Investigating power efficiency and co-location effects on heterogeneous HPC architectures

Nikolaos Koutsikos

August 23, 2013

MSc in High Performance Computing

The University of Edinburgh

Year of Presentation: 2013

Abstract

This project is researching in the area of energy efficiency of heterogeneous clusters. In particular, investigation of the relationship between clock frequencies and power efficiency takes place. Moreover, co-location of HPC workloads is researched and analysed, both in terms of performance and power efficiency. For evaluation purposes, several scientific applications were used, with the purpose of reproduction of real case scenarios.

Contents

List of Tables

List of Figures

Acknowledgements

This project would not have been possible without the kind contribution of others. I would like to thank my supervisor Dr Michele Weiland for her constant guidance and time she devoted to me. I am also grateful to Mr David Power from Boston Ltd for the help and support he provided for the ISC '13 Student Cluster Challenge. Finally, I would like to thank my parents Spiridon and Demetra for their unconditional love and support, without which I would not be able to do this Master of Science.

Chapter 1

Introduction

One of the key societal issues today is energy. Up until ten years ago as technology was advancing, products were becoming better, but they were also consuming more power. This situation was initially not an issue, however it soon became pressing. Energy prices started growing up and this had as a result that more and more people were getting concerned about energy. Nowadays every product takes serious consideration of the power consumption and a lot of research is going on trying to lower the power needs of products. The term "Green product" has been advertised so much, that there is no one not aware of the situation that exists during the last several years.

Computer products are not an exception. As they starting providing more performance, they were consuming more energy. However this has changed over the last years, and the term "power efficiency" entered our lives. Power efficiency is the ratio of the resulting performance to the power consumption. It is expressed in performance per Watt, where performance is whatever the application considers as performance metric. The first fact that indicated that energy efficiency was a real concern was the power supply efficiency. A certification program called 80 Plus was assessing power supply units (PSU) based on their efficiency, which is calculated with the ratio of the power provided to the system divided by the power usage of the PSU. [1] This programm promoted power efficiency on PSUs, as vendors are trying to better certifications for the PSUs they are offering.

This dissertation investigates the important issue of power efficiency in two ways. Firstly it researches the interdependence of power efficiency and frequency rates of compute units. Secondly it applies the recent idea of co-location and investigates the system behaviour. Explanation of the results is being made for all the presented results. Co-location is the technique of running two different scientific applications simultaneously on the same system.

1.1 Report organisation

This report consists of 7 chapters:

Chapter 1 is a quick introduction to prepare the reader for what is following

Chapter 2 presents some background information and motivation about this research. The reasons for deviating from the original project plan are analysed.

Chapter 3 describes the experiences from the ISC'13 Student Cluster Challenge and explains the decisions which had to be taken

Chapter 4 analyses the experimental procedure. It analyses the hardware and its specifications and lists the software stack used. Moreover it describes all the benchmarks used, explains the co-location procedure and details the power measurements.

Chapter 5 presents and explains the results obtained. Firstly it deals with the individual application testing and after having made some conclusions it proceeds to the co-location tests.

Chapter 6 summarises the results and experiences from the experimental procedure.

Chapter 7 proposes some future directions that should be followed for exploiting power efficiency on modern clusters and investigate more in depth in this area.

Chapter 2

Background Theory

2.1 The energy problem in HPC

Supercomputers become bigger every year in order to deliver higher levels of performance. As for today, the most powerful supercomputer the world is Tianhe-2. Its performance is 33.9 petaFLOPS and its power consumption is 17.8 Mwatt. [2] Systems like Tianhe-2 consume huge amounts of energy, and because of that, the cost of running the system throughout its life is much higher than the cost of acquiring it. As the world's top HPC systems are achieving petaFLOPS of performance, we are in the era of Petascale.

If we were trying to build a system like Tianhe-2, but we wanted it to achieve performance of exaFLOPS, it would need more than 500 MWatt. In order to supply this amount of power, a nuclear power station is needed next to the supercomputer for power. Moreover, no country could afford to run a system that consumes so much energy. Even if the most efficient HPC system is taken into account, it would need more than 300 MWatt. [3]

A lot of scientists are researching towards designing a HPC system that could achieve exaFLOPS performance levels. This is the Exascale Challenge and HCP scientists around the world are trying to address it. A fundamental report about this challenge was written by the Defence Advanced Research Projects Agency in 2008. [4] The growth trend of HPC systems' performance suggests that an exascale system will be feasible by 2018 [5], however DARPA aimed to achieve an exascale system by 2015. Nowadays however the aims are for 2018-2020. In order to achieve this, they have identified four problematic issues:

- a. power and energy,
- b. memory and storage,
- c. concurrency and locality and
- d. resiliency and fault tolerance.

While a lot of research is going in all these four categories, this dissertations is dealing with the energy problem. The most power efficient systems nowadays are exploiting the high efficiency of accelerators in some particular calculations. Having access to a small cluster equipped with accelerators, investigation of unconventional ways to increase power efficiency is being done. Due to the fact that root access to the system was available, we decided to take the advantage of it and research areas that we would not be able to do without this priviledge access.

2.2 Deviation from project plan

During the Project Preparation course, after having done research about the initial direction of the dissertation, a project plan was proposed, accompanied with a detailed workplan and risk analysis. The main idea about the dissertation was to investigate the power efficiency of novel HPC architectures and compare it with conventional x86 hardware. At the time this project plan was made, we were expected to acquire access to a novel ARM cluster called Boston Viridis [6] and we were in touch with people from Adapteva in order to purchase some Parallella development boards, which combine an ARM SoC with RISC coprocessor [7].

However, due to the very high price of the Boston Viridis, our vendor was not able to provide adequate number of ARM nodes and this could seriously affect our testing plans. The ARM clusters due to low power consumption and low performance, need a high number of nodes in order to get a good level of performance. However, when the number of nodes increases, scalability issues have to be taken into account. Thus, in order to compare an ARM with a x86 cluster, many ARM nodes are need. Unfortunately we could not get access to more than 8 nodes in time, so the Boston Viridis investigation had to abandoned.

Moreover, Adapteva Parallella was supposed to start shipping in April. Initially there was an announcement for a two months delay, which was manageable. However, when the shipping date was approaching, another delay was announced. This made this platform unavailable as well, as it was not possible to get it on time. In fact the shipments of Parallella have started just a few days before this dissertation was written.

With the two main platforms being unavailable, there was no way of sticking to the original project plan. The platform to be investigated had to be changed. The new platform is another emerging technology in the area of HPC, the heterogeneous clusters. As these clusters have been used for some time, a simple investigation of power efficiency would not be truly novel. Thus the relationship of CPU and GPU clock frequencies with power efficiency is going to be investigated.

A new research area in HPC is dealing with the co-location of HPC workloads on clusters. [8] Since almost no research has been done on heterogeneous clusters, this subject would make this work pioneer. However, power efficiency and application co-location are not faced as two different aspects of the research, but they are highly combined. In particular, the results from the power efficiency investigations have affected the way that co-location was tested.

In summary, despite the careful planning and risk analysis, it was inevitable for the initial project plan not to change. However, this should not be considered as a negative fact. Since there are novel ideas, there will always be areas for research.

Chapter 3

ISC '13 Student Cluster Challenge

Every year in June, the International Supercomputing Conference takes place in Germany. HPC vendors from around the world set up their booths and they try every year to exhibit revolutionary products. Big names like IBM, Intel, Nvidia, Samsung, Bull, Mellanox and many others are present at every ISC. Moreover, many presentations, talks and workshops are given by HPC experts, giving the visitor the opportunity to widen his knowledge. ISC is the annual meeting for the HPC professionals who want to learn about new exotic products, exchange ideas, socialise and of course have fun! It is considered to be a great event for High Performance Computing.

3.1 The Student Cluster Challenge

The HPC advisory council has organised the Student Cluster Challenge since 2012. [9] Small student teams from various Universities around the world are competing against each other within a friendly spirit. The teams have their own booths on the exhibition floor, where they set up their clusters and compete in real time in a series of HPC benchmarks, trying to achieve the highest performance. It is a perfect opportunity for students to showcase their HPC skills and exchange experiences and knowledge with the other competitors.

For fairness purposes, there is a power limit of 3000 Watts of maximum power consumption, capping the system that the teams can build. The power consumption is monitored via power distribution units (PDU) with a network interface. Each cluster is powered by its own PDU and all PDUs are monitored together in real time. After the system is set up and the first benchmark result is submitted, no one can physically

touch the cluster. The system must be turned on all the time (even at nights) and reboots are not permitted. Only in the case of a hardware failure can the system be touched or rebooted, and this has to happen under supervision of the organisers. [10]

The benchmarking part of the competition is done using several HPC applications, in order to determine the systems' capabilities both in terms of architecture choice and software stack setup. Of course the builds of the benchmarks play a major role as well. Every year, the standard benchmarks are the High Performance Linpack and the HPC Challenge. HPCC is a benchmark suite that has several benchmarks for assessing different subsystems of a cluster, in order to provide an overall image about its performance. Moreover, five scientific applications are used for benchmarking: three of them are known beforehand, so that the students can prepare builds and optimise them, but the other two are announced at the same day of the runs, in order to test the students experience, collaboration and work under pressure. The three known applications this year were GROMACS, MILC and WRF, while the secret applications were AMG and CP2K. [10]

The student teams have to find vendors to support their efforts, both by providing the necessary hardware and support prior to the competition. Our vendors were Boston Ltd and Viglen. A series of meetings and teleconferences with vendors staff and our MSc supervisors were organised, in order to help us decide the optimal configuration for our cluster and share their experiences, which could help us to make high performing builds of the scientific software.

3.2 The ARM days

Initially, we were working on a novel ARM cluster, called Boston Viridis. This system is equipped with processors made from Calxeda and called EnergyCore. Those processors are quad-core ARM Cortex-A9 and are characterised as Server-on-Chip which in fact means that they are SoCs with built-in interconnection (10 GbE). Each of these SoCs is considered a different node 48 nodes fit in a 2U chassis, summing up 192 cores with about 350 Watts of power consumption. [6]

Extended testing was done in order to exploit the system's potential. Due to the fact that this system was based on a completely different architecture than the usual HPC systems, we had to research everything from scratch. After many optimisations including optimised libraries for ARM, explicit compilations for MPI libraries, and extended configuration of our benchmarks, we concluded that this system could not be competitive for HPC use. The reason for this was that the ARM cores do not perform well on floating point calculations and all the HPC benchmarks rely heavily on FP performance. Moreover, the systems price was extremely high (about $50,000 per 2U chassis and we needed 8 chassis) and our vendors were struggling to invest this amount of money for the competition. For these reasons we had to abandon the ARM plan, sacrificing all the work that had been done for months.

3.3 The Kepler days

The final architecture decision included the safe choice of Intel Xeon CPUs and Nvidia Tesla GPUs. In particular we used a four node cluster, each node equipped with two 8-core Sandy Bridge Xeons, two Nvidia Tesla K20 (Kepler) and Infiniband QDR. We had remote access to the cluster two months before the competition, in order to gain experience with the system, tweak it and build the benchmarks. Our vendors showed complete trust in our knowledge, providing us root access to the systems and full freedom to experiment.

3.4 Hardware training

During our preparation for ISC, our vendors and EPCC came to an agreement of providing us exclusive hardware training. Thus, two weeks before the competition we went to London for three days, and attended intensive hardware training at our vendors' sites. In particular we spend two days in Boston's Ltd laboratory, in order to gain practical experience in handling, disassembling and assembling the hardware, as well as how to face possible hardware issues that might come up during the Cluster Challenge. Moreover, we experimented setting up a different Linux distribution, but after struggling making everything work (at last we succeeded), we decided that it was not worth it because of limited support and rolled back to Red Hat. We had the chance to identify a obscure hardware problem that one of the Tesla cards was facing and fixed it in time. In general this provided us with everything we should know in order to be ready to face any problem that might show up during the Cluster Challenge.

Moreover, we spend the last day at Viglen's site, were applications experts were helping us surpass any issues we had with the software part. Last but not least, Viglen had arranged an HPC expert from Nvidia to attend, who offered us his in-depth knowledge on exploiting every last bit of performance from the Nvidia Tesla cards. After this three-day training we returned to Edinburgh completely confident for the competition, looking forward to attending ISC.

3.5 Results and experiences

The ISC '13 Student Cluster Challenge lasted four long days. A lot of work was necessary during these days, in order to ensure that everything was working the way it should, and of course we had to deal with the secret applications as well. Despite the fact that our institution and us had no previous experience in Student Cluster Challenges, we managed some good results that are worth mentioning.

On High Performance Linpack we achieved a result of 8.321 TFLOPS which gave us the second place, only 0.134 below the first result. Having the smallest system of the eight teams participating, that result gave us the unofficial award of highest

performance per node. We also succeeded the first and second place for the two data sets used for MILC. [11]

Apart from the Cluster Challenge success, our experience from the ISC is completely positive. We had the chance to socialise with HPC experts from around the world, meet new friends and of course visit and spend some time on the numerous booths on the exhibition floor. We strongly believe that it was an “once in a lifetime” experience that anyone should chase.

Chapter 4

Experimental design

In this chapter, the experimental setup is presented. In particular the hardware specifications of the cluster used for the tests are detailed, as well as the installed software. Furthermore, the benchmarking applications are explained and the specifics of the tests used for assessing the test system are detailed. The power measurement equipment and procedure is also explained.

4.1 Hardware

The cluster used for the experimental procedure of this dissertation has many similarities with the system used for the ISC '13 Student Cluster Challenge. The system was provided by Boston Ltd, and consisted of three nodes of the Boston Venom GPU compute server. Each node was equipped with dual Intel Xeon E5 8-core CPUs, three Nvidia Tesla Kepler GPUs, 64 GB of RAM and Infiniband QDR interconnection. The exact specification of each compute node is detailed in Table 1.

CPU	2x Intel Xeon E5-2670, 8-cores, 2.6 GHz
Main Memory	64GB DDR3 ECC 1600 MHz
GPU	3x Nvidia Tesla K20, 2496 cuda cores
GPU memory	3x 5120 MB GDDR5 ECC 5200 MHz
Interconnection	Mellanox Infiniband QDR (ConnectX-3)
Power supply	1800 Watts, 80 Plus Platinum efficiency

Table 1: Compute node hardware specifications

4.1.1 The Intel Xeon E5-2xxx

The Intel Xeon E5-2670 CPU is based on the Sandy Bridge microarchitecture. In particular, the code name is Sandy Bridge-EP, where EP stands for Efficient Performance. It belongs to the E5-2xxx series which means that it can equip dual processor systems and it is manufactured at 32nm. It is equipped with 8 physical cores which have a base clock speed of 2.6 GHz and a maximum Thermal Design Power of 115 Watts. It has four memory channels that can use memory clocked up to 1600 MHz. [12] Our system had a 8 GB 1600 MHz ECC memory stick on each channel, accumulating a total of 32 GB per processor. It supports Hyper-Threading [13], which is Intel's proprietary simultaneous multithreading (SMT) implementation with 2 threads per core, but during the experimental procedure, Hyper-Threading was disabled from the system's BIOS. The reason for that choice is that Hyper-Threading increases the power consumption but the performance gains are not constant, and some times there is even a performance decrease. Nevertheless in most cases, Hyper-Threading decreases the power efficiency, thus its impact was out of scope for the current dissertation.

4.1.2 Turbo Boost explained

The Intel Xeon E5-2670 CPU supports Turbo Boost [14], which is also called Turbo mode. Turbo Boost is a technology that was first seen in the Nehalem microarchitecture, which enables the CPU to operate at a higher clock rate. It is activated when the operating system requests the highest performance state and it does not need any additional software or driver. The maximum amount of frequency boost that the processor can get depends on the number of cores that are being used and it is expressed by a number of Turbo steps, which in fact is a number added to the CPU's multiplier. For the Intel Xeon E5-2670 CPU the Turbo steps are 4/4/5/5/6/6/7/7, which means that the added frequency is 400 MHz when 8 or 7 cores are under load, 500 MHz when 6 or 5 cores are under load, 600 MHz when 4 or 3 cores are under load and 700 MHz when 2 or 1 cores are under load. However, in order for the processor to use these increased speeds, it has to remain within its specifications for power, current and thermal. In practice, when Turbo Boost is enabled, the CPU clock rate is not constant, but it fluctuates from base frequency up to maximum Turbo frequency.

