Journal of Computational Physics 237 (2013) 132–150

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

# Energy efficiency vs. performance of the numerical solution of PDEs: An application study on a low-power ARM-based cluster

Dominik Göddeke [a,*], Dimitri Komatitsch [b], Markus Geveler [a], Dirk Ribbrock [a], Nikola Rajovic [c,d], Nikola Puzovic [c], Alex Ramirez [c,d]

[a] Institut für Angewandte Mathematik, Fakultät für Mathematik, TU Dortmund, Germany
[b] Laboratory of Mechanics and Acoustics, CNRS/University of Aix-Marseille, France
[c] Computer Sciences Department, Barcelona Supercomputing Center, Barcelona, Spain
[d] Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain

ARTICLE INFO

*Article history:*
Received 28 July 2012
Received in revised form 14 October 2012
Accepted 21 November 2012
Available online 7 December 2012

*Keywords:*
High performance computing
Energy efficiency
Low-power processors
ARM processors
Parallel scalability
Finite elements
Multigrid
Wave propagation
Lattice-Boltzmann

ABSTRACT

Power consumption and energy efficiency are becoming critical aspects in the design and operation of large scale HPC facilities, and it is unanimously recognised that future exascale supercomputers will be strongly constrained by their power requirements. At current electricity costs, operating an HPC system over its lifetime can already be on par with the initial deployment cost. These power consumption constraints, and the benefits a more energy-efficient HPC platform may have on other societal areas, have motivated the HPC research community to investigate the use of energy-efficient technologies originally developed for the embedded and especially mobile markets. However, lower power does not always mean lower energy consumption, since execution time often also increases. In order to achieve competitive performance, applications then need to efficiently exploit a larger number of processors. In this article, we discuss how applications can efficiently exploit this new class of low-power architectures to achieve competitive performance. We evaluate if they can benefit from the increased energy efficiency that the architecture is supposed to achieve. The applications that we consider cover three different classes of numerical solution methods for partial differential equations, namely a low-order finite element multigrid solver for huge sparse linear systems of equations, a Lattice-Boltzmann code for fluid simulation, and a high-order spectral element method for acoustic or seismic wave propagation modelling. We evaluate weak and strong scalability on a cluster of 96 ARM Cortex-A9 dual-core processors and demonstrate that the ARM-based cluster can be more efficient in terms of energy to solution when executing the three applications compared to an x86-based reference machine.



## 1. Introduction and motivation

Energy efficiency and power consumption have become one of the most critical issues regarding the design and deployment of a high performance computing (HPC) facility, or a data centre in general. While the combined power consumption of HPC systems worldwide continues to be small in relative terms (2% of the total $CO_2$ emissions), in absolute terms the cost is

* Corresponding author.
*E-mail addresses:* dominik.goeddeke@math.tu-dortmund.de (D. Göddeke), komatitsch@lma.cnrs-mrs.fr (D. Komatitsch), markus.geveler@math.tu-dortmund.de (M. Geveler), dirk.ribbrock@math.tu-dortmund.de (D. Ribbrock), nikola.rajovic@bsc.es (N. Rajovic), nikola.puzovic@bsc.es (N. Puzovic), alex.ramirez@bsc.es (A. Ramirez).
*URLs:* http://www.mathematik.tu-dortmund.de/~goeddeke (D. Göddeke), http://komatitsch.free.fr/ (D. Komatitsch).

0021-9991/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcp.2012.11.031

already extremely high, around 200–300 billion kW h [1,2]. This trend will further increase in the near future, and the cost of energy is also increasing, which will soon lead to a critical situation owing to energy efficiency limits [3].

The most immediate concern is that the energy cost of an HPC installation over its lifetime (5–7 years) is already comparable to its initial cost of acquisition and deployment, see Section 3.2. This is important on all scales, and in both academia and industry. In academia for instance, it is common – at least for medium-scale installations hosted by a single university – that (governmental or regional) funding is only provided for the initial deployment of a machine, and the university or research institution has to cover the total cost of running the machine for its lifetime. For instance it was recently announced that the state of New Mexico, USA, would discontinue operations of its 'Encanto' system, which ranked #3 in the TOP500 list when it was deployed in 2007, due to lack of maintenance funds.[1]

Since 2007, the Green500 list [4] ranks supercomputers similar to the well known TOP500 list, but based on the energy efficiency (performance per watt) of the systems. Extrapolating the energy efficiency of the #1 system in the current Green500 list, a postulated 1 exaflops system, i.e., $10^{18}$ operations per second, would require 500 MW of power, corresponding to more than 500 million dollars per year for electricity alone. Many studies predict a feasible power envelope of 20 MW for exascale machines, which requires a 25× efficiency improvement over the current Green500 leader and a 50× improvement over current standard x86-based systems [5,6].

An analysis of current HPC systems shows that 40–60% of the energy consumption can be attributed to the compute nodes (processors and memories), 10% to the interconnect and storage systems, and the remainder (up to 50%) is consumed by the infrastructure itself, including lighting, power supply, and most of all, cooling [7,8]. Consequently, the largest returns can be expected from improving the energy efficiency of the compute nodes, since any improvement there will also translate into reduced cooling requirements.

It is also possible to improve the energy efficiency of a computer system by offloading parts of the application to a more specialised hardware accelerator that achieves higher performance at a lower energy cost for a particular type of computation. Current representatives of such heterogeneous compute systems couple conventional x86-based CPUs with GPU accelerators, or the recently introduced Xeon Phi. However, such improvements come at the cost of major changes in the application codes, which must be partitioned to offload computation to the accelerator and partly rewritten in the accelerator specific programming model. While these heterogeneous architectures offer significant improvements in energy efficiency, the investment required to rewrite production codes that have been used for years or decades is often difficult to handle, which increases the pressure to find more efficient general purpose solutions.

There are several ongoing research proposals and industrial initiatives that suggest the use of devices originally designed for the embedded and mobile markets in a high performance computing environment. These devices have been designed from scratch to operate in energy and temperature constrained environments such as cell phones, which operate on batteries and are stored in our pockets. Recent developments in mobile processors have included the addition of double-precision floating point units, which is a necessary condition for them to become accepted in HPC. Their higher energy efficiency as well as their large market volume, which drives prices down, make these devices a promising candidate in terms of performance per watt and performance per dollar. Nowadays, the majority of such mobile devices are built on the intellectual property of ARM Holdings plc. For instance, ARM's 2011 annual report to investors [9] quotes constant market shares of 95% for handheld devices (smartphones and tablet computers) since 2008.

For these devices and systems built from them, the software environment is favourable to porting production codes: The GNU/Linux tool chain has been supported on ARM processors for a long time, together with all major programming and scripting languages such as Fortran, C/C++ or Python, as well as parallel programming models such as MPI and OpenMP. The effort of getting an application up and running on an ARM-based system is thus identical to the effort on any classical architecture, i.e., usually small. The required investment in manual tuning for specific (micro-) architectural features of these processors is also comparable between classical and ARM designs, as their architecture is not as radically different as for instance that of GPUs. In this article, we discuss a range of tuning strategies that are particularly important.

Low power also often means low performance: Mobile devices offer 10 to 100 times lower peak performance than their high-end counterparts, but they do so at 100 to 1000 times lower power. As an example, the chips that we use in this work deliver a theoretical peak performance of 2 GFLOP/s at roughly 0.5 W, while our reference x86 designs require a TDP (thermal design power) of 95 W for a theoretical peak performance of 42.6 GFLOP/s. Thus, owing to the longer execution time, lower power may not translate into lower energy. To achieve the degree of performance required by actual applications, many low-power cluster nodes have to be used in one system. Very good weak and strong scalability of the (numerical) methodology *and* the implementation are required simultaneously. Only then is it possible to distribute large 'real-world' problems over more nodes with less memory each and to compensate for lower single-node performance by using more nodes.

For most non-trivial PDE (partial differential equations) applications this is highly challenging to achieve simultaneously, and the behaviour is strongly application dependent. In this article, we employ a prototype ARM-based cluster that, given its size, allows us to quantitatively analyse the power-performance trade-off (measured in time vs. energy to solution) for the first time at a scale of interest for future HPC systems and for production codes solving real-world problems. We perform this

[1] HPC Wire, July 12, 2012, 'New Mexico to Pull Plug on Encanto, Former Top 5 Supercomputer', http://www.hpcwire.com/hpcwire/2012-07-12/new_mexico_to_pull_plug_on_encanto_former_top_5_supercomputer.html.

analysis with three applications that address different performance and complexity characteristics; they cover several scientific domains and we believe that they are representative of a wide range of scientific computing codes to solve problems modelled by PDEs and discretised on structured and non-structured grids. These applications are a parallel locality-maximising finite-element geometric multigrid solver with very capable smoothers (FEAST), a fluid dynamics code based on the Lattice Boltzmann method (HONEI_LBM), and a petascale-proven production code that can perform forward or adjoint simulations of acoustic wave propagation modelling in the time domain at the scale of industrial or geophysical models based on a spectral-element method (SPECFEM3D_GLOBE).

