Ruprecht-Karls-Universität Heidelberg
Institute of Computer Science
Research Group Parallel and Distributed Systems

Masterthesis

Model and simulation of power consumption and power saving potential of energy efficient cluster hardware

Name	Timo Minartz
Student number	2676380
Supervised by	Julian M. Kunkel, Prof. Thomas Ludwig
Date of submission	Heidelberg, August 27, 2009

Timo Minartz
Matrikelnummer: 2676380

Diese Masterarbeit ist von mir selbstständig angefertig und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden. Weiter wurden die Grundsätze und Empfehlungen "Verantwortung in der Wissenschaft" der Universität Heidelberg beachtet.

Unterschrift

Heidelberg, den 27. August 2009

Danksagung

Herrn Prof. Ludwig danke ich für die Übernahme der Betreuung der Arbeit und seine Hilfestellungen und nützlichen Anregungen.

Entscheidend zum Gelingen dieser Arbeit beigetragen hat Julian Kunkel. Er hat mit mir fast täglich zusammengearbeitet und mich mit seinen Ideen, konstruktiven Beiträgen und seinem Engagement umfangreich unterstützt.

Stephan Krempel möchte ich für seine Hilfe bei der Nutzung der **ResourceUtilizationTracingLibrary** und der **PowerTracingLibrary** danken.

Jörg, Michael und Philipp möchte ich für das Korrekturlesen und den damit verbundenen Änderungsvorschlägen danken.

Der größte Dank gilt meinen Eltern und Großeltern, die mir das Studium in Heidelberg und Aachen überhaupt erst ermöglicht haben. Sie haben mir neben ihrer großzügigen finanziellen Unterstützung auch immer moralisch zu Seite gestanden.

Besonderer Dank gilt Isabell, für all ihre Geduld mit mir.

Abstract

In the last years the power consumption of high performance computing clusters has become a growing problem because number and size of cluster installations raised and still is raising. The high power consumption of the clusters results from the main goal of these clusters: High performance. With a low utilization the cluster hardware consumes nearly as much energy as when it is fully utilized. In these low utilization phases the cluster hardware can theoretically turned off or switched to an lower power consuming mode.

In this thesis a model is designed to estimate the power consumption of the hardware with and without energy saving mechanism. With the resulting software it is possible to estimate the cluster power consumption for different configurations of a parallel program. Further energy aware hardware can be simulated to determine an upper bound for energy savings without performance leakage.

The results show that is a great energy saving potential for energy aware hardware even in high performance computing. This potential should motivate research in mechanism to control the energy aware hardware in high performance clusters.

Contents

1. Introduction

In the last years the power consumption of high performance computing clusters has become a growing problem because number and size of cluster installations raised and still is raising. In times of "Green IT" the installations and especially their power consumption have to be legitimated.

For example the fastest public and official listed cluster has a peak power consumption of about 2483.47 kW (see table 1.2, discussed in section 1.2). If this cluster reaches a power saving of 1 %, this results in power savings of about 200 MWh per year (see equation 1.1). Assuming 0.05 € per kWh [1] these are savings of about 10.000 €/year. This saving is equivalent to about 108 t CO_2 produced when generating the energy in a power plant. The same amount of CO_2 is produced when driving about 630.000 km by car [2]. For the second listed cluster these values have to be triplicated based on the higher peak power consumption of this specific cluster.

$$\begin{aligned} &2483.47\,\mathrm{kW} * \frac{1}{100} * 24\,\mathrm{h} * 365 \\ &\approx 596.03\,\mathrm{kWh\ per\ day} * 365 \\ &\approx 217.55\,\mathrm{MWh\ per\ year} \end{aligned} \tag{1.1}$$

The high power consumption of the clusters results from the main goal of these clusters: High performance. This goal has impact on the hardware, the worst case cooling scenarios and the resulting energy consumption. If these clusters are fully utilized, this is maybe the most energy efficient way to get this calculation power. But with a low utilization the hardware consumes nearly as much energy as when it is fully utilized. Based on this relationship between peak and idle power consumption of components the loss of energy can take place. Of course it is tried to eliminate these low utilization phases, but they still arise due to hardware bottlenecks, load balancing problems and problem specific behavior.

In these low utilization phases the cluster hardware can theoretically turned off or switched to an lower power consuming mode if the component supports this feature.

In chapter 2 an overview of energy aware hardware in today's desktop and server systems is given to evaluate possible hardware for an energy efficient cluster.

Based on the approach to switch off unused hardware components a model is developed to estimate the power consumption of the hardware with and without energy saving mechanism in chapter 3. With "trace files" containing information about the utilization on the one hand and the knowledge of the components utilization based power consumption on the other hand the power consumption of each hardware component can be estimated. The component utilization based power consumption can be gained from experimental setups described in this work.

The software resulting from this model is described in chapter 4.

Chapter 5 assesses various parallel program runs on the cluster of the Research Group Parallel and Distributed Systems at the University of Heidelberg. Each program run and the associated measured power consumption is analyzed in consideration of the configuration, the run time and the power consumption.

With these program runs the model and the resulting software are evaluated in chapter 6, the measured power consumption values are compared with the estimated ones of the software. Further energy aware hardware is simulated to determine upper bounds for energy savings without performance leakage. This simulation is based on several strategies for switching to low power consuming

[1] a normal household price is about 0.15 €, but cluster operators get special prices

[2] based on average emission of vehicle registrations in Germany 2007, `http://www.watt.de/CO2_Rechner.aspx`, last checked on August 27, 2009

modes in low utilization phases. The estimated power consumption under usage of the different strategies is analyzed based on different program and hardware configurations to show the potential of energy savings.

1.1. Definitions

Table 1.1.: *Definitions*

Name	**Unit**	**Description**
Energy	J	$1\,\mathrm{J} = 1\,\mathrm{kg\,m^2\,s^{-2}}$, one joule is defined as the amount of work done by a force of one newton moving an object through a distance of one meter [Phy04]
Power	W	V A, watt, equivalent to energy per second ($\mathrm{J\,s^{-1}}$). Meaning the rate at which energy is generated and consumed [Phy04]
Energy	Wh	$1\,\mathrm{Wh} = 3600\,\mathrm{Ws} = 3600\,\mathrm{J}$ [Phy04]
Thermal design power (TDP)	W	represents the average maximum power the cooling system in a computer is required to dissipate
Performance	Flops	Floating point operations per second [Top]
Power Efficiency	Flops / W	Power efficiency measured in performance per power [Gre]
Speedup	–	$t_1\,t_n^{-1}$, t_1 = run time for one process, t_n = run time for n processes [Lud08a]

1.2. High Performance Computing Clusters

This chapter gives an introduction into cluster computing. It starts with the formal definition followed by some applications to motivate the building of cluster systems. Further the problem of energy consumption and some methods to resolve it are described.

Cluster computing is defined as the usage of lots of connected and homogeneous hardware components for high performance computing (HPC) [Lud08a]. In earlier days, clusters were built by connecting normal desktop PCs by a network. These clusters are called Beowulf-Clusters. Each of these PCs is called cluster node. The nodes have common hardware components like CPU, main memory (RAM), hard disks, network cards (NIC), main board etc. Because a cluster usually is controlled by a master node over network, the other nodes do not have Human Interface Devices (HID) like keyboard, mouse or monitor. To manage and monitor the cluster and its components an operating system (usually UNIX based) and cluster software is required which isn't described in detail here.

Today high performance clusters are mainly built by special business units of different companies like HP, IBM, Cray and SGI. An overview of high performance cluster systems is given by the organization Top500.org [Top]. This organization biannually ranks clusters from all over the world by performing a benchmark called LINPACK and publishes the results at the International Supercomputing Conference [3]. The benchmark used in LINPACK is to solve a dense system of linear equations [Top]. Based on this specific calculation and the time needed to perform this task performance is calculated in Flops. The fastest system of the June 2009 list reaches 1105000 GFlops while normal desktop computers reach between 1 and 8 GFlops (mainly dependent on the CPU).

[3]`http://www.supercomp.de`, last checked on August 27, 2009

Implied by the LINPACK benchmark HPC systems are mainly used for solving complex numerical problems. These problems are native to many domains of science like fluid mechanics, bio-informatics, physics, climate research etc.

For getting program code to run on a cluster environment, the original algorithm (called sequential program code) has to be parallelized. On these high performance clusters usually techniques such as MPICH [4], OpenMP [5] or a combination of both (Hybrid Programming) are used. While MPI (Message Passing Interface) communicates as the name implies with sending messages to other processes possibly located on other nodes (distributed memory), OpenMP is used for communication among threads in the same address space (shared memory, mainly the same node).

To parallelize code for a high performance cluster the main challenge apart from producing a correct program is to create a scalable program. In this context scalable means that incrementing the number of nodes or processes proportionally decreases the run time of the algorithm. Scalability of some parallel algorithm is limited based on the particular sequential algorithm. But if an algorithm is scalable in general, the programmer's challenge is to locate and minimize waiting times resulting from communication or hardware bottlenecks. To understand a parallel program (especially its communication schema) and to locate the bottleneck of the program its code can be instrumented to generate a trace file. This trace file contains information about the communication schema of the processes, hardware usage etc. By analyzing this file potential bottlenecks (software and/or hardware based) can be identified and further analyzed.

Because of the performance driven development of the hardware (and the software) the energy consumption of the hardware has become a real problem in the last years. This is mainly based on the high power density. For example Intel's Pentium-I processor (1993) has a maximal power consumption of 14 W at a chip size of 292 mm^2 and a resulting power density of about 4.8 W/cm^2. Intel's Itanium2 processor (Fanwood, 2004) has a maximal power consumption of 130 W, a chip size of 374 mm^2 and a resulting power density of about 34.7 W/cm^2 (about 7 times higher).

Out of this, not only the direct energy consumption of hardware has to be considered, the cooling of the hardware must be regarded as well. The power fed into the hardware results in heat which has to be dissipated. Because of the space requirement of a cluster installation which might be a few hundred square meters these rooms have to be fully air-conditioned. This results in a power consumption (for the Top 3 installations, see table 1.2) between 2 and 7 MW [6]. Interestingly, there is a big difference between the energy consumption of the three systems. While the peak performance differs in a range of 400 TFlops, especially the first and the second ranked systems differ in peak performance in about 70 TFlops (about 5 %). But the power consumption differs in about 4.5 MW (about 280 %).

Table 1.2.: *Abstract of TOP500 List (June 2009) for supercomputers [Top]. The power value is given in kW for the entire system, the R_{peak} value is given in TFlops (10^{12} floating point operations per second).*

Rank	**Computer**	**Vendor**	**Year**	**Cores**	**R_{peak}**	**Peak power**
1	Roadrunner	IBM	2008	129600	1456.70 TFlops	2483.47 kW
2	Jaguar	Cray Inc.	2008	150152	1381.40 TFlops	6950.60 kW
3	JUGENE	IBM	2009	294912	1002.70 TFlops	2268.00 kW

To compare installations on the basis of their power consumption and their performance, there is another ranking called Green500 [Gre]. The installations are ranked by MFlops/W to get the performance in relation to the power consumption (power efficiency). In table 1.3 the top 3 systems from table 1.2 and the first rank system are listed. It is interesting to see that the first system in the

[4] `http://www.mcs.anl.gov/research/projects/mpi/mpich1`, last checked on August 27, 2009
[5] `http://openmp.org/wp`, last checked on August 27, 2009
[6] enough to sustain a city of between 8,000 to 28,000 inhabitants

Green500 list nearly reaches 100 MFlops more per watt than the first system in the Top500 list. The aforementioned difference between the two first ranked systems in the Top500 list is clearly reflected in the Green500 list: The Jaguar system (with the highest peak power consumption of the system listed in table 1.2) only ranks at position 90.

Table 1.3.: *Abstract of Green500 List (June 2009) for supercomputers [Gre]. The total power value is given in kW for the entire system, the efficiency is given in MFlops (10^6 floating point operations per second) per watt.*

Rank	Computer	Efficiency	Peak power	TOP500 Rank
1	BladeCenter	536.24 MFlops/W	34.63 kW	422
4	Roadrunner	444.94 MFlops/W	1456.70 kW	1
18	JUGENE	363.98 MFlops/W	1002.70 kW	3
90	Jaguar	152.36 MFlops/W	1381.40 kW	2

But the trend to save energy (and lower the costs) is not new to big computer system installations. Datacenters for example are cluster installations with more general purpose to store and analyze data (for example Google stores information about search queries in large datacenters) or to maintain different servers (for example an Internet service provider, ISP). These datacenters have a commercial use and have to be economically in their costs. Due to this, the performance is not the primary goal (as in HPC). Also the different usage of the cluster allows to develop energy saving mechanisms. For example a data center of an ISP which rents web servers. Each web server is realized as a virtual machine and multiple virtual machines are located on one node. This can be done because web servers often are under utilized and do not need the performance of a whole node. If a server needs more performance, the virtual machine can migrate to another (yet unused) node. If the performance is not necessary, the unused node can be turned off to conserve energy.

Another approach is to use the waste heat. This heat can be used to heat water (e.g. for a heating system or a swimming pool). It is also possible to use the environment for cooling (for example cold winter air) instead of mechanical cooling.

Some of these concepts are already applied to HPC, but with savings in the energy consumption of the hardware these concepts can be improved.

1.3. Project Goals

This project's main goal is to motivate further research of energy saving mechanism in high performance computing environments. The usage of energy efficient components have been rejected in reference to the possible performance loss and the higher acquisition costs. But it is possible that the increased calculation time due to reduced performance results in a even better energy efficiency of the hardware.

The approach based on reducing the component idle power consumption affects the total power consumption twice: First the component power consumption is decreased and second less lost heat has to be dissipated. The usage of hardware that support different working modes for high and low utilization times promises power savings with small or even without performance loss if used in the right way.

This work is a first step with the first intention to assess the energy consumption of different cluster components and program configurations.

The second intention is to simulate the power consumption and possible power savings with and without reducing performance for each component to motivate further research of the mentioned ideas.

For this purpose a model is designed which calculates the power consumption for each component based on the component's utilization. Further the components power consumption at zero utilization (idle) and full utilization (load) is used to estimate the power consumption. The information about the utilization is gained from trace files, while the zero and full utilization values are gained from experiments.

With different strategies it is possible to estimate the power consumption with different hardware and power saving characteristics.

The estimated power consumption must be interpreted as an upper bound for power savings that can be reached with energy aware hardware. Based on the specific program traces the monetary evaluation of these hardware components is possible: The energy savings can be compared with the acquisition costs of the specific hardware.

This is interesting for clusters built for different application types, because the different types have different utilization profiles. For example some applications are communication intensive (high utilization of the network card), some other applications are IO intensive (high utilization of the disk subsystem). With different utilization profiles from different workload some resources aren't fully utilized (for example the performant disk subsystem for the communication intensive application). For clusters built up for one application type the hardware is usually optimized for this workload to prevent idle times.

Especially for waiting times resulting from hardware bottlenecks and/or problem specific behavior the estimation of the power consumption is useful. Phases with partial utilization of specific hardware devices can be identified. With modification of the parallel algorithm the device utilization can be changed to phases with full and zero utilization respectively to make efficient device power state changes possible.

The power savings can be monetary calculated for the specific program, configuration and the hardware. This has advantages for the cluster operator, but also advantages for the hardware vendor and further researches.

2. Energy Aware Hardware

This chapter gives an overview of energy aware hardware components which can be used to build up an energy efficient cluster. For this purpose, the energy saving potential of each component is discussed by analyzing several research approaches.

Energy aware hardware in the context of this thesis is hardware, that supports different operating modes, each of these with a different power consumption. When decreasing the power consumption from one mode to another mode, some functions of the specific component must be changed or turned off to save energy. To reuse these functions the component must switch back to the previous mode. This switching can result in latency and additional energy consumption, because the hardware must be reactivated (e.g. spin up the hard disk). This trade-off has to be considered when using energy aware hardware. The concrete capabilities of hardware components are discussed in the following section.

2.1. Overview of Hardware

CPU

The analysis starts with the most power consuming component, the Central Processing Unit (CPU). Because the CPU consumes between 15 % (in idle mode) and 50 % (CPU load) of a mobile device's power [MV04], a lot of research has already been done to reduce the energy consumption of this component. But until now, the developed energy saving mechanisms are not used in high performance cluster hardware because of potential performance loss.

The energy consumption of a CPU depends on several factors: The frequency f, the core voltage V, the level of chip activity α and a factor C dependent on the capacitance of the chip [MV04]. Equation 2.1 describes the power consumption of a CPU.

$$P_{\mathrm{CPU}} = \alpha\, C\, V^2\, f \tag{2.1}$$

The linear effect of the "level of chip activity" α shows that the CPU still consumes lots of energy, even if not utilized (because the chip is still active and α is not zero). This is the main problem of components without energy saving mechanisms. The core voltage V effects the power quadratically, while the other factors only have a linear effect. Further reducing the core voltage results in a reduced frequency. Out of this, the main approach is to reduce the core voltage (and so the frequency) in phases of low utilization. These technologies are called "SpeedStep Technology" (for Intel processors) and "Cool'n'Quiet" (for AMD processors). To manage the energy consumption without lowering the frequency, today's processor hardware components can be switched on and off with fine granularity. This way it is possible to disable whole cores (in case of multicore processors) or core functions, such as first/second level cache or Hyperthreading. Intel already introduced a technology to manage the processor's energy consumption at this level called "Intelligent Power Capability". Newer desktop, server and especially notebook processors provide the introduced energy saving features.

Disk

In today's computer systems the hard disk drive is a moderate power consuming components [MV04]. But in idle mode (drive not serving any requests) the platters are still spinning. This leads to a great potential for energy saving when platters are spinned down for idle times. Servers with a low mean utilization do not use the disks frequently and the platters could spin down. This is one of the most popular approaches, because the angular velocity ω of the motor has a quadratic effect on the power consumption (see equation 2.2, K_e is the motor voltage constant and R is the motor resistance) [GSKF03].

$$P_{\text{disk}} = \frac{K_e^2\,\omega^2}{R} \tag{2.2}$$

Today's hard disks support several modes to reduce energy consumption: Unloading the heads, parking the heads, reducing the RPM (revolutions per minute) when idle and powering of the motor. Reducing the RPM only is possible if the heads are parked in the parking zone because of the danger of head crashes.

The approach of DRPM (Dynamic Rotations Per Minute) [GSKF03] is a method explored for dynamically controlling the speed at which drives spin. With this approach, energy is saved while preserving high availability (because writing is still possible at lower RPM). But today's hard drives do not offer these multi-speed operating modes.

While most approaches try to reduce the power consumption of the mechanical parts of the disk, there also is an approach to reduce the power consumption of the electrical parts [HSRJ08]. This paper shows that the energy required to access data is affected by physical location on a drive. Also, the size of data transfers has measurable effects on power consumption.

Network

Research about the power consumption of the Internet [GS03] and Ethernet LANs [GS07b] shows that big networks like the Internet or even campus LANs have a low mean utilization, while network components like network interface cards and switches consume as much energy as when under load. Because of this, some research has been made in order to reduce power consumption of network interface cards and corresponding switches. Gupta and Singh designed a Dynamic Ethernet Link Shutdown (DELS) algorithm leading to significant benefits in energy savings with little noticeable impact on packet loss or delay [GS07a]. They also published an approach to use the low-power modes of network interfaces [GS07b]. This approach uses hardware that is designed to change link rate for saving power in battery-operated devices. They developed an algorithm to change the link rate if the link is idle or under-utilized. Another approach is to design a network interface card containing a secondary processor and an Embedded OS including the networking stack and a flash storage [AHC+09]. This NIC can handle network applications while the other components such as main CPU etc. are asleep. Under use of application-level triggers the host wakes up for specified applications or packets.

On the other hand research has been made for power management in LAN switches [GGS04]. This work examines methods to reduce the power consumption in switches dynamically using low-power states depending upon traffic activity at each interface and analyzes the impact of sleeping on layer 2 protocols (example for a layer 2 protocol: the Address Resolution Protocol (ARP)). Another approach is to manage the power consumption of network devices with power consumption policies. By monitoring the power consumption of "EnergyWise" devices it is possible to manually select different energy policy settings to start managing the network's energy consumption [Aud09].

Memory

Main memory of desktop or server systems often is optimized for performance. Out of this, at low utilization times the main memory only is partially used. Moona et al. [MCH07] analyzed the effect of changing memory performance for different applications and implemented Dynamic Memory Switching for Linux. A kernel daemon migrates pages to different banks (fixed sized sections of Rambus RDRAM modules) and turns unused banks off. Switching a bank off means that a bank switches from Active to Nap, similarly bank on means switching from Nap to Active (see table 2.1). The Nap state is a sleep state with a very small latency to wake up.

Another approach is based on the high peak power consumption of main memory. Diniz et al. propose techniques that limit consumption by adjusting the power states of the memory devices as a function of the load on the memory subsystem [DGJB07]. This leads to an optimization problem

Table 2.1.: *Specifications of Rambus RDRAM for a* 32 *MB bank [MCH07],* 2 *GB RDRAM consumes approximately* 16 *W.*

Power State	Power	Active Components
Active	300 mW	Refresh, clock, row, col decoder
Standby	180 mW	Refresh, clock, row decoder
Nap	30 mW	Refresh, clock
Powerdown	3 mW	Refresh

for the memory controller to select the different power states. This technique is not limited to RDRAM-based memory subsystems, they can treat entire DDR modules as well as single RDRAM chips.

Remaining components

There are some other remaining components in a cluster node which also consume energy, like the uninterrupted power supply (UPS), the main board with North- and Southbridge, various ports (USB etc.), graphics card and fans. These components are not discussed here because on one hand it is difficult to determine and control their actual power consumption, and on the other hand these components are more or less dependent on the already discussed components. But one component needs to be separately mentioned: the power supply. The power supply's task is to change the voltages from one level to another level, mainly from the power grid voltage (e.g. 220 V) to 3,5 V and 12 V for the node components. But this change is not lossless, this energy loss is generally transferred as heat to the power supply. The relation of the input and the output power of the power supply, called efficiency, ranges from 60 % up to 95 % (for a SMPS, switched-mode power supply) [AH03]. But the efficiency of the power supplies does not only differ from device to device, the main problem is that the efficiency depends on the load. So usually for low and high utilization the efficiency is better than for medium utilization. Due to the fact that there is no relation between the performance of a node and the efficiency of its power supply, there is no obstacle to use SMPS's (apart from the acquisition costs) in high performance clusters to reduce energy loss.

2.2. Managing of Energy Aware Hardware with ACPI

The presented energy saving mechanisms of the different components of a cluster are partially controlled by the component itself, but the operating system has a better overview over the components. Hence the energy management usually is performed by the operating system using the Advanced Configuration and Power Management Interface (ACPI) [CCC$^+$05]. The ACPI specification is an open standard for unified operating system-centric device configuration and power management. The system is divided into several states: The Global System States (G0-G3), the Device Power States (D0-D3), the Sleeping States (S1-S5), the Processor Power States (C0-C3) and the Device and Processor Performance States (P0-Pn). Generally state 0 is the "Working" state with the highest power consumption. To increase the state means to decrease energy consumption and to increase wake-up time (latency) of specified device or system. The last state implicates the "Mechanical Off" for the component (exception: Sleeping State S5 implies the "Soft Off" state). The Device and Processor Performance States imply the Processor State C0 and the Device State D0, but a different performance level (e.g. using a mobile device with battery or power supply).