In order to ensure system stability when Turbo Boost is engaged, the CPU's voltage (vcore) is increased. This has a negative effect on the CPU's power consumption and is proportional to the square of the voltage. In particular, processors' power consumption is approximated using the following formula [15], which calculates the switching power dissipation:

$$P = \alpha f C V^2$$

The α term is activity factor, which is a fraction between 0 and 1, and denotes what part of the processor subsystems are clocked and can be lowered with techniques such as clock gating. The f term is the processor frequency, the term C is the chip capacitance and the term V is the voltage applied to the chip.

The Xeon processor can operate frequencies starting from 1200 MHz up to 2600 MHz with a 100 MHz step, in addition to the Turbo Boost. All these frequencies can be controlled in the Linux Operating system by setting the parameter `scaling_max_freq` to the desired frequency. The `scaling_max_freq` is under the folder `/sys/devices/system/cpu/cpu*/cpufreq` and root permissions are necessary in order to be set. The Turbo Boost is enabled by setting the `scaling_max_freq` to the value max_freq+1, which in our case is 2601. Note that the frequencies needs to be changed for every available core. In order to change the CPU frequencies easily, a script was written.

4.1.3 The Nvidia Tesla K20

The Nvidia Tesla K20 is based on the Kepler microarchitecture and it uses the GK110 GPU. It is equipped with 2496 cuda cores operating at 705 Mhz and 5120 MB of ECC GDDR5 RAM operating at 5200 MHz and offering 208 GB/sec of bandwidth. The Nvidia K20 allows frequency scaling of the GPU, but unfortunately not of the device memory. The available frequencies are 758 MHz, 705 MHz, 666 MHz, 640 MHz and 614 MHz and they are set using the nvidia-smi utility [16]. In particular the command to change the frequencies to the highest available is `nvidia-smi -ac 2600,758` and requires root permissions. Note that the memory frequency is the actual frequency and not the DDR equivalent. The available frequencies can be listed by using the command `nvidia-smi -q -d SUPPORTED_CLOCKS`.

4.2 Operating System, drivers, compiler and libraries

The Operating System installed on the cluster nodes is a server Linux distribution CentOS [17], which is a Red Hat [18] derivative. Both Red Hat and CentOS are widely used in server environments, due to their increased stability and extended support from the hardware vendors. The compiler used was GCC [19], due to small performance differences from the Intel compiler. The GCC compiler is widely available, thus it allows reproducibility of the benchmark results. For the same reason Open MPI [20] was used as MPI library. The detailed software versions used are noted on Table 2.

Operating System	CentOS 6.2
Kernel	2.6.32 x86_64 GNU/Linux
Compiler	GCC 4.4.6
MPI library	Open MPI 1.6.4
CUDA	CUDA Toolkit 5.0
Nvidia driver	304.54

Table 2: Software stack versions

4.3 Benchmarks

In order to investigate the power efficiency of the test system several HPC applications were used. The choice of benchmarks has been done in a way that covers different kinds of applications as much as possible. By “kinds” we mean that we take into account both the underlying science and the computation specifics of each application.

4.3.1 High Performance Linpack

The High Performance Linpack benchmark [21] is probably the best known benchmark in the area of HPC. It is used to measure the performance of every HPC system found in the Top500 and Green500 lists. It is based on the original LINPACK, which was written in Fortran by Jack Dongarra, Jim Bunch, Cleve Moler and Gilbert Stewart. High Performance Linpack is a portable implementation for Distributed-Memory systems, written in C by Antoine Petitet, Clint Whaley, Jack Dongarra, Andy Cleary and it is using MPI for inter-node communication. The algorithm of High Performance Linpack solves NxN random linear equations (Ax=b) using LU factorisation with partial row pivoting. These calculations are done using double precision (64-bits) floating point arithmetic, thus the resulting performance is expressed in FLoating-point Operations Per Second (FLOPS). The fact that the benchmark result is a single number makes it easily comparable and databases of results can be easily formed.

High Performance Linpack is highly scalable because the parallel efficiency of the algorithm is maintained constant as long as the memory usage per processor remains constant. Moreover, Linpack has been highly optimised and it is producing performance levels very close to the maximum theoretical peak. For that reason it has been many times criticised, because the performance levels of Linpack cannot be obtained on any other real world HPC application, as dense linear systems are not representative of calculations usually applied in scientific computing. Thus it is widely believed that Linpack's performance does not necesserily represent the system's real world capabilities.

Nvidia has developed a CUDA accelerated version of High Performance Linpack. This version utilises both CPUs and GPUs. While the original version of High Performance Linpack uses MPI processes for CPU cores, the Nvidia version uses one MPI process per GPU, which is typical for GPU computing. Therefore, in order to exploit the CPU cores, it uses threading for each MPI process. The number of threads can be adjusted via an environment variable. Furthermore, the CUDA version has an extra parameter that needs to be set, which is called `GPU_DGEMM_SPLIT`, and it controls the percentage of DGEMM operation data that will be offloaded to the GPU. This parameter plays a major role at the performance results, as it is the main way of achieving load balance between the CPUs and the GPUs. There is no way other than testing to determine the most appropriate data split. Several system specifications affect the optimal value, such as the number of CPU cores, the CPU clock frequency, the GPU model, the number of GPUs and a few others. In fact, this parameter varies on every system.

The version used for the experimental procedure was optimised for the Kepler architecture. Unfortunately this version cannot be found online. On each node, two GPUs were used, as the third one did not provide the expected performance boost. For that reason, in order to be as power efficient as possible, we used two out of the three available cards. The Nvidia K20's low idle power (approximately 15 Watts) helped by not affecting the power consumption of the system much. At every CPU or GPU frequency change, the `GPU_DGEMM_SPLIT` parameter had to be tweaked again, in order to maximise the performance. This was a very time consuming task, as it included a lot of test runs. The problem size was set in order to use more than 90% of the available memory.

4.3.2 GROMACS

GROMACS (GROningen MAchine for Chemical Simulations) is a molecular dynamics versatile package designed to perform scientific simulations of proteins, lipids and nucleic acids [22]. Originally it was developed by the Biophysical Chemistry department of University of Groningen but nowadays is maintained and developed by many universities and research centers around the world. It is free and open source software, released under the GNU Lesser General Public Licence. It is written in C and C++ using a lot of intrinsic functions that are compiled to fast SIMD instructions. However the program it is derived from was written in Fortran77. GROMACS is being used extensively in the distributed computing project Folding@home, which simulates protein folding and computational drug design. However, in order to ensure data validity, Folding@home uses a closed licence version of the GROMACS code. [23]

GROMACS is the most widely used Molecular Dynamics simulation software mainly because of its speed. Typically it is 3-10 times faster than many of the other codes. [24] In order to achieve this, developers have applied a lot of algorithmic optimisations for reducing the complexity of the computations, but also they have written processor-specific optimised kernels. Because of its high performance and its extreme speed in calculating even the nonbonded interactions, many researchers are using GROMACS in non-biological simulations such as polymers.

GROMACS supports many schemes for efficient parallelisation. MPI has been used since the early versions of the code. It applies domain decomposition and supports automatic dynamic load balancing, which plays a major role in the resulting performance. Based on the modern trend of multiprocessor systems with high core counts, that have several non-uniform memory access (NUMA) areas within a node, GROMACS is also supporting OpenMP, which can be used in combination with MPI. In a pure MPI scheme, intra- and inter-node parallelisation are using the same network interface, and because of the NUMA effects that most modern system have, inter node communications are widely affected. The multi-level parallelisation can be used to address these NUMA and communication effects. The OpenMP implemetation is quite efficient, however when used together with MPI there is additional overhead. The performance of the mixed mode parallelisation is affected by various factors like the underlying architecture and the type of simulation. When used correctly it can provide

a significant performance boost. There is also a thread-MPI implementation that uses Posix pthreads and Windows threads. It targets on single machines that do not have MPI support and can run slightly faster than MPI. [25]

In GROMACS 4.6 there is highly efficient CUDA support in order to exploit GPU acceleration on Nvidia cards with compute capability 2.0 or more (Fermi and Kepler architectures). In particular, CPUs and GPUs are used simultaneously. The non bonded forces are calculated on the GPUs with one MPI process per GPU (multiple GPUs are also supported by applying domain decomposition), while the bonded forces and Particle Mesh Ewald electrostatics are calculated on CPUs in parallel using OpenMP.

The version used for benchmarking was 4.6.3. Two different compilations were used, a CPU-only one and a GPU accelerated. The CPU runs were fastest for the test system when using pure MPI. As for as the GPU runs are concerned, using more than two GPUs did not add any performance. That is happening because the K20 are very powerful cards and two of them are more than enough. Adding an extra card does not affect the performance because there is no more computation to be done on GPUs. Logic dictates that for using two GPUs, two MPI processes should be launched per node. However, OpenMP adds overhead as it does not scale well to large number of threads and because of the sufficiency of GPU compute power [25], testing showed that using four MPI processes per node (or else two MPI processes per GPU) was more efficient.

The performance metric for GROMACS is directly provided by the application and it is expressed in *nanoseconds per day* (ns/day). This metric corresponds to the actual time of simulation that can be run within a day.

4.3.3 Ludwig

Ludwig is a general purpose parallel Lattice-Boltzmann code specifically designed for simulating hydrodynamics of complex fluid problems. It is developed in C by EPCC, the University of Edinburgh. The original code is using MPI in order to be parallelised and can run efficiently on thousands of CPU cores.[26] There is also a CUDA enabled version of Ludwig, which takes advantage of the huge compute capabilities of modern Nvidia GPUs. All the computational expensive routines are offloaded to the GPU. By keeping the data resident on the GPU in order to minimise data transfers between CPU and GPU, by the application of new data layouts which exploit the GPU memory bandwidth and through other optimisations, perfect weak scalability is observed for thousands of GPUs. [27] Last but not least, Ludwig is highly efficient when using multiple GPUs per node. This is a result of minimising data transfers between GPUs through specialised data compression techniques and overlapping computations with MPI communications.

Two different builds were used for testing: a CPU-only version and a GPU-only version. The part of the simulation that has been ported to the GPUs is the main time step loop, thus the performance measurements concern this part. All nine GPUs of the test system were used (three per node), as the scalability was perfect and the speedup

over the CPU-only version was very good. However a drawback of the GPU version of Ludwig is that it can only be used for simulations that fit on the GPU's memory. For larger simulations the usage of the original version is compulsory.

Ludwig is only reporting timing results for the execution. However, it is not difficult to determine a performance metric. The simulation time is highly dependent on the problem size, which is the total amount of lattice units. The proposed performance metric for Ludwig is:

$$Ludwig\ performance = \frac{lattices * steps}{execution\ time}$$

and it is expressed in Mcells/sec (cells*10^6 / sec).

4.3.4 AMG

AMG is a parallel Algebraic MultiGrid solver for linear systems that use unstructured grids. [28] This benchmark is written in ISO standard C. As most of algebra algorithms, AMG is highly memory bandwidth dependant. Parallelisation is achieved by data decomposition both using MPI and/or OpenMP. However, the MPI implementation is more efficient and it is highly recommended, even when running on a single node. As the size of the problem increases, the MPI communication time starts to dominate the total execution time. Thus, on highly parallel environments, the interconnection performance plays a major role. Moreover, AMG performs one or two computations per memory access, so it can definitely be characterised as a memory bound application.

The AMG version used for benchmarking is dated 20 May 2013. It includes three different problems and four different solvers, which enumerate several combinations. With the purpose of results reproducibility, we tested the proposed setup for small systems benchmarking. In particular the problem being solved is a 3D Laplace on a cube (`-laplace` parameter) and the solver as the PCG with diagonal scaling (`-solver 2` parameter)

AMG provides directly a performance metric for the resulting performance. The performance is expressed by the formula:

$$AMG\ performance = \frac{cells * iterations}{solve\ time}$$

and it is expressed in Mcells/sec (cells*10^6 / sec).

4.3.5 Co-location tests

In order to investigate the effects that co-location of HPC applications has on performance and power efficiency, three different co-location tests were planned.

a. *Ludwig CPU – Ludwig GPU:* Co-locating the two versions of Ludwig can provide valuable data about the co-location behaviour of applications that are stressing different

processors of a modern system and that are both compute intensive. Also, it will be a good indication if it is worth investing time in developing a hybrid Ludwig code that exploits all the available hardware.

b. *AMG – Ludwig GPU:* Co-locating the memory bound AMG and the GPU-only Ludwig will show how two completely different applications coexist within the same cluster and how they are affecting each other.

c. *AMG – GROMACS:* Both of these applications are very demanding. AMG stresses the memory subsystem and GROMACS is a very optimised code that exploits any hardware on a modern system. Because both applications are using CPU cores, the process mapping on the CPU was managed using OpenMPI's rankfiles. Two different mappings were tested.

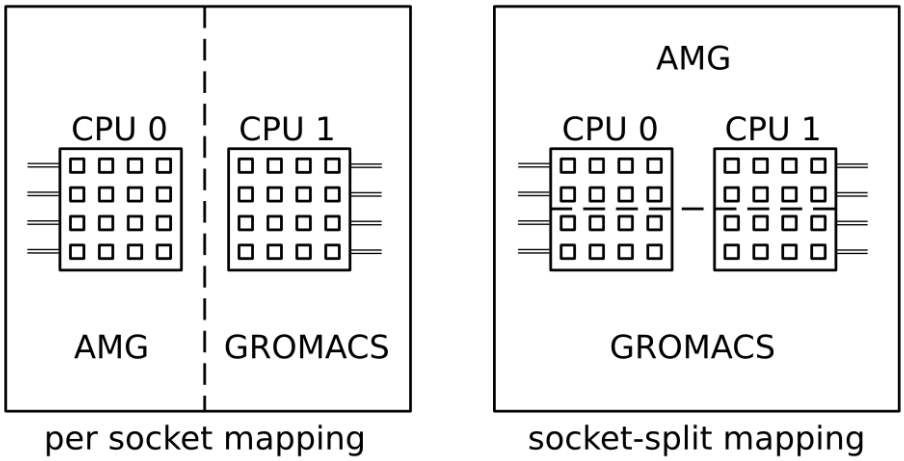

Figure 1: Types of core mapping

"Per socket" maps each application to an entire socket/CPU. This translates that each application will have its own caches and its own memory banks (specific memory bandwidth). On the other hand, "socket-split" maps each application to half the cores of both sockets. That means that applications will compete for caches and memory bandwidth. These subsystems are shared among the applications. The two mappings are visualised in Figure 1.

It should be noted that the co-location tests are reproducing the worst case scenario: the compute intensive kernels are aligned to coexist throughout their execution. In order to achieve this, the input data for each benchmark was carefully chosen in order to have almost the same execution time. Furthermore, depending on the length of the initialisation that each benchmark performs, appropriate sleeps were introduced in order to let the computational kernels start at the same time. This is visualised on Figure 2.

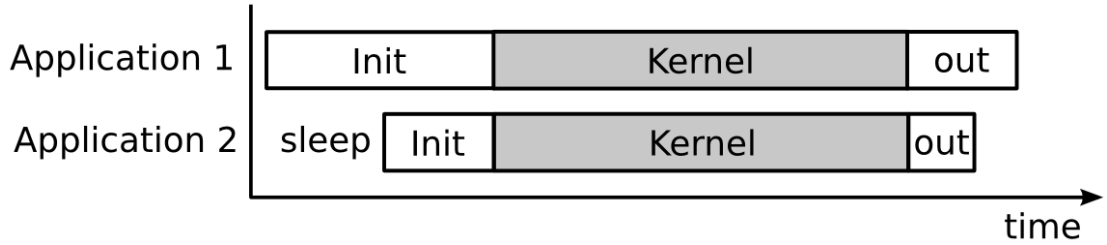

Figure 2: Co-location kernels alignment

4.4 Power consumption measurement

In order to measure the power consumed during the benchmarks, we used a professional power meter. It is made by HAMEG Instruments and the exact model is MH8115-2 [29]. This particular power meter reports true RMS voltage up to 500 V with resolution of 1V, true RMS (route mean square) current up to 16 A with resolution of 10mA and active power measurement up to 8000 W with resolution of 1W. The minimum logging interval is 1 second. The accuracy of the power mesurement is ±0.5%, which is low compared to generic power meter equipment. The advantage of using a power meter is that the actual power consumption is measured, taking into account every power loss that take place on a real system. In particular the power meter was measuring the wall power consumption of the three nodes and the infiniband switch, in order to make safe conclusions about the overall power usage.