## 2. Related work

In the field of mobile computing, a multitude of different applications have been ported to execute on ARM processors, including resource-hungry games and video and audio processing software. Gutierrez et al. [10] present benchmarks of the microarchitectural behaviour of such applications on modern smartphones. However, the literature on scientific computing on low-energy (ARM) processors and in particular ARM clusters is still pretty scarce; and most published articles restrict themselves to 'standard' benchmark problems on a single node. Fürlinger et al. [11] have built a small five-node cluster from Apple's ATV2 set-top boxes. The underlying hardware is based on Apple's A4 SoC (system-on-a-chip), which uses the ARM Cortex-A8 processor. That article also includes references to cluster prototypes from other unconventional consumer-level hardware that have been built before, e.g. based on the PlayStation 3 to benefit from the STI Cell processor [12], which subsequently has been used in the first system in the world to have sustained a petaflop/s, the RoadRunner supercomputer at Los Alamos National Laboratory (USA). Dasika et al. [13] evaluate various architectures, including ARM Cortex-A8 processors and (mobile) Intel CPUs and GPUs, with respect to their energy efficiency for certain benchmark problems.

Low-energy computing is traditionally the domain of embedded systems, which require specialised tools and are notoriously hard to program. A hybrid approach is pursued by various vendors coupling conventional processors with x86-socket mounted FPGAs, combined with compiler technology to more easily offload tasks to the FPGA. Augustin et al. [14] analyse a recent representative system of this kind in detail and report issues of the compiler in the generation of optimised co-processor code, and thus poor performance results for hybrid workloads despite a coherent shared memory space. Power and performance of hardwired vs. general purpose architectures are discussed by Fan et al. [15].

Dynamic voltage frequency scaling (DVFS) is a technique implemented in modern processors to reduce energy consumption by downclocking them when the load is lower. Etinski et al. [16] analyse software-controlled DVFS and its impact on execution time and in particular energy to solution for a set of representative HPC applications executing on medium-sized clusters; see also the references therein for an overview of the state of the art.

Adaptations of the GNU compiler suite to the ARM architecture, and in particular optimising for various size and energy vs. performance targets, are subject to active research [17,18]. Energy-aware scheduling both in clusters and in the cloud has also moved into focus recently [19,20]. Similar in spirit is the work by Lee et al. [21] on proactive cooling for large data centres.

Algorithmic approaches to improving energy efficiency are also being pursued. In the context of scientific computing, one prominent example is computing with reduced precision. The idea can easily be applied to schemes with explicit time stepping that do not suffer from ill-conditioning, such as the wave propagation and fluid dynamics codes that we evaluate in this study, see Section 5. Assuming a contiguous layout of data in memory, halving the computational precision enables moving twice the amount of values from memory to the CPU in the same time, more or less doubling the effective cache sizes modulo line size granularity effects, and doubling the effective SIMD width. Faster execution and data movement in single rather than double precision can thus give an immediate energy efficiency improvement due to less time to solution. In addition, low precision can also be more energy efficient for arithmetic and data movement in the hardware itself. For other applications, high precision is often not required at all stages and for all data. Mixed precision approaches (see [22] for a survey) can provide a way of improving computational density and energy efficiency in such situations. Anzt et al. [23] recently performed detailed measurements of mixed precision iterative refinement GMRES on multicore and hybrid CPU-GPU architectures and concluded that the reduction in energy consumption is proportional to the performance improvement of the scheme compared to executing entirely in high precision. In follow-up work they extend their approach to include more fine-grained control of power consumption by evaluating DVFS and idle-waiting instead of busy-waiting [24].

## 3. Background

### 3.1. The power wall problem

In electrical engineering, the term *power wall* is often used to describe the issues and hard physical limitations related to increasing compute performance without increasing the power envelope of a microprocessor design. We refer to [25–27] and the references therein for recent discussions of the general power/performance trade-offs in microprocessor design and on the optimisation problems that chip designers face. Table 1 summarises the physical background and implications on performance improvements for the 'old' situation of frequency scaling up to 90 nm processes and the current situation. The data in the left column highlight that halving the feature length yielded eight times the operation rate for the same

**Table 1**
Physical background of performance improvements until the 90 nm process (left) and for current feature sizes (right).

| | Constant field scaling (old) | Constant voltage (new) |
|---|---|---|
| Feature length | $L' = L/2$ | $L' = L/2$ |
| Voltage | $V' = V/2$ | $V' = V$ |
| Capacitance | $E' = CV^2/2 = E/8$ | $E' = E/2$ |
| Frequency | $f' = 2f$ | $f' = 2f$ |
| Area | $A' = L^2 = A/4$ | $A' = A/4$ |
| Power/area | $P' = Pf/A = P$ | $P' = 4P$ |
| Operations/s/area | $f'/A' = 8f/A$ | $f'/A' = 8f/A$ |

power in a fixed area. However with current feature length scales it is no longer possible to halve voltage proportionally with feature length, and consequently, achieving the same factor of eight for a fixed area requires four times the power. Thus, frequencies can no longer be doubled with each die shrink, issues due to leaking voltage are more dominant, and some trade-off between power and performance must be found.

Another important aspect of the power wall is the cost of moving data. The following example has been promoted by W. Dally, and has been subsequently used in many conference presentations. For details (albeit in the context of embedded computing with a stronger emphasis on the cost of moving not only floating point data but also instructions through pipelines) we refer to [27,28]. A 64-bit FPU (floating point unit) in a standard yet hypothetical contemporary processor clocked at 1.5 GHz consumes approximately 50 pJ for a multiply–add operation and occupies 0.1 $mm^2$ of die area (at some current feature size, the actual fabrication process does not matter for the sake of the argument). Moving one double precision word through a channel of length 1 mm consumes 25 pJ, which means that for a chip with 4000 FPUs (and hence an area of 20 $mm^2$), moving data across the chip requires 500 pJ. Moving the same data off-chip requires another factor of two, namely 1 nJ. This indicates that the well known memory wall problem does not stop at chip boundaries. Even more importantly, projections to feature sizes expected for 2018 at the beginning of the exascale era in [5] indicate that '*data movement across the system, through the memory hierarchy, and even for register-to-register operations, will likely be the single principal contributor to power consumption, with control adding to this appreciably*'.

### 3.2. System energy cost

Energy consumption data for scientific computing installations is rarely publicly available. Some data have been gathered for the Bay Area by the 'High-Performance Buildings for High-Tech Industries' study performed by Lawrence Berkeley National Laboratory (USA) [8]. One of the systems analysed is dedicated to computational science workloads, but due the anonymisation of the data, only conclusions and recommendations regarding the efficiency of the infrastructure of the machine can be deduced.

Nonetheless, we want to illustrate that the cost of operating a system for a typical lifetime of 5–7 years already amounts to a significant fraction of its acquisition cost, dominated for the largest part by energy and personnel cost.

The first system that we report on in order to support this claim is a capability installation, the 1 petaflop/s 'Hermit' Cray XE6 system installed in Stuttgart (Germany) at the HLRS. Officially published numbers[2] for this system indicate approximately two million euros annually for energy and personnel, compared to acquisition costs of 22.5 million euros.

The second system that we consider has been designed for capacity computing: The 'LiDOng' machine operated by TU Dortmund is a heterogeneous (i.e., different memory/CPU nodes) installation assembled by Hewlett–Packard in 2009. It achieves 23 TFLOP/s in Linpack, resulting in the 253th rank in the June 2009 TOP500 list. Overall, there are 432 nodes (predominantly 3.0 GHz Intel Xeon 'Harpertown' quad-core processors) with a total of 3584 CPU cores and 8 TB of memory. The interconnect is hierarchical, with a crossbar for all blades in each rack and a tree between the racks. 128 nodes are connected via DDR Infiniband and all nodes are connected via gigabit Ethernet twice, in the compute network and the service and I/O network. In addition to the compute nodes there are two management servers, two gateway servers, and eight fileservers that provide access to a 256 TB Lustre filesystem. We provide these details because we want to emphasise that LiDOng is a typical representative system. The raw energy costs of LiDOng amount to 260,000 euros per year. In other words, by 2013 its accumulated electricity cost alone will have exceeded its acquisition cost of approximately one million euros.

## 4. The ARM Cortex-A processor family and ARM-based systems-on-a-chip

### 4.1. ARM processor overview

In this article, we focus on the ARMv7 architecture family, or more precisely its so-called 'application profile' ARMv7-A (Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9 and Cortex-A15). ARM itself only licenses processor designs but does not man-

[2] http://www.uni-stuttgart.de/hkom/presseservice/pressemitteilungen/2011/119_hlrs.html.

ufacture actual chips. Processors by other companies that target similar markets and include to some extent intellectual property by ARM are the Qualcomm Scorpion/Dragonfly, Samsung Exynos, Texas Instruments OMAP, the Freescale i.MX series, or the Marvell Armada and Sheeva designs.

We begin the description of the hardware with a brief overview of how the Cortex-A and other ARMv7 cores have developed into general purpose processor cores. For more detailed information, we refer to the official programmer's guide and in particular the technical reference manuals for specific processor instantiations [29,30]. Much of the information summarised here can also (and sometimes only) be found in material provided by actual SoC (system-on-a-chip) vendors and/or technical press. In our case, the Cortex-A9 processor design is included in an NVIDIA Tegra 2 SoC [31].