An explicit overview of the different states is located in the appendix. Based on these state specifications, hardware vendors can implement reactions for state changes occurring for ACPI-aware components.

In reality in the power state the component could decide to disable internal circuits if it stays ready all the time. For instance the processor could disable the ALU per instruction if it can turn it on rapidly, i.e. without performance loss, by knowing the future instructions in the pipeline.

2.3. Analysis of Energy Saving Potential

The introduced energy saving mechanisms for the different hardware components are yet partially under development in today's computer systems.

The mechanisms of the CPU are well developed and used in today's energy aware computer systems. This is based on three aspects: At first, the CPU is one of the most power consuming components. At second, the CPU is one of the most heat generating components, so reducing the power consumption for the CPU means reducing the heat which reduces the fan RPM. This in turn reduces the energy consumption and -especially for mobile devices- the noise of the whole system. The third aspect is that switching the different power states for a CPU has (compared to the other components) the lowest wake-up times ranging between a few nanoseconds and some milliseconds.

The hard disk mechanisms are also used in today's energy aware systems, especially mobile devices. The wake up time for the disks are a multiple of the wake-up time of a CPU, but the operative point are the caching capabilities of the main memory for hard disk tasks. Data can be written to the main memory while the disk is powering on and the writing task itself can wait until the disk is in operating mode. This can be handled in the background without users notice (with small amounts of data).

The mechanisms for main memory, network cards and switches are not used in today's systems, because the hard- and software for these tasks are still in development. In the past, there has only been small interest in developing algorithms for these components, because the power consumption is only a fractional amount of a mobile system [MV04]. Other components, that are not relevant for high performance computing (e.g. LCD panels) are more interesting concerning the energy consumption.

In today's research the main memory, network cards and switches are also included based on the growing networks like the Internet or campus LANs, the increasing number of datacenters, clusters and also personal computers and mobile devices.

For high performance clusters these parts are more relevant, because of the sum of the individual components. If a network card has an energy saving potential of maybe 5 %, this is not really interesting for a desktop or even a mobile system. But for a cluster with about 6000 network cards, the aggregated savings are interesting.

But why are the mechanisms (e.g. for CPU and disk) not used in high performance clusters?

The main point is that in high performance computing the time overhead for switching between different power states can increase the computing time significantly even with small state change durations. High performance algorithms have lots of fine tuning to minimize time overheads. For this tuning, each algorithm is analyzed to find waiting times (e.g. *MPI_Barrier*) and these times have to be minimized. Lots of algorithms are also based on topology specific communication schemes and network responding times. Out of this, small influences for the responding time of components can significantly reduce the performance. If the performance is reduced, the calculating time for the whole cluster increases and hence the power consumption.

Another point is that the caching capabilities for the data to be written to disk are restricted. Some parallel programs are using the whole main memory for calculation. These parameters are specified by the developer to increase the calculating performance. A typical use case is called checkpointing. For this, the working datastructures (in this case, the bulk part of the main memory) are written to disk. This is done for backup purposes, because most algorithms calculate for days, weeks and even months or years. If the program crashes or a hardware failure occurs, the calculation can be restarted using the checkpoint data. If a checkpoint should be written, disks have to be in working state (because no caching is possible) to avoid performance degradation. Because usually the operating system manages the ACPI states of the hardware, no information for the future disk use is available and so the influence on the computing time can not be minimized.

The main requirements to control energy saving without reducing performance is the information which hardware component is used when. There are two main ideas how this information can be gained: First, the programmer of the parallel algorithm knows when components are used, so he can publish this information (like instrumenting the code). Secondary the program trace can be analyzed to gain the typical usage of the components. A second run of the same (or similarly) program can use this profile created in the first run. With this information, the components can change their ACPI levels to decrease the power consumption and wake up before the specific component is used.

But these concrete approaches are not discussed in this work.

3. Estimation of Cluster Power Consumption

In this chapter different models for estimation of power consumption of a cluster environment are explained and discussed. After the discussion of these models the chosen model is described in detail. Section 3.2 contains an approach to determine the input values for the modeled components. To estimate the power consumption different strategies are described in sections 3.3 and 3.4.

There are two main ideas to estimate the power consumption of a specific program run on a cluster. The first idea is to gain knowledge about the power consumption of specific operations in a computer system (like writing a block to disk or assigning a variable). With this information, a program run can be simulated and the power consumption can be estimated. A simulator to analyze parallel programs and the underlying cluster hardware is in development at the Research Group Parallel and Distributed Systems. The power consumption has a high accuracy because the data about the specific operations is cumulated in a very granular way via the simulator. To calculate the power consumption with different energy-aware features of the hardware, the simulator offers a realistic environment to develop different strategies to use these features.

But this realistic environment has the disadvantage of its high complexity. Intelligent hardware and software is using lots of mechanisms to optimize the hardware access such as caches. Therefore, if a program for example writes a block on disk, there is almost no knowledge about when this block is really written on disk. It is possible that this block is held in the cache and only if enough blocks are in the cache, the data is written to disk. It is also possible that the block should be overwritten or deleted, so the writing does not have to be performed. The only way to sustain the real hardware access by the simulator is to implement the caching algorithms. But this a complex feature on the one hand and on the other hand the information about the caching algorithms is typically are kept secret by the hardware vendors.

Apart from the hardware caching mechanisms there are other optimizations on different layers, starting at the compiler optimizations over operating system based optimizations to optimizations of the file system, especially with intelligent cluster file systems such as PVFS [1]. Of course there are also optimizations for other hardware components, especially CPU and NIC, with similar behavior.

The second approach is to collect the utilization data in a coarse-granular way. This data can be collected in fixed timesteps (for example by the operating system) and appended to a trace file containing information about the specific program (such as MPI calls). The trace file can be analyzed after the program run to estimate the power consumption based on the components utilization. With this utilization data it is possible to evaluate the future component's utilization when analyzing the trace. The inactive times can be identified and considered in the calculation of the power consumption. It is possible to consider sleeping times for the affected devices without affecting the calculating time. In this way an upper bound for energy saving can be calculated.

The granularity of the utilization values is based on the mechanism used to collect the data. If data is collected too frequently, the CPU is busy with collecting the utilization data. That is why the frequency has to be adjusted to a level with adequate granularity without affecting the performance.

This second model will be described detailed in the next section.

[1] `http://www.pvfs.org`, last checked on August 27, 2009

3.1. Model

In general, the power consumption of a cluster environment is the sum of the power consumption of its components (nodes, switches and cooling environment). The power consumption for the cooling environment simply can be included in a model. This consumption depends on the power consumption of the other components. Every watt needed by a component like a CPU or a switch results in waste heat which has to be cooled down. The power consumption for cooling can be easily estimated by multiplying the consumption of all other components with an environment based factor. For today's HPC installations this usually is the factor 2, each watt produced has to be cooled down using one watt for air conditioning. But in future installations this factor will be decreased (e.g. increasing the operating temperature of cluster environments so that the outer air can be used for cooling).

The power consumption of the switch is nearly independent of the net utilization, it consumes as much power when active as when it is inactive. Hence, the power consumption of a switch is a constant, which can be added to the node's power consumption. For this reason, the estimation of the node's consumption is the main challenge. The power consumption of a node is the sum of the node's hardware 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 almost every component the power consumption is based on the utilization (a calculating CPU uses more energy than an idle one).

The chosen model contains abstractions for the following components: CPU, memory, disk, NIC and the power supply. These components are grouped to a node. The sum of the power consumptions for all components of a node is the node consumption. It is possible to include a percental overhead for other components like main board etc.. The power consumption of each component based on the utilization is linearly interpolated from two values: The first one is the power consumption at zero utilization and the second is the power consumption at full utilization. This results in a power consumption / utilization graph like shown in figure 3.1.

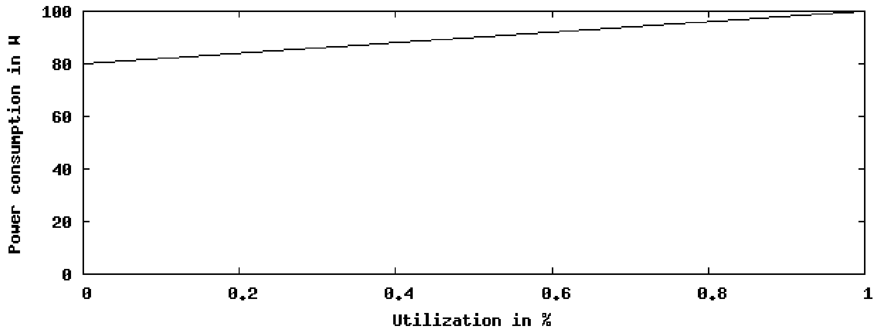


Figure 3.1.: *Power consumption / utilization graph for a specific component. At zero utilization this component has a power consumption of* 80 *W, at full utilization* 100 *W. For every other utilization value the power consumption is interpolated linearly.*

In reality, the power consumption / utilization graph is not linear and the characteristics are different for each component. But this information is not easy to obtain: The majority of hardware vendors does not publish this information and the equipment for taking detailed measurements has not been applicable. To measure an indicated value for a specific component special equipment is needed, because the power consumption for a CPU (without main board) for different utilization values is not measurable easily. Out of this, only the two values for high and low utilization are used. The concrete measuring is described in section 3.2.

Further the power consumption for modeling the different ACPI states is needed. The power consumption for Device Power State 0 is as already mentioned linearly interpolated. For all other non-working states ($1-3$) no interpolation has to be done, because the components are in a sleeping mode where the power consumption is independent from the utilization. Different working states where for example the CPU is used at a lower frequency (C-States) are not modeled here, because the trace file does not contain this information and the cluster's hardware (see next chapter) does not support this feature (in general, it is possible to get statistics about the CPU frequency, e.g. with PowerTop [2]). To switch to a different ACPI Power State two further values are needed: The duration for switching and the energy consumption for switching. If a disk is switched from ACPI Device State 3 (deeper sleep) to Device State 0 (working), this takes a few seconds. And also some energy is needed to spin up the disk to the required revolutions per minute. These values are depending on the component and the concrete change, so for every state change (incrementing and decrementing) values can be considered. The switching between two not sequenced states is realized by sequential incrementing or decrementing respectively until the desired state is reached.

Because the components are grouped in nodes, the ACPI Global System States are all associated components in the equivalent ACPI Device State.

Based on this model for the power consumption in each ACPI Device State and the utilization from the trace file the power consumption can be estimated. This process is modeled as a Replay: For each fixed step (defined by the stepsize of the utilization values in the trace file) the power consumption for each component is calculated. This process (without usage of the ACPI Power States) is the simplest strategy for power estimation and adequate to the real power consumption, because the cluster hardware is (with one exception, see section 5.1) not ACPI capable. However with the input values of the different ACPI State consumptions a potential power saving can be calculated, when putting components to a higher ACPI State (e.g. lower power consumption) when inactive. Different strategies for putting components to sleep (mainly based on the character of the component and the utilization) have been developed and implemented. These strategies are described in detail in the next section.

3.2. Determination of Component Power Consumption

As discussed in the last section the following values are needed to estimate the component power consumption:

- power consumption without utilization
- power consumption with full utilization
- power consumption in ACPI Device Power States $1 - 3$ (sleeping states)
- energy consumption and duration for switching the ACPI Device Power States

One way to identify the power consumption can be using data sheets of each hardware vendor. But the data published in these data sheets is mainly based on marketing strategies. So nearly every vendor defines or re-defines some power consumption values, but the concrete data is not significant in most of the cases, especially for different utilization phases. To distinct these values standardized benchmarks are needed, but these benchmarks do not exist yet.

But if the power consumption of a component doesn't differ significantly between high and low utilization phases, or in general it has a low power consumption, benchmarks are too inaccurate. The measuring of the component based power consumption is in this respect a problem as not each component has a direct power cable supply. The components located on the main board (such as CPU, main memory, network card) are served by the main board, so the specifying energy into individual component consumption is only approximately possible. When using a stresstest for different components the relation between the components have to be considered (e.g. an Ethernet card possibly utilizes the CPU). This second level utilization is not insignificant, but is not easy to measure because of fast switching between high and low utilization.

Of course this component based measuring is the most accurate way, but is not possible without special measuring equipment.

With the measuring equipment available at Research Group Parallel and Distributed Systems it is possible to measure the power consumption at node level. The different utilization of each component

[2]`http://www.lesswatts.org/projects/powertop`, last checked on August 27, 2009

has influence on the node power consumption and out of this the difference between the component power consumption with different utilization can be measured.

In this approach especially the characteristics of the power supply have to be considered. On one hand the power supply's overhead (for converting the voltage) have to be deducted to determine the components power consumption. On the other hand the charge process of the supply's capacitors is dependent on the specific device and not urgently results in equivalent power consumption. When components need more power for a short time period this is not recognizable when measuring the node consumption, because the needed power is stored in the power supply's capacitors. If for example a component increases its power consumption for 1 millisecond by 10 W, this can possibly result in a different power consumption for the power supply (e.g. increasing power consumption for 10 milliseconds by 1 W).

The resulting procedure to determine the power consumption of selective components is combination of measuring the node power consumption and using the hardware vendor data sheets. To determine the idle power consumption different sets of components can be removed from one cluster node and the power consumption can be measured when entering the BIOS (basic input/output system) of the node (each resulting power consumption is named $BIOS_{components}$). For the utilized power consumption a stresstest has been implemented to stress different sets of components (each resulting power consumption is named $STRESS_{components}$). Based on the limited capability to stress and remove components the modeling of components is limited (e.g. the stressing of the network card results amongst other in stressing the network controller whose increased power consumption is included in the power consumption of the network card).

The idle and utilized power consumption for each component based on the total power consumption is calculated as follows: At first, each measured value (from the stresstest and the idle power consumption) is multiplied by the power supply efficiency to get the power consumption without the overhead of the power supply. With the node consumption with different hardware configurations in idle mode the idle power consumption for each component (except the CPU) can be calculated ($MEM_0 = BIOS_{MEM,others} - BIOS_{others}, DISK_0 = BIOS_{DISK,MEM,others} - BIOS_{MEM,others}, \dots$). Also the difference between the idle and utilized hardware can calculated subtracting the idle power consumption ($CPU_{diff} = STRESS_{CPU} - BIOS_{ALL}, MEM_{diff} = STRESS_{CPU,MEM} - STRESS_{CPU}, \dots$). With these values and the idle power consumption the utilized power consumption for the components (without the CPU) can be calculated ($MEM_1 = MEM_0 + MEM_{diff}, \dots$). When using the utilized power consumption for the CPU from the vendors data sheet the idle power consumption of the CPU can be calculated the same way.

The resulting power consumption values for idle and utilized components can be compared with further hardware vendor data sheets if available.

The power consumption for the other ACPI Device Power States ($1-3$) and the energy consumption and duration for switching the ACPI Device Power States must be extracted from various data sheets of comparable hardware, because the concrete hardware is not ACPI capable.

3.3. Power Consumption Estimation Strategies

To estimate the power consumption of a cluster four different strategies have been implemented. These strategies are component based, so every component in a node can use a different strategy. The four strategies are named Simple Strategy, Optimal Strategy, Approach Strategy and MultipleState Strategy.

The first strategy estimates the power consumption without the usage of any power saving mechanism. For the Simple Strategy, every single timestep is evaluated independently. The other strategies are look ahead strategies, which take different decisions based on future utilization values. The Optimal Strategy decides to put a component into sleep mode if the future utilization is zero for sufficient timesteps and wakes it up before it is used again. Because such zero utilization times are not frequently for some components such as a CPU, the Approach Strategy puts a component into sleep if the utilization is under a specified level, e.g. 5 %. At the end of the low utilization phase an equivalent phase with high utilization is performed to adjust the utilization. The last strategy assigns different levels of utilization to different ACPI Device States. This is useful for components like the main memory. If the main memory utilization is under 50 %, half of the banks can theoretically be switched off to save energy. These four strategies are explained and discussed in detail in the next subsections. For this, an example utilization graph is created for a single component. Based on this utilization, for each strategy the resulting power consumption graph is shown to explain the differences between the strategies.

3.3.1. Simple Strategy

To estimate the real power consumption without energy aware hardware the power consumption for each hardware component has to be interpolated. To interpolate, the consumption values for low and peak utilization have to be defined. The power consumption P_{curr} for a given utilization u_{curr} and the power consumption at ACPI Device State 0 $P_{\text{ACPI0}_{0\,\%}}$ (zero utilization) and $P_{\text{ACPI0}_{100\,\%}}$ (full utilization) is calculated in equation 3.1.

$$P_{\text{curr}} = (P_{\text{ACPI0}_{100\,\%}} - P_{\text{ACPI0}_{0\,\%}})\, u_{\text{curr}} + P_{\text{ACPI0}_{0\,\%}} \tag{3.1}$$

Each timestep of the given utilization is evaluated (the step power consumption is calculated with equation 3.1) independently, hence the power consumption values are as granular as the utilization values. The power consumption for the component is the accumulated sum of the step power consumptions (power consumption per step), the power consumption for the node is the accumulated sum of its components power consumption.

Because the calculated power consumption only depends on the utilization and the ACPI Device Power States this strategy can also be used to estimate the power consumption with energy efficient hardware. Energy efficient hardware means hardware that consumes less energy when not utilized.

3.3.2. Optimal Strategy

The Optimal Strategy uses the different ACPI Device Power States of the component. This strategy calculates the minimal power consumption without reducing the performance, because components only change to sleep mode if the component is not used and wake up before the component is used again. Only the ACPI Device Power States 0 (working) and 3 (deepest sleep) are used. The decision if a component switches to sleep mode depends on different factors: The duration of the zero utilization phase, the power saving potential of the state change (P_{diff}, difference of power consumption ACPI0 and ACPI3), the duration of the state change (t_{change} , sum duration ACPI0 to ACPI3 and duration ACPI3 to ACPI0) and the energy of the state change (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 which sleeping reduces the power consumption without reducing performance (see equation 3.2).

$$t_{\text{optimal}} = \frac{E_{\text{change}}}{P_{\text{diff}}} + t_{\text{change}} \tag{3.2}$$

The algorithm 1 describes the procedure in detail for a node with Optimal Strategy.

In difference to the power consumption with the Simple Strategy the component uses the sleep mode at the zero utilization phase and wakes up before it is used again. Therefore the power consumption is decreased without increasing the calculating time. Because the future utilization in concrete runs of parallel programs is not known, the output of the Optimal Strategy is an approximated upper bound for power saving.

Algorithm 1 Optimal Strategy

```
Require: All components in state ACPI0 (working)
  for $step$ = 1 to $countSteps$ do
    for $device$ in $nodeComponents$ do
      if $device$ is working then
        if $device$'s future utilization is zero for at least $t_{\text{optimal}}$ steps then
          $device$ go to sleep mode (ACPI3)
        end if
      else
        {$device$ is sleeping}
        if $device$'s future utilization is not zero for at least $t_{\text{ACPI3-ACPI0}}$ steps then
          $device$ go to working mode (ACPI0)
        end if
      end if
    end for
  end for
```

3.3.3. Approach Strategy

Because some components do not have zero utilization phases but low utilization phases, the Approach Strategy has been developed. The intention is to simulate more energy efficient characteristics. If a component works for example 10 seconds with a low utilization of 10 %, the same work could be done working 1 second with a utilization of 100 %. This results in 9 seconds with zero utilization where the component can switch to sleep mode. The rearrangement of load can be thought of reordering the code. However, in reality this might not be possible due to dependencies between calls etc..

The calculation of the minimal time for an efficient state change is now dependent on a tolerance value δ. The timesteps with a utilization lower than δ will be rearranged if the count of timesteps under the specified tolerance is greater or equal t_{approach} (see equation 3.3). The rearrangement of load results in a zero utilization phase (t_{optimal}) and the utilization phase containing the equivalent load (t_{load}). The equivalent load is the aggregated utilization for the duration of t_{optimal} before the rearrangement.

$$t_{\text{approach}} = t_{\text{optimal}} + t_{\text{load}} \tag{3.3}$$

The algorithm 2 describes the procedure in detail for a node with Approach Strategy.

This strategy provides also an upper bound for power saving. In contrast to the Optimal Strategy the strategy rearranges the utilization, which more likely than not affects the calculating time of the parallel program because of the dependencies inside the code and/or the hardware. If for example the CPU is utilized 10 seconds with a mean value of 10 % for sending a message over network, the bottleneck is the NIC or the bus system. And a utilization of the CPU with 100 % for one second would not solve that problem. In this case another approach is needed, so it could be determined if other calculations in the parallel program can be pushed to utilize the CPU with the remaining 90 %. Out of this, the output produced by the Approach Strategy shows scopes where power saving could be scheduled. In the above example an approach to eliminate the bottleneck (e.g. a faster NIC) can increase the performance and possibly decrease the power consumption. It must be evaluated in the concrete use case if the faster NIC consumes more power than the CPU saves.

3.3.4. Multiple State Strategy

The last strategy is an extension of the Optimal Strategy to use multiple ACPI Device Power States. This allows to model another use case: Its possible for a component to use different states with different power consumptions and wake up times. A component to use this strategy is the main memory, because the main memory will never reach a zero utilization phase. Neither could it be put

Algorithm 2 Approach Strategy

```
Require: All components in state ACPI0 (working)
  for $step = 1$ to $countSteps$ do
    for $device$ in $nodeComponents$ do
      if $device$ is working then
        if $device$'s future utilization is lower than tolerance $\delta$ for at least $t_{\text{approach}}$ steps then
          $device$ go to sleep mode (ACPI3)
          rearrange $device$'s utilization
        end if
      else
        {$device$ is sleeping}
        if $device$'s future utilization is not zero for the next $t_{\text{ACPI3-ACPI0}}$ steps then
          $device$ go to working mode (ACPI0)
        end if
      end if
    end for
  end for
```

to sleep if only utilized with 10 %. But if the memory is split into multiple banks on the main board, some of these banks can be turned off if not utilized. It is also possible to use this strategy for the ACPI Processor Performance States (P-States), because if the CPU is for example utilized about 50 %, it is possible to lower the frequency, but it is not recommended to go in sleep mode.

For the Multiple State Strategy multiple utilization levels (u_1, u_2, u_3, u_4) have to be defined to trigger a state change. If the utilization is under the specified value (for example $u_1 = 0.75$) the state change from ACPI Device Power State 0 to ACPI Device Power State 1 is triggered. For increasing the ACPI Device Power State (and decreasing the power consumption) the time t_{min} can be calculated based on the future utilization steps and the next higher utilization level. The algorithm 3 describes the procedure in detail for a node with Multiple State Strategy.

Algorithm 3 Multiple State Strategy

```
Require: All components in state ACPI0 (working)
Require: $level[4]$ containing utilization levels $u_1, u_2, u_3$ and $u_4$
  for $step = 1$ to $countSteps$ do
    for $device$ in $nodeComponents$ do
      $currentState = device$'s current ACPI Device Power State
      if $currentState < 3$ then
        calculate $t_{\text{min}}$ based on $currentState$ and $currentState+1$
        if $device$'s future utilization is lower $level[currentState + 1]$ for the next $t_{\text{min}}$ steps then
          $device$ go to State ACPI$currentState + 1$
        end if
      end if
      if $currentState > 0$ then
        for each $state$ of the lower states do
          calculate $t_{\text{wakeUp}}$ for the change from $currentState$ to $state$
          if $device$'s future utilization is greater $level[state]$ in one of the next $t_{\text{wakeUp}}$ steps then
            $device$ go to state ACPI$state$
          end if
        end for
      end if
    end for
  end for
```

3.4. Example Power Estimation

In this section the previous described strategies are used to estimate the power consumption for an example component. Based on this component and a given utilization the different approaches of the strategies are visualized and compared.

