MAIN__
  0.00 MPI_Init
  0.00 benchmarking.benchmark_pre_
  0.00 files.openfiles_
  0.00 stamp_
  0.00 setup_
    70.67 MPI_Bcast  1)
    0.00 tree_utils.locaddress_real8_8_
    0.00 tree_utils.locaddress_int4_8_
  0.00 pepc_setup_
  0.00 configure_
    0.00 special_start_
      0.00 MPI_Reduce
      29.33 MPI_Bcast  2)
      0.00 MPI_Scan
      0.00 MPI_Allgather
      0.00 par_rand_
      0.00 TRACING
  0.00 module_fmm_framework.fmm_framework_init_
  0.00 benchmarking.benchmark_inner_
  0.00 MPI_Barrier
  0.00 pepc_fields_
  0.00 particle_pusher.velocities_
  0.00 particle_pusher.push_
  0.00 energies.energy_cons_
  0.00 benchmarking.gather_particle_diag_
  0.00 benchmarking.benchmarking_dump_diagnostics_
  0.00 benchmarking.benchmark_post_
  0.00 pepc_cleanup_
  0.00 cleanup_
  0.00 files.closefiles_
  0.00 benchmarking.benchmark_end_
  0.00 MPI_Finalize
0.00        29.33          100.00
0.00    55.67 (29.33%)     189.80
```

```
Call selection percent
System tree    Topology 0
- Linux Intel
  0.51 jj25c65
  0.15 jj25c64
  0.21 jj25c59
  0.37 jj25c58
  0.54 jj25c57
  0.32 jj25c53
  0.55 jj25c52
  0.69 jj25c36
  1.65 jj25c35
  9.95 jj25c34
  0.65 jj25c31
  0.16 jj25c30
  0.76 jj25c29
  - jj25c28
    0.02 Process 104
    18.99 Process 105  3)
    0.02 Process 106
    0.02 Process 107
    0.13 Process 108
    0.25 Process 109
    0.13 Process 110
    0.13 Process 111
  0.70 jj25c27
  0.63 jj25c25
  0.28 jj25c24
  0.36 jj25c10
  0.51 jj24c95
  0.62 jj24c93
  0.49 jj24c92
  0.02 jj24c91
  0.66 jj24c90
  0.87 jj24c89
  0.33 jj24c87
  0.41 jj24c86
  0.55 jj24c85
  0.47 jj24c84
  0.50 jj24c83
  0.33 jj24c82
  0.65 jj24c81
  0.15 jj24c80
0.00        18.99          100.00
0.00    10.57 (18.99%)     55.67
```

Fig. 9 Cube3 Screenshot of PEPC run on Juropa displaying the distribution of the Late Broadcast ESP while busy waiting. We see that 70.67% of the energy-saving potential is in the MPI_Bcast called in setup (1) and 29.33% in the MPI_Bcast in special_start (2). 18.99% of that could be saved on Process 105 (3)

Table 4 The three most severe Patterns on all machines. The AMD Opteron outperforms the Xeons in the ESP, but the Xeons slightly lead in the ESP_BW

Machine	Pattern	Time (s)	Energy (J)	ESP (J)	ESP (%)	ESP_BW (J)	ESP_BW (%)
Juropa	Late Sender	1863.74	1.10e5	4.99e4	45.36	2.92e4	25.55
	Wait at Barrier	1.21e5	7.09e6	3.36e6	47.39	2.16e6	30.47
	Wait at $N \times N$	1.48e5	8.7e6	4.00e6	45.98	2.49e6	28.62
Xeon	Late Sender	129.46	4618.96	2127.97	46.07	1307.84	28.31
	Wait at Barrier	1835.81	5.48e4	2.57e4	46.9	1.65e4	30.11
	Wait at $N \times N$	8896.07	3.17e5	1.47e5	46.37	9.16e4	28.9
Opteron	Late Sender	878.57	1.15e4	7458.22	64.85	2836.78	24.66
	Wait at Barrier	1864.9	1.41e5	1.04e5	73.76	4.13e4	29.29
	Wait at $N \times N$	2.62e4	3.43e5	2.40e5	69.97	9.34e4	27.2



🟥 7458.22 ESP Late Sender
□ 0.00 ESP Late Receiver
🟥 1.04e5 ESP Wait at Barrier
🟥 600.67 ESP Barrier Completion
🟥 2.40e5 ESP at N x N
🟥 930.43 ESP N x N Completion
□ 0.00 ESP Early Reduce
🟥 5140.93 ESP Early Scan
🟥 27.93 ESP Late Broadcast
🟥 2836.78 ESP BW Late Sender
□ 0.00 ESP BW Late Receiver
🟥 4.13e4 ESP BW Wait at Barrier
🟥 9.34e4 ESP BW at N x N
□ 0.00 ESP BW Early Reduce
🟥 2296.45 ESP BW Early Scan
🟥 0.66 ESP BW Late Broadcast
□ 0 Power-state switching possibility