*Development of application-oriented ARM processors.* The main technical reasons for the current Cortex-A processor family becoming more and more interesting for scientific computing tasks are that

1. processors prior to the ARMv7-A architecture did not support native floating point nor SIMD instructions,
2. symmetric multiprocessing ('multi-core') became available with the Cortex-A5, and
3. clocking regimes of 1 GHz and more became available with the Cortex-A8.

These three features combined with (a) area efficiency, (b) energy efficiency and (c) availability/cost-efficiency (all three being a consequence of the architecture originating in the field of embedded systems) make them increasingly interesting in scientific computing. Here (a) as well as (b) mainly result from a smaller instruction-decode unit than in x86 and specially-designed instruction sets leading to small transistor counts and better code-density: Even compared to the lowest performance Intel Atom core, Cortex-A9 processors have roughly half the die size and number of transistors, and they are an order of magnitude smaller than high-end Intel Xeon or AMD Opteron processors. Note however that such comparisons from core to core are a bit difficult to perform because for instance the Cortex-A processors use a level-2 (L2) cache that is implemented on the SoC rather than in the processor, and thus not included in the per-core transistor count and die area. Nonetheless, the factor of 1/2 is a good average commonly found in the literature. On the other hand, their cost efficiency is driven by the power of consumer markets. Up to early 2011, a total of more than 20 billion ARM processors have been shipped, with estimates by ARM itself that a good quarter of all electronic devices contain at least one ARM processor [29]. For comparison, ARM states three billion devices in total for the x86 architecture [29]. The first number is clearly influenced by ARM's diverse set of target markets, in particular including all sorts of microcontrollers, the Cortex-M and Cortex-R profiles, and other non-Cortex designs. In addition to the huge market volume, emerging performance-hungry applications for mobile devices such as video encoding, gaming and communication – currently and even more in the future – imply high production amounts at comparatively low cost and a fast development of mobile processors towards better performance without losing energy efficiency.

*Hardware details of ARMv7-A processors and development of the Cortex-A9.* The ARMv7-A is a 32-bit, Harvard Load/Store (or formerly RISC – Reduced Instruction Set Computer) architecture. The Load/Store attribute is however not the most important distinction between this architecture and x86, as for the latter the boundaries between CISC and RISC are blurred: Complex instruction sets for instance are clearly a CISC feature while SIMD attributes are close to RISC features. In the ARMv7-A processors, storage and signal pathways are physically separate for instructions and data, and arithmetic instructions work on registers only whereas dedicated operations realise access to main memory. The Cortex-A series continuously adopts features known from high performance platforms, which can be seen for instance in its pipeline design: Where the older Cortex-A8 had a 13-stage pipeline (and a separate 10-stage one for the SIMD instructions), the Cortex-A9 features a 9-stage, dual-issue superscalar out-of-order pipeline with dynamic branch prediction. This is the major reason for the speedup between these generations at the same clock speed when not considering SIMD capabilities. In addition, the memory subsystem has seen much improvement since the first Cortex-A5, which only comprised a small data cache on a single level. Where the Cortex-A8 already supported up to 1 MB of L2 cache, the Cortex-A9 extends this to an (optional) amount of up to 8 MB of L2 cache. The L1 cache of the Cortex-A9 has 32 kB for instructions and 32 kB for data. Even more important is the ability of the Cortex-A9 to be arranged into multi-core designs (the so-called MPCore configuration) comprising up to 4 cores. This is enabled by hardware support for cache coherency in a shared L2/ private L1 cache partition. More sophisticated production processes and higher admissible clock rates have come along with the economic success of the ARMv7: Newer Cortex-A based SoC are expected to be built in a 28 nm process and clock speeds of 1.5 GHz are becoming common.

The Cortex-A9 is produced in a 40 nm process, and typical clock rates are 1 to 1.2 GHz. Regarding floating point capabilities, it is equipped with a fully IEEE-754 compliant FPU (called VFPv3-D16) capable of single and double precision arithmetic as well as conversion between half and single precision. Fused multiply-accumulate and square root instructions are supported natively, and the FPU is considered to be the most augmented of all processors of the series owing to reduced operation latencies, out-of-order completion of load and store instructions, and support for speculative execution [32]. One important aspect is that the advanced SIMD unit, called NEON, is an optional part of a Cortex-A9 MPCore. When available, the NEON unit provides packed operations for 64 and 128 bit vector registers supporting integer and single precision floating point arithmetic only, without native division nor square root. NEON is not available in the Tegra 2 SoC on which our experimental evaluation is based.

Taking a brief look at the more advanced Cortex-A15, the trends in development continue: The pipeline (superscalar and dynamic branch prediction) as well as caches (shared L2 no longer optional, error correction) and the memory system (more

translation lookaside buffers) are augmented, and clock rates of up to 2 GHz are possible. As the newest Cortex series representative, the Cortex-A7 is meant to be a lower-power counterpart of the Cortex-A15 and will ship in 2014.

### 4.2. The NVIDIA Tegra 2 SoC

The design of a SoC typically includes the integration of different processor cores/multi-cores or ASICs (application specific integrated circuits) alongside some memory system and bulk main memory. In case of the Tegra 2 this means a customised dual Cortex-A9 MPCore combined with a customised memory system (1 MB of L2 cache plus main memory) as well as an ultra-low-power NVIDIA GeForce GPU. Here, we explicitly refer to a Tegra 2 of the first generation, based on the 'Harmony' development board similar to the SECO Q7 carrier board built into our test cluster. There are other Tegra 2 designs that are not covered here. We deliberately ignore all other SoC components such as audio, video and wireless networking, see Section 6. Main memory is low power DDR2 (LPDDR2) in single channel mode ($1\times32$ bit with 667 timings, presumably clocked at 400 MHz) resulting in an approximate peak throughput of slightly less than 2 GB/s. Its Cortex-A9 is customised with a 1 GHz clock rate and the optional NEON units in the cores have been left out. Hence, there are two very important aspects of this design to notice: This version of the Tegra 2 is not representative of current high-end ARM-based SoC designs, as only two cores without SIMD are present; and a slow off-chip memory system is used. With the maximum possible LPDDR2 configuration (1066-timings, 533 MHz clock rate, dual channel mode) more than 4.2 GB/s can in principle be achieved. However, note that reduced bandwidth and lower clock rates are one of the major approaches to reducing energy consumption of a SoC. The GPU only supports single precision and is not programmable by means of CUDA/OpenCL; this is expected not to change with the recently introduced Tegra 3, but only with the Tegra 4 code-named 'Wayne'. Programming the Tegra 2 GPU component for general purpose computations is in principle possible using 'legacy GPGPU' techniques through graphics APIs [33,34], an avenue that we do not pursue in this article.

### 4.3. Energy efficiency of ARM-based SoC designs and future trends

With respect to performance and energy efficiency we observe that mobile SoC designs begin to trade (extreme) energy efficiency for performance. The major techniques that are power-hungry but beneficial for performance of a single core in x86 have been or are being adopted, that is superscalarity, dynamic instruction issuing, large multi-level caches with advanced replacement policies, and complex branch prediction. Taking into account techniques for reducing energy consumption at the SoC/socket-level common to both architectures, such as dynamic branch prediction, dynamic voltage/power scaling (DVFS), translation lookaside buffers and gated clocks, the current and future ARM-based designs can still excel at several points:

Explicit exclusion of instruction-set backward compatibility as well as compressed instruction sets such as the 16 bit Thumb/Thumb2 sets makes the architecture more slim compared to x86 ones, minimising energy consuming transitions on the instruction bus. In addition, many application specific tasks are performed by and offloaded to on-chip specialised components such as digital signal processing (DSP) cores and other ASICs. Here the SoC synthesis is very flexible and close-to-optimal energy efficiency could be reached, at the cost of increasing market prices. However, the most important improvement for these processors regarding floating point arithmetic performance together with energy-efficiency for scientific computing would be a fast and rigorous augmentation of the advanced SIMD units, which can enhance performance with a very low increase in energy consumption. In this regard, x86 is far ahead, as illustrated for instance by AVX, the successor of SSE.

A number of general trends can be deduced from announcements of SoC designs for products to be introduced in the market in 2013/14, like the NVIDIA Tegra 4, the Qualcomm Snapdragon S4 or the Texas Instruments OMAP 5 (OMAP5430): All designs are based on the Cortex-A15, with quad-core MPcore components, clock frequencies higher than 1.7 GHz and larger L2 caches. Single precision SIMD (NEON) will always be integrated, with double precision SIMD looming on the horizon. There is also a trend of making the SoC even more heterogeneous by integrating a lower-energy general purpose core, for instance to execute the operating system and leaving the cores free for compute. Another important improvement that has been announced is the switch from 32 bit to 64 bit. The GPU components are expected to support OpenCL, and in case of NVIDIA designs, also CUDA C/C++. Finally, designs that are more tailored towards scientific computing or gaming rather than mobile computing are being pursued, for instance NVIDIA's 'Echelon' research chip [35].