First the input values for the example component have to be specified (see table 3.1). These values are chosen to see the difference of the strategies and aren't based on a real component.

Table 3.1.: *Power schema for example power estimation component. The power schema contains the power consumption for the different ACPI Device Power States and the energy and duration for changing these.*

Power schema entry	Value
$P_{\text{ACPI0}_{0\,\%}}$	80 W
$P_{\text{ACPI0}_{100\,\%}}$	100 W
P_{ACPI1}	75 W
P_{ACPI2}	50 W
P_{ACPI3}	25 W
Level u	$[1.0, 0.75, 0.5, 0.25]$
$E_{\text{ACPI0-ACPI1}}, E_{\text{ACPI1-ACPI2}}, E_{\text{ACPI2-ACPI3}}$	0 Wh
$E_{\text{ACPI3-ACPI2}}, E_{\text{ACPI2-ACPI1}}, E_{\text{ACPI1-ACPI0}}$	0.01 Wh
$t_{\text{ACPI0-ACPI1}}, t_{\text{ACPI1-ACPI2}}, t_{\text{ACPI2-ACPI3}}$	0 ms
$t_{\text{ACPI3-ACPI2}}, t_{\text{ACPI2-ACPI1}}, t_{\text{ACPI1-ACPI0}}$	500 ms

Not all strategies are using all entries in the power schema, so the definition of the level u and the power consumption for the ACPI Device Power State 1 and 2 is only used for the Multiple State Strategy (P_{ACPI1} and P_{ACPI2} respectively). The energy consumption for switching the ACPI Device Power States and the associated duration is used in the strategies Optimal Strategy and Approach Strategy, because the energy consumption (and duration respectively) of the change from state 0 to state 3 is the sum of the changes from state 0 to state 1, 1 to 2 and 2 to 3. The specification of these values is of course not necessary for Simple Strategy, because no energy saving mechanism is considered in this strategy. Only the power consumption values for zero and full utilization in ACPI Device Power State 0 are used ($P_{\text{ACPI0}_{0\,\%}}$ and $P_{\text{ACPI0}_{100\,\%}}$ respectively).

Second the input utilization has to be specified (see figure 3.2(a)). The domain axis is divided in timesteps, timestep 1 is the timestep from $0 \rightarrow 1$ and so on. This utilization has been chosen to see effects for each strategy.

The resulting power consumptions for each strategy are summarized in table 3.2.

Table 3.2.: *Summary of example component energy consumption and savings for each strategy.*

Strategy	Total energy consumption	Saving
Simple	0.38 Wh	
Optimal	0.334 Wh	12.1 %
Approach	0.303 Wh	20.3 %
Multiple State	0.278 Wh	26.8 %

The energy consumption of 0.38 Wh for the Simple Strategy is equivalent to the energy consumption without using ACPI Device Power States. The visualization of the power consumption is shown in figure 3.2(b)). When comparing the power consumption with the utilization (figure 3.2(a)) the characteristics are mainly the same, because the power consumption graph is the upset and shifted graph of the utilization. The maximal power consumption is 100 W at the full utilization phase at the beginning, decreasing with the utilization to 80 W after 10 seconds where the zero percent utilization phase occurs.

With the Optimal Strategy the power consumption of this zero utilization phase can be decreased to 25 W (see figure 3.2(c)) when switching to ACPI Device Power State 3. This results in savings of 12.1 % compared to Simple Strategy. The switching is possible, because the duration of the zero utilization phase is 3 seconds, and the minimal duration for an efficient state change is $t_{\text{optimal}} = 2654\,\text{ms}$ (see equation 3.4).

$$\begin{aligned} t_{\text{optimal}} &= \frac{E_{\text{ACPI0-ACPI3}} + E_{\text{ACPI3-ACPI0}}}{P_{\text{ACPI0}} - P_{\text{ACPI3}}} + (t_{\text{ACPI0-ACPI3}} + t_{\text{ACPI3-ACPI0}}) \\ &= \frac{0.01\,\text{Wh}}{80\,\text{W} - 25\,\text{W}} + 2000\,\text{ms} = \frac{360000\,\text{Wms}}{55\,\text{W}} + 2000\,\text{ms} \\ &\approx 2654\,\text{ms} \end{aligned} \tag{3.4}$$

Using the Approach Strategy with a tolerance $\delta = 0.1$ phases with a utilization lower than 10 % will possibly be rearranged. In this example the energy consumption can be decreased in the 9 % utilization phase before the zero utilization phase when rearranging the utilization. The three steps with 9 % utilization result in one step with 27 % utilization, thus $t_{\text{load}} = 1\,\text{sec}$.

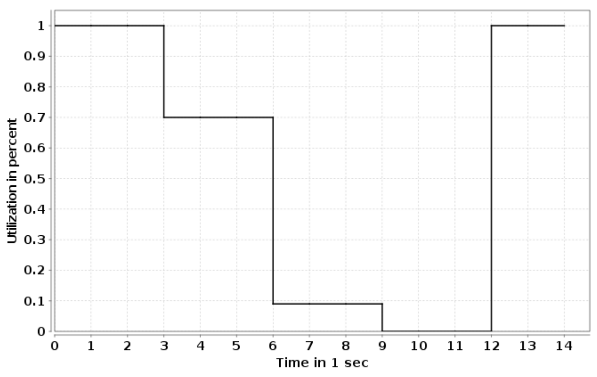
The rearranging is possible, because the duration of the phase under the specified level δ is 6 seconds, and the minimal duration for an efficient state change is $t_{\text{approach}} = 3654\,\text{ms}$ (see equation 3.5).

$$\begin{aligned} t_{\text{approach}} &= t_{\text{optimal}} * t_{\text{load}} \\ &\approx 2654\,\text{ms} + 1\,\text{sec} \\ &= 3654\,\text{ms} \end{aligned} \tag{3.5}$$

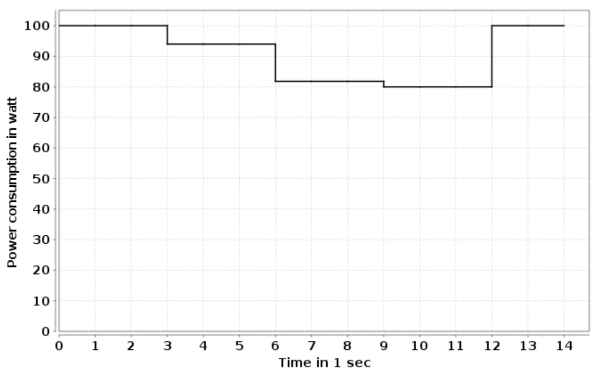
Figure 3.2(e) shows the recalculated utilization based on figure 3.2(a). The utilization phase lower than δ (from timestep 7 to times step 12) is summarized in a 27 % utilization phase at the end (timestep 12). The resulting power consumption is shown in figure 3.2(f) and is equivalent to the calculation with the Optimal Strategy and the rearranged utilization.

When comparing the Multiple State Strategy power consumption (see figure 3.2(d)) with the one calculated with Optimal Strategy multiple ACPI Device Power States are used. With this granular levels a energy saving of about 26.8 % for this example is possible, because every utilization change in the example results in a different ACPI Device Power State.

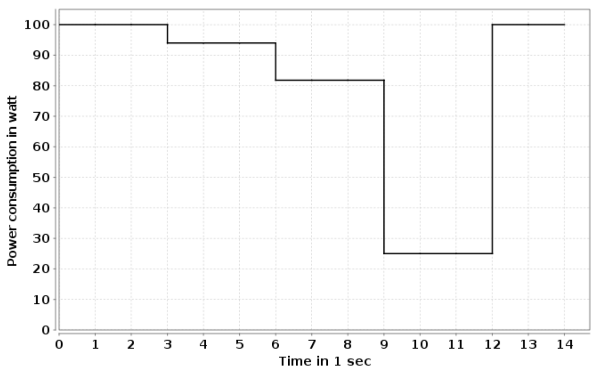
As seen in the previous section the energy saving increases with each strategy, in the case of the Multiple State Strategy a saving of 26.8 % is possible. But based on the input values a quantification of the energy saving potential is still not possible, but it comes clear that an energy saving potential is existent. Which the choice of the strategy the power saving varies: When using the Approach Strategy in the main case this strategy saves more energy as Optimal Strategy. But this strategy recalculates the utilization, thus the state changes of the Approach Strategy are hints for the developer of the algorithm to review possible code modifications to change the hardware utilization. The Optimal Strategy is an upper bound for energy saving because the hardware switches to power saving modes in every utilization phase where it is efficient possible. With Simple Strategy it is possible to estimate the power consumption of the hardware as granular as the input values (for the utilization and the state power consumption) is specified.



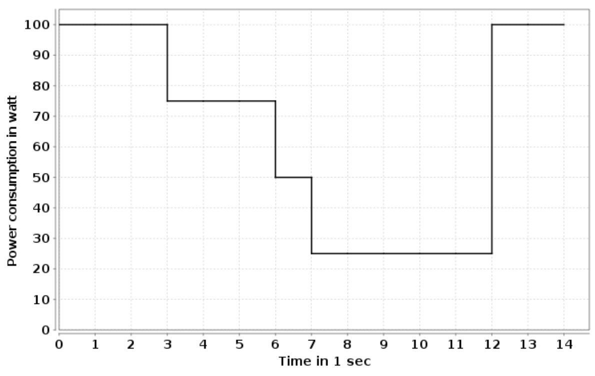
(a) Example utilization to compare the different strategies for power estimation.



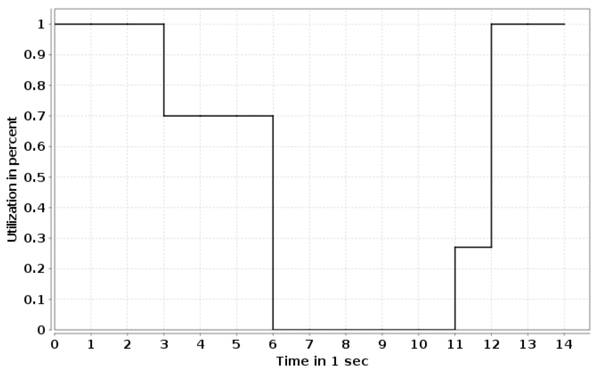
(b) Power consumption with Simple Strategy based on utilization from figure 3.2(a).



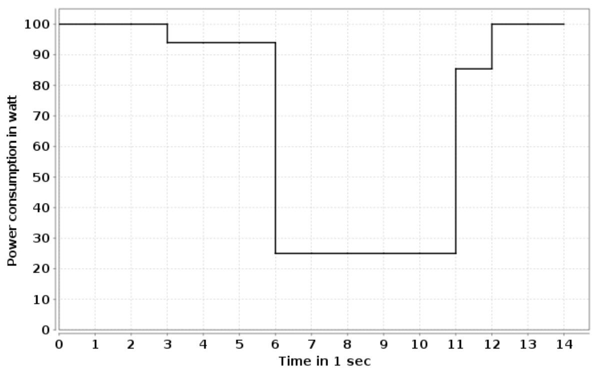
(c) Power consumption with Optimal Strategy based on utilization from figure 3.2(a).



(d) Power consumption with Multiple State Strategy based on utilization from figure 3.2(a).



(e) Rearrangement of utilization with ApproachStrategy based on figure 3.2(a).



(f) Power consumption with Approach Strategy based on utilization from figure 3.2(e).

4. Software Design

This chapter is a description of the software design to realize the power estimation model and the stresstest. Furthermore the requirements for the implemented software are discussed. After describing the related projects at Research Group Parallel and Distributed Systems to create and analyze traces, the concrete implementation model of the software components and its interfaces will be illustrated. Hence an overview of the testing concept and the external package dependencies is given.

4.1. Software environment

The project **HDPowerEstimation** has multiple related projects developed at the Research Group Parallel and Distributed Systems. These projects are used to create and visualize program traces. To create traces of MPI programs a wrapper has been implemented (**HDMPIwrapper**). This wrapper uses the **ResourcesUtilizationTracingLibrary** and the **PowerTracingLibrary** to capture utilization of components and the power consumption of nodes respectively. While the **ResourcesUtilizationTracingLibrary** queries the components utilization from the operating system, the **PowerTracingLibrary** uses an external power meter to measure the power consumption. The mentioned projects are building the tracing environment to generate traces. Each of these projects is implemented in the C programming language. A graphical overview of the tracing components is given in the next chapter in figure 5.1. To visualize the trace files a viewer has been implemented based on Jumpshot [1] named **Sunshot**. This viewer is implemented in Java. To support the traces the Java project **HDTraceFormat** has been implemented. This project includes the model of the trace data and reading, editing and writing support for the trace files. A detailed interaction description of the Java-based projects is given in the subsection "Interfaces" in this chapter.

4.2. Stresstest

The already mentioned stresstest is implemented as an independent MPI program in C++ programming language.

The stresstest is divided into six phases, in which a different set of components is stressed. In the first phase no component is stressed (idle phase). In the second phase the network and one CPU are stressed when sending a specified count of blocks to another MPI process located on another node. In the third phase the CPUs are stressed, first only one followed by both CPUs (or cores, based on the concrete hardware). The stressing of the CPUs is realized as an assignment in a loop with a specified count of iterations. In the fourth phase the main memory and one CPU are stressed allocating the whole memory and writing values into the allocated memory. The fifth phase stresses the disk, the main memory and the CPU. After allocating the whole memory the memory is written to disk multiple times. In the sixth and last phase all components are stressed performing the sum of the further mentioned phases.

Each component based stresstest is implemented using threads. Out of this, the parallelization of the work is possible (for example the last phase in the stresstest, which starts different parallel jobs).

[1] `http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm`, last checked on August 27, 2009

4.3. Implementation

*The requirements for the concrete implementation of **HDPowerEstimation** are mainly based on the previous described model and the interfaces. As first the interfaces are explained followed by the concrete implemented model. An overview about the testing concept finishes this section.*

4.3.1. Interfaces

The main goal is to integrate this project into the existing tracing environment. Figure 4.1 shows the relevant interfaces to the projects.

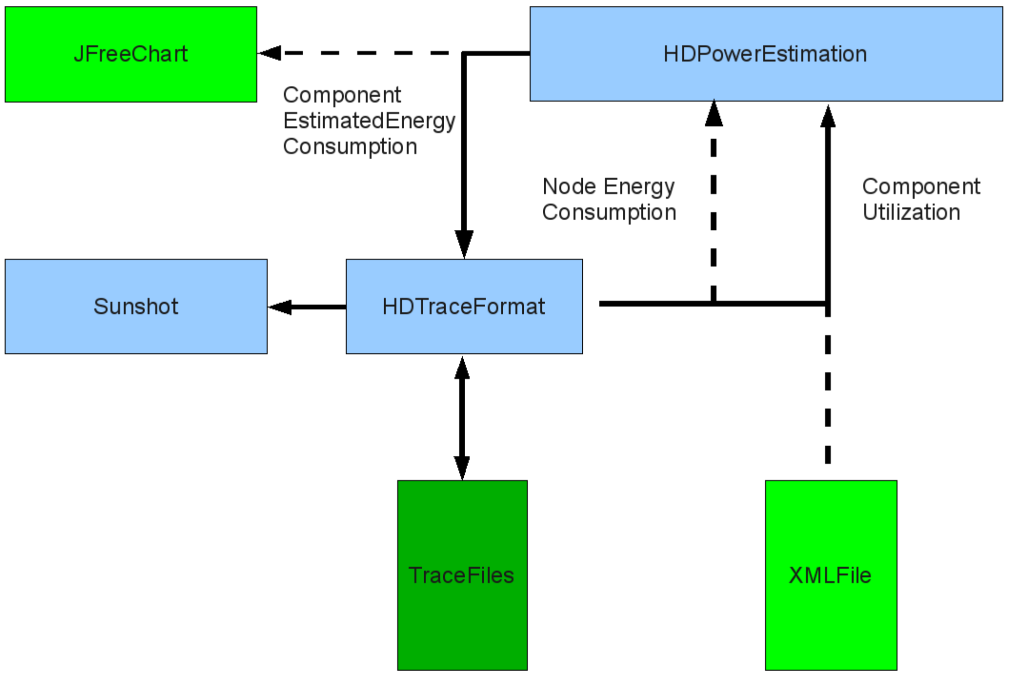


Figure 4.1.: *Interfaces of **HDPowerEstimation**. The component utilization can be read from an XML-File or from **HDTraceFormat**. The estimated power consumption can be visualized using **JFreeCharts** or using **Sunshot**.*

The main interface is the one to **HDTraceFormat**, because this project contains the capability to read and write traces. Reading the traces is important to get the component based utilization values. It is also possible to get the power consumption per node from the trace file, but this is not the regular use case. This capability has been implemented to verify the estimation and is used in the following chapter to evaluate the software. Writing the trace file is important to visualize the estimated component based energy consumption. Out of this, it is possible to analyze the MPI calls, the component utilization and the estimated energy with **Sunshot**.

But the project should also work stand-alone without any dependencies to the tracing environment. Out of this, the software has capabilities to read the input values specification (via XML or commandline) and visualize the estimated power consumption with **JFreeChart** [2]. **JFreeChart** has various types of diagrams to visualize different types of data sets. Because the utilization data is organized in steps, this array based data can be converted into a XYSeriesCollection. This can be visualized using a XYStepChart which represents the data in fixed steps (as given from the trace files). The resulting JChart has a user interface which allows the user to change the appearance of the plot. It is also possible to print and to save the plot (in PNG format).

[2] `http://www.jfree.org/jfreechart`, last checked on August 27, 2009

4.3.2. Model

To realize the previous discussed model some central datastructures are implemented in Java language. The class ACPIDevice contains all information specific for a node's component, especially the power schema, the current utilization, the current ACPI state, the current power consumption in watt and the total power consumption in Wh. The instances of ACPIDevice can be grouped using the abstract classes Node and Cluster. This grouping also contains the current and total power consumption with overhead for components that are not modeled (main board etc.) and potential overhead of the power supply. To distinct the utilization based power consumption a class Replay is implemented. Each Replay has a list of ReplayItems for the specific replay. The ReplayItem contains an instance of the class ReplayDevice which encapsulates the ACPIDevice and associated data like utilization and power consumption for each step. ReplayItem also contains an implementation of the interface PlayStrategy. For each step in the specific replay and the specific strategy the utilization of the ACPIDevice changes and stores the power consumption for the last step using class DeviceData. After replaying the class DeviceData contains the step based utilization and power consumption for a specific component. The DeviceData can be visualized using the implementation of the interface Visualizer (current implementations CommandLineVisualizer, LineChartVisualizer and StepChartVisualizer). It is also possible to export the specific Replay with the ReplayExporter into the trace format.

The model of the replay is visualized in figure 4.2 as an class diagram showing the interaction of the mentioned classes.

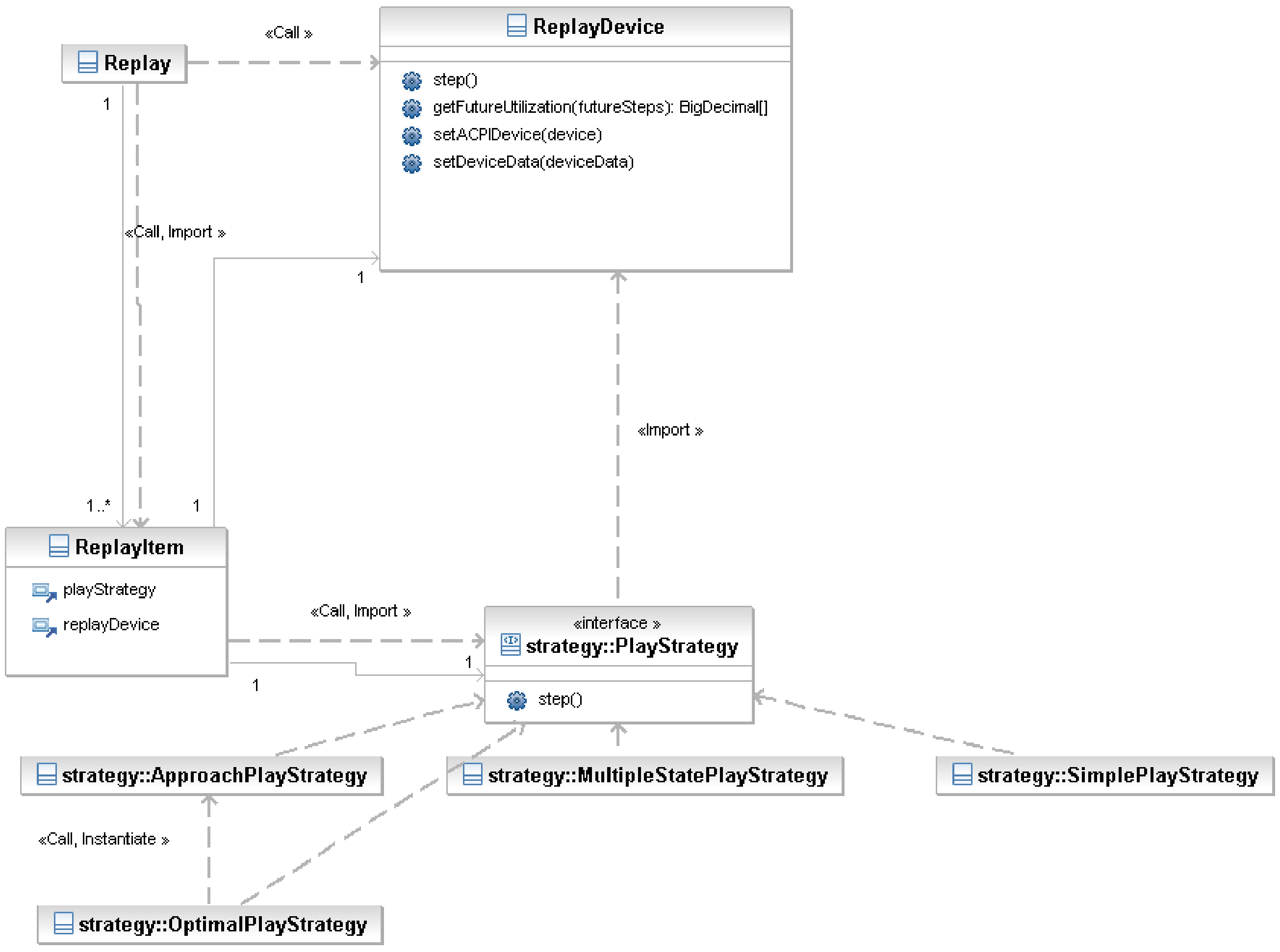

Replay
«Call »
ReplayDevice
step()
getFutureUtilization(futureSteps): BigDecimal[]
setACPIDevice(device)
setDeviceData(deviceData)
«Call, Import »
«Import »
1
1
1..*
1
ReplayItem
playStrategy
replayDevice
«Call, Import »
«interface »
strategy::PlayStrategy
step()
1
1
strategy::ApproachPlayStrategy
strategy::MultipleStatePlayStrategy
strategy::SimplePlayStrategy
«Call, Instantiate »
strategy::OptimalPlayStrategy

Figure 4.2.: *Class diagram of the replay package*

4.3.3. Tests and Guidance

The different packages of the project are tested using **JUnit** [3], a unit testing framework. The code coverage of the tests ensures the program functionality and shows the usage of each unit. To ensure the functionality after code changes a test suite running all tests is located in the test folder, which ought to be executed when finished editing. The test case PVSClusterTest shows the usage of **HDPowerEstimation**. The devices used to replay a trace file are adjusted to the cluster specification at the research group (see next chapter). To adjust a fine granular logging level (class specific) the logging functions are implemented using **log4j** [4].