🟧 5.53e4 P-State 2 switch possibility
□ 0 P-State 3 switch possibility
🟩 1.21e4 P-State 4 switch possibility
🟦 2317 P-State 5 switch possibility
□ 0 P-State 6 switch possibility
□ 0 Other P-State switch possibility
🟥 1.56e5 No P-State switch possibility
□ 0 Power-state switching possibility w
🟦 3214 P-State 2 switch possibility
🟦 4166 P-State 3 switch possibility
□ 0 P-State 4 switch possibility
🟧 2.30e4 P-State 5 switch possibility

🟥 4.13e4 MAIN__

□ - Linux Cluster
🟨 1.36e4 amd5
🟦 4840.87 amd4
🟩 9772.96 amd3
🟨 1.30e4 amd2

0.00 4.13e4 (100.00%) 4.13e4
0.00 4.13e4 (100.00%) 4.13e4
0.00 4.13e4

Fig. 10 Cube3 Screenshot of PEPC run on Opteron nodes displaying the distribution of the Wait at Barrier ESP BW. For idle waiting higher power-states are more important while lower power-states are dominant for busy waiting

References

1. Dong Y, Chen J, Yang X, Yang C, Peng L (2008) Low power optimization for MPI collective operations. In: Young computer scientists, International conference for, pp 1047–1052
2. Feng X, Ge R, Cameron KW (2005) Power and energy profiling of scientific applications on distributed systems. In: Proceedings of the 19th IEEE international parallel and distributed processing symposium, IPDPS'05. IEEE Computer Society, Washington, vol 1, p 34. doi:10.1109/IPDPS.2005.346
3. Freeh VW, Pan F, Kappiah N, Lowenthal DK, Springer R (2005) Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In: Proceedings of the 19th IEEE international parallel and distributed processing symposium, IPDPS'05, vol 1. IEEE Computer Society, Washington. doi:10.1109/IPDPS.2005.214
4. Ge R, Feng X, Cameron KW (2005) Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In: Proceedings of the 2005 ACM/IEEE conference on supercomputing, SC'05. IEEE Computer Society, Washington, p 34. doi:10.1109/SC.2005.57
5. Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW (2009) PowerPack: energy profiling and analysis of high-performance systems and applications. IEEE Trans Parallel Distrib Syst 99:658–671
6. Geimer M, Wolf F, Wylie BJN, Abraham E, Becker D, Mohr B (2010) The Scalasca performance toolset architecture. Concurr Comput: Pract Exp 22(6):277–288
7. Gibbon P (2003) PEPC: Pretty efficient parallel Coulomb-solver. Forschungszentrum Jülich

8. Hsu CH, Feng WC (2005) A power-aware run-time system for high-performance computing. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, SC'05. IEEE Computer Society, Washington, p 1. doi:10.1109/SC.2005.3
9. Huang S, Feng W (2009) Energy-efficient cluster computing via accurate workload characterization. In: Proceedings of the 2009 9th IEEE/ACM international symposium on cluster computing and the grid, CCGRID '09. IEEE Computer Society, Washington, pp 68–75. doi:10.1109/CCGRID.2009.88
10. Kappiah N, Freeh VW, Lowenthal DK (2005) Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, SC'05. IEEE Computer Society, Washington, p 33. doi:10.1109/SC.2005.39
11. Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS, Nagel WE (2008) The Vampir performance analysis tool-set. In: Tools for high performance computing, Proceedings of the 2nd international workshop on parallel tools. Springer, Berlin, pp 139–155
12. Lim MY, Freeh VW, Lowenthal DK (2006) Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, SC'06. ACM, New York. doi:10.1145/1188455.1188567