## 5. Applications

In this section we present the PDE applications used to evaluate the energy vs. performance trade-off in our experiments and discuss the performance limitations that each code faces on conventional x86-based architectures as well as their respective scalability challenges. We refer to a recent publication by some of the authors [36] for an initial performance assessment for typical HPC (micro-) benchmarks such as STREAM and LINPACK.

### 5.1. FEAST – hardware-oriented geometric multigrid solvers with strong smoothers

FEAST ('Finite-Element Analysis and Solution Tools') is a toolkit providing finite-element discretisations and massively parallel multilevel solvers [37–39]. It has been designed to carefully balance numerical capabilities (flexibility in the discretisation, $h$- and $\Omega_i$-independent convergence rates, robustness with respect to high degrees of anisotropy in both the differential operators and the underlying mesh etc.) with hardware exploitation. Starting from an unstructured coarse mesh of quadrilateral patches $\Omega_i$ that are distributed to the available MPI processes, refinement proceeds in a generalised tensor product fashion and a finite-element discretisation is applied to this hierarchical sequence of meshes. This approach retains sufficient flexibility in resolving complex geometries, and all numerical linear algebra routines are performed in parallel on the patches with overlapped communication of the patch edges. A linewise numbering of the degrees of freedom in each patch results in local banded matrices, e.g., nine bands for bilinear conforming finite elements. This fixed structure is exploited in the implementation of numerical linear algebra routines that do not require indirect addressing, which both improves cache utilisation and makes much better use of the available memory bandwidth. This is important because the performance of sparse linear solvers is memory-bound on almost all modern architectures. In FEAST, this functionality is implemented by the SBBLAS library ('sparse banded BLAS', [40]).

Multigrid solvers are the only algorithmically scalable and asymptotically optimal iterative schemes for sparse linear systems. However, in practice there are a number of issues with achieving textbook convergence, in particular due to the block-Jacobi (patchwise additive) character of the smoother, an approach that is often applied to reduce communication requirements compared to fully multiplicative Schwarz decompositions. On complex meshes, convergence of block-Jacobi schemes depends on the relative size and distribution of the patches, resulting in the loss of $h$- and $\Omega_i$-independent iteration numbers. The core idea of FEASTs underlying solver concept called ScaRC ('scalable recursive clustering') to alleviate this problem is to hide local anisotropies as much as possible. In each patch, the banded matrix structure is again exploited in the design of very capable hardware-friendly smoothing operators, and in this work we employ an alternating direction implicit (ADI) variant of a combination of a Gauß–Seidel method with a tridiagonal solve (TRIGS), i.e., the application of one smoothing step translates to inverting a system with all subdiagonals and the first superdiagonal, followed by computing a global residual, virtually switching the numbering from row- to columnwise and again inverting an almost lower-triangular banded system. Mesh transfer routines are tailored to the underlying finite-element space and exploit the local structure in a similar way. Coarse grid problems in ScaRC are currently solved on a master node with the sparse direct UMFPACK solver [41].

#### 5.1.1. Problem configuration

In this article, we do not execute full applications built on top of FEAST, such as the Navier–Stokes solver or the elasticity code presented elsewhere [37,38]. Instead, we employ FEAST as a 'mini-application', which allows us generalise our findings for this important numerical building block to any low-order finite-element method and other iterative solvers. We solve a standard Poisson problem

$$-\Delta u = f \tag{1}$$

using variations of the unstructured 2D 'flow around a cylinder' grid [42] as the coarse grid. Homogeneous Dirichlet boundary conditions are prescribed on the outer boundary and inhomogeneous ones on the inner. The right-hand side is set to the one-vector. The problem is discretised with conforming bilinear finite elements. We employ the ScaRC solver with the ADI-TRIGS smoother described above, executing in an $F$-cycle with two pre- and post-smoothing steps each, and overrelaxation by $\omega = 1.2$. The stopping criterion is set to a reduction of the initial residual by eight digits. Double precision is required because the condition number of the linear system behaves as $O(h^{-2})$. We deliberately do not apply a mixed precision scheme as proposed in previous work [22] even though it would be advantageous, because the two other applications presented in this article execute completely in single precision already.

#### 5.1.2. Performance limitations on x86

As described above, the careful exploitation of the local structure for numerical linear algebra and multigrid smoothers is realised in the SBBLAS library. The bulk of the work in FEAST is performed in two phases, the finite-element assembly of the linear systems and the parallel solver. As is classical for (low-order) finite-element codes, the assembly and set-up phase for communication are on the fringe between being compute- rather than memory bound, with a non-trivial amount of integer arithmetic to account for the unstructured coarse grid. The solver is limited by memory bandwidth alone but there is considerable room for cache exploitation, in particular in the matrix–vector multiplications and the ADI-TRIGS smoother. The SBBLAS library is not cache oblivious. Instead, offline calibration runs are used to determine optimal cache blocking parameters for each new architecture.

#### 5.1.3. Scalability discussion

Parallel multilevel methods may easily suffer from unfavourable communication-to-computation ratios and fewer opportunities to overlap on coarser levels of the multigrid hierarchy, especially since we employ an $F$-cycle to improve robustness. Due to the unstructured nature of the coarse grid of patches, the amount of communication that each process performs cannot be balanced perfectly despite careful scheduling to achieve equidistribution. Weak scaling is realised by subsequently doubling the number of patches in the coarse mesh. As the patches exhibit strongly varying shapes (aspect ratios, inner an-

gles), algorithmic weak scalability in terms of iteration numbers is challenging. As usual, strong scaling is realised by distributing a fixed number of patches to more and more processes. As the load per node reduces, inexpensive shared memory communication between patches and bulk-transferring data of all patches per process can no longer be exploited. The sequential solve of the coarse grid problem can become a bottleneck for very large runs.

### 5.2. HONEI_LBM – CFD simulations based on a Lattice-Boltzmann method

The second application is a parallel computational fluid dynamics (CFD) solver for, e.g., flows and free surface waves in oceans, rivers or water tanks, supporting arbitrary geometries and bed surface profiles as well as dry states. The physical behaviour is governed by the laminar 2D shallow water equations:

$$\frac{\partial h}{\partial t}+\frac{\partial (hu_j)}{\partial x_j}=0 \quad \text{and} \quad \frac{\partial hu_i}{\partial t}+\frac{\partial (hu_iu_j)}{\partial x_j}+g\frac{\partial}{\partial x_i}\left(\frac{h^2}{2}\right)=S_i^b, \tag{2}$$

where $h$ is the fluid depth, $\mathbf{u}=(u_1,u_2)^T$ its velocity in the $x$- and $y$-direction, and $g$ is the gravitational acceleration. In addition, we apply a source term $S_i^b$ that internalises forces acting on the fluid due to the slope of the bed and material-dependent friction.

A Lattice-Boltzmann method (LBM) is then employed to solve these equations numerically. Based on a two dimensional Lattice with nine velocities (D2Q9) and a Bhatnagar–Gross–Krook (BGK) approximation of the collision operator and simple bounce-back boundary conditions [43], the method can be summarised as

$$f_\alpha(\mathbf{x}+\mathbf{e}_\alpha\Delta t,t+\Delta t)=f_\alpha(\mathbf{x},t)-\frac{1}{\tau}(f_\alpha-f_\alpha^{eq})+\frac{\Delta t}{6e^2}e_{\alpha i}S_i^b, \quad \alpha=0,\ldots,8, \tag{3}$$

where $f_\alpha$ is the particle distribution corresponding to the lattice-velocity $\mathbf{e}_\alpha$ and $f_\alpha^{\mathrm{eq}}$ defined as

$$f_\alpha^{\mathrm{eq}}=\begin{cases} h\left(1-\frac{5gh}{6e^2}-\frac{2}{3e^2}u_iu_i\right) & \alpha=0 \\ h\left(\frac{gh}{6e^2}+\frac{e_{\alpha i}u_i}{3e^2}+\frac{e_{\alpha j}u_iu_j}{2e^4}-\frac{u_iu_i}{6e^2}\right) & \alpha=1,3,5,7 \\ h\left(\frac{gh}{24e^2}+\frac{e_{\alpha i}u_i}{12e^2}+\frac{e_{\alpha j}u_iu_j}{8e^4}-\frac{u_iu_i}{24e^2}\right) & \alpha=2,4,6,8. \end{cases} \tag{4}$$

Macroscopic fluid depth and velocity can be obtained via

$$h(\mathbf{x},t)=\sum_\alpha f_\alpha(\mathbf{x},t) \quad \text{and} \quad u_i(\mathbf{x},t)=\frac{1}{h(\mathbf{x},t)}\sum_\alpha e_{\alpha i}f_\alpha. \tag{5}$$

The solver is built on top of the HONEI libraries[3] in order to be able to use different target hardware platforms efficiently, in particular an SSE backend for x86-type CPUs and an MPI backend for parallel computations. We refer to [44] and the references therein for details on this approach as well as implementations and evaluations on several modern target platforms including the Cell BE, GPUs, multicore CPUs, and clusters thereof.