[3]`http://www.junit.org`, last checked on August 27, 2009

[4]`http://logging.apache.org/log4j`, last checked on August 27, 2009

5. Assessing Energy Efficiency of Jacobi PDE Solver

In this chapter the energy efficiency of a specific parallel program on a cluster is assessed. Section 5.1 describes the education cluster at Research Group Parallel and Distributed Systems at the University of Heidelberg. In section 5.2 the concrete parallel program is described. The following sections contain the testing methodology and the conducted experiments. In the next chapter the energy saving potential is analyzed based on the assess of the energy consumption of different cluster components gained from this chapter.

5.1. Cluster Environment

Determining the power consumption of hardware components in a cluster environment is heavily dependent on the specific hardware, a composition of different sources for the data is needed. For this specific test setup the following components are modeled: CPU, NIC, main memory, disk, power supply and an overhead for the remaining components. A node at the PVS cluster (education cluster of Research Group Parallel and Distributed Systems) has 2 Intel Xeon Northwood CPU's (dual socket, single core) with 1024 MB RAM main memory divided into two 512 MB memory bars. Each node has a local P-ATA disk and a Gigabit network card using TCP/IP. The specific hardware of a PVS-Cluster node is ACPI aware. The concrete energy savings features are not in use, because these are too old to be effectively manageable. The processor it the only exception. The processor lowers its power consumption when in zero utilization phases using Advanced Power Management (which performs CPU Idle calls). Additionally the processor has a temperature control mechanism: If a specified temperature (between 63 to 70 degree Celsius) is exceeded, the processor uses NOOP's to lower the temperature and so the power consumption does.

The temperature control mechanism depends on the utilization, the uptime and the cooling capabilities. Because the tracing environment at its present state does not collect temperature values this characteristic is ignored at the moment.

A node of the PVS cluster is running Ubuntu 8.04 (SMP-Kernel 2.6.30) via NFS with mpich2 version 1.0.8 and PVFS (version 2.8.1) as parallel filesystem. If using a parallel filesystem such as PVFS as underlying filesystem further performance gains are possible. The parallel filesystem is spread over a variable count of disks, each administrated by a PVFS data server. As a result files are spread over multiple disks to increase the writing and reading performance.

The tracing can be realized by a software wrapper which intercepts a set of function calls and logs information on their invocation. Each function call is implemented to write some statistics (time of call, duration etc.) and to call the original function. In the case of a parallel MPI program usually the library calls the wrapped MPI library. For this, the program code does not need to be modified, it only has to be linked against the wrapper library instead of the original MPI library. When the instrumented program code is started, each process generates a trace file. These process-specific trace files can be merged to create a global communication schema. In the case of **HDMPIwrapper** additional information is logged to each trace file: The **ResourcesUtilizationTracingLibrary** logs the utilization of each loggable component 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. A graphical view on the tracing environment is given by figure 5.1.

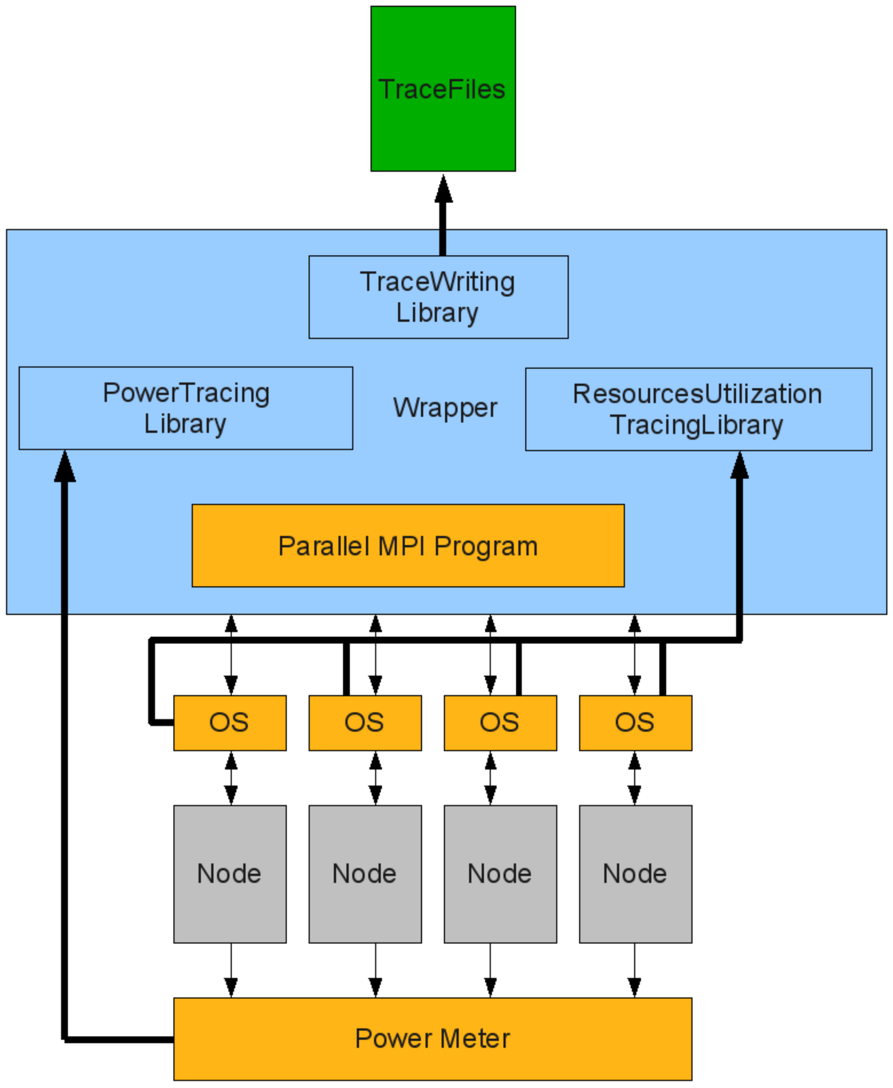


Figure 5.1.: *Tracing environment at the PVS cluster. Each node is attached to an external power meter. The node based power consumption and the component based utilization are included in the trace file generated by the MPI library wrapper.*

5.2. Description of Jacobi PDE Solver

As an example parallel program partdiff-par has been chosen. This program solves a PDE (partial differential equation) with an iterative method (Jacobi iteration method). The PDE uses a quadratic matrix with given boundary values as input and calculates the inner values through iteration (the calculating complexity depends on the boundary values). In each iteration for each matrix value a new value is calculated based on the surrounding matrix values of the previous iteration. To calculate the matrix values with a set of processes the matrix can be divided into almost equal sized blocks. Each parallel process (identified by a rank) works with one of these blocks. After each iteration (when every process finished its block) the last matrix line of process r has to be exchanged with the first matrix line of process $r+1$ to gain the input values for the next iteration. The data partitioning is shown in figure 5.2 for a 11×11 matrix and 3 processes. Each of the processes gets a 5×11 matrix and calculates a 3×9 sub matrix (because of the surrounding values). If one iteration finished, rank 0 sends line 4 to rank 1 and rank 1 sends line 5 to rank 0. Similarly lines 7 and 8 are exchanged.

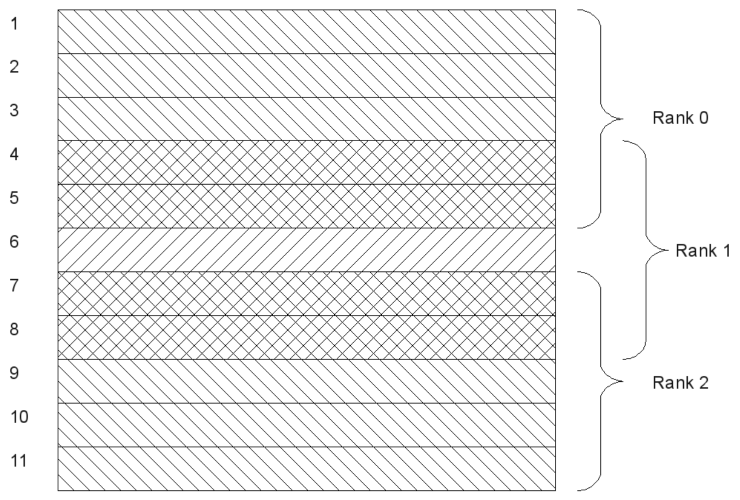


Figure 5.2.: *Partitioning of the matrix in partdiff-par. Each rank gets an equal part of the matrix (5 lines) while the border lines are saved in each neighbor rank. These are the lines that have to be exchanged after each iteration.*

The resulting overhead (exchanging the matrix lines, partitioning the matrix etc.) is the "parallelization overhead". This parallelization overhead affects the speedup of a parallel program. The speedup itself is limited by a fragment of not parallelizable actions in the code. For this reason the speedup in general is limited and in the best case the doubling of the calculating nodes divides the program run time by two (called optimal scaling). For partdiff-par the communication overhead is not constant (for each new process two more lines have to be exchanged per iteration) and there is no additional dynamic load balancing implemented. That is why the scaling of partdiff-par is not optimal.

As already mentioned the calculation complexity depends on the boundary values of the matrix. With the boundary values the program can be switched to calculation intensive and communication intensive mode respectively. Because the PVS cluster uses TCP/IP network, the communication results in frequent utilization of the CPU. With other network types (for example Myrinet [Lud08a]) the utilization of the CPU can be reduced.

To avoid data loss in case the program crashes (e.g. due to hardware failures) it is possible to use checkpointing. For this, after a fixed count of iterations every process writes its current working matrix to disk. With this checkpointing data the program can be restarted. The writing to disk uses an implementation of the MPI-IO specification [1] which allows to parallelize the IO.

To sum up the program partdiff-par has been chosen for experiments due to the following features:

- the calculation-communication-relation is flexible based on the input values for the boundary values of the matrix
- the component utilization depends on the checkpointing frequency and the calculation-communication-relation
- the consideration of a parallel filesystem is possible
- partdiff-par is a real application and no synthetic benchmark

With a set of different test setups and resulting trace files on the one hand the general relation between the test setup and the power consumption can be analyzed. On the other hand the measured power consumption can be compared with the estimated one. Some test setups are configured to fill the main memory with the working datastructures (the matrix) to avoid caching of the writing operations. The matrix size has to be specified in interlines, the resulting size in MB is calculated as follows:

$$(\text{interlines} * 8 + 9)^2 * 2 * 8/1024/1024 \tag{5.1}$$

The first bracket contains the count of elements in the matrix. This factor is multiplied with 2, because the Jacobi iterative calculation method is based on the values from the last iteration and so the matrix is stored twice. The factor 8 is the size of a double value on a 32-Bit architecture. With 950 interlines this results in approximately 880 MB for the two matrices. The operating system uses about 100 to 150 MB of the main memory and so the node main memory with 1024 MB is completely filled. When using the checkpointing mechanism to write the matrix to disk of course only one matrix is written (about 440 MB). When using multiple nodes the matrix is split into equal sized parts. This results in 1300 interlines for two nodes, 1600 interlines for three nodes and 1800 interlines for four nodes.

5.3. Methodology

The previously described parallel program partdiff-par has been traced with different setups: Multiple counts of nodes, multiple counts of MPI processes, multiple counts of PVFS servers and different program parameters (different count of iterations, different matrix size and different boundary values to increase the calculating part of the program).

An overview about the measured energy consumptions is given in table 5.1. In the table the count of interlines (to determine the matrix size), the PVFS nodes (nodes on which PVFS servers are running) and the MPI nodes (nodes on which MPI processes are running) identify one table row. The standard setup is to iterate five times and to start one process (PVFS and MPI respectively) per node. The checkpointing mechanism is activated to write every iteration the matrix to disk. This is unrealistic for real programs, but appropriate for tests.

For all mentioned setups the non-collective IO version is chosen. In this setup using collective and non-collective IO results in a similar performance and energy consumption. A table with all measured and estimated setups is located in the appendix (table A.3).

For some of the setups the power has been estimated to gain comparable values, the accuracy of the estimation is discussed in chapter 6. The program partdiff-par prints out the calculation time after finishing its job. The calculation time includes the checkpointing, but does not include the final output of the results. Based on the setup the calculation time and the overall run time differ significantly for short jobs, for longer jobs its nearly the same.

[1] `http://www.mpi-forum.org/docs/docs.html`, last checked on August 27, 2009

Table 5.1.: *Energy consumption and calculation time for different setups of* partdiff-par.

Setup			**Energy consumption in Wh**					**Calculation time in sec**
Interlines	**PVFS nodes**	**MPI nodes**	**node06**	**node07**	**node08**	**node09**	**Total**	
800	-	node06	0.249	-	-	-	0.249	5
800	node06	node06	2.125	-	-	-	2.125	44
800	node06-07	node06,node09	1.117	1.060	-	0.879	3.056	22
800	node06	node06-09	2.113	1.674	1.625	1.633	7.045	43
800	node06-07	node06-09	1.207	1.111	0.96	0.943	4.221	23
800	node06-09	node06-09	0.628	0.675	0.607	0.612	2.522	13
800	NFS	node06-09	2.785	2.743	2.861	3.055	11.443	73
950 (power estimated)	node06	node06	3.835	-	-	-	3.835	81
1100	node06-07	node06,node09	2.205	1.953	-	1.744	5.902	42
1400	node06	node06,node08-09	9.316	-	7.542	7.289	24.147	201
1600	node06	node06-09	13.252	10.338	10.402	10.710	44.703	285
1600	node06-09	node06-09	3.375	3.237	3.056	3.162	12.831	65
1600	-	node06-09	0.561	0.556	0.695	0.537	2.348	6
1600	node06-09	node06-09 ($\times 2$)	3.208	3.577	3.354	3.222	13.362	69
1600	-	node06-09 ($\times 2$)	0.534	0.511	0.495	0.653	2.192	4
1600 (50 iterations)	node06-09	node06-09	33.484	32.385	35.664	32.312	133.845	773
1600 (calc intensive)	node06-09	node06-09	5.168	5.589	5.266	5.179	21.202	112
1600 (calc intensive)	node06-09	node06-09 ($\times 2$)	4.618	4.432	4.518	4.908	18.476	93
1800 (power estimated)	node06-09	node06-09	6.503	7.07	7.08	6.989	27.641	146
1800 (power estimated)	node06-09	node06-09 ($\times 2$)	6.197	5.978	6.102	5.715	23.992	126

The first presumption from looking at table 5.1 is that the proportion between the energy consumption per node and the program run time is linear. The plot in figure 5.3 shows the proportion between the two quantities, it is nearly linear. But there is also variance between the energy consumption values which will be analyzed in the following with comparing different setups.

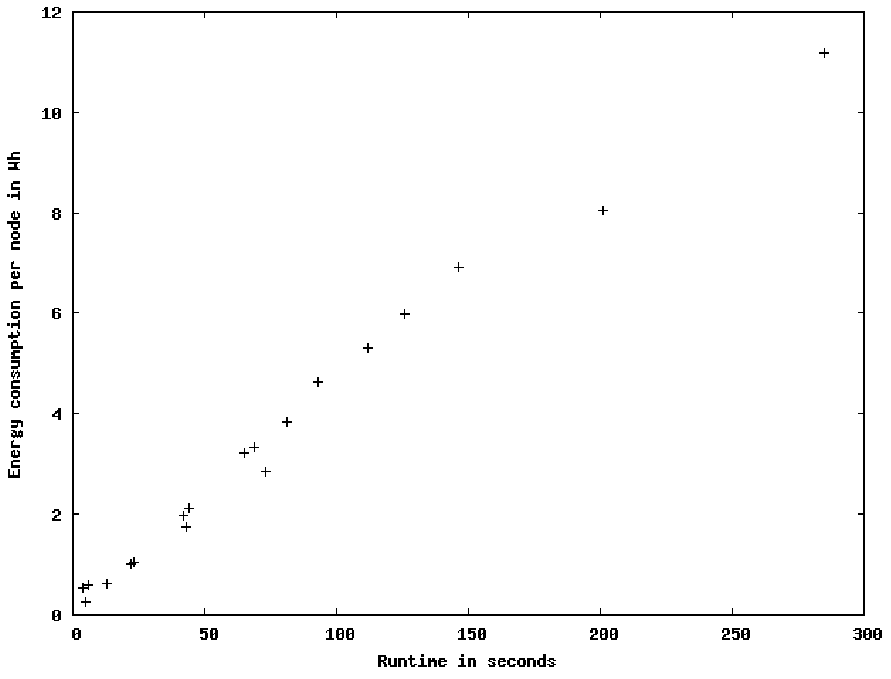


Figure 5.3.: *Energy consumption per node in relation to program run time in seconds.*

5.4. Experiments

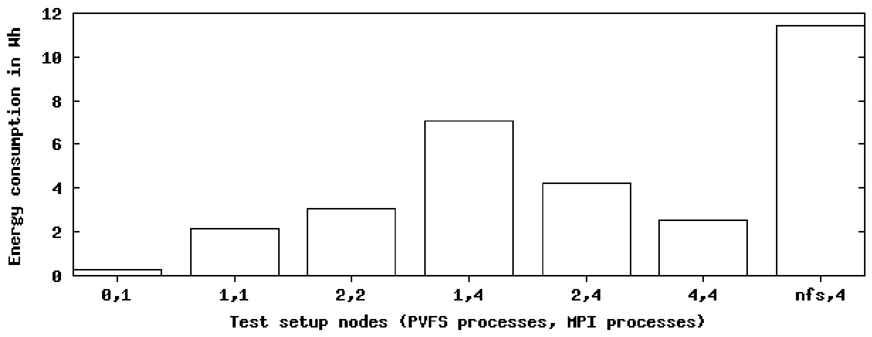
5.4.1. Fixed Problem Size

At first the power efficiency for solving a fixed sized problem is analyzed. The setup includes a different number of nodes, different file systems and various count of PVFS servers. The problem size is 800 interlines, so caching the write operations is possible (even if using a single node to solve the problem). Table 5.2 shows the different setups, the total energy consumption, the mean power consumption per node (total energy consumption / run time / number of nodes) and the run time. For one setup the IO (the checkpointing) is turned off, this results in a run time of 5 seconds and a energy consumption of 0.249 Wh for one node. When turning IO on (checkpointing every iteration) and using one PVFS server the resulting energy consumption is 2.125 Wh (850 %). The run time is 44 seconds (880 %) so the main cause for the energy consumption of the IO is the increased run time. But more interesting is the proportion between the count of nodes and the energy consumption. Figures 5.4(a) and 5.4(b) show the energy consumption and the run time for each test setup.

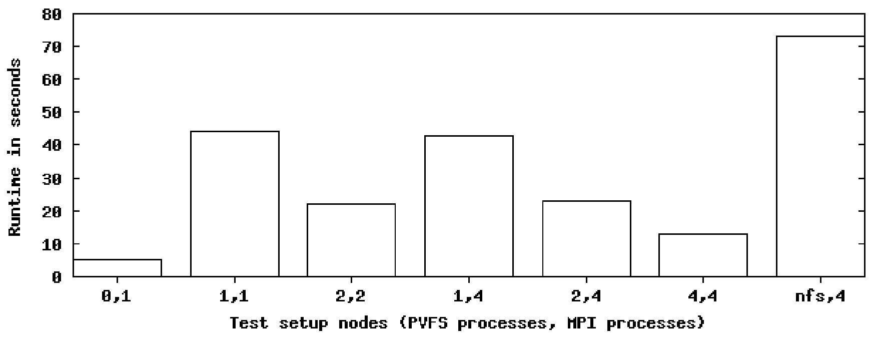
The setup with one node (one PVFS server, one MPI process) solves the problem in 44 seconds, while the setup with two MPI and PVFS processes takes only 22 seconds. This acceleration is mainly based on the fact that three nodes are used. On one node only one single PVFS server is started and on another node only one single MPI process. The usage of three nodes increases the energy consumption to 3.056 Wh. To sum up: The run time is divided by 2 while the energy consumption is multiplied with 1.5, hence the energy efficiency is better. When using four nodes with four MPI processes and PVFS servers respectively the run time is 13 seconds (about 70 % savings) while the total energy consumption is 2.522 Wh (about 19 % overhead). This is even more energy efficient than the configuration with two MPI processes.

Table 5.2.: *Results of tracing with fixed interlines 800. If not specified, one process per node (PVFS and MPI respectively). The mean power consumption is per node.*

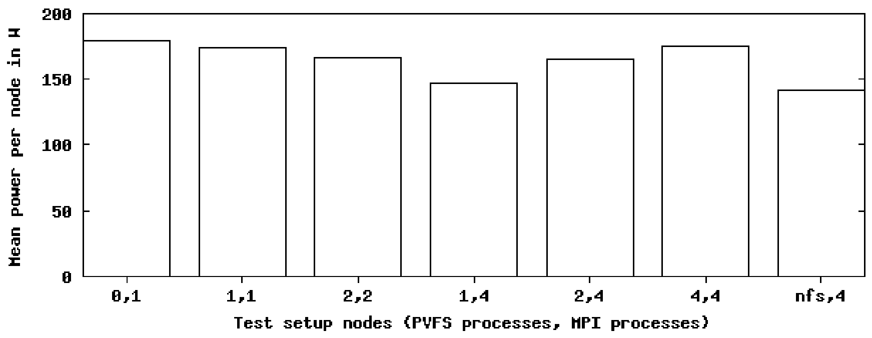
Setup			Consumption		Calculation time
Config	PVFS nodes	MPI nodes	Total energy	Mean power	
0,1	-	node06	0.249 Wh	179.29 W	5 sec
1,1	node06	node06	2.125 Wh	173.86 W	44 sec
2,2	node06-07	node06,node09	3.056 Wh	166.75 W	22 sec
1,4	node06	node06-09	7.045 Wh	147.42 W	43 sec
2,4	node06-07	node06-09	4.221 Wh	165.13 W	23 sec
4,4	node06-09	node06-09	2.522 Wh	174.74 W	13 sec
NFS,4	NFS	node06-09	11.443 Wh	141.09 W	73 sec



(a) Total energy consumption



(b) Run time



(c) Mean power consumption per node

Figure 5.4.: *Fixed problem size. For each setup the problem size is* 800 *interlines and the number of nodes and PVFS and MPI processes varies.*

If using less PVFS servers than MPI processes the energy consumption and the run time increases. For the example configuration of one PVFS server and four MPI processes this is mainly based on the increased waiting time: When using one PVFS server and MPI process on one node, both CPUs are utilized (one for calculation, one for communication) and the waiting times are minimized. Using less than four PVFS servers for four MPI processes results in under-utilization of one CPU on each node where no PVFS server is started. The relation of one PVFS server to multiple MPI nodes results in a network bottleneck, because all MPI processes have to send their matrix parts to the one node where the PVFS server is started. Because of the CPU intensive TCP/IP protocol this results in further utilization of the CPU of the PVFS server node.