HAMEG provides its own software for monitoring the power meter from a PC. However this software is not compatible with Linux. In order to have access to the monitoring software, the power meter was connected on a Windows machine that had Remote Desktop Connection enabled for accepting incoming connections. In that way, using simple software for RDC, direct control of the monitoring software was achieved. The logging produces a simple text file. Each line of the file corresponds to a taken measurement and it reports the timestamp, the voltage, the current and the power consumption.

In order to be able to identify the exact time when the computational kernels started executing, a simple routine for reporting the time was written in C. Routine calls were injected in the benchmarks' code, one just before the computational kernel and one after it. The cluster's and the Windows machine's clocks were synchronised using the same Network Time Protocol servers. Thus there was direct correlation between the times that the two systems reported.

Before starting any benchmark run, the cluster was idling for at least one minute, in order to let it cool down, let the air-cooling fans to decrease their speed and get it stabilised to the lower power consumption. After ensuring through the power readings that the cluster had stabilised, the power consumption logging started, 5 seconds before launching the benchmark and stopped 5 seconds after the benchmark was completed.

The power consumption logs were imported to a spread sheet and the execution time logs were isolated. The computational kernel power consumption was calculated by averaging the power readings for the time that the kernel was executing (as reported from the timing routines).

Chapter 5

Results and Analysis

In this chapter, results from the benchmarking of our heterogeneous cluster are presented. The results are divided in two sections. Initially, investigation of the performance and power efficiency results of each application is done separately. This includes the behaviour of each application in relation to CPU and GPU clock speeds. Having analysed each application, co-location of different combinations of the test applications is applied, in order to identify possible gains in total performance and power efficiency.

5.1 High Performance LINPACK

In order to identify the power characteristics of the well-known LINPACK benchmark, thirty-six different combinations of CPU and GPU clock speeds were tested. The reason for testing all the possible combinations is that LINPACK is a hybrid code, thus it is affected both from the frequencies of CPU and GPU. It should be noted that LINPACK achieves very high levels of hardware utilisation. As a result the observed performance is very close to the theoretical maximum, while the power consumption is higher than any other benchmark.

The graphs are grouped by the GPU clock speed, as the resulting performance is highly dependant by the GPU computation capabilities. It should be noted that in order to emphasise the performance and power consumption differences, the axes of the LINPACK figures do not start from zero, but from the minimum observed value.

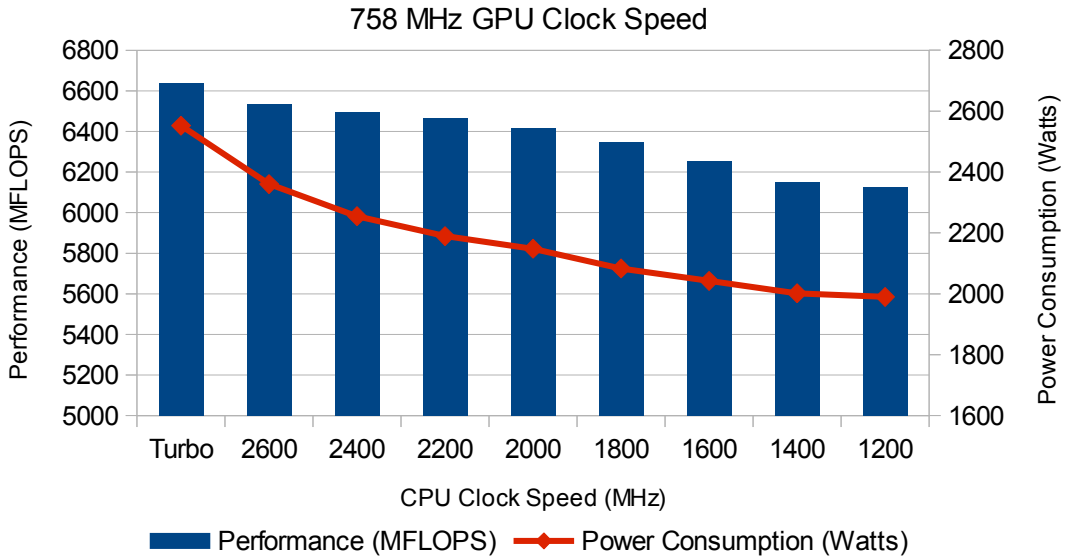

Figure 3: LINPACK performance and power consumption for 758 MHz GPU clock

As expected the highest performance is achieved when the GPU is clocked at the highest rate. The observed performance from the three-node cluster, using two Nvidia Tesla K20 per node, is 6636 MFLOPS. In Figure 3 the GPU is clocked at the maximum rate, at 758 MHz. It is clear that the power consumption decreases in an exponential way as the CPU clock rate drops, while the performance drop is almost linear. This fact shows that LINPACK is more power efficient for lower CPU clock rates.

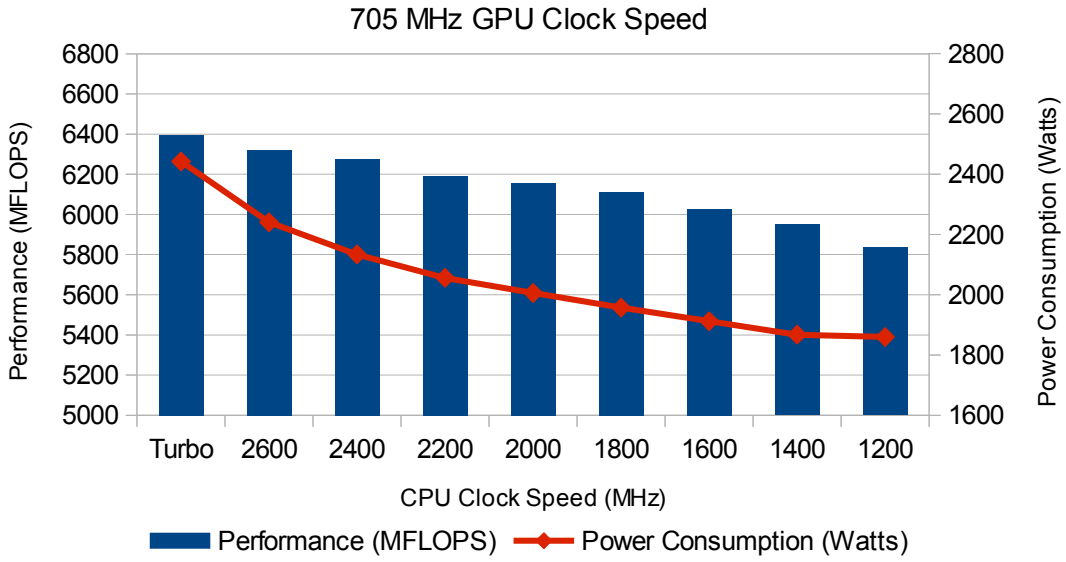

Figure 4: LINPACK performance and power consumption for 705 MHz GPU clock

In Figure 4, GPUs are used at their default clock rates, 705 MHz. The power consumption again is dropping in an exponential way, while performance drops in a lower rate, suggesting that lower CPU speeds result in higher efficiency. By comparing the performance results of Figure 3 and Figure 4 the performance was an average of 3.8% lower across all CPU rates, when GPUs are clocked at their default speed. However the GPU clock speed is 7% lower and this fact shows that the core clock speed is not the absolute limiting factor for the performance. Unfortunately there is no way to change other parameters, such as the memory clock rate, which could help the higher clocked GPU cores to be adequately fed with data.

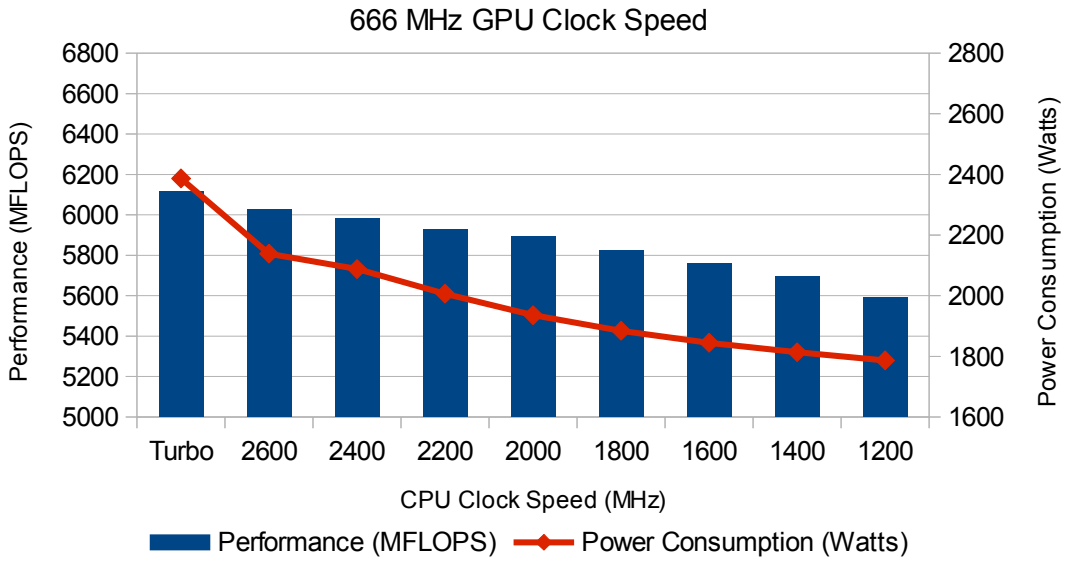

Figure 5: LINPACK performance and power consumption for 666 MHz GPU clock

In Figure 5 the GPU clock frequency is 666MHz. Setting the GPU clock speed even lower, decreases both performance and power consumption. Performance is again bound to the CPU clock rate in a linear way and power consumption changes exponentially. However it is worth noting that Turbo mode has a bigger impact over the power consumption. This is happening because turning Turbo on, adds about 200 Watts to the overall power consumption which becomes a greater part proportionally, as the overall power consumption is decreased by lowering the GPU clock speed.

By comparing Figure 4 and Figure 5, the performance differs on average by 4.4% while the GPU clock rate differs by 5.5%. Once again the performance does not keep up with the clock rate, however the effect now is a lot less observable as the memory speed to core speed ratio is higher.

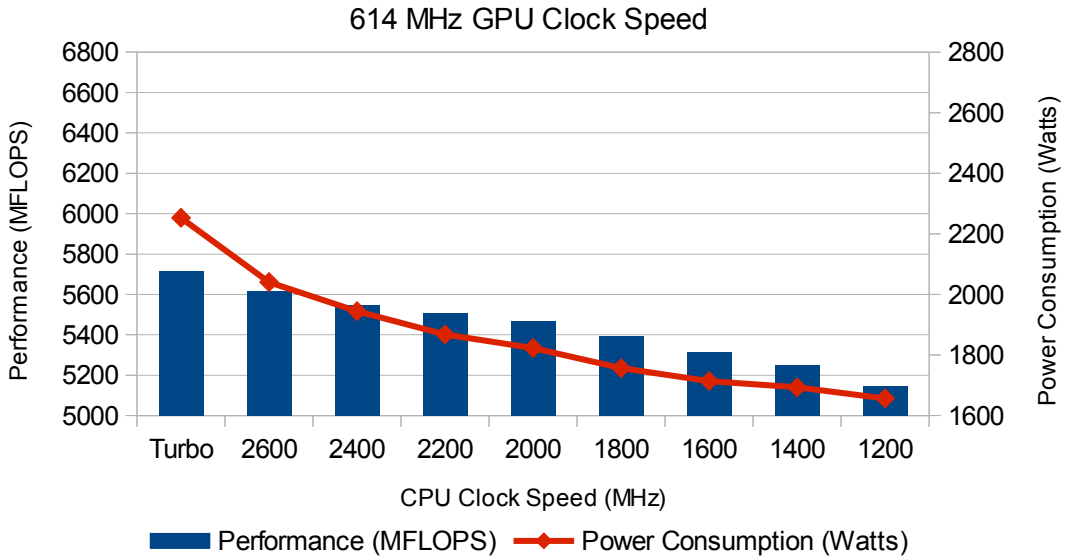

Figure 6: LINPACK performance and power consumption for 614 MHz GPU clock

In Figure 6 the GPU clock is set to the lower available rate, 614 MHz. This change has the biggest impact on performance compared to the other GPU clock changes. Performance is decreased in a linear way as CPU clock decreases while the power consumption decreases exponentially, which gives an advantage to the lower CPU clock rates, in terms of power efficiency. The performance differences between Figure 5 and Figure 6 is 7.3% while the GPU clock rate difference is 7.8%, which suggests that the memory is adequately clocked and the limiting factor is the GPU core.

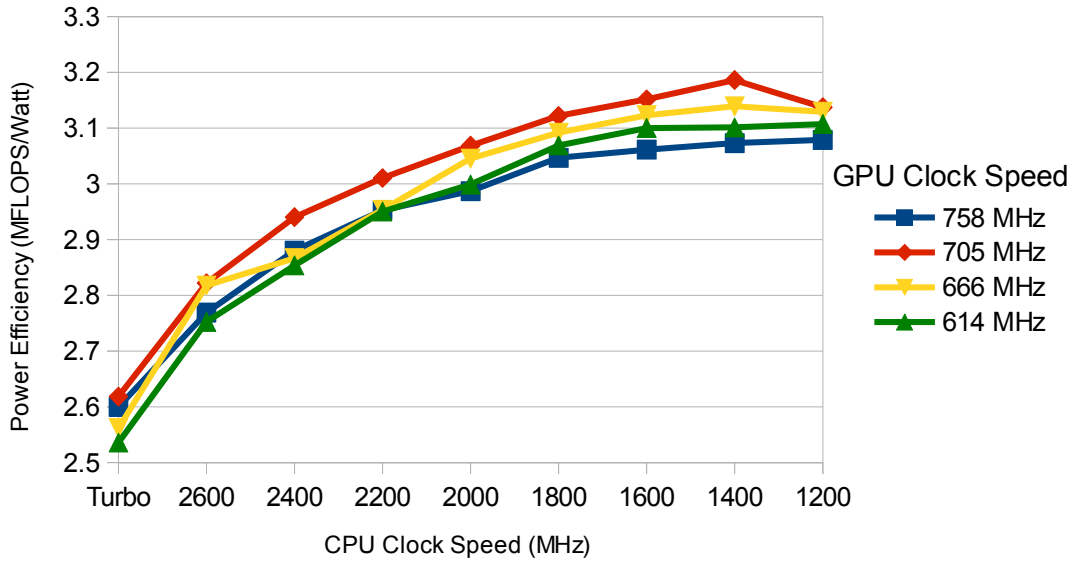

Figure 7: LINPACK power efficiency in relation to CPU and GPU clock speeds

Figure 7 compares the observed power efficiency across all combinations of CPU and GPU clock speeds. It can be seen that the power efficiency is maximised at the lower CPU speeds, and in particular it is maximum for 1400 MHz. This is happening because by lowering the CPU clock rate and by tweaking the percentage of the computation data split between the CPU and the GPU, more and more data are sent to the GPU. GPU devices are very efficient for heavy floating point computations such as LINPACK, which performs huge amounts of basic vector and matrix operations. Thus by using GPUs to compute the greater proportion of the computations, power efficiency is increased.