#### 5.2.1. Problem configuration

As a benchmark, we apply a full dam-break scenario over a 50 m × 50 m square region with initial water depth $h_0=5$ m and a collapsing cuboidal water region of dimensions 5 m × 5 m × 3 m placed in the centre of the region. We also use an uneven bottom bed topography defined by the function $b(x,y)=x^2+y^2$ in order to apply the force terms described above.

#### 5.2.2. Performance limitations on x86

Since the x86-optimised version of HONEI uses SSE instructions, it cannot be used on the ARM target platform (which even lacks NEON SIMD units). We thus use a carefully tuned generic C++ backend for ARM.

We carry out all benchmarks in single precision, which is commonly done in explicit LBM codes for this kind of benchmark scenario. The kernels involved in the computation are mainly memory bound with the exception of the force calculation, which is slightly compute bound.

#### 5.2.3. Scalability discussion

Parallelisation is performed based on domain decomposition of the lattice at the level of the Lattice-Boltzmann distribution functions, since virtually all work performed for each lattice site is independent of all other sites. We use a packed lattice approach and pad each local array with a few ghost entries, allowing it to synchronise with its predecessor and successor. As we use a one-dimensional data layout, each partition has only two direct neighbour parts to interact with. After each time step each part sends its own results corresponding to subdomain boundaries to the ghost sites of its direct neighbours. As soon as each part has finished these independent, non-blocking transfers, the next time step calculation can begin. Since the Lattice Boltzmann method is explicit in time, the time step must decrease with increasing problem size to honour a Courant–

[3] http://www.honei.org.

Friedrichs–Lewy (CFL) stability condition. The computations performed for all time steps are identical and thus what is typically measured in LBM codes is the throughput in terms of elapsed time per time step or lattice updates per second.

### 5.3. SPECFEM3D_GLOBE – acoustic wave propagation modelling

The software package SPECFEM3D_GLOBE[4] is an established petascale-proven [45] production code that simulates three-dimensional seismic wave propagation in the time domain based upon the spectral-element method (SEM) on an unstructured mesh of high-order hexahedral finite elements. It is widely used in geophysics (seismology) and in acoustic wave propagation. The SEM is a continuous Galerkin finite-element method with optimised efficiency owing to its tensorised basis functions [46–49]. In particular, it can accurately handle very distorted mesh elements [50].

SPECFEM3D_GLOBE targets the numerical modelling of seismic wave propagation in the whole Earth, or in large parts of the Earth, following earthquakes. Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D model of the crust of the Earth, its ellipticity, topography and bathymetry, the presence of the oceans, of rotation, and of self-gravitation are included. Adjoint modelling capabilities and finite-frequency kernel simulations to solve inverse (imaging) problems are also part of the code [47].

We seek to determine the displacement field produced by an earthquake in a finite Earth model with volume $\Omega$. The boundary of this volume is a stress-free surface $\partial\Omega$ on which waves are totally reflected. The Earth model may have any number of internal discontinuities, such as interfaces between geological layers or faults. The displacement field $\mathbf{u}$ produced by an earthquake is governed by the momentum equation, which in differential (strong) form is

$$\rho\,\partial_t^2\mathbf{u} = \nabla\cdot\boldsymbol{\sigma}, \tag{6}$$

where density is denoted by $\rho$ and the stress tensor $\boldsymbol{\sigma}$ is linearly related to the displacement gradient $\nabla\mathbf{u}$ by Hooke's law, which in an elastic, anisotropic solid may be written in the form

$$\boldsymbol{\sigma} = \mathbf{c} : \nabla\mathbf{u}. \tag{7}$$

The elastic properties of the Earth model are determined by the fourth-order elastic tensor $\mathbf{c}$, which can have up to 21 independent components in the case of general anisotropy.

In finite-element techniques one uses an integrated (weak) form obtained by dotting the momentum equation (6) with an arbitrary vector $\mathbf{w}$, integrating by parts over the model volume $\Omega$, and imposing the stress-free boundary condition, which makes the boundary term vanish. This gives

$$\int_\Omega \rho\,\mathbf{w}\cdot\partial_t^2\mathbf{u}\,\mathrm{d}^3\mathbf{x} = -\int_\Omega \nabla\mathbf{w} : \boldsymbol{\sigma}\,\mathrm{d}^3\mathbf{x}. \tag{8}$$

The SEM has very good accuracy and convergence properties [51–53]. It admits spectral rates of convergence and allows exploiting *hp*-convergence schemes. All SPECFEM3D_GLOBE software is written in Fortran90 + MPI with full portability in mind and conforms strictly to the Fortran95 standard. In what follows we have employed the full production code, which comprises about 80,000 lines of source code.

#### 5.3.1. Problem configuration

In a preprocessing step, we mesh the region of the Earth in which the earthquake occurred and then split the mesh into slices, one per processor core/MPI process. The mesh in each slice is unstructured in the finite-element sense, but the whole mesh is block-structured, i.e., all the mesh slices are topologically identical and thus the whole mesh is composed of a regular pattern of identical unstructured slices. Thus, all mesh slices and cut planes have the same number of elements, faces, edges and points, which implies perfect load balancing and identical communication patterns between all MPI processes. In practice, in this article we mesh 1/6th of the Earth using a so-called 'mesh chunk' of angular size $90^\circ \times 90^\circ$ centred on the North pole and we implement an earthquake source near the North pole. The model of the structure of the Earth is the elastic isotropic version of the PREM model without the ocean layer [54], which is a standard reference Earth model widely used in the geophysical community.

In our spectral-element technique, to represent the wave fields we use Lagrange polynomial interpolants of degree $N$, defined on $[-1,1]$, built from $N+1$ Gauß–Lobatto–Legendre (GLL) points. In practice we use degree $N=4$ and thus each spectral element contains $(N+1)^3 = 5\times5\times5$ GLL points.

An important property of spectral-element methods for linear acoustic or seismic wave propagation is that single precision suffices for all the calculations performed in the code.

#### 5.3.2. Performance limitations on x86

SPECFEM3D_GLOBE is a heavily optimised code that runs in production mode on large and very large scale systems worldwide [55,45,56]. It is well tuned for cache locality [56] and also supports GPU-enhanced clusters [57,58].

[4] http://www.geodynamics.org.

Of the three main computational kernels, two perform simple updates on large global vectors of unknowns, which typically contain a few million degrees of freedom per processor core; thus they have the advantage of being trivially parallel, but they have the disadvantage of being memory bound because only a few operations are performed on each degree of freedom accessed in memory. The second computation kernel is more complex, it performs mechanical force calculations and finite-element assembly at each time step based in part on small local matrix–matrix products. The matrices correspond to cutplanes along the three directions of each $(N+1)^3$ spectral finite element described above; they thus have a size of $5 \times 5$, which is too small to benefit from calling a BLAS3 matrix product library; in SPECFEM3D_GLOBE these products are thus implemented and unrolled manually, following an implementation introduced by Deville et al. [52] and explained in more details in Section 7.5, which minimises the number of memory accesses to perform per finite element.

#### 5.3.3. Scalability discussion

In previous work on accelerators for SPECFEM3D_GLOBE [57,58] we underlined the need for excellent weak scaling. The main argument was that if a scientist in the wave propagation field is given a larger machine, she/he will often fill it up to 90% typically in order to solve larger problems. But in many situations of practical interest in wave propagation, for instance in geophysics, there is an upper limit in terms of the resolution that one can be interested in. A good illustration of this is seismic wave propagation modelling at the scale of the entire Earth: in seismic data for the whole Earth there is a practical upper limit around 1 Hz above which the data recorded in the field are contaminated by noise or too strongly affected by viscoelastic attenuation [59]. Because of the very high cost of simulations at such high frequencies, currently the seismic modelling community has not yet reached this upper limit and the goal is still to increase the maximum seismic frequency at which calculations can be performed; the current largest runs ever performed on the largest supercomputers in the world are still typically a factor two to five below that limit [45,60], and doubling the maximum seismic frequency of a given run implies an increase of a factor of eight in memory storage and a factor of 16 in compute time for explicit time integration methods. Once this limit is reached, weak scaling (i.e., solving larger problems if given a larger machine) will cease to be interesting, and strong scaling will start to be the main goal again. This will be particularly true for adjoint/inverse acoustic or seismic tomography problems, for which thousands of forward simulations like the one we investigate in this article need to be performed [47,61–63].

Regarding our strong scaling study, in practice owing to the way the SPECFEM3D_GLOBE mesh is designed, a chunk of the Earth cannot be cut into any arbitrary number of mesh slices and only certain combinations are allowed; below we will use all possible combinations in order to get as many measurement points as possible. Regarding the SPECFEM3D_GLOBE weak scaling study, we keep the time step of our time scheme constant by using a larger (physical) mesh when we use more mesh slices, as in [57]. Doing so is fine, since the amount of work per MPI slice remains constant, which is the definition of weak scaling.