Figure 5.4(c) shows the mean power consumption in watt for each setup. The setups with the highest total energy consumption have the lowest mean power consumption. When comparing the mean power consumption of the previously mentioned PVFS setup (about 150 W) with the power consumption of a node with one CPU active (about 170 W, see previous section) the nodes are fully under-utilized. And based on the high idle power consumption and the increased run time the total energy consumption increases.

For the NFS setup the mean power consumption is even lower (about 140 W), the run time even higher (both by reason of further increased waiting times) and the resulting total energy consumption increases to the maximum of this test set.

To sum up this test set: The energy efficiency is scaling down with the utilization of the nodes (especially the CPU's). Out of this, the solution of a fixed sized problem with parallelization increases the energy efficiency if the component's utilization is constant and the run time decreases. Hence, it is more energy efficient (and even more performant) to use multiple nodes with caching than using only one node.

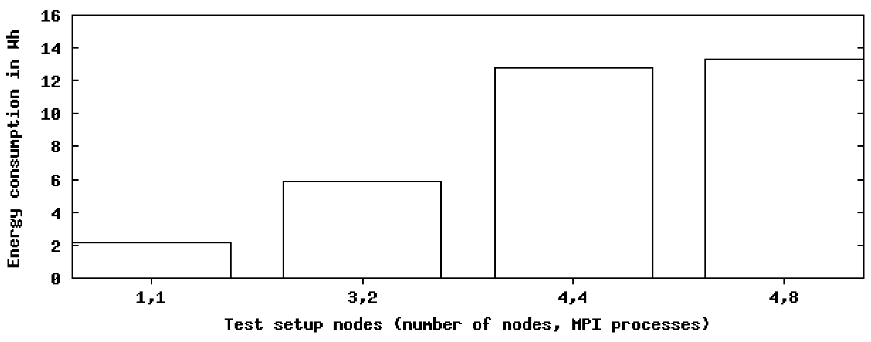
5.4.2. Increasing Problem Size with Number of Nodes

The following test set contains different setups with parameters to adjust the problem size, hence for every node added to the test setup the matrix size is increased. Using 800 interlines (about 630 MB) for one node this results in 1100 interlines for two nodes, 1400 interlines for three nodes and 1600 interlines for four nodes. On each node a PVFS server is started to handle the IO issues. To solve the 1200 MB problem with this configuration the running time is 42 seconds and the total energy consumption is 5.902 Wh (see figures 5.5(a) and 5.5(b)).

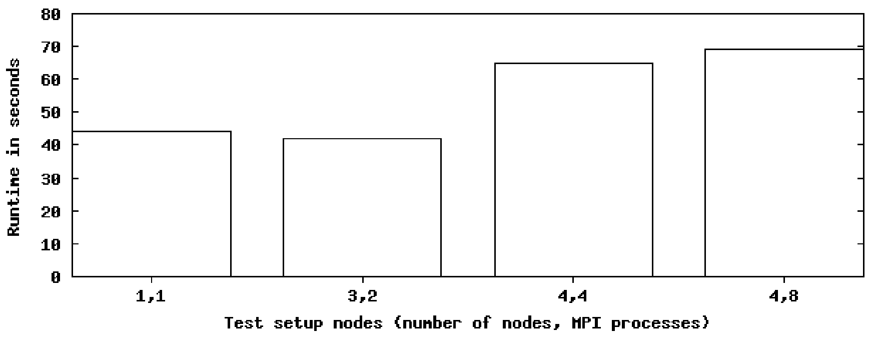
Table 5.3.: *Results of tracing with adjusted problem size. If not specified, one process per node (PVFS and MPI respectively) and 5 iterations for the Jacobi algorithm. The mean power consumption is per node.*

Setup				**Consumption**		**Calculation time**
Config	**Inter.**	**PVFS nodes**	**MPI nodes**	**Total energy**	**Mean power**	
1,1	800	node06	node06	2.125 Wh	173.86 W	44 sec
3,2	1100	node06-07	node06,node09	5.902 Wh	168.60 W	42 sec
4,4	1600	node06-09	node06-09	12.831 Wh	177.67 W	65 sec
4,8	1600	node06-09	node06-09 (×2)	13.362 Wh	174.31 W	69 sec

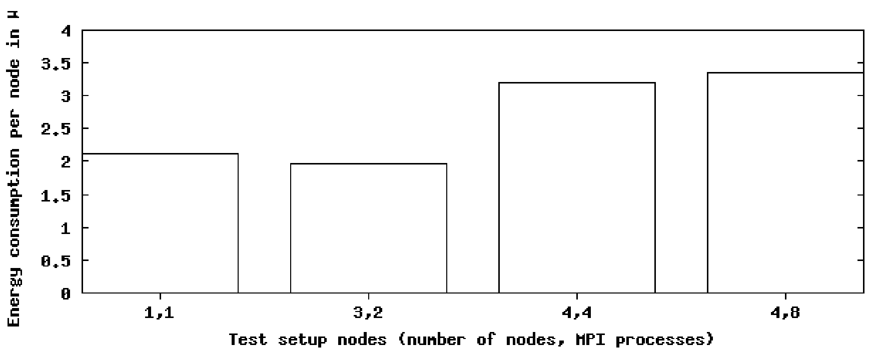
Comparing these values with the setup for one node these are savings of about 5 % for the run time and an increasing of the energy consumption of about 179 %. Solving the equivalent problem with the four node setup (with four MPI processes) the run time is 65 seconds with a energy consumption of 12.831 Wh. This is an increasing of the run time (about 47 %) as well as of the energy consumption (about 500 %).



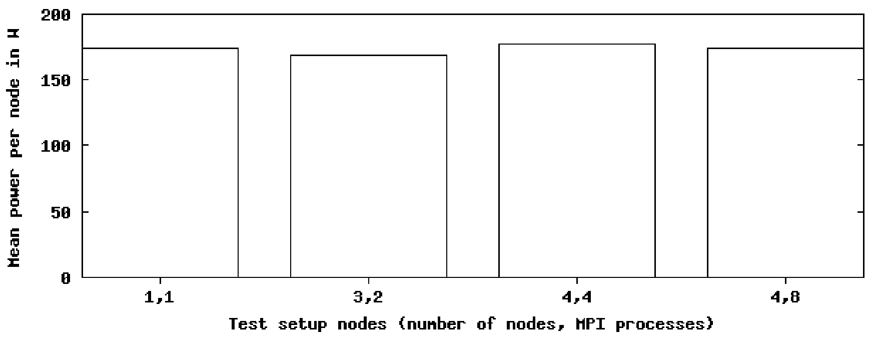
(a) Total energy consumption



(b) Run time



(c) Energy consumption per node



(d) Mean power consumption per node

Figure 5.5.: *Increasing problem size with number of nodes*

The parallelization of this problem results in a higher energy consumption per node (see figure 5.5(c)), but a more or less equal mean power consumption of about 175 W (see figure 5.5(d)). Out of this the parallelization does not further increase the node's mean utilization, but the communication overhead resulting in a higher power consumption by reason of the run time increase. Hence, the decreasing of the run time (for one node it would be $44\,\text{sec} * 4 = 176\,\text{sec}$ instead of 65 sec) results in an increasing of the energy consumption (for one node: $2.125\,\text{Wh} * 4 = 8.5\,\text{Wh}$ instead of 12.831 Wh). Using both CPUs mainly for calculating (setup with eight MPI processes) does consolidate this character. This configuration increases the run time and the energy consumption instead of showing a profit. This is based on the limited scaling: The parallelization overhead (communication, data partitioning etc.) consumes also calculating time and especially energy.

5.4.3. Communication Intensive vs. Calculation Intensive

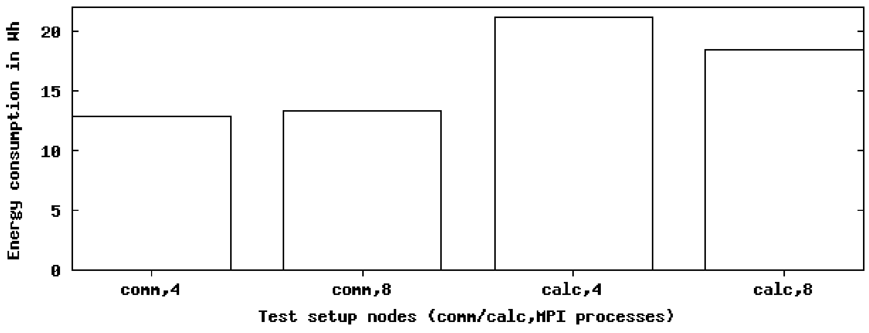
For the next test set the problem size and the number of nodes is constant, on each node a PVFS server is running. For each setup the boundary values of the input matrix are adjusted to control the communication / calculation proportion. When increasing the calculating time and of IO. Of course increasing the calculation time results in increasing the run time, because the same data has to be written to disk. The different setups are started with four and eight MPI processes per node respectively. When comparing the energy consumption and duration for each run (figures 5.6(a) and 5.6(b)) increasing the calculation time results in higher energy consumption and duration.

Table 5.4.: *Results of tracing with 1600 interlines. On four nodes four and eight MPI processes are started respectively (extract from table 5.1). The mean power consumption is per node.*

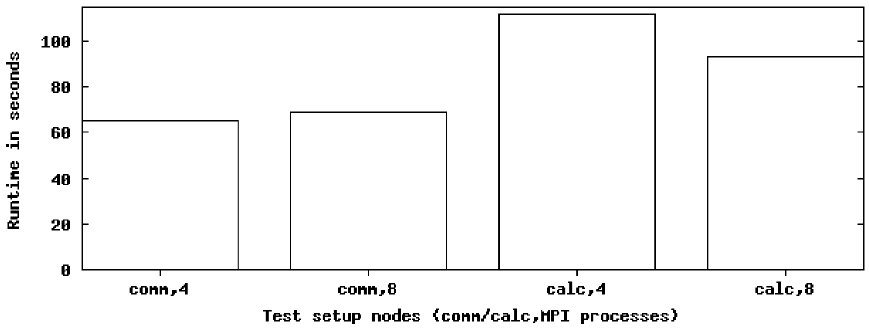
Setup			**Consumption**		**Calculating time**
Config	**MPI processes**	**Intensive**	**Total energy**	**Mean power**	
comm,4	4	communication	12.831 Wh	177.66 W	65 sec
comm,8	8	communication	13.362 Wh	174.29 W	69 sec
calc,4	4	calculation	21.202 Wh	170.37 W	112 sec
calc,8	8	calculation	18.476 Wh	178.80 W	93 sec

Starting eight MPI processes result in increasing run time and energy consumption for the communication intensive setup. This is mainly based on the parallelization overhead, which prevails the performance increase. For the calculation intensive program the increasing of the MPI processes results in performance increases because the second CPU can be used efficiently for calculation. The parallelization overhead from switching from four MPI processes to eight MPI processes is still the same, but the proportion of the total run time decreases.

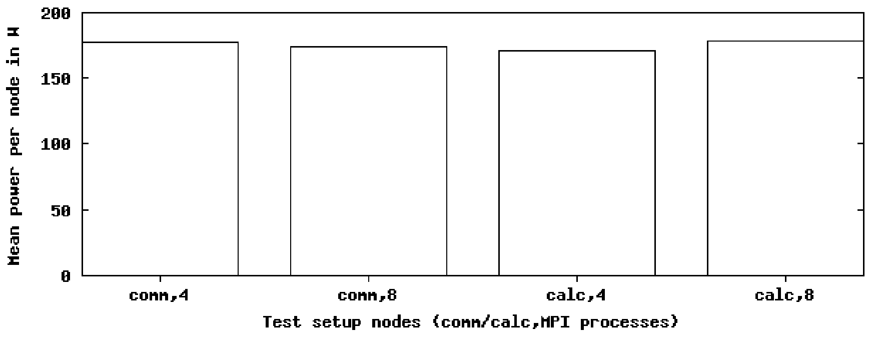
The mean power consumption of each node (figure 5.6(c)) only differs within a small range: The lower utilization of the calculation intensive setup with four MPI processes (because one CPU is most of the time idle) results in a lower mean power consumption. The setup with eight MPI processes increases the mean power consumption because both CPUs are used for calculation. For the communication intensive setup this is not the case, because the waiting times of the CPUs (based on the parallelization overhead), decreases the mean power consumption when changing from four to eight MPI processes.



(a) Total energy consumption



(b) Run time



(c) Mean power consumption per node

Figure 5.6.: *Communication intensive vs. calculation intensive. Test setup with 1600 interlines, four nodes and four PVFS-servers. The setups are calculation and communication intensive respectively, each with four and eight MPI processes*

5.4.4. Non-Cached I/O

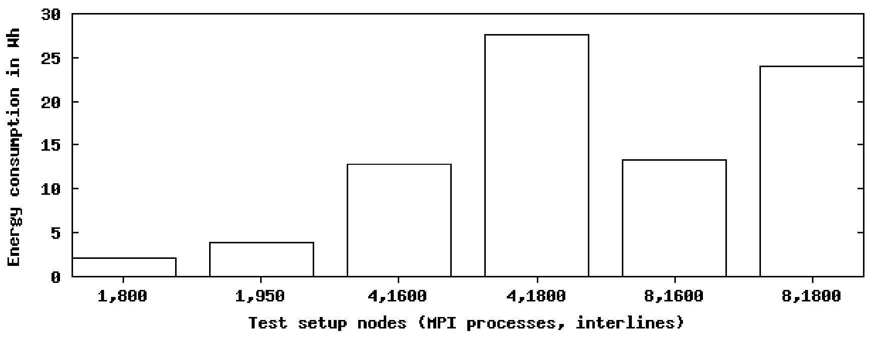
The following test setup is partially based on estimated energy consumption values. The different results are summarized in table 5.5. The goal of this setup is to compare different problem sizes (to avoid caching of the IO operations) based on different count of nodes and MPI processes respectively. With a matrix size of 800 interlines caching of the IO operations is possible for one node, with a matrix size of 950 interlines no caching optimization is possible (same with 1600 and 1800 interlines respectively for four nodes).

Table 5.5.: *Consumption and run time for different setups of* partdiff-par *with growing interlines and number of calculating nodes (no cache version). The energy consumption for* 950 *and* 1800 *interlines respectively is estimated. The mean power consumption is per node.*

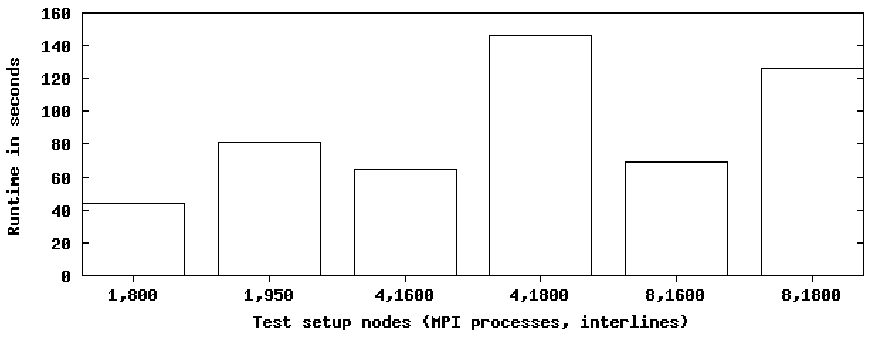
Setup				**Consumption**		**Calculation time**
Config	**Inter.**	**PVFS nodes**	**MPI nodes**	**Total energy**	**Mean power**	
1,800	800	node06	node06	2.125 Wh	173.86 W	44 sec
1,950	950	node06	node06	3.835 Wh	170.44 W	81 sec
4,1600	1600	node06-09	node06-09	12.831 Wh	177.67 W	65 sec
8,1600	1600	node06-09	node06-09 (×2)	13.362 Wh	174.31 W	69 sec
4,1800	1800	node06-09	node06-09	27.640 Wh	170.38 W	146 sec
8,1800	1800	node06-09	node06-09 (×2)	23.991 Wh	171.37 W	126 sec

While the problem size from 600 interlines to 850 interlines increases of about 13 % the energy consumption increases by about 80 % (see figure 5.7(a)) and the run time by about 84 % (see figure 5.7(b)). When increasing the problem size from 1600 interlines to 1800 interlines (27 %) the energy consumption is increased by 115 % for four processes and by 70 % for eight processes while the run time is increased by 125 % and 83 % respectively. The energy consumption in each case heavily depends on the run time. In the cached setup the change from four to eight processes doesn't decrease the run time in contrast to the non-cached setup. This mainly is based on the nearly equal mean power consumption per node in the non-cached setup (see figure 5.7(d)) and the decreased run time when using eight MPI processes.

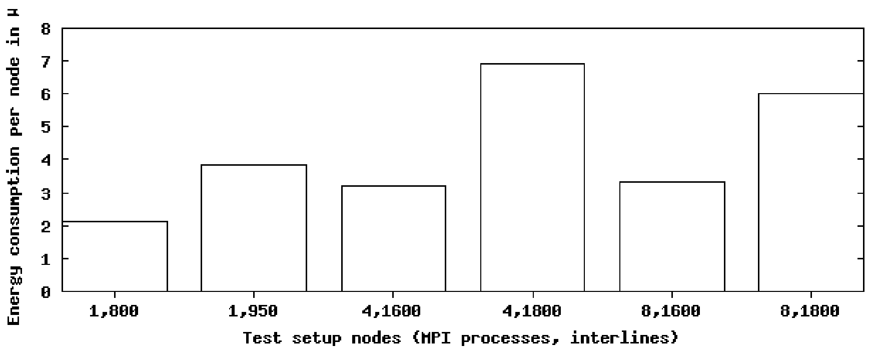
The energy consumption is mainly dependent on the program run time and the component utilization. The run time is mainly dependent on the program run and the scaling of the concrete algorithm. If the run time is increased based on component's idle times the total energy consumption is also increased, because the power consumption between the idle and utilized power consumption of a node is too small to reach energy savings with an increased run time and lower utilization. If comparing the mean power consumption of lower utilization runs with higher utilization runs the max measured difference is about 10 *W. When using efficient device power states to reduce power consumption the increasing of the run time results in much better energy efficiency.*



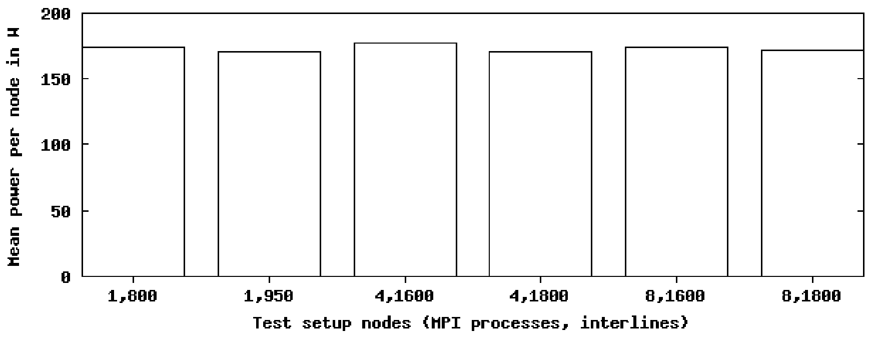
(a) Total energy consumption



(b) Run time



(c) Energy consumption per node



(d) Mean power consumption per node

Figure 5.7.: *Non-Cached I/O. Different configurations of* partdiff-par *with growing interlines and number of calculating nodes (no cache version).*

6. Evaluation

After assessing the measured energy consumption of different configurations of partdiff-par *with the PVS cluster hardware the model to estimate the component power consumption is evaluated. The goal of this chapter is to test the power estimation model described in the last chapters and to perform first experiments. The first step is to determine the component power consumption of a node at the specific cluster at Research Group Parallel and Distributed Systems at the University of Heidelberg. Using a power meter one of the cluster nodes is measured in different hardware configurations and utilization phases. Using the traces generated in the last chapter and the component consumptions as input for* ***HDPowerEstimation*** *it is possible to estimate and analyze the power consumption with different strategies and compare these values with the measured power consumption. The sections 6.4 and 6.5 contain first experiments with simulated energy aware hardware.*

6.1. Component Power Consumption

To determine the component power consumption as discussed in section 3.2 the zero utilization values the nodes' power consumption is measured with different hardware configurations. To estimate the power consumption of the utilized components the stresstest described in section 4.2 is used. The power consumption in the ACPI Device Power States 3 *is also estimated for each component in this section.*

To estimate the component's power consumption of the modeled components three assumptions have been made.

- At first, the ACPI power saving consumption of the processor (see section 5.1) is taken as zero utilization power consumption. This results in an increased gradient for the utilization / power consumption graph for this component, but the processor uses this power saving mode in real program runs therefore it has to be considered.
- At second the efficiency of the power supply has to be defined. For the specific power supply a constant efficiency of 65 % is assumed [AH03]. In general a utilization based efficiency is more precise and should be used if known.
- At third the overhead for the other node's components is assumed to be utilization independent.

For determining the zero utilization values the nodes' power consumption is measured with different hardware configurations. The resulting power consumptions are shown in table 6.3.