The power efficiency grows exponentially, just like the way the power consumption is decreasing when lowering the CPU clock rate. It can clearly be seen that the most efficient GPU clock speed is the default one, 705 MHz, while the overclocked 758 MHz frequency is the least efficient, probably because of the low memory to core clock ratio. It seems that the Nvidia Tesla K20 are highly optimised for LINPACK's computations.

5.2 GROMACS

In order to investigate the behaviour of a real-world application, GROMACS was used. GROMACS is a Molecular Dynamics code and it is widely used among the HPC community. Two different builds of the code were used: a CPU-only MPI version and a GPU accelerated version.

5.2.1 CPU-only GROMACS

For the CPU-only version, different CPU clock speeds were tested in terms of performance and power consumption. The simulation runs across the three nodes of the cluster, utilising all 16 cores per node.

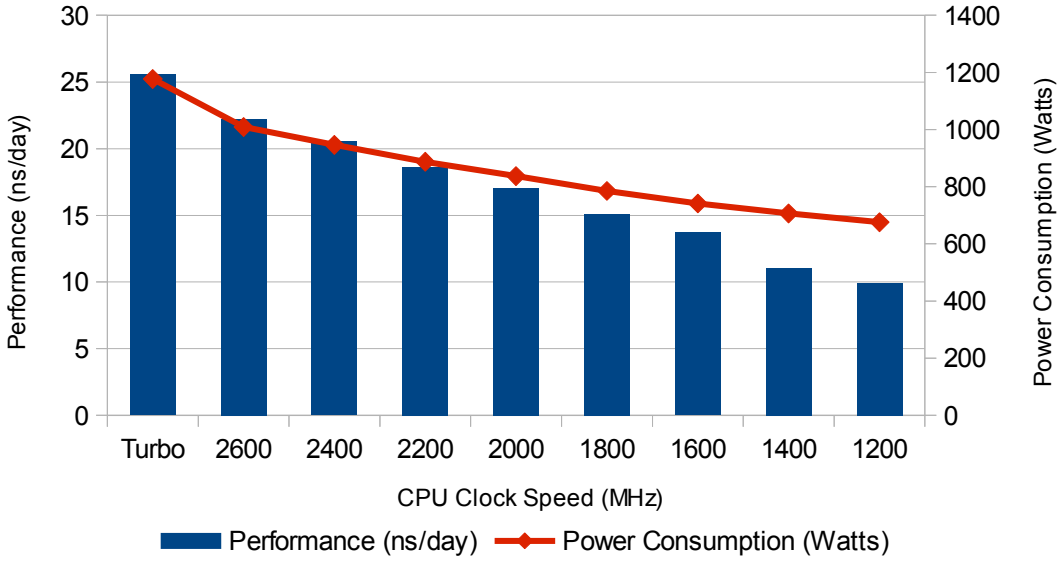

Figure 8: GROMACS CPU-only performance and power consumption

Figure 8 shows that performance is decreasing in a higher rate than power consumption, as the CPU clock rate decreases. This has as a result decreased power efficiency factor for the low CPU speeds. From the results it seems that GROMACS is a compute intensive code that is highly dependent on the compute performance of the CPU. Thus the performance obtained is completely proportional to the CPU clock speed. For example the performance when the clock speed is set to 2400 MHz is double than the performance of the 1200 MHz clock speed.

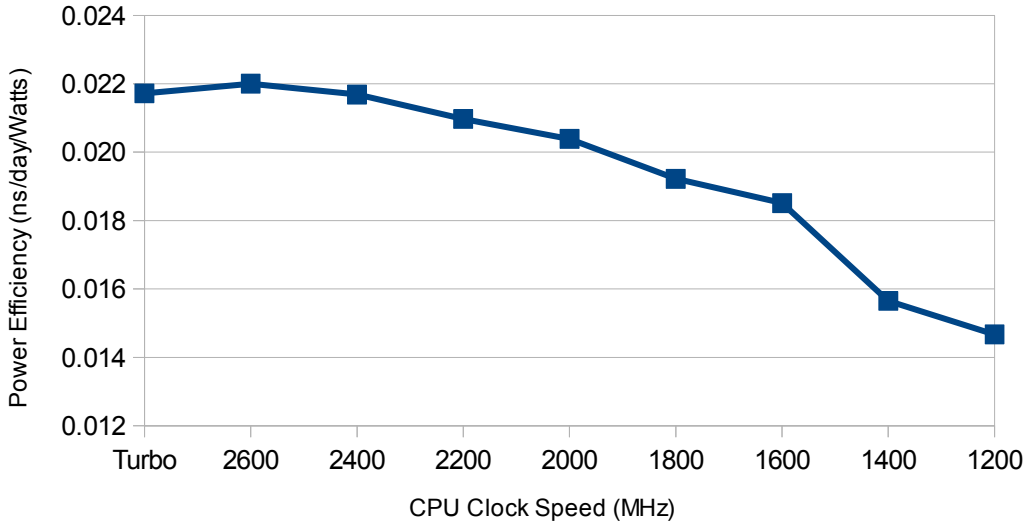

Figure 9: GROMACS CPU-only power efficiency

Turbo mode usually is not very power efficient. This happens because when Turbo mode is engaged, not only the clock frequency is increased, but also the voltage in order to ensure system stability. This results in even higher power consumption which is not on par with the performance gains. However, due to the fact that GROMACS is a compute intensive code, the gains from the Turbo mode in performance are high and the power efficiency stays in high levels as well. The best power efficiency for GROMACS is obtained when the CPU clock speed is set to the maximum non-Turbo speed of 2600 MHz. As the CPU frequency decreases, so does the power efficiency.

5.2.2 GPU accelerated GROMACS

The GPU accelerated version of the code is affected both by the frequencies of CPU and GPU, thus all thirty-six frequency combinations were tested. The graphs are grouped by the GPU clock speed, in order to evaluate the performance impact.

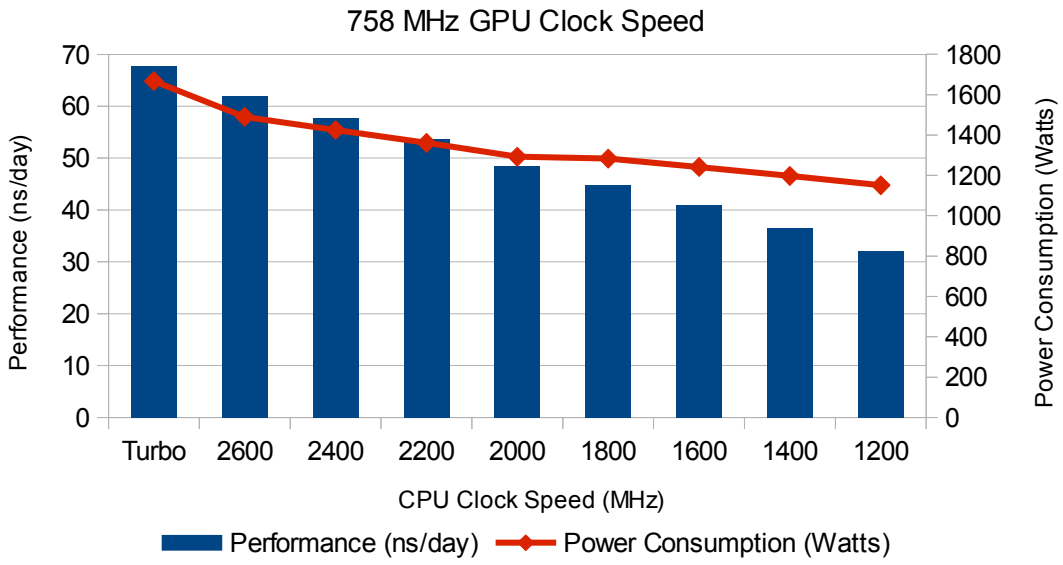

Figure 10: GROMACS performance and power consumption for 758 MHz GPU clock

The highest performance is achieved when the GPU is clocked at the highest rate. The observed performance from the three-node cluster, using two Nvidia Tesla K20 per node, is 67.75 ns/day. As with the CPU-only version, the performance decreases linearly as the CPU frequency drops. However, it is decreased at a slower rate than the CPU-only version, due to the fact that the performance now is depending on the GPU frequency as well. Figure 10 shows that below 2000MHz the power consumption decreases by a small amount, while the performance decreases a lot. As a result, the power efficiency is low below 2000MHz.

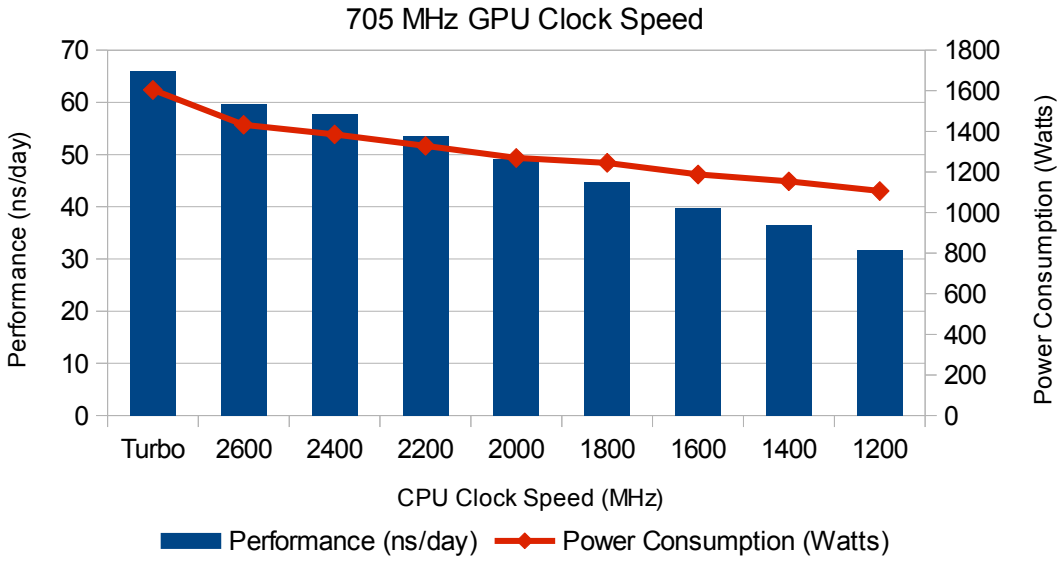

Figure 11: GROMACS performance and power consumption for 705 MHz GPU clock

At Figure 11 the behaviour remains the same. The performance differences between the 758 MHz and 705 MHz case are small, however the power consumption seems noticeably lower, resulting in higher power efficiency.

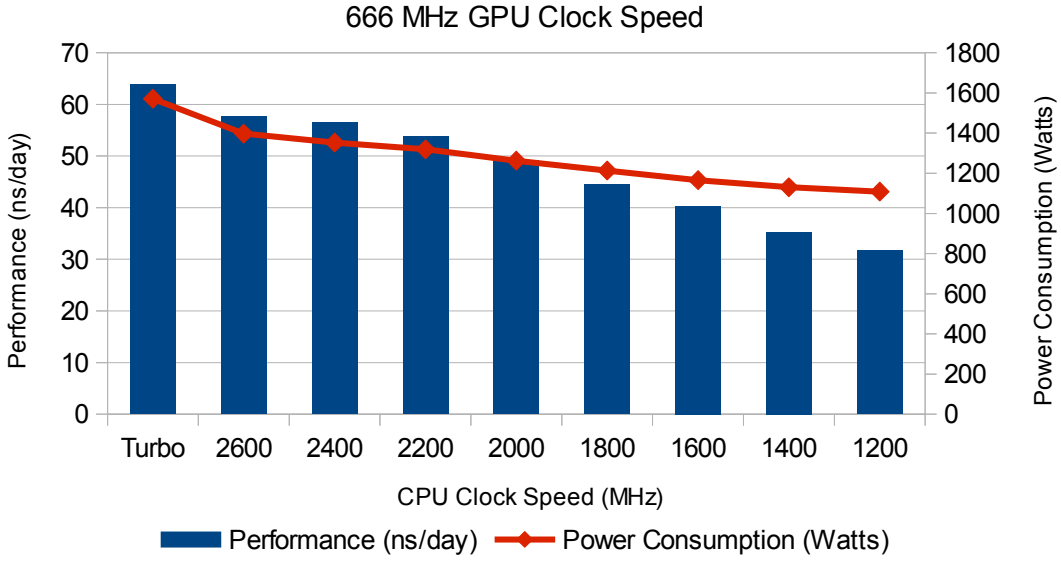

Figure 12: GROMACS performance and power consumption for 666 MHz GPU clock

Once again, the performance decrease on Figure 12 is small and so is the power consumption decrease. The high frequencies are the the most efficient.

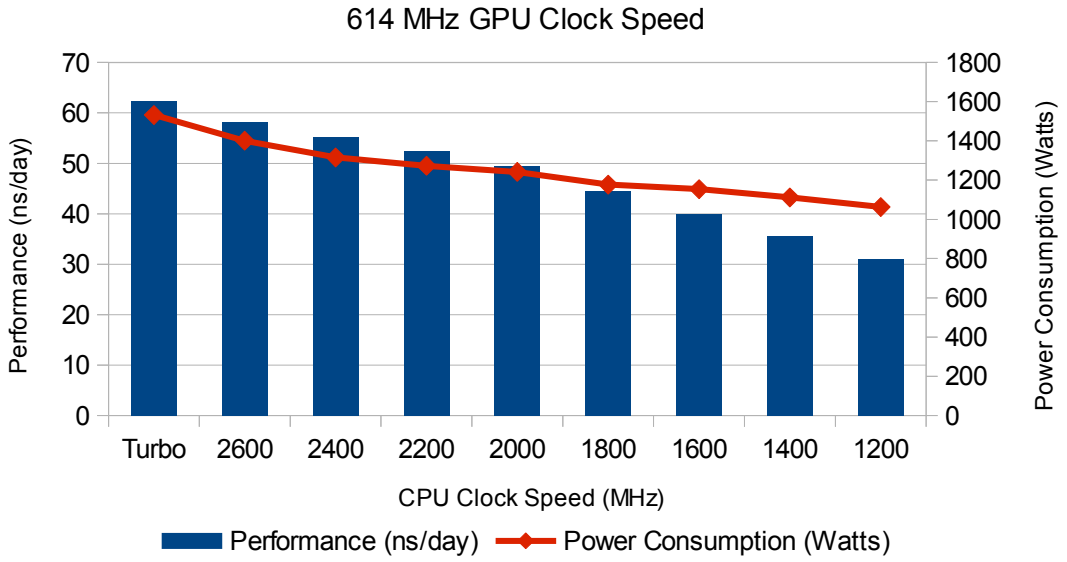

Figure 13: GROMACS performance and power consumption for 614 MHz GPU clock

Performance on Figure 13 is almost unchanged but the power consumption is slightly lower. In order to extract some conclusions about the behaviour of GROMACS, power efficiency figures are needed for comparison.