## 6. Tibidabo – The first Tegra 2 based ARM cluster

The ARM cluster 'Tibidabo' at the Barcelona Supercomputing Center in Barcelona, Spain, is the first sufficiently large ARM prototype enabling research towards energy efficient HPC systems, as pursued in the 'PRACE'[5] and the 'Mont-Blanc' projects.[6] Tibidabo has 128 nodes, which are organised into blades. Each blade has eight nodes and a shared power supply unit (PSU). The system runs an Ubuntu version 10.10 GNU/Linux operating system. Nodes, built around the SECO Q7 carrier board [64], consist of an NVIDIA Tegra 2 SoC with a dual-core ARM Cortex-A9 processor and 1 GB of LPDDR2-667 memory with a single-channel interface. The maximum amount of memory that is available to Linux is 896 MB (as 128 MB are allocated to the Tegra's GPU, which we do not use), and we found that allocating more than 865 MB for applications makes the system unstable. Each node also integrates two network interface controllers (NIC) – 1 GBit Ethernet connected through PCIe for MPI communication and 100 MBit Ethernet connected through USB for a remote network file system (NFS) holding both user and system data, since the nodes of Tibidabo are diskless. The shared, relatively narrow-bandwidth high-latency 100 MBit Ethernet connection to the file system constitutes a potentially severe bottleneck. We therefore made sure that the codes used in our evaluation perform only the minimum amount of I/O necessary to output the timing results required for our scalability studies, and we exclude costly preprocessing I/O (namely, the SPECFEM3D_GLOBE mesher as described in Section 5.3) from our measurements.

The network topology of Tibidabo is arranged in a tree-like fashion. Switches are cascaded, with each group of 32 nodes being connected through a first-level full-crossbar switch. Each node is reachable within three hops in the cascade.

According to the specifications, the CPU cores consume about 0.5 W [65] each, memory is responsible for an additional 0.4–0.5 W [66] and Ethernet controllers consume 1.1 W in total [67,68].

We measure 6.3 W for a single node when idle, and between 6.8 and 7.6 W under full load, depending on the application. There is a substantial energy waste in system integration that does not contribute to computing, like circuitry and interfaces for (unused) SATA links, HDMI, USB, RTC etc. Due to the prototypical nature of the machine and the fact that carrier boards are designed for embedded development and not for HPC, these components (and also the GPU and the other unused com-

[5] http://www.prace-project.eu.
[6] http://www.montblanc-project.eu.

ponents in the SoC itself) are not power-gated. Furthermore, power supplies are shared between each group of eight nodes, resulting in roughly 7 W PSU waste (0.9 W per node).

## 7. Application performance tuning

In this section, we report on our experience of adapting and tuning the three codes to the Cortex-A9 architecture, and on the microbenchmarks that we performed. Owing to the existing GNU tool chain, the effort we invested has been comparable to that of tuning the codes for any other x86-based architecture. In particular, the effort of porting codes to GPUs with CUDA or OpenCL is substantially higher.

### 7.1. Compiler optimisations

All our codes are compiled with GCC/GFORTRAN version 4.6.2, and we use MPICH2 version 1.4.1. In earlier versions of our measurements we used GCC 4.4.3 and found that the newer compiler gave roughly 5% speedup, indicating continuous improvement in the optimisation backends of the compiler (see below and Section 2). We apply the following generic compiler optimisation flags: `-O3 -foptimize-register-move -fprefetch-loop-arrays -funroll-loops`. Furthermore, we apply several architecture-specific optimisation flags: `-mcpu=cortex-a9-march=armv7-a-mtune=cortex-a9-mfloat-abi=softfp-mfp=vfpv3-dl6-mfpu=vfp`. We refer to the corresponding GCC manual pages and Section 4.1 for details about which features of the processor are enabled by these flags. Detailed experiments revealed that explicitly specifying the floating point ABI and the floating point unit (`-mfloat-abi=softfp-mfp=vfpv3-dl6`) is mandatory for acceptable performance, indicating that the GNU compiler suite optimises for size (and energy) by default [18,17]. For various microbenchmarks, we measured up to 40% performance improvement by applying the latter two flags in single-core single-node benchmarks.

### 7.2. Sustaining the available memory bandwidth

Despite explicitly specifying optimising compiler flags, we still measured only about half of the expected throughput for STREAM-like microbenchmarks using only a single core. This was surprising, since the chip has only one memory controller. We conclude that the available memory bandwidth of the ARM Cortex-A9 processor cannot be fully exploited with serial codes. Adding OpenMP parallelisation to our microbenchmarks alleviated this issue: For large array sizes, we observed a twofold performance improvement, i.e., we were able to achieve the asymptotically expected bandwidth. These findings are supplemented by our timing results of the same benchmark for smaller array sizes, indicating that the OpenMP parallelisation performed by the GCC compiler incurred substantial overhead: Even for a simple vector copy kernel we needed – according to performance counters provided by PAPI [69] – at least an array length of 10,000 values to accommodate for the overhead, substantially larger than for any x86-based architecture that we evaluated.

### 7.3. Floating point subnormal handling

Wave propagation codes with explicit time stepping such as SPECFEM3D_GLOBE potentially suffer from the way subnormal (denormalised) numbers and underflows are treated, i.e., values that fall below the zero threshold of the floating point format: Only floating point noise is computed at mesh points that the propagating waves have not reached yet, leading to almost-zero values multiplied by a time step smaller than one being added to almost-zero values, which results in underflows. Many current microprocessors, including all x86-based designs that we have tested, switch from hardware to software treatment of subnormal numbers. Forcing subnormal treatment to 'flush-to-zero' mode can thus yield substantial performance improvements, as it removes the fallback to software.

We performed a quick set of experiments. On x86, the Intel Compiler Suite provides the -ftz flag to flush subnormals, while the GNU compiler collection requires an explicit assembly language call to enable this mode.[7,8] However, our experiments for the Cortex-A9 architecture indicate that it flushes subnormals to zero by default, similarly to the first generation of fully programmable GPU designs. Consequently, no additional compiler flag is necessary to achieve good performance for SPECFEM3D_GLOBE.

### 7.4. Flat MPI vs. hybrid MPI + OpenMP

Motivated by the microbenchmarks related to achieving full memory bandwidth described in Section 7.2, we extended the mini-application based on FEAST of Section 5.1 to fully support a hybrid MPI + OpenMP implementation. To this end, we extended all kernels in the SBBLAS library to spawn two OpenMP threads as soon as the input array length exceeds the approximate limit of 10,000. The results were non-ambiguous: The hybrid version was always slower than the flat

[7] http://stackoverflow.com/questions/9314534/why-does-changing-0-1f-to-0-slow-down-performance-by-10x.
[8] http://software.intel.com/en-us/articles/x87-and-sse-floating-point-assists-in-ia-32-flush-to-zero-ftz-and-denormals-are-zero-daz/.

MPI version that scheduled two processes per node, both for measurements of the solver alone and for the full mini-application: More precisely, using two MPI processes per node reduced the run time by 60% compared to using a single core (100% would mean ideal scaling), compared to 40–50% for the hybrid version. Consequently, we use a flat MPI implementation of all three codes in our experimental evaluation in Section 8.

### 7.5. Cache blocking

The SBBLAS library of FEAST described in Section 5.1 contains several implementations of the numerical linear algebra kernels required by the solver. The majority of the run time is however spent in only two kernels: residual calculation (matrix–vector multiplication) and inversion of a six-banded 'almost lower-triangular' matrix (smoother application). We thus analysed the matrix–vector multiplication kernel in more detail. In a loop over all entries of the result vector, it reads the nine matrix bands and the right-hand-side vector. Data reuse via caching is only possible for the coefficient vector. This 'naive' implementation already achieves quite good cache usage since the corresponding PAPI counters indicate that the coefficient vector is read from memory approximately three times compared to nine times without any caching. A more advanced 'sliding window' implementation [40] reads in data from three different chunks of the coefficient vector corresponding to the three groups of matrix bands. Indeed, we measure approximately 30% fewer cache misses for this variant, and consequently 30% fewer load instructions to off-chip memory. However, this version is overall 20% *slower* than the naive implementation. The reason for this non intuitive behaviour is that the nested loop structure and the more involved index arithmetic incur substantial overhead, despite aggressive manual unrolling.

Regarding SPECFEM3D_GLOBE, the cache optimisation in the internal mechanical force computation kernel, in which the code spends approximately 90% of its time, implements an optimised approach proposed by Deville et al. [52] in which a pointer to the linear memory segment that corresponds to one finite element is seen as different 2D sub-arrays with varying leading dimension, resulting in substantially fewer cache accesses in the small per-cutplane dense matrix–matrix multiplications, since each point is read from cache a minimised number of times and re-used from registers much longer. We benchmarked both this version and the conventional implementation with strided indexing as well as redundant cache accesses coming from nested loops but found no substantial performance difference on ARM. On x86-type CPUs however, the [52] optimised implementation can be up to about twice faster [63].

## 8. Scalability evaluation and energy vs. time to solution

### 8.1. Weak and strong scalability

The per-core execution rate of the Cortex-A9 is slow compared to x86-based processors. Thus, weak scaling is relatively easy to achieve because there is ample opportunity to overlap communications with computations. In strong scalability, the same problem is distributed to an increasing number of nodes. The challenge then lies in the proportional reduction of the load per node, making it increasingly difficult to overlap communication with computation. For example, the largest strong scaling run below with SPECFEM3D_GLOBE only schedules a local problem size of 9 MB per node, which is very small and thus results in a loss of scalability, at least on x86.