The stresstest trace is visualized in figure 6.1, each of the six phases is labeled with roman numerals:

I. Idle (no component is stressed)

II. NIC

III. CPUs

IV. Main memory

V. Disk

VI. All (all components are stressed)

Table 6.1.: *Node power consumption with different hardware without utilization. The power consumption includes all possible overheads because the power meter is connected between the power grid and the node's power supply.*

Hardware included	**Node power consumption**
2 CPUs	120.9 W
2 CPUs + main memory	137.8 W
2 CPUs + main memory + disk	144.6 W
2 CPUs + main memory + disk + NIC	145.8 W

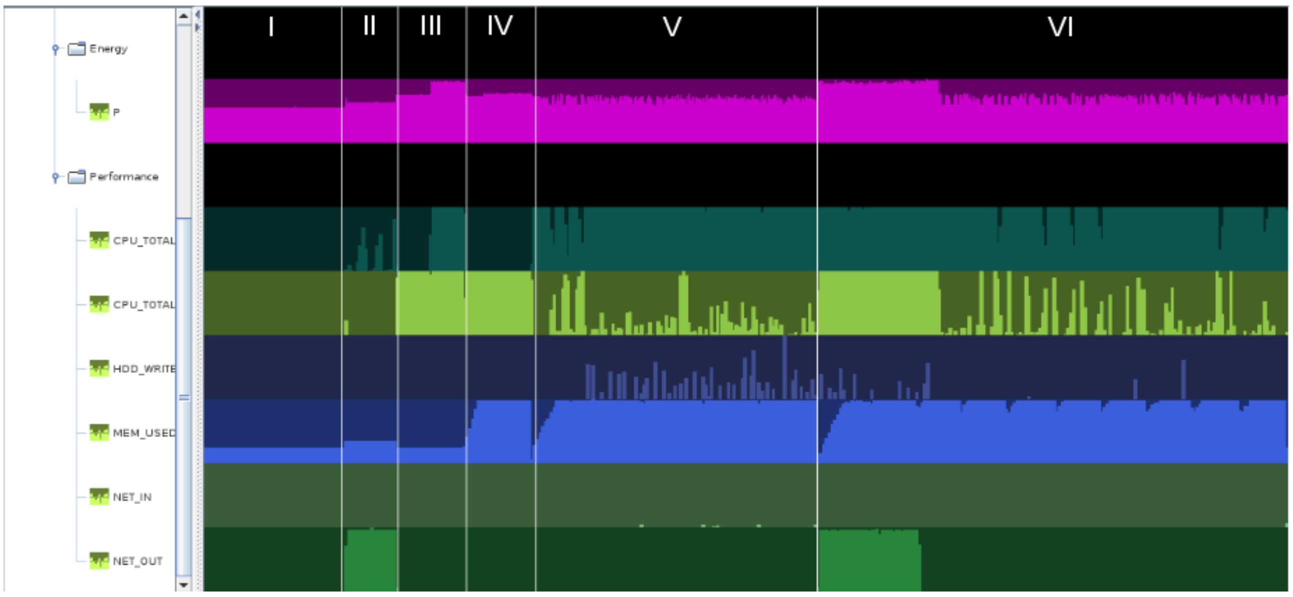


Figure 6.1.: *Trace visualized with* Sunshot *for one stressed node. The first time line visualizes the power consumption measured with the power meter in fixed timesteps of* 100 *milliseconds. The next six lines visualize the different components utilization in timesteps of* 1 *second. The different components are (from top to bottom): CPU_0, CPU_1, hard disk, memory usage, network card (incoming traffic) and network card (outgoing traffic).*

The full utilization power consumption of the disk and the network card are not measurable with the current measurement equipment, because the power consumption is lower than the one for the stressed CPU even though the CPU is utilized while stressing disk and NIC. This most likely is based on fast switching between high and low utilization for the CPU resulting in a lower power consumption than for full utilization. Out of this, the difference between the low and high utilization of the disk and the network card respectively is assumed to be small and is approximated.

Table 6.2.: *Node power consumption in different phases of the stresstest (see figure 6.1). These values include also the power supply overhead.*

Hardware stressed	Node power consumption
− (idle)	124 W
1 CPU	170 W
2 CPUs	216 W
1 CPU + NIC	142 W
1 CPU + main memory	174 W
1 CPU + main memory + disk	142 W
2 CPUs + main memory + disk + NIC (full utilization)	220 W

With the TDP value of the CPU's vendor data sheet [Fol04] all component's consumptions can be calculated based on the described model in section 3.2 using the measured differences for the power consumption (see tables 6.2 and 6.3).

Table 6.3.: *Component power consumption for zero and full utilization without power supply overhead.*

Hardware component	Power consumption	
	0 %	100 %
CPU	28.1 W	58 W
main memory	10.985 W	13.585 W
network card	0.78 W	2 W
disk	4.42 W	7.02 W
overhead (rest of system)	6.305 W	6.305 W

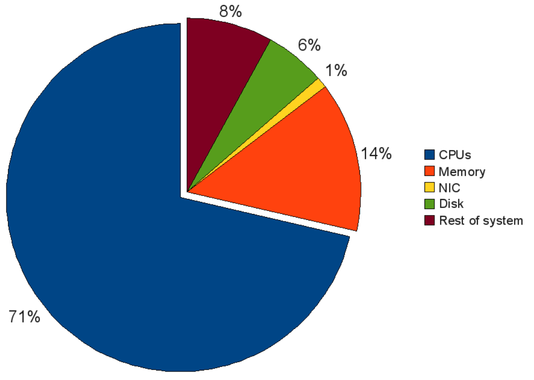
When comparing the components power consumption contingent (figure 6.2) the two CPUs consume about 71 % of the idle system and about 80 % of the utilized system.

For estimating the power consumption the ACPI Device Power State 3 consumptions must be specified. Table 6.4 shows the components' consumption in ACPI Device Power State 3 for the modeled components for one PVS cluster node (typical values from desktop and mobile components).

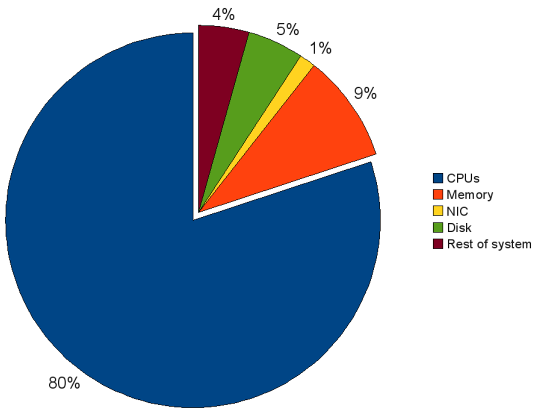
The ACPI values for the CPU are based on the values for a Intel Centrino Core Duo 1.8 GHz CPU (mobile edition). This CPU consumes about 53 W when fully utilized and 12 W, 7 W and 1 W in different ACPI Device Power States [Int09]. Hence, a value of 4 W for the power consumption in ACPI Device Power State 3 has been chosen. The duration for decreasing the ACPI Device Power States is taken from ACPI table of the operating system [1] which is 17 μs = 0.017 ms for the Centrino Duo processor.

The power consumption values of the main memory are estimated based on Moona's assumptions [MCH07]. The ACPI Device power state consumption for the disk are extracted from Hylic's work [HSRJ08], while the NIC values are extracted from Agarwal's work [AHC+09].

[1]File /proc/acpi/processor/CPU0/power on Ubuntu 8.04



(a) Idle power consumption, all components are utilized 0 %.



(b) Load power consumption, all components are utilized 100 %.

Figure 6.2.: *Visualization of the component power consumption (without power supply overhead) in percent.*

Table 6.4.: *Power schema (inclusive ACPI values) of a PVS cluster node*

Power schema entry	**CPU**	**Main memory**	**Disk**	**NIC**
$P_{ACPI0_{100\,\%}}$	58 W	13.585 W	7.02 W	2 W
$P_{ACPI0_{0\,\%}}$	28.1 W	10.985 W	4.42 W	0.78 W
P_{ACPI3}	4 W	0.1 W	2 W	0.2 W
$E_{ACPI0-ACPI3}$	$2.74 * 10^{-7}$ Wh	$1.1 * 10^{-8}$ Wh	0.001 Wh	$1 * 10^{-6}$ Wh
$E_{ACPI3-ACPI0}$	$2.74 * 10^{-7}$ Wh	$1.1 * 10^{-8}$ Wh	0.026 Wh	0.001 Wh
$t_{ACPI0-ACPI3}$	0.017 ms	0.006 ms	1000 ms	0.1 ms
$t_{ACPI3-ACPI0}$	0.017 ms	0.006 ms	4000 ms	0.1 ms

Missing values have been approximated using the max power consumption. For example the energy consumption for increasing the CPU ACPI Device Power State has been determined by multiplying the duration with the max power consumption of 58 W.

As seen in this section the component based power consumption is based on multiple assumptions. Most of these assumptions are based on the fact that measuring the node's power consumption is not precisely enough to determine the component power consumptions (as described in section 3.2). Hence, the measurement environment is adequate to distinct the node power consumption (for example of different program runs), but provides only limited insight into the component power consumption.

6.2. Model Verification

*The following section contains the evaluation of the implemented program **HDPowerEstimation**. For this purpose, the knowledge about the component specific power consumption gained in this chapter and the utilization of each component described from the different traces is used as input for the program. After discussing the accuracy of **HDPowerEstimation** the project is used to determine the energy saving potential for different traces when using ACPI aware hardware and energy efficient devices in general respectively.*

Table 6.5 shows the results with different strategies for different traces already discussed in the last sections. The deviance between the measured energy consumption values and the ones estimated with Simple Strategy (without usage of ACPI states) ranges from 1.2 % to 3.2 % for the specific traces. Based on the non determinism of the parallel program the measured energy consumption values have a deviance of about 1 % for multiple runs. Out of this, a deviance of about 3 % is acceptable. For short program traces (about a few seconds) the energy estimation deviance with Simple Strategy can reach about 20 %. This is based on the tracing methodology: The **PowerTracingLibrary** is started before the **ResourcesUtilizationTracingLibrary** and finished after it. This can result in different durations of the utilization and the power consumption of about one second, which is not recognizable for longer traces, but for shorter ones. The estimation of the energy consumption for all mentioned traces is appended to this work (see table A.3).

Table 6.5.: *Measured and estimated energy consumption for different setups of partdiff-par*

Setup	**Energy consumption in Wh**		
	Measured	Estimated	Deviance in %
stresstest	16.697	16.468	1.370
calculation intensive	21.202	20.571	2.977
communication intensive (50 iterations)	133.845	138.126	3.198
communication intensive	12.831	12.620	1.638
calculation intensive (×2)	18.476	18.251	1.187
communication intensive (×2)	13.362	13.557	1.453

The traces only contain a small number of nodes (maximum four nodes because of the limited number of power outlets of the power meter) and the maximum run time of a traced program is about 10 minutes. To complete the evaluation of the accuracy different traces from different parallel programs should be used with an increased run time and a multiple number of nodes. Also the estimation heavily depends on the input values for the component consumption. The quality of these values could be improved using special power meters at the component level (e.g. power meter plug between the power supply and the hard disk to determine the hard disk's power consumption).

In the following the energy saving potential of the different ACPI strategies is evaluated based on the estimated power consumption (with Simple Strategy).

Table 6.6.: *Estimated energy consumption with different strategies for different setups of* partdiff-par. *Each setup with* 5 *iterations and* 1600 *interlines (except the stresstest).*

Setup		**Energy consumption in Wh**			**Savings in %**	
Config	**Intensive**	**Simple**	**Optimal**	**Approach**	**Optimal**	**Approach**
stresstest	-	16.468	14.852	14.083	9.813	14.482
4,calc	calc	20.571	19.071	18.260	7.290	11.230
-	comm (50 iterations)	138.126	132.640	129.08	3.971	6.546
4,comm	comm	12.620	12.183	11.719	3.460	7.142
8,calc	calc (×2)	18.251	17.394	17.027	4.723	6.731
8,comm	comm (×2)	13.557	12.957	12.591	4.420	7.116

Based on these ACPI Device Power State values the Optimal Strategy reaches power savings of nearly 9.8 % for the stresstest trace (see table 6.6). This replay does not include changes in the component utilization, out of this the value of 9.8 % is an upper bound for power saving without changes to the parallel program.

With the Approach Strategy a decrease in energy consumption of about 14.5 % is possible. But this strategy assumes the parallel program to better utilize its component to get longer and more low utilization phases for using the more energy efficient ACPI states. In general, savings of 3.5 % to 9.8 % are possible for all traces using the Optimal Strategy, in case of the Approach Strategy savings of 6.5 % up to 14.5 % are possible (see figure 6.3).

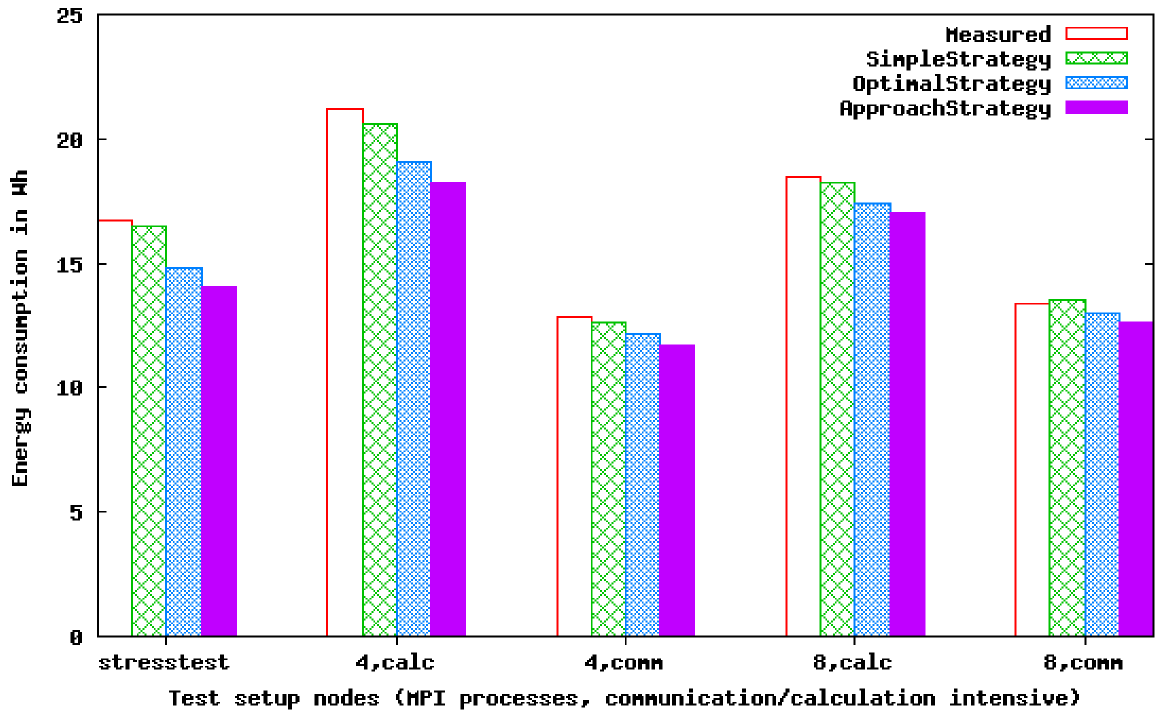


Figure 6.3.: *Energy consumption for different traces (see tables 6.5 and 6.6).*

The savings of about 9.8 % in case of the stresstest results from the high idle times, extra idle time is integrated in the program to determine the idle power consumption of the node. Also the CPU's are only partially utilized based on the specific stressed components. On the other hand the saving of only 3.5 % for the setup with four MPI processes results from the high utilization of the components (both CPUs are used: One for calculating, one for IO). Hence, the low utilization phases to use the ACPI states are rare. But this heavy IO load of this configuration is not typical for parallel programs. For the setup with the higher calculation proportion the energy consumption can be further decreased based on the idle times of the PVFS server. Using eight instead of four MPI processes results in a smaller saving because the idle times are decreased.

6.3. Strategies Applied to Program Runs

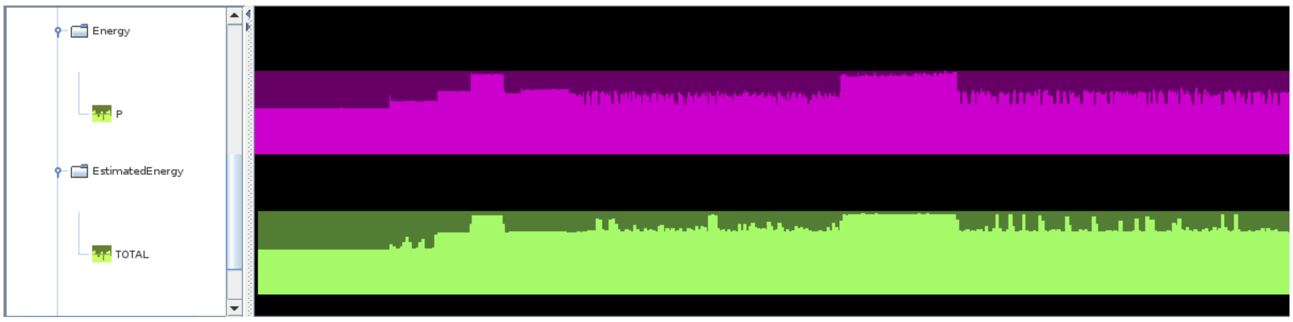


Figure 6.4.: *Total power consumption of measured value and the simulated one.*

In the following the estimated power consumption for each strategy is discussed in detail for the stresstest trace. Figure 6.4 visualizes the measured power consumption (with the power meter) and the estimated one (with Simple Strategy) for the stressed node with **Sunshot**. The characteristics of the time lines are similar, the utilization based power consumption model seems to reflect the measured power consumption. But there are also some phases of the estimated time line (e.g. the network stress phase) where the characteristics differ. This is based on multiple facts:

- the trace utilization in not as granular as the measured power consumption (interval size 1 sec vs. 100 ms)
- the stepsize is not exact 1 sec and 100 ms respectively for each step (the specification of **HD-TraceFormat** allows varying step sizes)
- this difference can be based on the effects of the power supply (see subsection 3.2).

In figure 6.5 the estimated power consumption for each component and the associated utilization is shown. The sum of the components and the node overhead is multiplied with the power supply overhead and the resulting total power consumption is shown in figure 6.4. The power consumption of each component is only depends on the utilization and the component specific utilization / power consumption relation (as defined in the model).

Figure 6.6 visualizes the power consumption in watt for each component of the stressed node with **HDPowerEstimation**. After the zero utilization phase of about 20 seconds both CPUs frequently change their utilization and the resulting power consumption. The power consumption for this component has its minimum at 26 W and the maximum at 58 W. The power consumption of the both CPUs affect the node power consumption, because the other components consume max. 3 W more based on utilization.

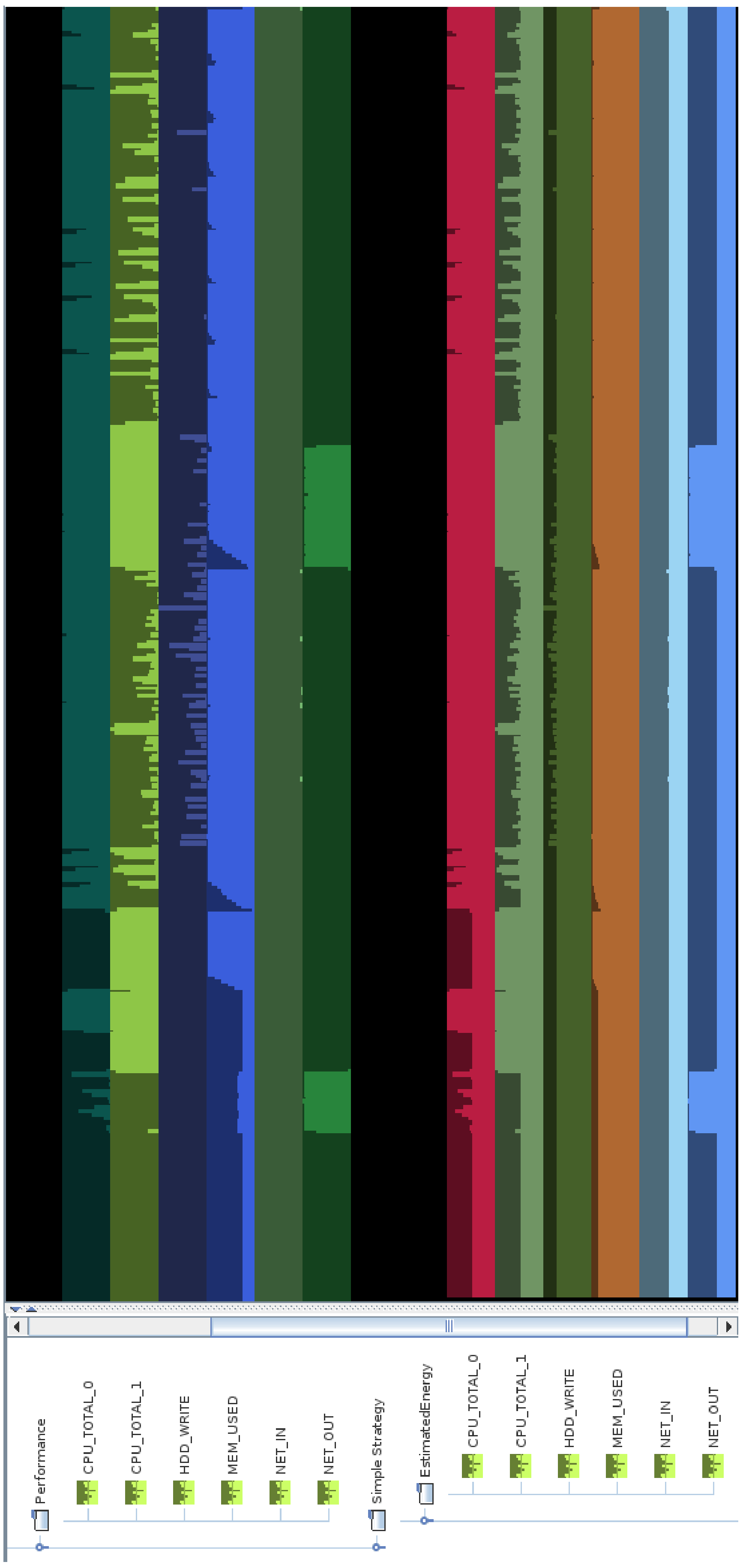


Figure 6.5.: *Component based power consumption and measured utilization.*

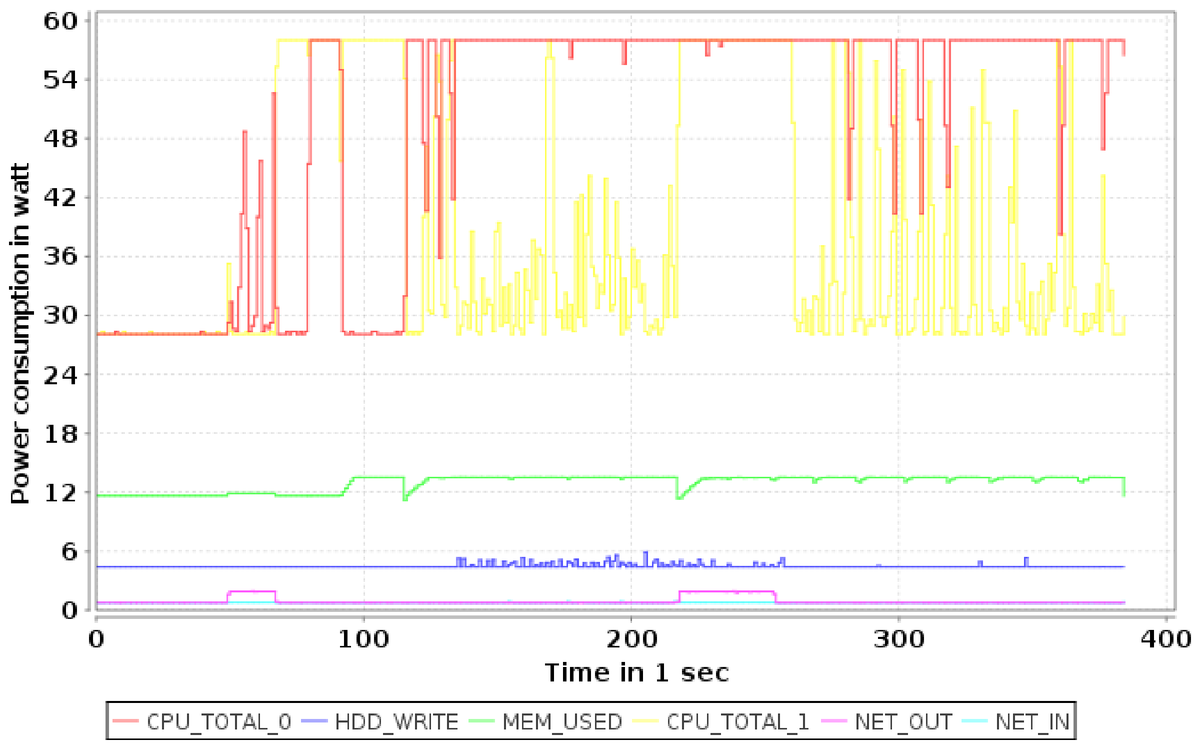


Figure 6.6.: *Stresstest with Simple Strategy*

When using the Optimal Strategy (see figure 6.7) for the estimation of the power consumption the frequency of the power consumption changes is much higher as for the Simple Strategy because of the additional ACPI state changes. The power consumption of the CPU now varies between 4 W (power consumption in ACPI Device Power State 3) and 58 W (power consumption in ACPI Device Power State 0 with full utilization). Also the other components are using the ACPI Device Power States, especially the networking interfaces and the disk (based on their low total utilization). Only the main memory is not using any states, because it has no zero utilization phases.