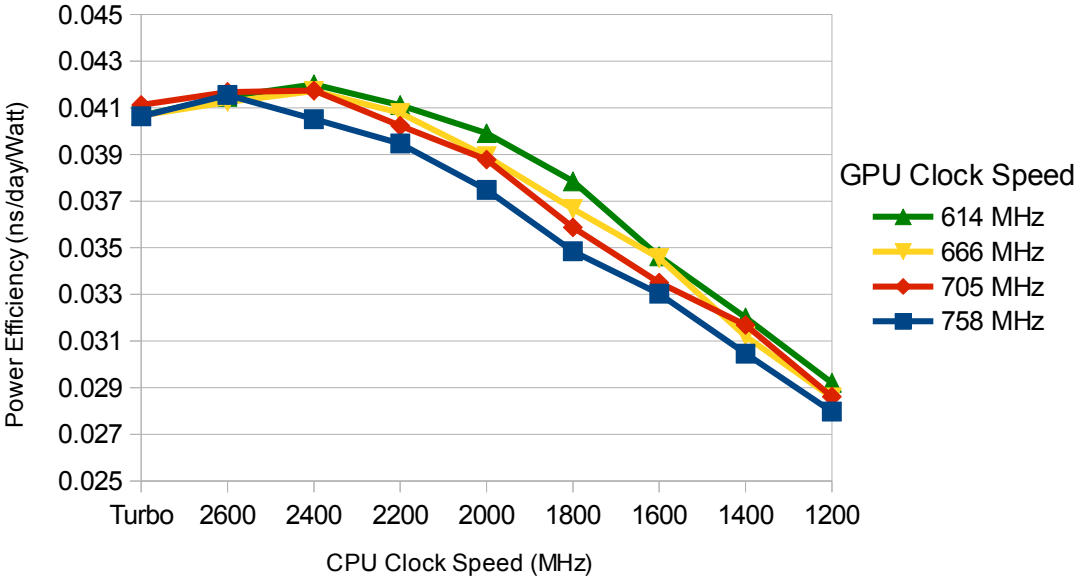

Figure 14: GROMACS power efficiency in relation to CPU and GPU clock speeds

It is clear from Figure 14 that the power efficiency shows a large drop below 2000 MHz CPU frequency for all GPU frequencies. It is interesting to investigate more what happens in the high efficient area. As a general conclusion, the lower the GPU

frequency is, the more power efficient the simulations are. This fact is an indication that the GROMACS simulations are not GPU bound when using two GPUs per node. This can be also confirmed by the fact that Nvidia's GPU monitoring utility, nvidia-smi, never reported more than 75% GPU utilisation during the benchmark tests.

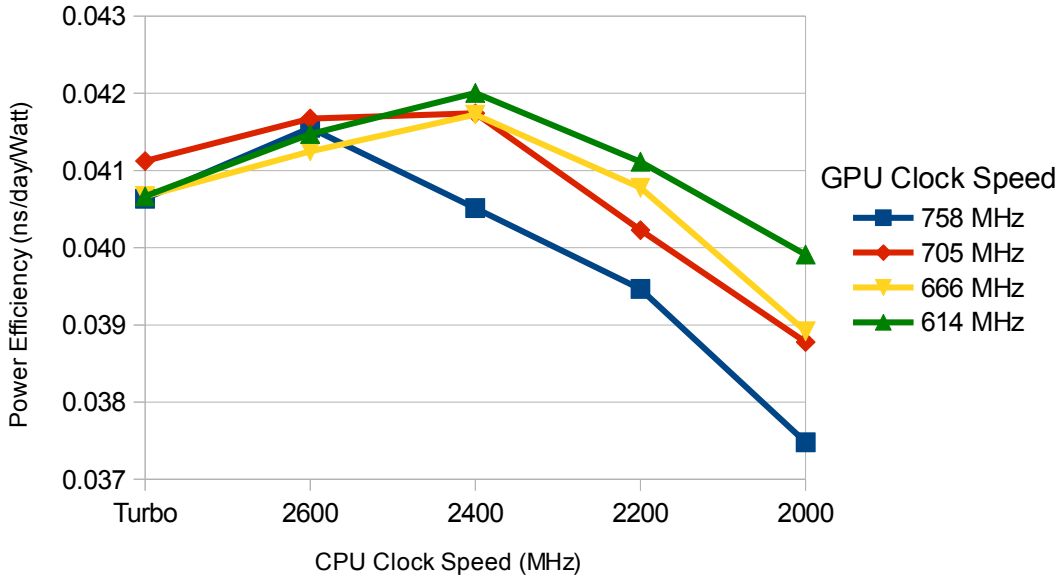

Figure 15: GROMACS high efficiency area zoomed-in

Figure 15 is a close-up of the high efficient area, in order to investigate the behaviour of GROMACS. It is clear that from 2400 MHz and below, the lower the GPUs are clocked, the higher the observed power efficiency is. In particular, a GPU frequency of 614 MHz combined with a 2400 MHz CPU frequency provide the best result, 0.042 ns/day per Watt. However for 2600 MHz of CPU frequency, the default 705 MHz becomes the most power efficient, and this also applies when the Turbo mode is enabled. The reason for this change is that the higher CPU frequencies make the CPU computations faster, making the simulation dependent of the GPU speed. It seems that the default GPU frequency provides the optimal balance in terms of power efficiency, when the CPUs are highly clocked. However, the differences are very small (1%).

In conclusion, GROMACS takes advantage of the high CPU frequencies, making even Turbo mode a fairly power efficient choice. When using two powerful GPUs (such as Nvidias K20) per node, their computational capabilities are not needed when the CPUs are not highly clocked (2600 MHz and above) and thus can be clocked down in order to save some power and become more efficient.

5.3 Ludwig

The third application used is EPCC's Ludwig code. Ludwig is a complex fluid mixture simulation code and comes in two variants: CPU-only and GPU-only.

5.3.1 CPU-only Ludwig

For the CPU-only version, different CPU clock speeds were tested in terms of performance and power consumption. The simulation run across the three nodes of the cluster, utilising all 16 cores per node.

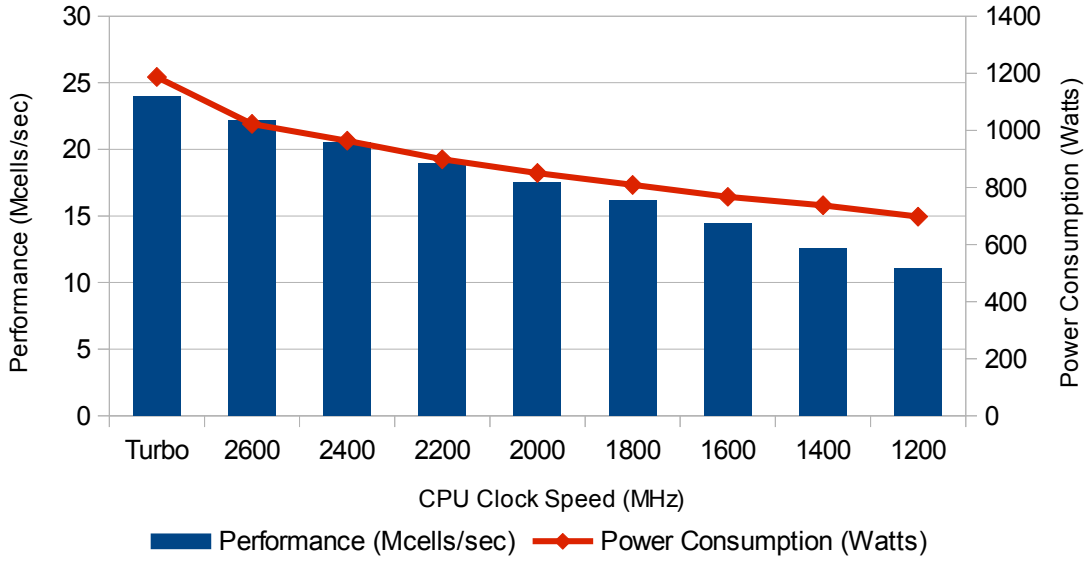

Figure 16: Ludwig CPU performance and power consumption

Performance decreases in a linear way, as CPU frequency decreases. However, Figure 16 shows that power consumption decreases with slower rate. This affects the power efficiency, which becomes lower as the CPU frequency decreases. Therefore, the tested kernel could be characterised as compute intensive due to the fact that the performance to frequency ratio is almost 1:1.

With Figure 17 it becomes clear that, excluding Turbo mode, as the CPU frequency decreases, so does the power efficiency. Furthermore, below 1800 MHz power efficiency decreases in a higher rate. As far as the Turbo mode is concerned, it is clear that it affects the power efficiency in a negative way. This is happening because of the voltage increase that Turbo mode applies, which increases the power consumption exponentially.

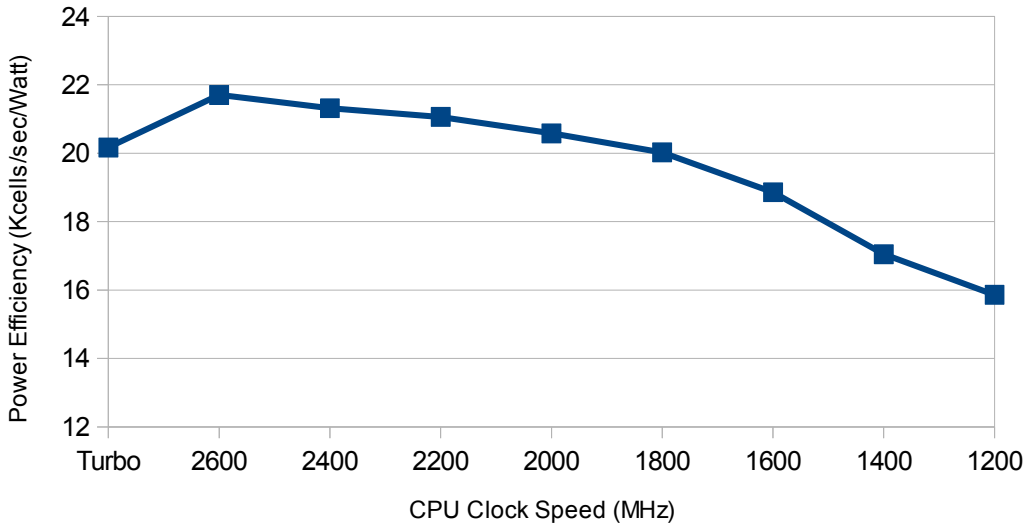

Figure 17: Ludwig CPU power efficiency

5.3.2 GPU-only Ludwig

The GPU-only version of the code is affected mainly by the GPU core frequency. For that reason only GPU frequencies were tested. As Ludwig scales perfectly across many GPUs, all three Nvidia K20 were used on each of the three nodes. Please note that the axis of the performance and power consumption figures do not start from zero, in order to emphasise the differences.

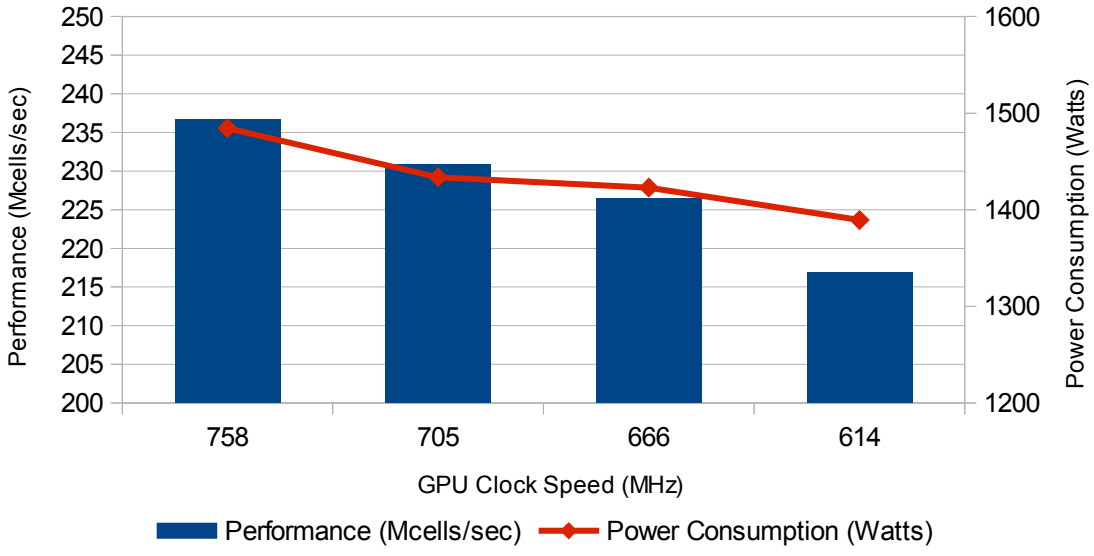

Figure 18: Ludwig GPU performance and power consumption

Lowering the frequency from 758 to 705 MHz (7%) decreases the performance by 2.5%. Lowering from 705 to 666 MHz (5.5%), decreases the performance by 1.9% and lowering from 666 to 614 MHz (7.8%) decreases the performance by 4.2%.

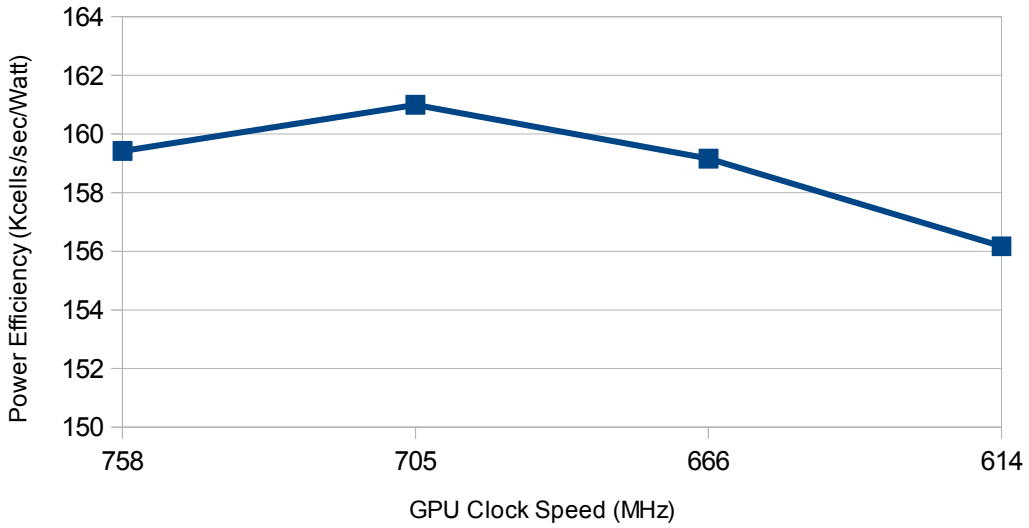

Figure 19: Ludwig GPU power efficiency

The default GPU frequency is the most power efficient. This happens because the GPU code of Ludwig is highly efficient, meaning that it exploits the hardware capabilities. The default ratio of GPU and memory frequency is optimal for this kind of application. By decreasing the GPU frequency we are capping the GPU computation ability and by increasing it the memory cannot keep up with the faster GPU.

5.4 AMG

The last application used is a parallel Algebraic Multigrid solver. AMG is a CPU-only application. Algebraic calculations are widely used in HPC, so the investigation of the behaviour of such codes is of high importance. As many algebraic algorithms, AMG is a memory bound code. In order to investigate its characteristics, the tests were performed using different number of CPU cores on each node, and the results are grouped in this way. Once again, performance and power consumption in relation to the CPU clock frequency were measured.

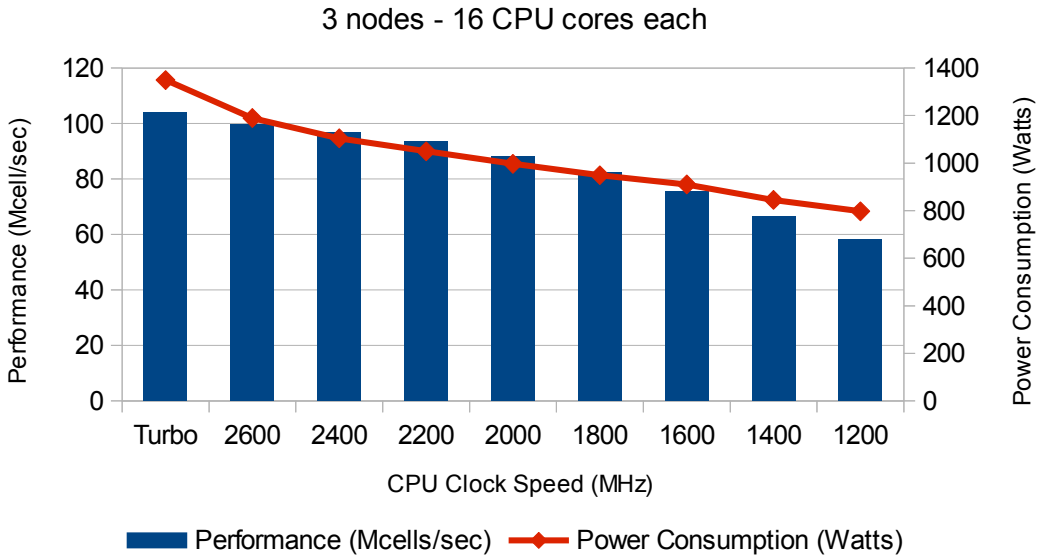

Figure 20: AMG performance and power consumption for 100% populated nodes

In Figure 20 it is shown that for CPU frequencies down to 2000, the performance decrease is small and not in line with the frequency decrease. This is typical for memory-bound applications, because the limiting factor for performance is the memory bandwidth. Thus, by having high clocked CPU cores, computation power is added but cannot be exploited because the cores cannot acquire enough data for computation from the main memory. However, as the CPU frequency decreases, the amount of data fed to the CPU remain the same but the CPU process them at a slower rate, but the power consumption decreases normally, resulting into increased power efficiency. For lower frequencies, the performance starts to drop in a linear way. Because of this memory-bound behaviour, utilisation of lower number of CPU cores per node was tested.