13. Lu YH, Benini L, De Micheli G (2000) Operating-system directed power reduction. In: Proceedings of the 2000 international symposium on low power electronics and design, ISLPED '00. ACM, New York, pp 37–42. doi:10.1145/344166.344189
14. Minartz T, Kunkel J, Ludwig T (2010) Simulation of power consumption of energy efficient cluster hardware. Comput Sci Res Dev 25:165–175
15. Minartz T, Knobloch M, Ludwig T, Mohr B (2011) Managing hardware power saving modes for high performance computing In: Proceedings of the 2nd international green computing Conference (to appear)
16. Minartz T, Molka D, Knobloch M, Krempel S, Ludwig T, Nagel W, Mohr B, Falter H (2011) eeClust—energy-efficient cluster computing. In proceedings of the CiHPC: competence in high performance computing, HPC status konferenz der Gauß-Allianz e.V. (to appear)
17. Nikolopoulos DS (2009) Green building blocks—software stacks for energy-efficient clusters and data centers. ERCIM News **79**
18. Pinheiro E, Bianchini R, Carrera EV, Heath T (2001) Load balancing and unbalancing for power and performance in cluster-based systems. In: Workshop on compilers and operating systems for low power.
19. Snowdon DC, Petters SM, Heiser G (2007) Accurate on-line prediction of processor and memory energy usage under voltage scaling. In: Proceedings of the 7th ACM & IEEE international conference on embedded software, EMSOFT'07, pp 84–93
20. Zamani R, Afsahi A (2010) Adaptive estimation and prediction of power and performance in high performance computing. Comput Sci Res Dev 25:177–186

Michael Knobloch is a research scientist at Forschungszentrum Jülich, Germany since receiving his diploma in Mathematics from Technische Universität Dresden in 2008. He is currently working on the eeClust project with the aim to improve the energy-efficiency of HPC clusters. His main research interests are tools to analyze the performance and energy consumption characteristics of large-scale parallel programs.

Bernd Mohr started to design and develop tools for performance analysis of parallel programs already with his diploma thesis (1987) at the University of Erlangen in Germany, and continued this in his Ph.D. work (1987 to 1992). During a three year Postdoc position at the University of Oregon, he designed and implemented the original TAU performance analysis framework. Since 1996 he is a senior scientist at Forschungszentrum Jülich, Germany's largest multi-disciplinary research center and home of the most parallel machine of the world, a 294,912 core BlueGene/P. He is the author of several dozen conference and journal articles about performance analysis and tuning of parallel programs. Since 2000, he is the team leader for the group "Programming Environments and Performance Optimization". Besides being responsible for user support and training in regard to parallel programming at the Jülich Supercomputing Centre (JSC), he is leading the Scalasca performance tool efforts in collaboration with Prof. Dr. Felix Wolf.

Timo Minartz received his M.Sc. degree in computer science at the University of Heidelberg in 2009. Now he is a research scientist at the University of Hamburg and contributes to the eeClust project. His major research interests are energy-efficiency aspects in high performance computing.