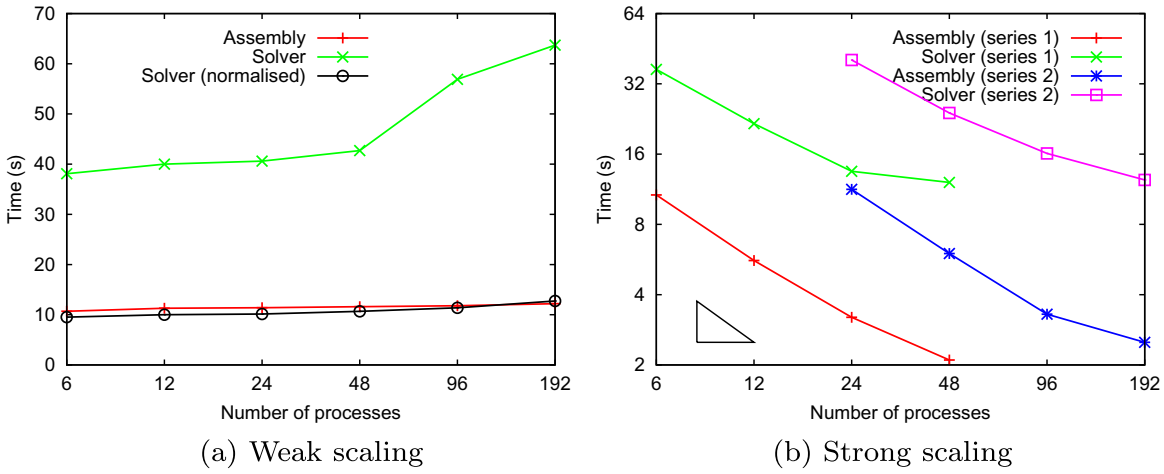
#### 8.1.1. Weak scalability

For FEAST, the underlying configurations are chosen to consume roughly 430 MB of memory per process, corresponding to 95% of the available memory per node. Each patch is refined eight times for a local problem size of $257 \times 257$ grid points ($256 \times 256$ elements) for a total of 24 patches per node. The largest FEAST run comprises 152 million degrees of freedom.

For the HONEI_LBM runs, we choose a comparatively small initial grid size of $250 \times 250$, resulting in a final grid size of $2000 \times 2000$ lattice cells for 192 processes. This small problem size per node allows us to investigate even better how well the communication can be overlapped.

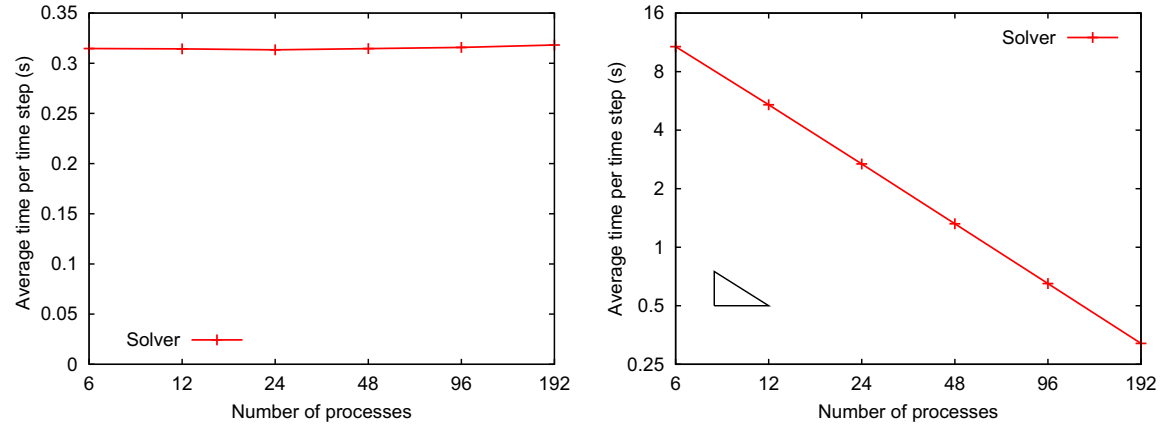
To evaluate SPECFEM3D_GLOBE, we again maximise the local problem size because this is common practice in the geophysics community as explained in Section 5.3. By allocating close to 419 MB of memory per process, the largest run corresponds to 680 million degrees of freedom.

Figs. 1–3(a) depict our weak scaling measurements for up to 192 MPI processes (not including the master process for FEAST), i.e., 96 nodes (out of 112 available) of the Tibidabo cluster. HONEI_LBM and SPECFEM3D_GLOBE scale almost perfectly, as expected. For FEAST, we analyse the finite-element assembly (which includes the serial symbolic and numerical factorisation of UMFPACK on the master node) and the linear solver separately. While there is a small loss in scalability of the assembly due to UMFPACK being tasked with increasing problem sizes, scalability is still very good. Recall that we execute the solver in an *F*-cycle and the depth of the mesh hierarchy is eight, so there are seven coarse grid solves for each multigrid cycle. The run time increase of the solver for the two largest runs is a consequence of an increase in iterations from four to five. To enable better comparison the graph also depicts normalised timings per cycle, indicating excellent scalability of the solver despite performing the coarse grid solves serially, which involves all-to-one and one-to-all communication.



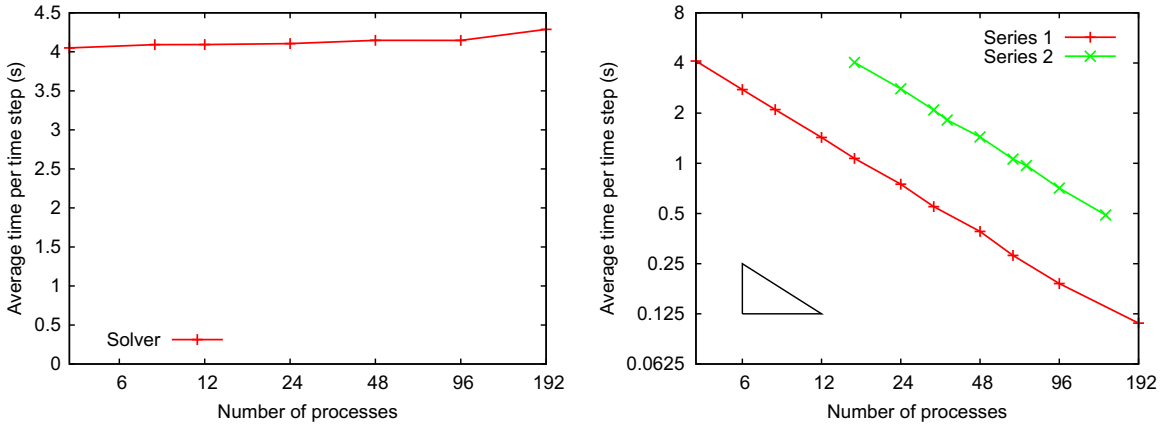

(a) Weak scaling (b) Strong scaling

**Fig. 1.** Weak and strong scaling of FEAST on the Tibidabo ARM cluster, showing absolute time in seconds. A horizontal line corresponds to perfect weak scaling. Normalised timings are obtained by dividing the measured time by the number of solver iterations. Strong scaling is depicted as a log–log plot of absolute time, so that a line with slope −1, as shown by the small black triangle, corresponds to perfect strong scaling.




(a) Weak scaling (b) Strong scaling

**Fig. 2.** Weak and strong scaling of HONEI_LBM on the Tibidabo ARM cluster, showing average time per time step in seconds. A horizontal line corresponds to perfect weak scaling. Strong scaling is depicted as a log–log plot of average time per time step, so that a line with slope −1, as shown by the small black triangle, corresponds to perfect strong scaling.




(a) Weak scaling (b) Strong scaling

**Fig. 3.** Weak and strong scaling of SPECFEM3D_GLOBE on the Tibidabo ARM cluster, showing average time per time step in seconds. A horizontal line corresponds to perfect weak scaling. Strong scaling is depicted as a log–log plot of average time per time step, so that a line with slope −1, as shown by the small black triangle, corresponds to perfect strong scaling.

*8.1.2. Strong scalability*

For FEAST and SPECFEM3D_GLOBE we consider two different starting points taken from the weak scalability series and increase the number of nodes from there. The two FEAST configurations correspond to 4.7 and 19 million degrees of freedom, and the two SPECFEM3D_GLOBE runs are based on 13.8 and 56 million degrees of freedom respectively. The LBM code solves for $2000 \times 2000$ lattice cells, so that the starting point is not aligned with the weak scaling series.

Scalability of FEAST (Fig. 1(b)) seems poor at first for both the finite element assembly and the solver, especially in the last doubling of resources. The main reason is that we suffer from a granularity effect in last step, i.e., the load per node is no longer equally shared between the two cores in each node but rather split 2:1. If we factor this out, we observe the same general trends as already discussed in the weak scalability analysis and the results are promising despite executing the coarse grid solve sequentially. This component alone contributes a constant 0.1 and 0.2 s overall for the two series under investigation, not including the required all-to-one and one-to-all communication. We observe speedups of 3.1 and 3.2 when increasing the number of processors by a factor of eight; and when not taking the largest node number into consideration, the speedups are 2.7 and 2.5 for quadrupled resources.