In Figure 21 the performance decrease as the CPU frequency is lowered is almost linear. This is happening because memory bandwidth is sufficient now, because fewer cores are using the same memory banks. Power consumption drops in a slower rate than performance and this has as a result that the high clock rates achieve high power efficiency. Turbo mode is adding a lot of power consumption, minimising power efficiency. In order to ensure that the memory-bound effects no longer exist, an even lower utilisation of the nodes was tested.

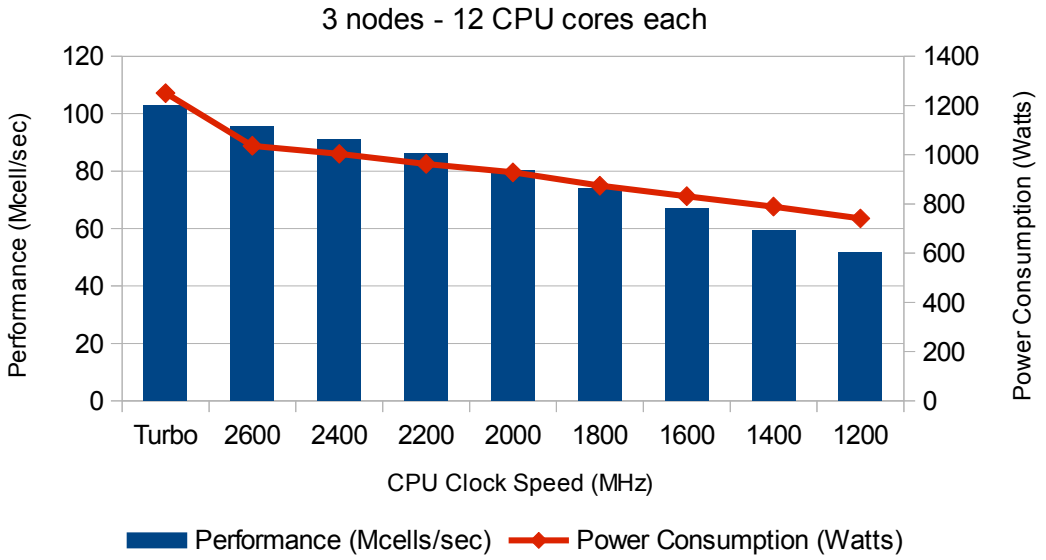

Figure 21: AMG performance and power consumption for 75% populated nodes

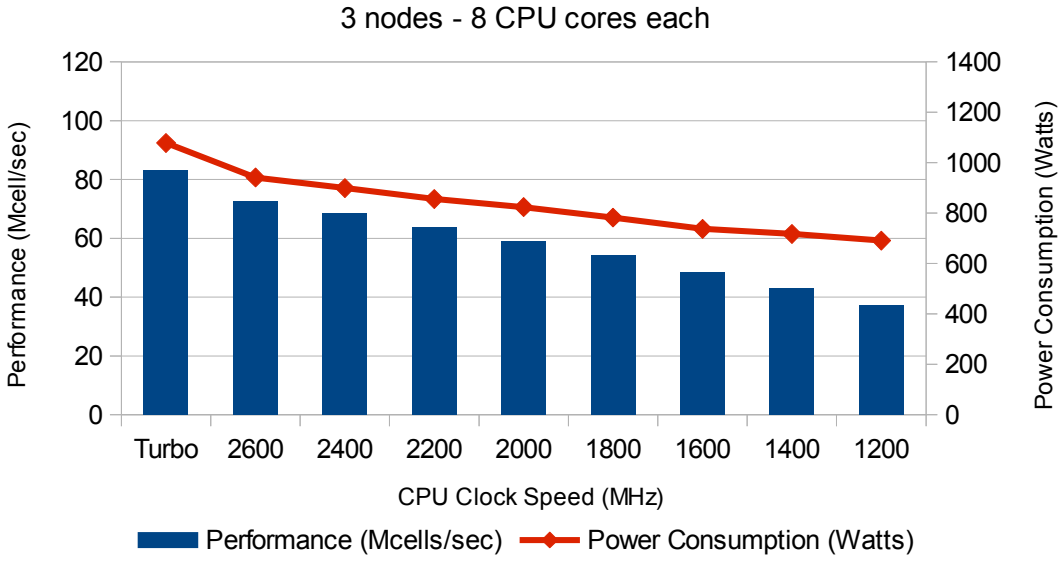

Figure 22: AMG performance and power consumption for 50% populated nodes

When reducing the cores per node from 12 down to 8, the performance drop is in line with the core count decrease (-25%). Memory is for sure no longer the limiting factor. Thus performance is decreasing linear as CPU frequency is lowered.

In Figure 23 it is clearly shown that Turbo mode ruins the power efficiency of the memory-bound AMG solver. The fully populated cluster achieves high power efficiency when the CPU cores are running at 2200 MHz, while the half populated cluster is not memory-bound and thus the efficiency is decreasing as the CPU frequency gets lowered. The maximum power efficiency is achieved when using twelve out of sixteen cores on each node (75% utilisation), at 2600 MHz CPU speed.

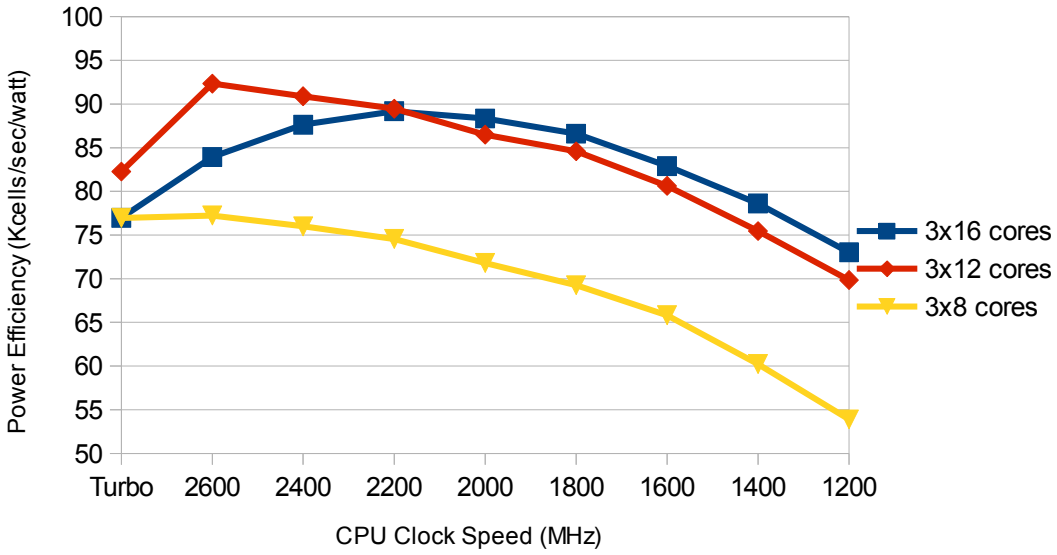

Figure 23: AMG power efficiency

5.5 Ludwig CPU with Ludwig GPU co-location

The Ludwig GPU version provides by far more performance compared to the CPU version. However, one disadvantage of the GPU code is that it only utilises the GPUs, leaving CPU cores unutilised, and thus sacrificing performance and computation time. In order to investigate the results of co-locating Ludwig, two different simulations were running simultaneously; one on the three GPUs and the other on the 12 CPU cores.

Figure 24 shows the performance and power consumption of the co-location. Because both parts of the run are simulating the same science, the performance results can be added. Performance-wise, when running at the default GPU frequency (705 MHz), the maximum performance achieved by co-location was 237.7 Mcells/sec, which is 3% higher than the 230.8 Mcells/sec that the GPU-only version achieved. In absolute performance numbers, co-location can offer a marginal 3% better performance, which can be a considerable amount of simulation time when running very large simulations.

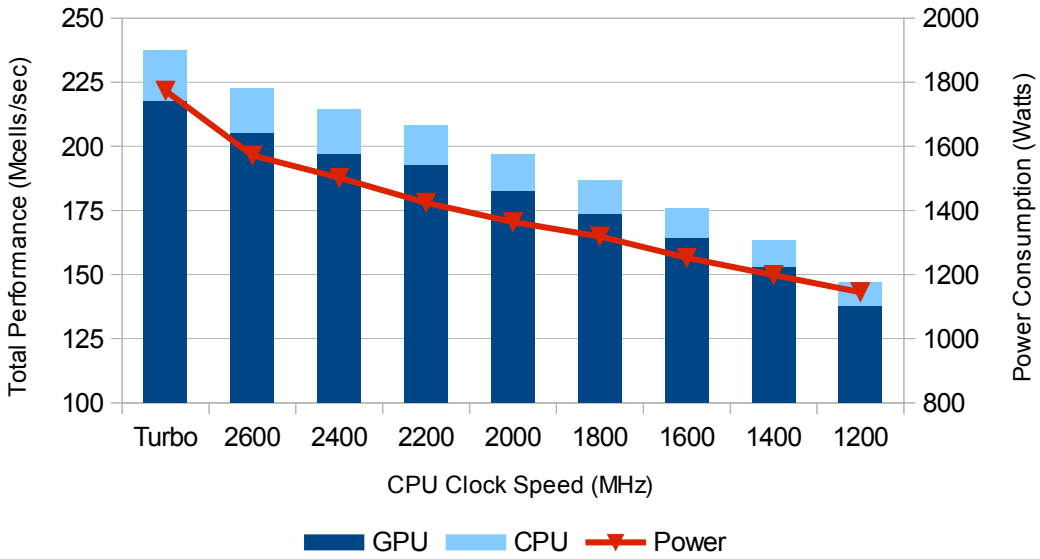

Figure 24: Ludwig CPU - GPU co-location performance and power consumption

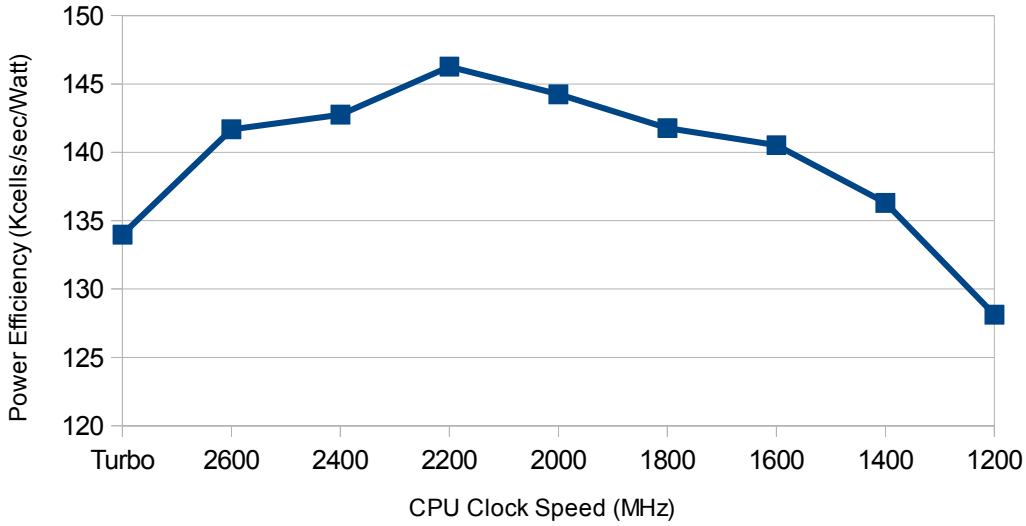

Figure 25: Ludwig CPU - GPU co-location power efficiency

Figure 25 shows that the maximum efficiency is achieved at 2200 MHz CPU clock rate and is 146 Kcells/sec per Watt. This is considerably lower than the 161 Kcells/sec per Watt achieved by the GPU-only version of Ludwig. When running single Ludwig simulations, the power efficient CPU frequency was 2600 MHz. The power efficient frequency has shifted to 2200 MHz, because via co-location higher utilisation of the compute units of the system is achieved, which probably results in making memory the limiting factor. Thus the higher clocked CPU cores cannot add as much performance as power consumption.

If for some reason it is necessery to run a CPU-only simulation of Ludwig (for example, because of a large simulation that does not fit into the GPU's memory), co-locating Ludwig simulations can be advantageous for power efficiency as well.

At the most power efficient frequency of 2200 MHz, the GPU performance is 193 Mcells/sec and the CPU performance is 15.4 Mcells/sec, while the power consumption is 1424 Watts. When running standalone GPU performance is 231 Mcells/sec and it is consuming 1434 Watts and CPU performance is 22.2 Mcells/sec and it is consuming 1023 Watts.

	GPU-only	CPU-only	co-location
Performance (Mcells/sec)	231	22.2	193 + 15.4
Power Consumtion (Watts)	1434	1023	1424

Table 3: Ludwig performance comparison at the most efficient configurations

Assuming that the co-location run lasts 1 hour, the total power consumed for GPU and CPU will be *1424 Wh.*

During this hour, the GPUs will compute:

*GPU problem size = 193 Mcells/sec * 3600 secs = 694800 Mcells*

and the CPUs will compute:

*CPU problem size = 15.4 Mcells/sec * 3600 secs = 55440 Mcells*

If the same simulations run sequentially:

GPU run time = 694800 Mcells / 231 Mcells/sec = 3008 secs = 50.1 mins

*GPU power consumption = 1433 W * 50.1/60 h = 1197 Wh*

CPU run time = 55440 Mcells / 22.2 Mcells/sec = 2497 secs = 41.6 mins

*CPU power consumption = 1023 W * 41.6/60 h = 709 Wh*

In total the two simulations would consume:

Total power consumption = 1197 Wh + 709 Wh = 1906 Wh

which is 33.8% more energy used for the same computations. Performance-wise, the co-located simulations finish in 1 hour, while the separate runs finish in a total of 1 hour 31.7 mins, which is 52.8% more time.

In conclusion, co-locating the two versions of Ludwig results in **25% less power** consumption and **35% less computation time**. It is clear that when a CPU simulation needs to run, co-location with other GPU simulations is definitely advantageous both in performance and power efficiency.

5.6 AMG with Ludwig GPU co-location

AMG is a memory-bound application that cannot take advantage of all the CPU cores, as the extra computing power adds very little to the resulting performance. In Chapter 5.4 it is shown that the best power efficiency is achieved when using twelve out of sixteen CPU cores on each node. On the other hand, the GPU version of Ludwig, only uses one core per GPU, leaving most of the CPU cores unutilised. By co-locating two different applications such as AMG and Ludwig, almost full utilisation of the cluster is achieved, and the simulations run using power efficient configurations.

From Figure 26 it is clear that except from Turbo mode, the power consumption is reduced in a linear way, as CPU frequency is lowered. However, performance of both AMG and Ludwig drops in a low rate until 2000 MHz. This has a positive effect on the power efficiency, as shown on Figure 27. AMG's power efficiency is maximised for the CPU frequency of 2200 MHz and Ludwig's power efficiency is maximised for the CPU frequency of 2000 MHz. The best overall power efficiency is achieved at 2200 MHz. Once again the efficient CPU frequency is lower for both AMG and Ludwig compared to individual runs. For AMG this happens because the maximum CPU frequency is able to exploit all the memory bandwidth when it is the only calculation running on the system, but when another simulation is running occupying the rest hardware, memory can become the limiting factor again.