The table 6.7 shows the detailed energy saving for each component, the count of state changes (to ACPI Device Power State 3 and back to ACPI Device Power State 0) and the sleeping time (time in ACPI Device Power State 3).

Table 6.7.: *ACPI overview for stresstest trace with Optimal Strategy, run time 385 sec*

Component	**E**$_{simple}$	**E**$_{optimal}$	**Savings**	**State changes**	**Time in ACPI state 3**
CPU0	5.312 Wh	4.844 Wh	8.8 %	11	70 sec
CPU1	4.160 Wh	3.524 Wh	15.3 %	39	95 sec
Disk	0.480 Wh	0.414 Wh	13.9 %	1	130 sec
NIC (in)	0.100 Wh	0.100 Wh	0 %	0	0 sec
NIC (out)	0.084 Wh	0.084 Wh	0 %	0	0 sec
Main memory	1.389 Wh	1.389 Wh	0 %	0	0 sec
Total	11.524 Wh	10.354 Wh	10.2 %	51	295 sec

The percental savings are higher than the ones from table 6.6, because in this table only the ACPI aware components are considered, the node and power supply overhead is ignored (because both are constant for this setup). Even if the components are sleeping about 15 % of the run time, the saving is only about 10 %. This is based on the small (estimated) energy saving potential of the disk and the NIC, because of the small difference between the idle power consumption and the power consumption

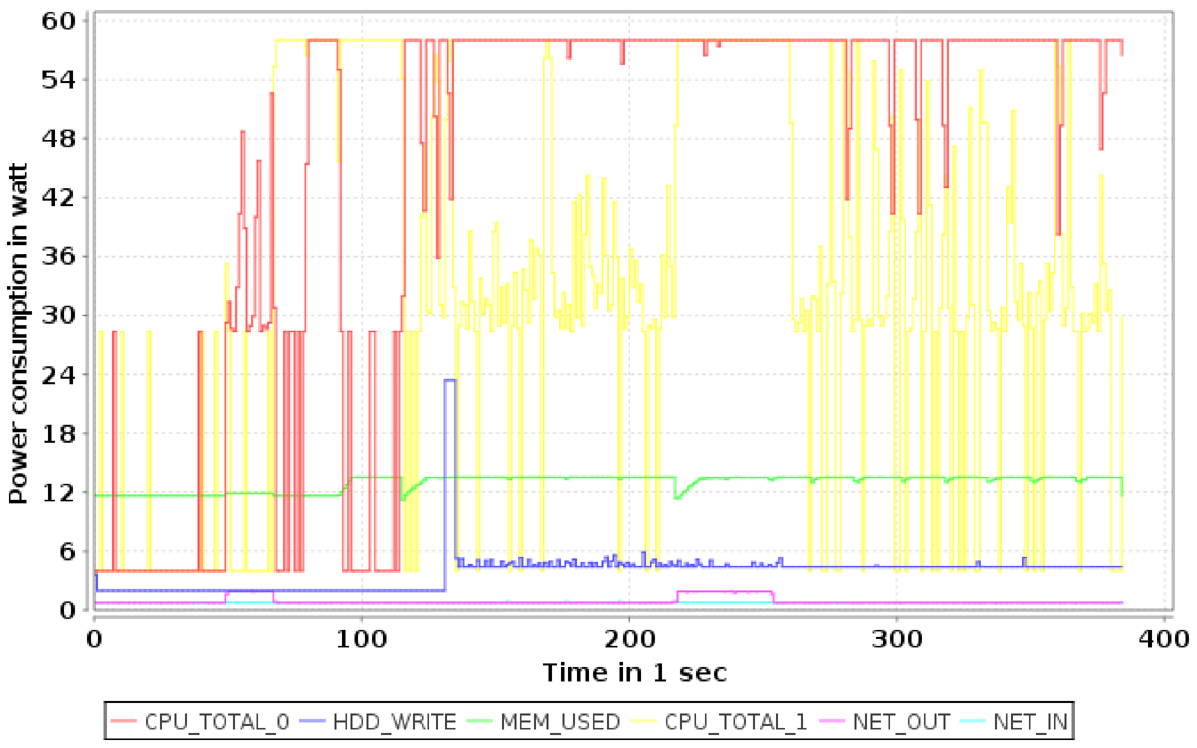


Figure 6.7.: *Stresstest with Optimal Strategy*

in ACPI Device Power State 3 compared with the CPU.

If reducing the disk energy consumption about 13 % lots of energy can be saved in a cluster with a much higher count of disks. The greatest savings can be gained with the second CPU on the stressed node, a saving of about 15 % is possible. The energy saved by this device (0.636 Wh) is about 50 % of the total saved energy (1.17 Wh), for both CPUs about 90 %. As expected the component with the highest power consumption has also the highest energy saving potential. Based on the small state changes duration the CPU can change frequently and without much run time overhead and power loss to energy efficient operation modes. The main memory and the network card are not changing their ACPI Device Power State, because the utilization of these components is not zero.

Replaying the stresstest trace with Approach Strategy results in further decreasing of the power consumption (see figure 6.8). If the utilization of a component is below the specified level of 10 % the Approach Strategy tries to use efficient ACPI Device Power State changes. This results in smoothing the power consumption of each component. But if the utilization is greater than 10 % (for one timestep) the component changes back to ACPI Device Power State 0. In the worst case this step is followed by another low utilization phase and the component switches back to ACPI Device Power State 3. This strategy has to be improved to avoid this peak utilization phases.

The table 6.8 shows the detailed energy savings for each component and the count of state changes for Approach Strategy. This strategy decreases the power saving by about 4 % (compared to the Optimal Strategy) while the count of state changes is increased by nearly 130 % (based on the recalculation of the component's utilization). Especially for the second CPU the number of state changes is increased (based on the frequent utilization changes). Also the main memory performs one state change based on a short low utilization period of nearly one second, because the changes for the main memory are assumed to be neither energy nor time consuming (compared to the e.g. the disk). The sleeping time for the NIC is increased (especially for the incoming traffic) because the mean utilization for these devices is low and these phases result in sleeping times with the Approach Strategy. But based on the low peak power consumption for this component this behavior does not decrease the total energy consumption significantly, even if the total percental sleeping time is now about 50 %. With this strategy the energy savings of the CPU's are also nearly 90 % of the total savings.

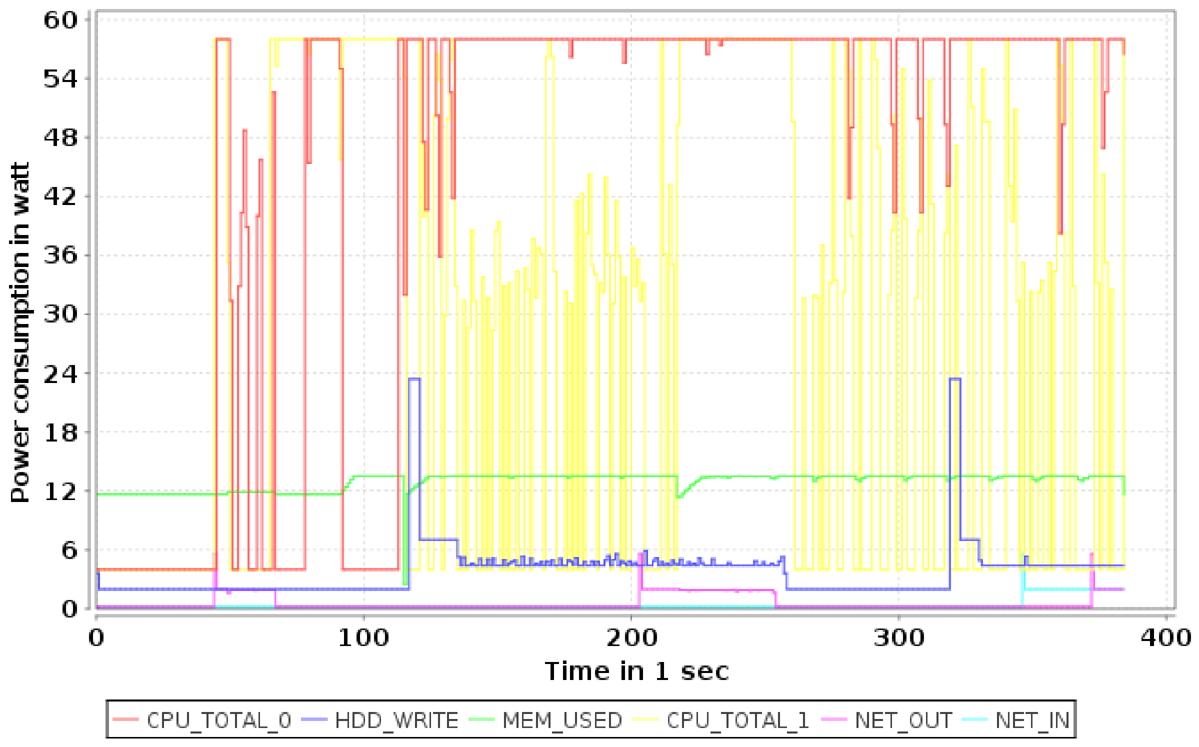


Figure 6.8.: *Stresstest with Approach Strategy*

Table 6.8.: *ACPI overview for stresstest trace with Approach Strategy, run time 385 sec*

Component	**E**$_{simple}$	**E**$_{approach}$	**Savings**	**State changes**	**Time in ACPI state 3**
CPU0	5.312 Wh	4.800 Wh	9.7 %	17	156 sec
CPU1	4.160 Wh	3.103 Wh	25.4 %	93	265 sec
Disk	0.480 Wh	0.418 Wh	13.0 %	3	307 sec
NIC (out)	0.100 Wh	0.066 Wh	33.7 %	3	298 sec
NIC (in)	0.084 Wh	0.042 Wh	49.9 %	1	346 sec
Main memory	1.389 Wh	1.386 Wh	0.2 %	1	1 sec
Total	11.524 Wh	9.815 Wh	14.8 %	118	1373 sec

Figure 6.9 visualizes the power consumption with Approach Strategy for the stresstest with **Sunshot**, especially the estimated energy states for each component are visualized. The energy states for Simple Strategy are not interesting, because each component is in ACPI Device State 0 (working state, blue colored fractions of the time line). The ACPI Device State 3 (sleeping state) is colored green. Having a look at the component "HDD_WRITE" the power consumption increases where the state change from state 3 to state 0 occurs (because the disk has to be accelerated) at about 120 seconds in the time line. The state change is marked as pink fragment in the state time line. After the state change the disk power consumption is the same as in Simple Strategy. But before this change the disk (in state 3 and sleeping) consumes only a fractional part of the power consumption with Simple Strategy. The change in general is possible because the disk is not utilized for the first 120 seconds. For the other components these state change fragments do not occur because of the durations for the changes are supposed to be smaller (5 seconds for the disk and 0.03 milliseconds for the CPU).

*In this section has been shown that the accuracy of **HDPowerEstimation** ranges between* 1 *and* 4 *% for traces with a duration longer than a few seconds. With this estimation of the power consumption of a cluster node it is possible to determine the power consumption of a cluster based on a specific program trace.*

On the basis of the stresstest trace the further strategies Optimal Strategy *and* Approach Strategy *have been evaluated. With* Optimal Strategy *savings on device level of about* 10 *% are possible. This means if an algorithm exist to switch unused devices off without any resulting time overhead, the max savings are* 10 *% for this specific trace.*

These savings are on device level, therefore the total savings can be much higher: If the power supply produces a percental overhead, the total overhead is also reduced. This total node power consumption has also impact on the air conditioning system, because the saved energy does not produce any waste heat which has to be dissipated.

The experiments in this subsection include basic analysis of energy saving potential based specific configurations of the parallel program partdiff-par. *These configurations result in higher utilization of the hardware and performance increments. The alternative is to use configuration resulting in lower utilization of hardware to get the possibility of switching to energy efficient sleeping modes.*

Further the power saving potential of energy efficient devices in general (without sleeping phases) is estimated to motivate further research in the development of energy efficient devices.

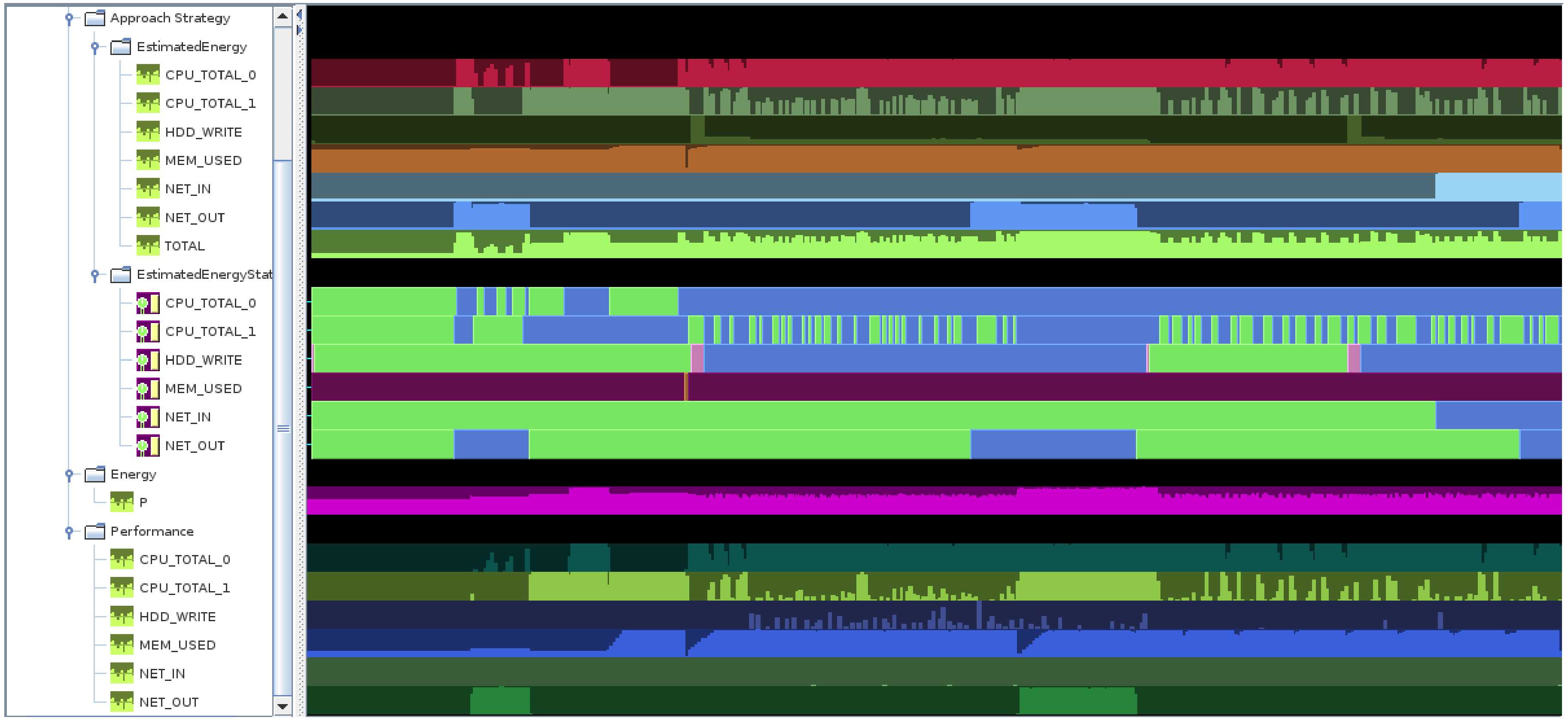


Figure 6.9.: *Estimated component based power consumption (Approach Strategy) of stresstest visualized with Sunshot*

6.4. Energy Efficient Sleeping

In this test set multiple program configurations without checkpointing and a various count of MPI processes are traced. Based on these traces the proportion between calculating and sleeping devices is analyzed. Table 6.9 shows the estimated energy consumption and the measured run time for different setups. The program partdiff-par is configured with a various count of iterations, each of these runs with 4 and 8 MPI processes respectively.

Table 6.9.: *Estimated energy consumption for different setups on four cluster nodes. All setups do not use checkpointing. The count of interlines is* 1600 *and constant for each setup, only the count of iterations and MPI processes is different. The savings are based on the simple consumption.*

Setup		**Energy consumption in Wh**			**Savings in %**		**Time in sec**
Iter.	**MPI processes**	**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	66.253	66.218	6.1	6.2	334

The different setups with 5 iterations have only a run time of 6 and 4 seconds respectively, based on the count of MPI processes. With 8 MPI processes the run time is 2 seconds smaller and the estimated energy consumption with Simple Strategy is about 0.05 Wh smaller, mainly based on the shorter run time because all 8 CPUs are used for calculating and less CPU idle times appear. If comparing the estimated energy consumption with Optimal Strategy, the saving is decreased to 0.008 Wh compared to the setup with 4 MPI processes, but the run time is still constant for both setups. The 4 idle CPUs of the 4 MPI process setup can switch to ACPI Device State 3 and sleep, while the other 4 CPUs are calculating. The parallelization overhead of the 8 MPI process setup, especially the additional communication overhead that increases the CPU utilization and the energy savings of the 4 MPI process setup decreases the total power consumption difference between the two setups. Using the Approach Strategy the estimated energy consumption for 4 MPI processes is even lower than for 8 MPI processes. But this strategy affects the run time of the program, this saving can only be reached if the hardware utilization changes (e.g. by editing the program code or by heuristics implemented in the operating system).

Figure 6.10 visualizes the estimated energy consumption (with and without sleeping) of each setup.

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 prevails the performance growth. Using 100 and 200 iterations respectively for the Jacobi method, 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. But the total power saving decreases.

With 500 iterations the performance growth prevails the overhead of the parallelization and using eight CPUs for calculating decreases the total energy consumption in comparison to the setup with four CPUs.

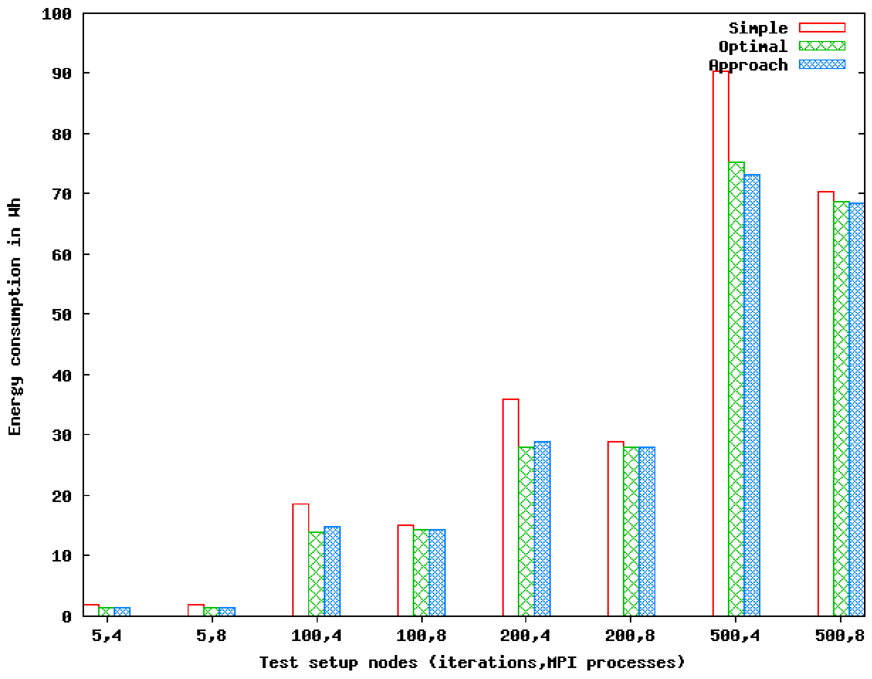


Figure 6.10.: *Estimated energy consumption in Wh for different setups. All setups use four nodes. The count of interlines is* 1600 *and constant for each setup, the checkpointing mechanism is not used. Only the count of iterations and MPI processes is different. This figure is based on the values in table 6.9.*

6.5. Efficient Devices

The project **HDPowerEstimation** allows to change the specifications of the underlying devices to compare different specifications for a trace. The resulting energy consumption (and the resulting costs) can be monetary evaluated based on the hardware acquisition costs. In the following test set multiple traces have been replayed with the Simple Strategy to estimate the power consumption without energy aware hardware.

This test set contains the specification for each node device as explained in earlier section. As already discussed the power consumption for zero utilization is relative high for each device. When this overhead could be technical reduced by the hardware vendors the power efficiency can be increased. Figure 6.11 shows the proportion of utilization and power consumption for one CPU at a PVS cluster node with the current specification and with an optimal specification. The CPU with the optimal specification would consume zero watt if not utilized, the mentioned overhead would be omitted.

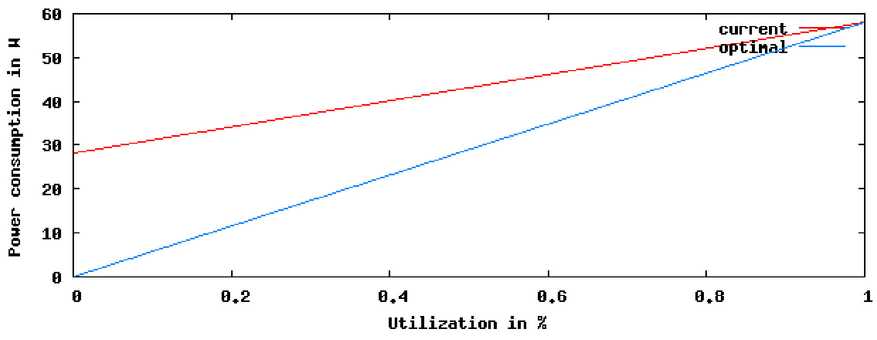


Figure 6.11.: *Utilization / power consumption graph of the PVS cluster node CPU.*

Table 6.10 shows the estimated power consumption with the Simple Strategy and with efficient devices (CPUs, NIC, main memory and hard disk).