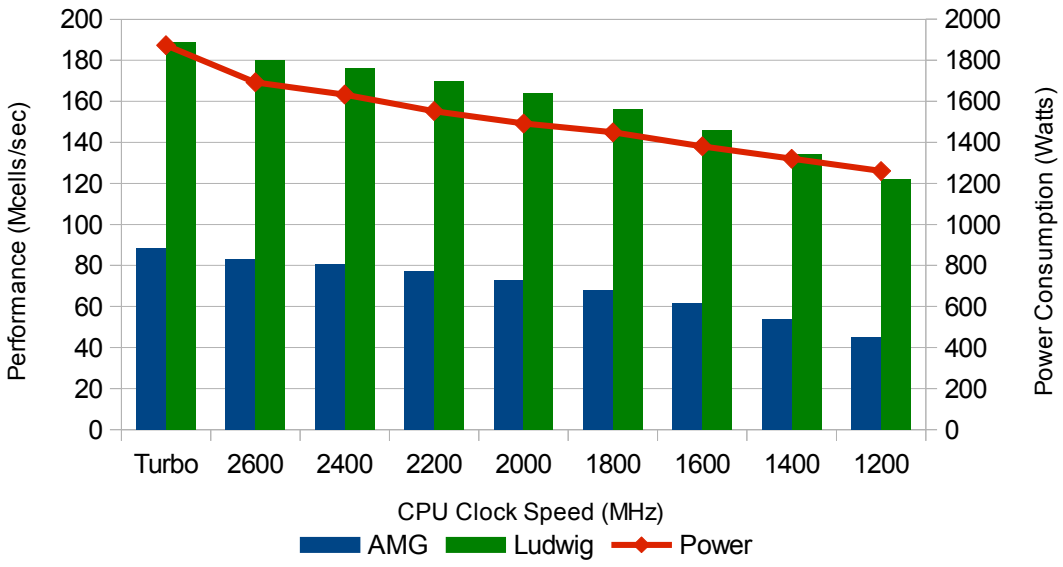

Figure 26: AMG - Ludwig GPU co-locatino performance and power consumption

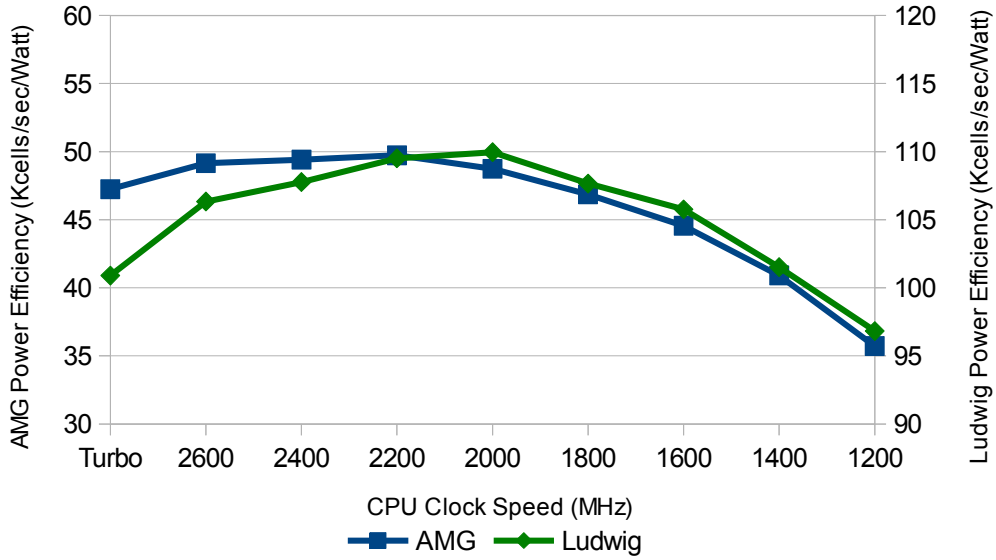

Figure 27: AMG - Ludwig GPU power efficiency

At this power efficient frequency (2200 MHz), the performance of AMG is 77.2 Mcells/sec and Ludwig performance is 170 Mcells/sec, while the power consumption is 1552 Watts. When running alone, AMG's most efficient performance is 95.7 Mcells/sec with power consumption of 1036 Watts and Ludwig's most efficient performance is 231 Mcells/sec with power consumption of 1434 Watts. In percentages, AMG is 24% faster when running alone and Ludwig is 36% faster.

	AMG	Ludwig GPU	co-location
Performance (Mcells/sec)	95.7	231	77.2 + 170
Power Consumtion (Watts)	1036	1434	1552

Table 4: AMG and Ludwig performance comparison at the most efficient configurations

Assuming that the co-location run lasts 1 hour, the total power consumed for AMG and Ludwig will be *1552 Wh.*

During this hour, AMG will compute:

*AMG problem size = 77.2 Mcells/sec * 3600 secs = 277920 Mcells*

and Ludwig will compute:

*Ludwig problem size = 170 Mcells/sec * 3600 secs = 612000 Mcells*

If the same simulations run sequentially:

AMG run time = 277920 Mcells / 95.7 Mcells/sec = 2904 secs = 48.4 mins

*AMG power consumption = 1036 W * 48.4/60 h = 836 Wh*

Ludwig run time = 612000 Mcells / 231 Mcells/sec = 2649 secs = 44.2 mins

*Ludwig power consumption = 1434 W * 44.2/60 h = 1056 Wh*

In total the two simulations would consume:

Total power consumption = 836 + 1056 = 1892 Wh

which is 21.9% more energy used for the same computations. Performance-wise, the co-located simulations finish in 1 hour, while the separate runs finish in a total of 1 hour 24 mins, which is 40% more time.

In conclusion, co-locating AMG and Ludwig results in **18% less power consumption** and **29% less computation time**. It is clear that co-location is definitely advantageous both in performance and power efficiency.

5.7 AMG with GROMACS GPU accelerated co-location

The third co-location test includes the memory-bound AMG and the GPU accelerated version of GROMACS. This test is of different nature from the test in Chapter 5.6, because GROMACS is a hybrid code and in particular its performance is highly dependent on the CPU performance. Both of these applications are highly demanding in terms of hardware, and the co-location of them is a stress test for the cluster.

The CPU cores on each node are divided in half and assigned to each application. The tests include two different mappings of the applications to the CPU cores. The first mapping divides the CPU cores by socket, assigning each application to a different socket. This mapping has a negative a effect on the memory-bound AMG, as the memory bandwidth of a socket is not enough when all CPU cores are being used. The second mapping splits the cores of each socket in half, giving the first half of the cores to AMG and the other half to GROMACS. This mapping has a positive effect for AMG, as it provides more memory bandwidth.

The GPUs are tested in the two most efficient frequencies for the standalone GROMACS runs, which is the default 705 MHz and the lowest 614 MHz, as shown in Chapter 5.2.2. The charts are grouped in terms of GPU clock speed and CPU core distribution.

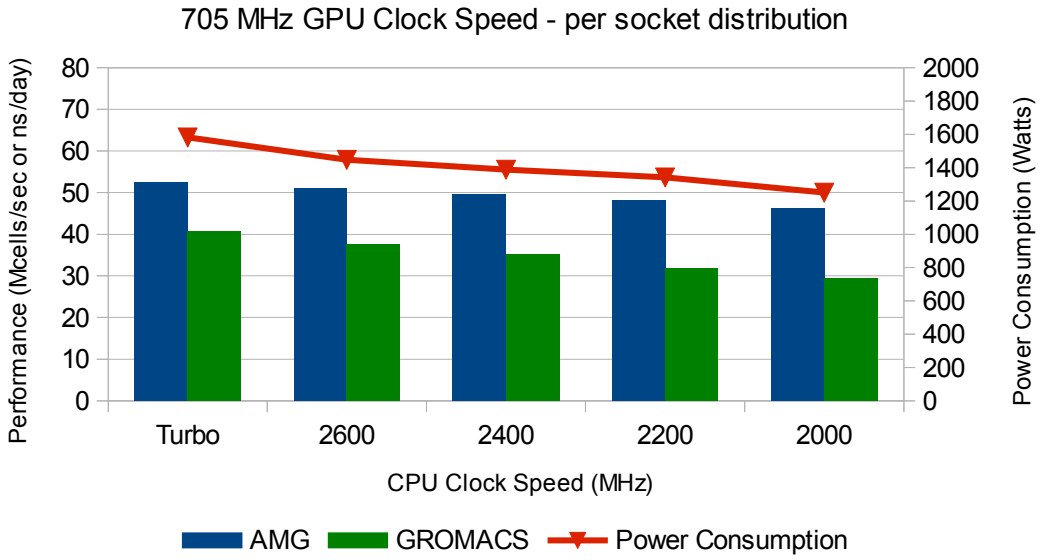

Figure 28: AMG - GROMACS co-location for 705 MHz GPU and per socket mapping

In Figure 28 it is clear that the performance of GROMACS is more affected by the CPU clock speed than AMG, which has a very little decrease by the frequency lowering. As expected, the per socket mapping affects the performance of AMG highly, which is 28% lower than the presented performance in Figure 22.

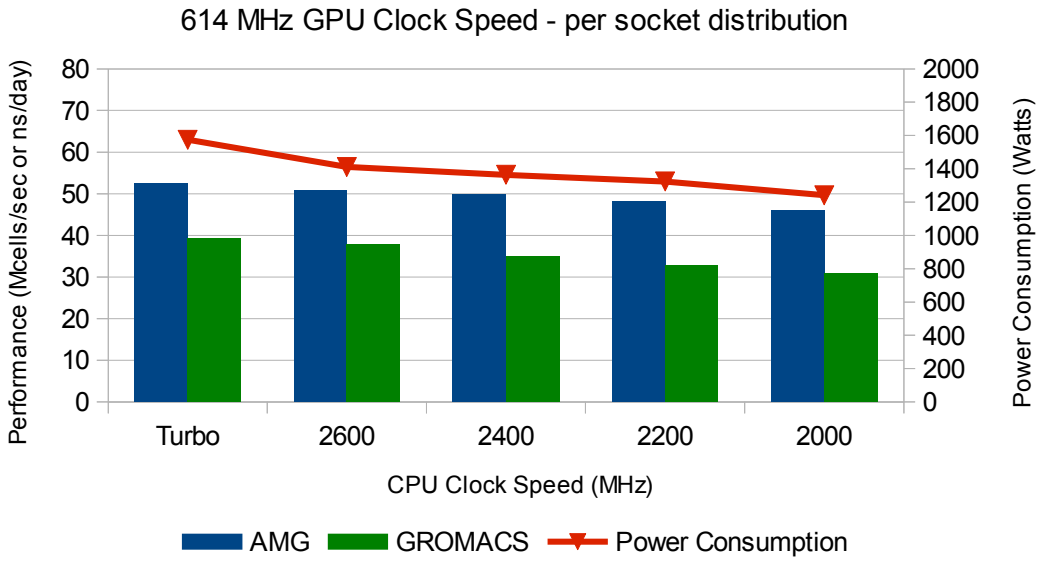

Figure 29: AMG - GROMACS co-location for 614 MHz GPU and per socket mapping

In Figure 29 the GPU clock rate is lowered to 614 MHz. As expected, this has no effect on the performance of AMG, but it also does not have any effect on the performance of

GROMACS either. The reason for this behaviour is that the two Nvidia K20 provide more computational capabilities than GROMACS can exploit, as explained in Figure 15. Thus by lowering the GPU frequency, the only effect is a slight decrease in power consumption, which will favour the power efficiency.

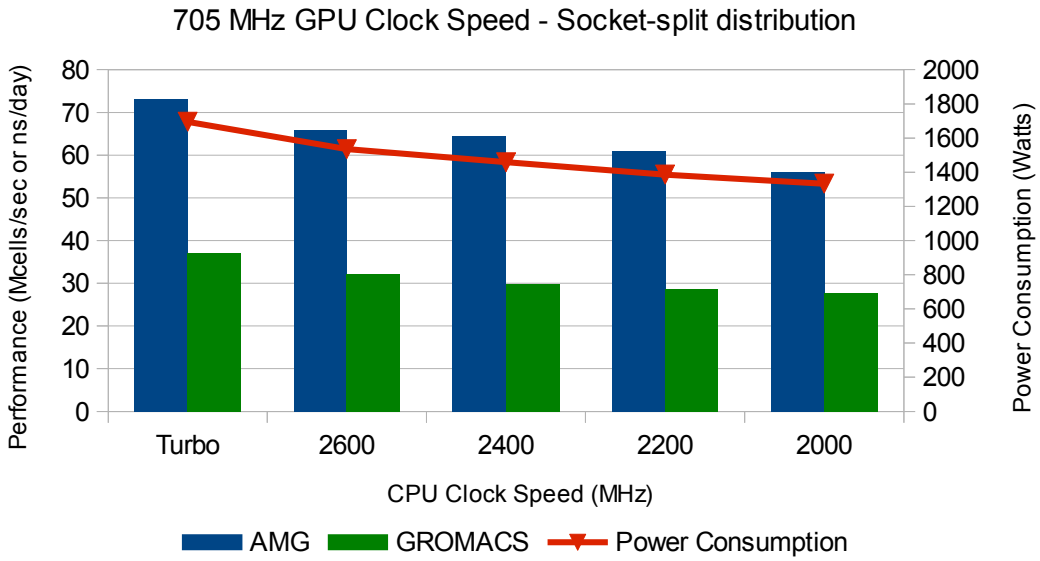

Figure 30: AMG - GROMACS co-location for 705 MHz GPU and Socket-split mapping

In Figure 30 the socket-split mapping greatly affects the performance of AMG, which is much higher than the per-socket mapping. In particular compared to Figure 22, the performance of AMG is only 6% lower. However the performance of GROMACS is around 11% lower compared to the per-socket mapping. The extra memory bandwidth provided to AMG has improved its performance a lot, but on the other hand this acts competitively for the performance of GROMACS. However, the performance of AMG gains are more than the GROMACS loses.

In Figure 31 the decrease of GPU clock speeds has not any effect on the performance of the two applications. The power consumption is slightly decreased and this has an positive effect on power efficiency.

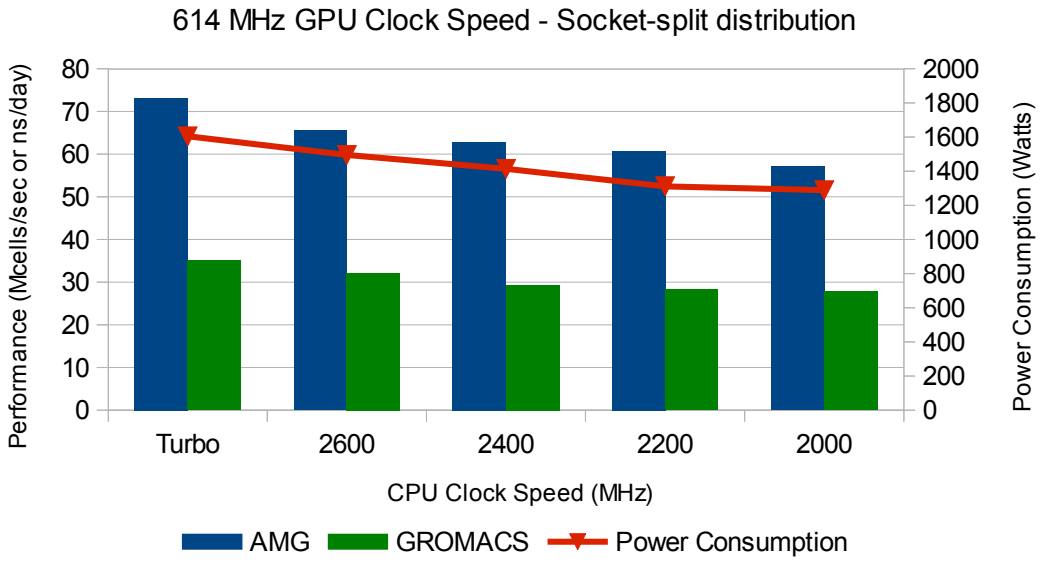

Figure 31: AMG - GROMACS co-location for 614 MHz GPU and Socket-split mapping

In order to compare the different CPU core mappings, power efficiency charts are needed.