Table 6.10.: *Estimated power consumption with Simple Strategy and efficient devices.*

Setup				**Energy consumption in Wh**		
Config	**Inter.**	**Iter.**	**Intensive**	**Simple Strategy**	**Efficient Devices**	**Savings in %**
stresstest	-	-	stresstest	16.468	11.554	29.83
4,calc	1600	5	calc	20.571	14.662	28.72
-	1600	50	comm	138.126	104.300	24.49
4,comm	1600	5	comm	12.620	9.1503	27.49
8,calc	1600	5	calc (×2)	18.251	13.973	23.46
8,comm	1600	5	comm (×2)	13.557	9.9154	26.85
1 PVFS	800	5	comm, 1 PVFS	6.2007	2.2676	63.430
no IO	1600	50	comm (×2), no IO	70.383	63.826	9.3157

For the stresstest trace savings of about 30 % are estimated, 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 for four nodes) potential savings of only 9 % have been estimated. For the trace with lots of idle times (using one PVFS server and four MPI processes on four nodes with checkpointing) the estimated saving with efficient devices is about 63 %.

Of course the savings for well-utilized hardware are much smaller than for idle hardware. Hence, the savings are heavily dependent on the concrete traced program and the configuration.

In the current configuration the calculation includes a percental overhead of the power supply of 35 %. It is possible to reduce this overhead, for example by a switched-mode power supply. When using this power efficient supply the percental overhead can be reduced to about 5 % (device specific). Hence, the resulting power consumption can be simply calculated for specific power supplies (see equation 6.1).

$$E_{\text{node}_{5\,\%}} = E_{\text{node}_{35\,\%}} * \frac{105}{135} \tag{6.1}$$

Also the percental energy savings can be calculated using equation 6.2.

$$1 - \frac{E_{\text{node}_{5\,\%}}}{E_{\text{node}_{35\,\%}}} = 1 - \frac{E_{\text{node}_{35\,\%}} * \frac{105}{135}}{E_{\text{node}_{35\,\%}}} = 1 - \frac{105}{135} \approx 22.2\,\% \tag{6.2}$$

Figure 6.12 visualizes the energy consumptions with

- efficient power supply (SMPS)
- efficient devices and
- efficient devices and the efficient power supply.

The saving when using a SMPS are constant for each setup (22 %) of the energy consumption estimated with Simple Strategy and efficient devices respectively.

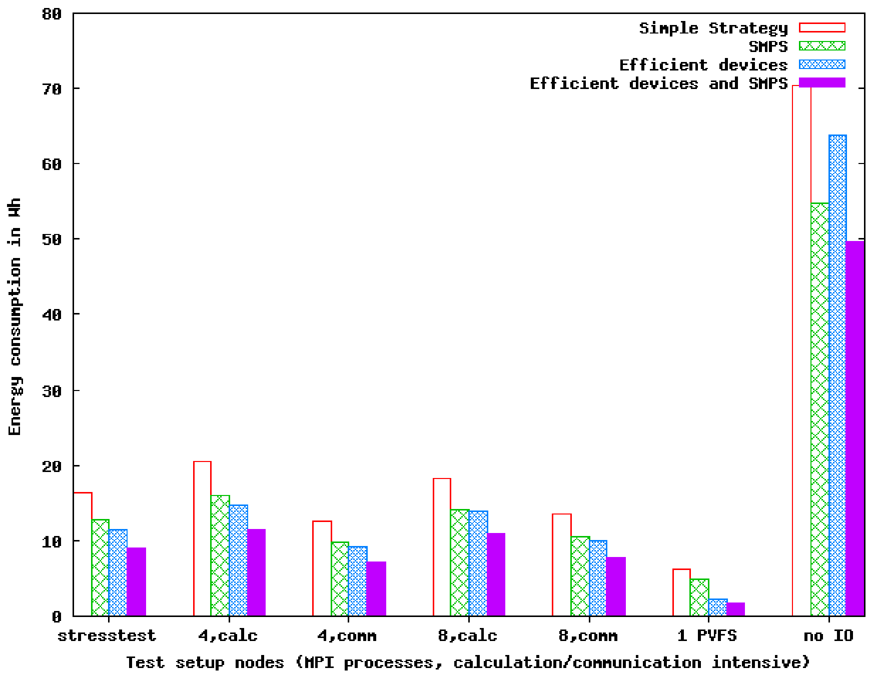


Figure 6.12.: *Energy consumption with efficient devices and SMPS*

*The experiments in this section have shown the analysis capabilities of **HDPowerEstimation**. Using multiple traces and device power specifications the resulting energy consumption can be estimated. Especially the usage of energy efficient devices (with a low zero utilization power consumption) has great energy saving potential for clusters in general. For specific application traces and hardware specifications savings of about* 60 % *have been calculated. Also has been shown that the configuration of the parallel environment (number of MPI processes) has influence on the power efficiency. Based on parallelization overheads and/or hardware bottlenecks the performance growth by using other configurations can result in increased power efficiency, because idle devices can switch to energy saving modes.*

7. Summary, Conclusion and Future Work

Computer hardware in general and cluster hardware in particular consume plenty of energy. The power consumption mainly depends on the hardware utilization as shown in this thesis. Because cluster components usually do not have low energy consuming modes (as opposed to mobile devices) the power consumption in idle phases is nearly as much as in high utilization phases.

This work contains on overview of hardware with low energy consuming modes (energy aware hardware) to evaluate possible hardware for an energy efficient cluster.

Based on this overview a model has been designed to estimate the power consumption of (energy aware) hardware components. With this model it is possible to calculate the power consumption based on program traces containing information about the components utilization. Therefore the components utilization based power consumption has to be specified. The power consumption can be estimated using different strategies to simulate low power consuming modes. These results have to be treated as upper bounds for the energy savings, because the analysis of the trace file allows to anticipate the future utilization of each component.

Using the education cluster of the Research Group Parallel and Distributed Systems at the University of Heidelberg the implemented software has been evaluated. In this context the utilization dependent power consumption for different components of a cluster node are identified using a stresstest. Further multiple parallel program setups have been traced to analyze the node based power consumption with an external power meter.

Experiments showed that power consumption mainly depends on the configuration of the parallel program and the resulting run time. Program configurations not using all hardware resources in an efficient way result in increased run times and decreased power efficiency. Especially program configurations resulting in hardware bottlenecks (e.g. using one PVFS server for four MPI processes on four nodes results in a network bottleneck) are very energy inefficient.

Further the parallelization overhead has an influence on the energy efficiency by increasing the run time without additional calculation. Setup comparisons between different number of MPI processes and nodes show differences in energy efficiency: Even with decreased run time using more MPI processes and nodes, the power efficiency might not increase due to the specific proportion of the parallelization overhead.

On the other hand increasing the proportion of the calculation work results in a lower proportion of the parallelization overhead leading to a more energy efficient behavior for this specific parallel algorithm.

It has been shown that the difference of the power consumption per node depending on the utilization is about 10 W (comparing a well-utilized and an under-utilized setup). This variation is small compared to the difference of the run times of the setups. For this reason the well-utilized setup consumes more energy by utilizing the components better but the lower run time leads to a better energy efficiency.

Simulating energy aware hardware with low power consuming states (with the same performance in working mode) noticeable increases the mentioned difference of about 10 W. The upper bounds for power savings are between 0 % and 28 % depending on the hardware utilization. If less idle times appear for a parallel program, less power can be saved using energy aware hardware. The average power saving for all traced setups is about 11 % using an optimal strategy for device sleeping.

The energy aware hardware in setups with low utilized hardware results in better power savings compared to setups with high utilized hardware and decreased run time. In this setup the comparison between four and eight MPI processes on four nodes results in a better energy efficiency of the four MPI processes setup. Due to greater difference between idle and utilized power consumption the calculation proportion has to be further increased to reach a more energy efficient setup with eight

MPI processes.

Further the power consumption using energy efficient components has been estimated. For this experiment the power consumption of zero utilized components is assumed to be zero. This is a best case behavior. The resulting energy consumption are the energy costs to run the application on the cluster. With this efficient components savings between 9 % and 58 % have been estimated, also for this setup mainly based on the components utilization. The average power saving for all traces is about 32 %.

Conclusion and Future Work

The estimated power consumption mainly depends on the utilization granularity and the input power consumption for zero and full utilization. The process to determine the component's power consumption can be improved using a component based power meter. Also the power estimation strategies can be improved: The Approach Strategy can be revised to not wake up on every peak utilization value. The mechanism can be adjusted to use the mean utilization of the future steps as input and not only the next step's utilization. If the trace file is extended to contain information about the power states of the components (e.g. using PowerTop) a strategy has to be implemented that estimates the power consumption with this further information (for real energy aware hardware).

The evaluation chapter has been shown that it is possible to estimate and monetary evaluate the power consumption of different hardware and applications in general. This is especially interesting for clusters designed for more than one application type. Because these clusters are designed to be high performant for a rich variety of applications, this results in idle times of components for each specific application. In this idle phases the hardware can be switched off to save energy. These energy savings can be calculated against the energy aware hardware acquisition costs.

This work shows that there is a hidden energy saving potential for energy aware hardware even in high performance computing. This potential should motivate research in mechanism to control the energy aware hardware in high performance clusters. An idea is to analyze program traces of applications for idle phases. Based on this analysis heuristics can be developed which try to anticipate the future utilization of the devices. This can be done for special application types. For example the usage of the disk subsystem in the parallel program partdiff-par discussed in this work: The checkpointing mechanism is called after a fixed sized count of iterations. Only in this phase the disk subsystem has to be active for the duration of the IO. This event occurs in more or less fixed timesteps. The disk subsystem can switch to low power consuming mode the time it is not used and switch to operating mode shortly before it is going to be used (due to the heuristics). This scenario includes possible performance losses, because the heuristics can fail (because of load balancing problems etc.). With **HDPowerEstimation** the saved energy of this scenario can be estimated to monetary evaluate the savings.

Another potential approach is to identify phases of low utilization (e.g. with Approach Strategy) to optimize the hardware utilization: Increase the hardware utilization to decrease the run time to save energy or to decrease the utilization and put the hardware to sleep. The algorithm can be optimized for both procedures. In case of the systematic decreasing of the utilization the hardware can switch to low power consuming modes. This can be done using for example program hints to instruct the operating system to switch the operating mode of the specific component. Using again the example of checkpointing: Shortly before the writing starts, a command sent to the IO library (e.g. MPICH library) instructs the device to change its operating mode. Following this approach the power consumption can be decreased without affecting the performance.

The great power savings by using energy efficient components estimated in this work are not surprising, but the capability to monetary evaluate concrete program traces shows the potential of this approach. Of course, when engineering energy efficient components the resulting power consumption is decreased. Due to the great savings with energy efficient components it is definitely imaginable to build an high performance cluster with energy efficient components (e.g. components from mobile devices) [faw]. The power consumption of this cluster would be a fractional amount of the power consumption of traditional high performance clusters. The performance would be decreased, but the concrete energy efficiency has to be evaluated.

In general the evaluation of the energy efficiency of (cluster) components is difficult because no standardized benchmarks are existing yet.

Due to the growing importance of energy awareness modeling a benchmark will be one of the first steps in direction to energy efficient high performance clusters.

A. Appendix

Table A.1.: *Summary of Global System States for ACPI [CCC+ 05]*

Global system state	Software runs	Latency	Power consumption	OS restart required	Safe to disassemble computer	Exit state electronically
G0 Working	Yes	0	Large	No	No	Yes
G1 Sleeping	No	< 0, varies with sleep state	Smaller	No	No	Yes
G2/S5 Soft Off	No	Long	Close to 0	Yes	No	Yes
G3 Mechanical Off	No	Long	RTC Battery	Yes	Yes	No

Table A.2.: *Summary of Device Power States for ACPI [CCC+ 05]*

Device state	Power consumption	Device Context Retained	Driver Restoration
D0 - Fully-On	As needed for operation	All	None
D1	$D0 > D1 > D2 > D3$	$> D2$	$< D2$
D2	$D0 > D1 > D2 > D3$	$< D1$	$> D1$
D3 - Off	0	None	Full initialization and load

Table A.3.: *Energy consumption of all non-collective trace files. For all trace files the power consumption has been estimated using the different strategies and effective devices respectively.*

Setup					**Energy consumption in Wh**								
					Measured	**Simple**		**Optimal**		**Approach**		**Efficient devices**	
Interlines	Iterations	Intensive	PVFS	MPI	Total	Total	Deviance	Total	Saving	Total	Saving	Total	Saving
800	5	comm	0	1	0.2487	0.1970	20.771 %	0.1699	13.742 %	0.1720	12.684 %	0.1132	42.514 %
800	5	comm	1	1	2.1246	2.0373	4.1080 %	2.0373	0 %	2.0099	1.3457 %	1.6405	19.473 %
800	5	comm	1	4	7.0446	6.2007	11.979 %	4.6752	24.602 %	4.4083	28.906 %	2.2676	63.430 %
800	5	comm	2	4	4.2213	3.8313	9.2389 %	3.2355	15.550 %	3.1067	18.911 %	2.0390	46.779 %
800	5	comm	4	4	2.5218	2.3811	5.5790 %	2.3450	1.5163 %	2.2235	6.6190 %	1.5985	32.865 %
950	5	comm	1	1	-	3.8348	-	3.8348	0 %	3.8157	0.4971 %	3.2527	15.178 %
1400	5	comm	1	3	24.146	22.160	8.2270 %	16.606	25.062 %	16.037	27.628 %	10.412	53.011 %
1600	5	comm	0	4	2.3481	1.9855	15.444 %	1.5161	23.641 %	1.3887	30.056 %	0.8497	57.200 %
1600	5	comm	1	4	44.702	40.794	8.7434 %	29.370	28.002 %	27.490	32.610 %	16.875	58.631 %
1600	100	comm	0	4	-	18.495	-	13.962	24.505 %	14.760	20.195 %	11.938	35.453 %
1600	200	comm	0	4	-	35.975	-	27.973	22.240 %	28.927	19.589 %	23.621	34.339 %
1600	500	comm	0	4	-	90.447	-	75.180	16.880 %	73.244	19.019 %	60.270	33.364 %
1600	5	comm	0	8	2.1923	1.9320	11.872 %	1.5078	21.958 %	1.4767	23.569 %	0.9447	51.101 %
1600	100	comm	0	8	-	15.056	-	14.387	4.4480 %	14.368	4.5703 %	13.083	13.104 %
1600	200	comm	0	8	-	28.840	-	27.930	3.1538 %	27.883	3.3165 %	25.850	10.367 %
1600	500	comm	0	8	-	70.383	-	68.761	2.3044 %	68.582	2.5586 %	63.826	9.3157 %
1600	5	comm	4	4	12.830	12.620	1.6377 %	12.187	3.4332 %	11.725	7.0904 %	9.1503	27.495 %
1600	5	calc	4	4	21.202	20.570	2.9767 %	19.077	7.2599 %	18.270	11.180 %	14.662	28.723 %
1600	50	comm	4	4	133.84	138.12	3.1981 %	132.67	3.9487 %	129.14	6.5046 %	104.30	24.486 %
1600	5	comm	4	8	13.362	13.556	1.4532 %	12.960	4.3947 %	12.597	7.0734 %	9.9154	26.859 %
1600	5	calc	4	8	18.476	18.256	1.1874 %	17.399	4.6972 %	17.036	6.6864 %	13.973	23.461 %
1800	5	comm	4	4	-	27.640	-	26.179	5.2851 %	25.443	7.9508 %	20.956	24.183 %
1800	5	comm	4	8	-	23.991	-	22.457	6.3963 %	22.035	8.1531 %	17.613	26.587 %
stresstest	-	-	0	1	16.697	16.468	1.3701 %	14.857	9.7835 %	14.091	14.433 %	11.554	29.837 %
Average:							7.1858 %		11.367 %		13.381 %		32.823 %

Table A.4.: *Energy consumption of all collective trace files. For all trace files the power consumption has been estimated using the different strategies and effective devices respectively.*

Setup					**Energy consumption in Wh**								
					Measured	**Simple**		**Optimal**		**Approach**		**Efficient devices**	
Interlines	Iterations	Intensive	PVFS	MPI	Total	Total	Deviance	Total	Saving	Total	Saving	Total	Saving
800	5	comm	1	1	2.1034	1.9935	5.2287 %	1.9844	0.4528 %	1.9748	0.9338 %	1.6089	19.289 %
800	5	comm	2	2	2.2119	1.8832	14.859 %	1.6395	12.942 %	1.5898	15.579 %	1.0732	43.010 %
800	5	comm	2	4	4.2061	3.8547	8.3541 %	3.3131	14.050 %	3.1652	17.886 %	2.0860	45.882 %
800	5	comm	4	4	2.7965	2.5925	7.2942 %	2.5203	2.7855 %	2.4451	5.6836 %	1.7931	30.831 %
1100	5	comm	2	2	3.9491	3.6516	7.5311 %	3.1371	14.090 %	3.0581	16.253 %	2.2095	39.491 %
1400	5	comm	1	3	25.450	23.481	7.7356 %	18.007	23.314 %	16.951	27.809 %	11.235	52.154 %
1600	5	comm	1	4	45.574	41.640	8.6325 %	30.651	26.388 %	28.141	32.418 %	17.269	58.528 %
Average:							8.5193 %		13.432 %		16.652 %		41.312 %

Tools and Software

This thesis has been created under usage of the following tools and software.

The project **HDPowerEstimation** has been implemented using **Eclipse Ganymede** (version 3.4.2). The source code was maintained using **Subversion** (version 1.5.4). The class diagram 4.2 has been created using the **Eclipse** plug in **Omondo** (evaluation version).

For identifying the CPU at a PVS cluster node the tool **CPUInfo** [1] has been used.

The screenshots of **Sunshot** have been partially reworked using **GIMP** [2] (version 2.6.6) and **OpenOffice Draw** [3] (version 3.0.1) respectively . The figures of **HDPowerEstimation** are created using the PNG export function of **JFreeCharts**.

Further the bar charts and plots are created with **Gnuplot** [4], the pie charts are created with **OpenOffice Draw**.

This document has been created with LaTeX [5] (using the KOMA script, BibTeX and a modified version of the alphadin style) and the text editor **Emacs** [6].

The software and tools used are almost all free, thanks to the efforts of Free Software Foundation [7], and thousands of dedicated and conscientious software developers all around the world.

[1] `http://www.alasir.com/software/cpuinfo`, last checked on August 27, 2009
[2] `http://www.gimp.org`, last checked on August 27, 2009
[3] `http://www.openoffice.org`, last checked on August 27, 2009
[4] `http://www.gnuplot.info`, last checked on August 27, 2009
[5] `http://www.latex-project.org`, last checked on August 27, 2009
[6] `http://www.gnu.org/software/emacs`, last checked on August 27, 2009
[7] `http://www.fsf.org`, last checked on August 27, 2009

List of Figures

List of Tables

List of Algorithms

Bibliography

[AH03] Aebischer, Bernard (editor) ; Huser, Alois (editor) ; Proceedings of the 3rd International Conference on Energy Efficiency in Domestic Appliances and Lighting (EEDAL '03) (organizer): *Energy efficiency of computer power supplies.* 2003

[AHC+09] Agarwal, Yuvraj (editor) ; Hodges, Steve (editor) ; Chandra, Ranveer (editor) ; Scott, James (editor) ; Bahl, Paramvir (editor) ; Gupta, Rajesh (editor) ; USENIX Symposium on Networked Systems Design and Implementation (organizer): *Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage.* 2009

[Aud09] Audin, Gary: *Managing LAN Switch Power.* `http://www.nojitter.com/blog/archives/2009/06/managing_lan_sw.html`. Version: June 2009, last checked: August 27, 2009

[Bas08] Bastian, Peter: *Paralleles Höchstleistungsrechnen.* Oktober 2008. – Lecture script

[CCC+05] Corporation, Hewlett-Packard ; Corporation, Intel ; Corporation, Microsoft ; Ltd., Phoenix T. ; Corporation, Toshiba: *Advances Configuration and Power Interface Specification.* December 2005

[DGJB07] Diniz, Bruno (editor) ; Guedes, Dorgival (editor) ; Jr., Wagner M. (editor) ; Bianchini, Ricardo (editor) ; ISCA (organizer): *Limiting the Power Consumption of Main Memory.* 2007

[faw] Vasudevan, Vijay (editor) ; Franklin, Jason (editor) ; Andersen, David (editor) ; Phanishayee, Amar (editor) ; Tan, Lawrence (editor) ; Kaminsky, Michael (editor) ; Moraru, Iulian (editor): *FAWNdamentally Power-efficient Clusters.* – accepted to HotOS XII

[Fol04] Folsom, PAE Tech Pubs I.: *Pentium(R) 4 Processor with 512-KB L2 Cache on 0.13 Micron Process and Pentium(R) 4 Processor Extreme Edition Supporting HT Technology.* February 2004

[GGS04] Gupta, Maruti (editor) ; Grover, Satyajit (editor) ; Singh, Suresh (editor) ; IEEE ICNP (organizer): *A feasibility study for power management in LAN switches.* 2004

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns - Elements of Reusable Object-Oriented Software.* Addision-Wesley, 1995

[Gre] Green500: *Green500 List.* `http://www.green500.org`, last checked: August 27, 2009

[GS03] Gupta, Maruti (editor) ; Singh, Suresh (editor) ; ACM SIGCOMM'03 (organizer): *Greening of the Internet.* 2003

[GS07a] Gupta, Maruti (editor) ; Singh, Suresh (editor) ; IEEE ICC'07 (organizer): *Dynamic Ethernet link shutdown for power conservation on Ethernet links.* 2007

[GS07b] Gupta, Maruti (editor) ; Singh, Suresh (editor) ; IEEE INFOCOM (Minisymposium) (organizer): *Energy conservation with low power modes in Ethernet LAN environments.* 2007

[GSKF03] Gurumurthi, S. ; Sivasubramaniam, A. ; Kandemir, M. ; Franke, H.: Reducing Disk Power Consumption in Servers with DRPM. In: *IEEE Computer'03* (2003), December

[HSRJ08] Hylick, Anthony (editor) ; Sohan, Ripduman (editor) ; Rice, Andrew (editor) ; Jones, Brian (editor): *An Analysis of Hard Drive Energy Consumption.* 2008

[Int09] Intel, MPG Tech P.: *Intel Core2 Duo Mobile Processor for Intel Centrino Duo Mobile Processor Technology Datasheet.* September 2009

[Lud08a] Ludwig, Thomas: *Cluster Computing.* April 2008. – Lecture script

[Lud08b] Ludwig, Thomas: *Hochleistungs- Eingabe/Ausgabe- Systeme.* January 2008. – Lecture script

[man08] Managing Energy Use in a Network with a New SNMP Power State MIB. In: Blanquicet, F. (editor) ; Christensen, K. (editor): *Proceedings of the IEEE Conference on Local Computer Networks*, 2008, S. 509–511

[MCH07] Moona, Prof. R. ; Chole, Sharad ; Harneja, Sanchay: Memory Management using Dynamic Memory Switching / Department of Computer Science and Engineering, Indian Institute of Technology Kanpur. 2007. – Project Report

[MV04] Mahesri, Aqeel (editor) ; Vardhan, Vibhore (editor) ; Workshop on Power Aware Computing Systems, 37th International Symposium on Microarchitecture (organizer): *Power Consumption Breakdown on a Modern Laptop.* 2004

[NC05] Nordman, B. (editor) ; Christensen, K. (editor) ; IEEE 802 LAN/MAN Standards Committee Plenary Session (organizer): *Reducing the Energy Consumption of Network Devices.* 2005

[Phy04] Physikalisch Technische Bundesanstalt: *The legal units in Germany.* February 2004

[Top] Top500: *Top500 List.* http://www.top500.org, last checked: August 30, 2009