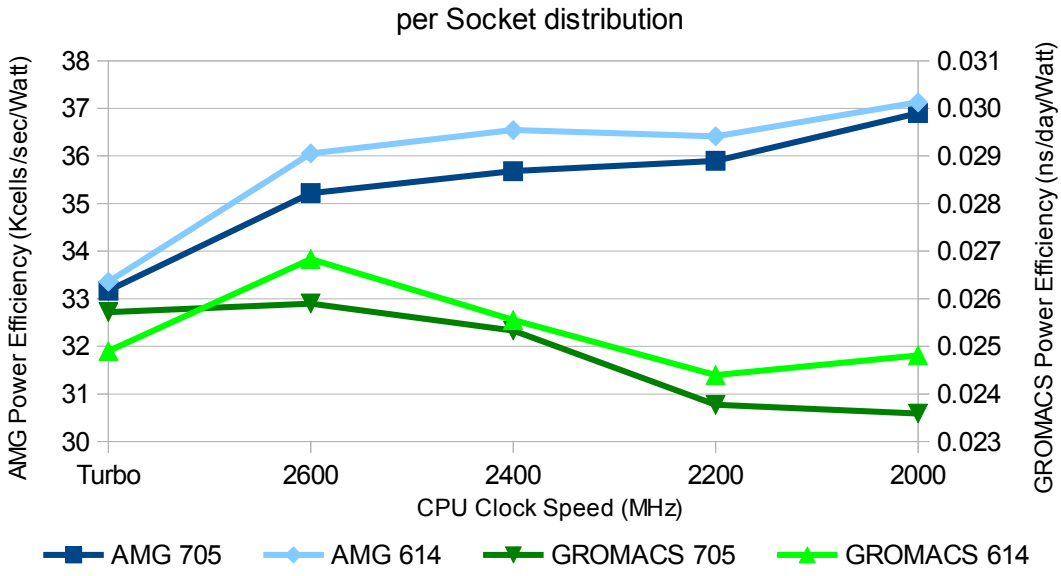

Figure 32: AMG - GROMACS co-location power efficiency for per socket mapping

In Figure 32 it becomes clear that lower GPU clock rates have a positive effect on power efficiency. AMG best power efficiency comes for 2000 MHz CPU frequency, while GROMACS best power efficiency comes for 2600 MHz. However, the 2600 MHz CPU frequency is an overall best speed in terms of power efficiency, providing 26 Kcells/sec per Watt for AMG and 0.027 ns/day per Watt for GROMACS.

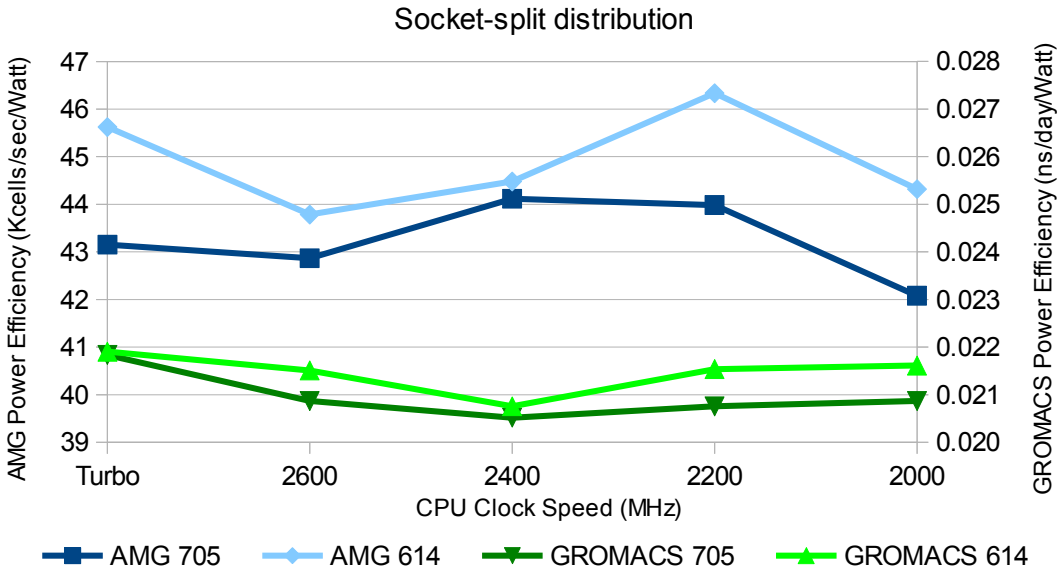

Figure 33: AMG - GROMACS co-location power efficiency for socket-split mapping

Figure 33 shows again that the lower GPU frequencies improve power efficiency. AMG best power efficiency is presented at 2200 MHz CPU clock speed, however, Turbo mode provides a very close alternative. For GROMACS the fluctuation is very low, making almost all frequencies acceptable, while the maximum is when the Turbo mode is enabled. Thus, the best combination is on Turbo mode, both in terms of power efficiency and performance.

The socket-split mapping provides the overall best power efficiency. With Turbo mode enabled, the performance of AMG is 73.2 Mcells/sec and the performance of GROMACS is 35.2 ns/day, while the power consumption is 1605 Watts. When running alone, AMG's most efficient performance is 95.7 Mcells/sec with power consumption of 1036 Watts and GROMACS's most efficient performance is 55.3 ns/day with power consumption of 1316 Watts. In percentages, AMG is 31% faster when running alone and GROMACS is 57% faster.

	AMG	GROMACS	Co-location
Performance	95.7	55.3	73.2 + 35.2
Power Consumtion (Watts)	1036	1316	1605

Table 5: AMG and GROMACS performance comparison at the most efficient configuration

Assuming that the co-location run lasts 1 hour, the total power consumed for AMG and Ludwig will be 1*605 Wh.* During this hour, AMG will compute:

*AMG problem size = 73.2 Mcells/sec * 3600 sec = 263520 Mcells*

and GROMACS will compute:

GROMACS problem size = 35.2 ns/day / 24h = 1.47 ns

If the same simulations run sequentially:

AMG run time = 263520 Mcells / 95.7 Mcells/sec = 2754 secs = 45.9 mins

*AMG power consumption = 1036 W * 45.9/60 h = 793 Wh*

*GROMACS run time = 1.47 ns / 55.3 ns/day * 24 h = 0.64 hours = 38.2 mins*

*GROMACS power consumption = 1316 W * 38.2/60 h = 838 Wh*

In total the two simulations would take

Total power consumption = 793 Wh + 838 Wh = 1631 Wh

which is 1.6% more energy used for the same computations. Performance-wise, the co-located simulations finish in 1 hour, while the separate runs finish in a total of 1 hour 24 mins, which is 40% more time.

In conclusion, co-locating AMG and Ludwig results in **1.6% less power consumption** and **29% less computation time**. It is clear that co-location of even two very demanding applications is definitely advantageous for performance.

Chapter 6

Conclusions

Heterogeneous System Architectures are expanding across the HPC world constantly. Having a look at the Top500 list can verify this fact: As for June 2013, 4 out of 10 top supercomputers are making use of accelerators in order to reach high levels of performance. The accelerators have stopped from being just a trend and moved to be considered as a permanent fact. They affect the way that HPC systems are utilised and of course the way that scientific applications are developed. More and more scientific applications and HPC codes are ported to accelerators, in order to take advantage of the performance boost that accelerators can give.

However, the heterogeneous nature of modern HPC systems has added complexity to both the architecture and the applications. A lot of research is ongoing, trying to show the path to exploiting the full potential of these systems. One of the most discussed topics of last years is the power efficiency issue. The energy problems that have arisen during the last decade have definitely affected the HPC world and a lot of research is going on in order for this linear growth in the performance of supercomputers to continue and not be saturated.

This is where this dissertation is contributing. By investigating the power efficiency of modern systems and identify ways of increasing it, we add to the systems of tomorrow. The investigation has been done in two domains: clock frequency scaling of the compute devices and co-location of HPC workloads. For measuring and quantifying the power efficiency, scientific applications were used: the algebraic (LU factorisation) Linpack, the Molecular Dynamics GROMACS, the Lattice-Boltzmann Ludwig and the algebraic (multigrid) AMG.

When running individually, due to the different nature of these codes, each application produced behaved differently. The high hardware-stressing Linpack showed very high levels of power efficiency for very low CPU clock rates (1400 MHz). The huge amount of DGEMM operations that Linpack performs, are very efficiently ported on GPUs, thus minimising the computation on the CPUs maximises power efficiency. On the other hand, the complex calculations of GROMACS require high frequencies of the CPU clock (2400 MHz) in order to achieve high levels of power efficiency. Moreover, an efficient code such as Ludwig shows best behaviour in terms of power efficiency at

the default clock rates, both on CPUs and on the highly efficient port on the GPUs. As expected, a memory bound application such as AMG runs more efficiently in terms of power at slightly lower than default CPU frequency (2200 MHz), or even better at a 75% populated cluster at default clock speeds. It should also be noted that Turbo Boost had a significantly negative effect on power efficiency, mainly because of the voltage increase it applies.

When co-locating HPC applications, the behaviour of the system changes because of the different load that is applied. The aim was to compare the most efficient configurations of co-location with the individual runs. When co-locating the two versions of Ludwig, the co-location saves 25% of power and 35% faster than running two sequential simulations. Co-locating AMG and Ludwig GPU-only results in 18% power consumption and 29% less computation time. Finally, when co-locating AMG and GROMACS the power consumption is only 1.6% less but the computation times is again 29% less.

It is clear that tweaking the clock frequencies of the compute units of a modern cluster can result in high power efficiencies. When the absolute performance is not the main target, users can sacrifice some time to get the results and get significant power savings. Furthermore, co-location of HPC workloads seems to be a very promising technique that when carefully applied, can result not only in less power consumption, but also in smaller time-to-result overall, without changing even a single line of code!

Chapter 7

Future work

Future work in the areas of power efficiency and co-locating could investigate numerous aspects of the possible applications of these techniques. The most important directions that this research should have are outlined below:

- Application categorisation / Metrics: The power efficiency of more HPC applications should be investigated in order to acquire a more general view of the power efficiency's dependence of clock frequencies. Moreover profiling results of the applications could be combined with the power efficiency behaviour, in order to generate some general rules and metrics about them.

- Frequency scaling control: In order to have control over the frequency scaling, the user must have root permissions. However this is not applicable to HPC and and workstation environments. However, frequency scaling control could be integrated within the batch systems. The user could set a parameter in the batch submission script that informs the batch system that he is willing execute his job at a lower frequency. The batch system could then set the desired frequency to the nodes that the job is going to be executed on, and probably charge the user less, as he will be consuming less power.

- Voltage scaling: The presented results are about the power efficiency gains due to frequency scaling. Power consumption is proportional to the frequency, thus frequency affects power efficiency. However, lower frequencies need lower voltage levels to operate. Moreover, power consumption is also proportional to the square of voltage. In this manner, voltage scaling could have greater affection to power efficiency. On low-power devices were power efficiency plays a major role, like smartphones where it is affecting battery life, voltage scaling is already functional. The increasing power limitations dictate that extended voltage scaling should be also applied on HPC systems, as power consumption is already acting as a limiting factor.

- Co-location batch systems / queues: In order to co-locate applications on a HPC system, the user should have direct access on the compute nodes. In most cases, users only interact with the batch system. However, current versions of batch

systems do not support co-location at all. This could change in two ways. Firstly the user should have the ability to declare in the submissions scripts which jobs that he is willing to collocate. In this way, the user has direct control over the jobs that are going to be co-located and he is responsible for an efficient co-location. Secondly, there could be dedicated lower priced queues for users that do not care about absolute performance but power efficiency. The batch system should co-locate jobs in order to exploit the increased power efficiency that co-location provides. The user could use parameters to help the batch system to co-locate the jobs efficiently, by declaring the type of his job. For example, jobs could be categorised as floating point compute intensive, integer compute intensive, memory-bound, GPU- accelerated, GPU-only.

In conclusion, work needs to be done on several levels. Researchers need to investigate more on power efficiency and co-location effects and ideally conclude to some rules. Operating systems need to adapt the voltage scaling ability in order to exploit the reduced frequencies even more. Lastly, batch systems have to integrate the ability to scale the frequencies and co-locate applications.

References

[1] Dr. Arshad Mansoor , et al., Generalized Test Protocol for Calculating the Energy Efficiency of Internal Ac-Dc and Dc-Dc Power Supplies, Revision 6.6

[2] Top500 List, June 2013, http://www.top500.org/lists/2013/06/ (Accessed 15-08-2013)

[3] Top500 List, June 2013, http://www.green500.org/lists/green201306 (Accessed 15-08-2013)

[4] P. M. Kogge, et al., "Exascale Computing Study: Technology Challenges in Achieving Exascale Systems", DARPA Information Processing Techniques Office, Washington, DC, September 28, 2008.

[5] Top500 Performance Development, http://www.top500.org/statistics/perfdevel/ (Accessed 15-08-2013)

[6] Boston Viridis Datasheet, http://download.boston.co.uk/downloads/9/3/2/932c4ecb-692a-47a9-937d-a94bd0f3df1b/viridis.pdf (Accessed 16-08-2013)

[7] Epiphany Architecture Reference Manual, http://www.adapteva.com/wp-content/uploads/2012/12/epiphany_arch_reference_3.12.12.18.pdf (Accessed 16-08-2013)

[8] Alex D. Breslow , et al., The Case For Colocation of HPC Workloads, DOI: 10.1002/cpe

[9] HPC Advisory Council, Student Cluster Challenge, http://www.hpcadvisorycouncil.com/events/2013/ISC13-Student-Cluster-Competition/index.php (Accessed 17-08-2013)

[10] HPC Advisory Council, Student Cluster Challenge – Rules, http://www.hpcadvisorycouncil.com/events/2013/ISC13-Student-Cluster-Competition/rules.php (Accessed 17-08-2013)

[11] EPCC MSc Student team in ISC '13 SCC, http://www.epcc.ed.ac.uk/blog/2013/07/11/epcc-msc-student-team-isc13-scc (Accessed 17-08-2013)

[12] Intel Xeon E5-2670 official specifications, http://ark.intel.com/products/64595 (Accessed 18-08-2013)

[13] Hyper-threading, wikipedia article, http://en.wikipedia.org/wiki/Hyper-threading (Accessed 18-08-2013)

[14] Intel Turbo Boost, wikipedia article, http://en.wikipedia.org/wiki/Intel_Turbo_Boost (Accessed 18-08-2013)

[15] Stefanos Kaxiras, Margaret Martonosi, Computer architecture techniques for power efficiency, Mark D. Hill Series editor

[16] Nvidia System Management Interface program, https://developer.nvidia.com/nvidia-system-management-interface (Accessed 19-08-2013)

[17] CentOS Operating System, http://www.centos.org/

[18] RedHat Operating System, http://www.redhat.com/

[19] GCC, The GNU Compiler Collection, http://gcc.gnu.org/

[20] Open MPI, A High Performance Message Passing Library, http://www.open-mpi.org/

[21] High Performance Linpack benchmark, www.netlib.org/benchmark/hpl/

[22] GROMACS, http://www.gromacs.org/About_Gromacs

[23] Folding@Home, http://en.wikipedia.org/wiki/Folding@home

[24] GROMACS, wikipedia article, http://en.wikipedia.org/wiki/GROMACS (Accessed 20-08-2013)

[25] GROMACS documentation, Acceleration and Parallelisation, http://www.gromacs.org/Documentation/Acceleration_and_parallelization (Accessed 21-08-2013)

[26] EPCC News, Issue 69, Sping 2011

[27] Alan Gray, et al., Lattice Boltzmann for Large-Scale GPU Systems

[28] AMG benchmark, http://www.nersc.gov/systems/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/amg

[29] HAMEG HM8115-2 Manual