As illustrated in Figs. 2(b) and 3(b) the two other applications both scale very well since their behaviour is only limited by the ability to overlap communications with computations, and such overlapping is clearly achieved on the ARM cluster under study for the problem sizes under consideration. With SPECFEM3D_GLOBE we observe a speedup of 37 in the first series from 4 to 192 processes, i.e., an increase in resources by a factor of 48; in the second series the factor is 3.3 when quadrupling the resources. HONEI_LBM even scales superlinearly (34-fold speedup for 32 times the resources) because the chosen configuration is small enough in the largest two runs to benefit from additional caching effects.

*8.2. Energy vs. time to solution on Tibidabo and an x86-based reference system*

As both weak and strong scalability on the Tibidabo machine are equally good for our three representative applications, there is hope that compensating for slower execution compared to x86 by using more nodes is not offset by requiring more *energy to solution*, i.e., power consumption integrated over the run time of the application. In what follows, we quantify this aspect.

*8.2.1. Reference x86 system*

Our comparison is based on the Nehalem sub-cluster of the LiDOng machine installed at TU Dortmund, see also Section 3.2. Each node comprises a dual-socket quad-core Intel Xeon X5550 processor clocked at 2.66 GHz. Hyperthreading is turned off in the BIOS. The clock frequency is reduced to 1.6 GHz for each core separately when it is idle. There are eight 2 GB DDR3 memory modules in each node and the network is twofold, with one 1 Gbit Ethernet connection for the Lustre filesystem and one 1 Gbit Ethernet connection for MPI. The switch for MPI is a full crossbar.

*8.2.2. Comparison methodology*

The relation of memory capacity and core count between the two machines enables us to evaluate different energy-performance regimes. Our baseline problem instances are those that we employed in the weak scalability analysis for Tibidabo in the previous section. We now vary the mapping of these problem instances onto the LiDOng machine, see also Table 2. From a memory point of view, 32 LiDOng nodes correspond to 192 Tibidabo nodes. In Configuration 1, we strictly use the same load per core as on Tibidabo and do not change the problem partitioning, i.e., we schedule six processes onto each LiDOng node except for some of the small problem size runs that lead to infeasible partitions for SPECFEM3D_GLOBE, see Section 5.3. In Configuration 2, we compensate for not using all available cores per node in Configuration 1 and repartition all

**Table 2**
Details of the mapping onto various LiDOng nodes for FEAST and HONEI_LBM (top) and SPECFEM3D_GLOBE (bottom).

| Tibidabo | | Configuration 1 | | Configuration 2 | | Configuration 3 | | Configuration 4 | |
|---|---|---|---|---|---|---|---|---|---|
| Cores | Nodes | Nodes | Cores/node | Nodes | Cores/node | Nodes | Cores/node | Nodes | Cores/node |
| 6 | 3 | 1 | 6 | 1 | 8 | 1 | 8 | 1 | 8 |
| 12 | 6 | 2 | 6 | 2 | 8 | 1 | 8 | 1 | 8 |
| 24 | 12 | 4 | 6 | 4 | 8 | 1 | 8 | 2 | 8 |
| 48 | 24 | 8 | 6 | 8 | 8 | 2 | 8 | 4 | 8 |
| 96 | 48 | 16 | 6 | 16 | 8 | 3 | 8 | 6 | 8 |
| 192 | 96 | 32 | 6 | 32 | 8 | 6 | 8 | 12 | 8 |
| | | | | | | | | | |
| 4 | 2 | 1 | 4 | | | 1 | 8 | 1 | 8 |
| 8 | 4 | 1 | 8 | | | 1 | 8 | 1 | 8 |
| 12 | 6 | 2 | 6 | | | 1 | 8 | 1 | 8 |
| 24 | 12 | 4 | 6 | | | 1 | 8 | 2 | 6 |
| 48 | 24 | 8 | 6 | | | 2 | 8 | 3 | 8 |
| 96 | 48 | 16 | 6 | | | 3 | 8 | 6 | 8 |
| 192 | 96 | 32 | 6 | | | 6 | 8 | 12 | 8 |

problem instances to use all eight cores per node. This configuration is not possible with SPECFEM3D_GLOBE for more than half of the problem sizes, so we skip it completely.

These two configurations substantially under-utilise the available memory per node on LiDOng. In Configuration 3, we evaluate the schedule of the problems to as few nodes as possible for a given problem size, and twice the amount of that in Configuration 4. These configurations thus represent the other extreme end of the configuration space and a reasonable intermediate point to evaluate granularity effects in terms of energy to solution.

As already argued previously, we continue to measure time and now also energy to solution. The only difference with respect to the previous results is that for FEAST we sum up the timings for the assembly and the linear solver for simplicity.

*8.2.3. Energy measurement approach*

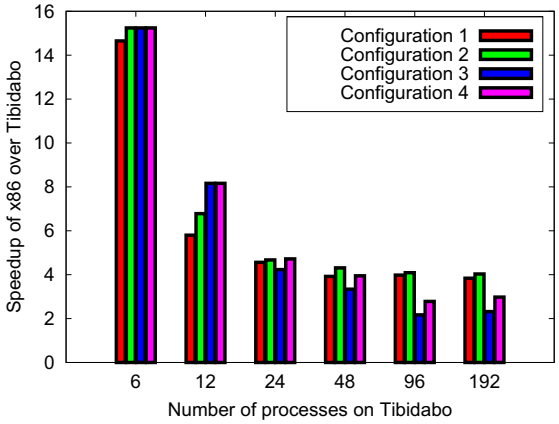
All LiDOng numbers are based on samples obtained by the HP-Health-Monitor software daemon. This software is only installed on one node and thus we cannot generate true energy profiles for full runs. This daemon service samples at a granularity of one second into some log file, and all power consumption readings for LiDOng are based on histograms obtained from matching time stamps from the actual runs with time stamps in this log. Since the load per node does not vary within one run, sampling a single node is not a limitation for our tests. We pursue the same generic measurement approach on Tibidabo and restrict the measurements to a single node. As a baseline sample point we state a measured 6.3 W for a completely idle Tibidabo node and 58 W for a completely idle LiDOng node when all its cores have downclocked.

We emphasise that in all our tests we explicitly exclude the power of network switches, but do include the energy consumed by the network card and also by the local disk drive in a LiDOng node. Given that current Ethernet technologies do not have a linear relationship between the network usage and power consumption, a 10% network utilisation may lead to usage of over 90% of peak switch power [70]. We purport that both machines could be equipped with a similar full-crossbar switch, which would only offset all our measurements (integrated or rather summed up over all nodes) by a constant additive term, that furthermore would be the same for all applications. For Tibidabo configurations that use up to 48 cores, both Tibidabo and LiDOng will use a single network switch, and we can assume that the additive term for switching power is the same in both cases, and for larger configurations Tibidabo will spend more power for the network. Due to significant difference in multicore density of the two systems (2 cores per node in Tibidabo and 8 cores per node in LiDOng), Tibidabo will always require more nodes and network switches to execute the same problem size as LiDOng, and we expect this to change in the next generation of ARM-based products with more cores per chip, so in order to have a fair comparison we exclude the switching power.

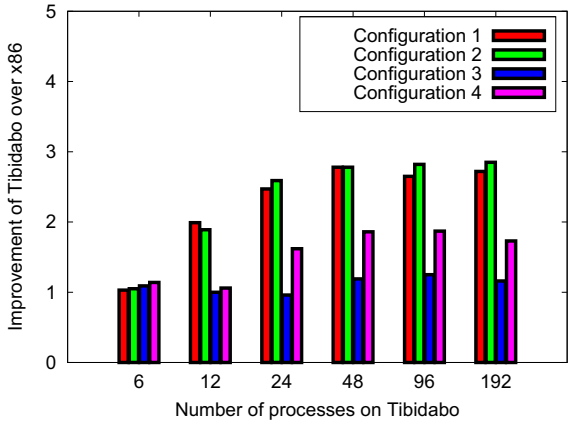
On the Nehalem nodes we observe substantial differences in power consumption depending on the load of the machines (amount of active cores, amount of allocated memory); energy measurements all lie between roughly 210 and 310 W. On Tibidabo, the actual application (and memory load) does not have such a large impact owing to the overall large fixed, static power consumption of the node (see Section 6): In our tests, power consumption varies between 6.8 and 7.6 W per node depending on the various applications' dynamic load.

*8.2.4. Time and energy to solution*

For FEAST, Fig. 4(a) and (b) show the speedup of the four Dortmund configurations over the baseline executions on Tibidabo, and the improvement in terms of energy to solution of Tibidabo, See also Table 2 for auxiliary data related to the mapping of the various configurations to cores and nodes. The most important observation is that for all problem sizes and mappings to the LiDOng machine, Tibidabo is more energy-efficient. We highlight the most important representative data points from the two plots: Looking at the smallest problem first, we see that Tibidabo is only 3–14% more energy efficient,




(a) Speedup LiDOng over Tibidabo (time to solution)



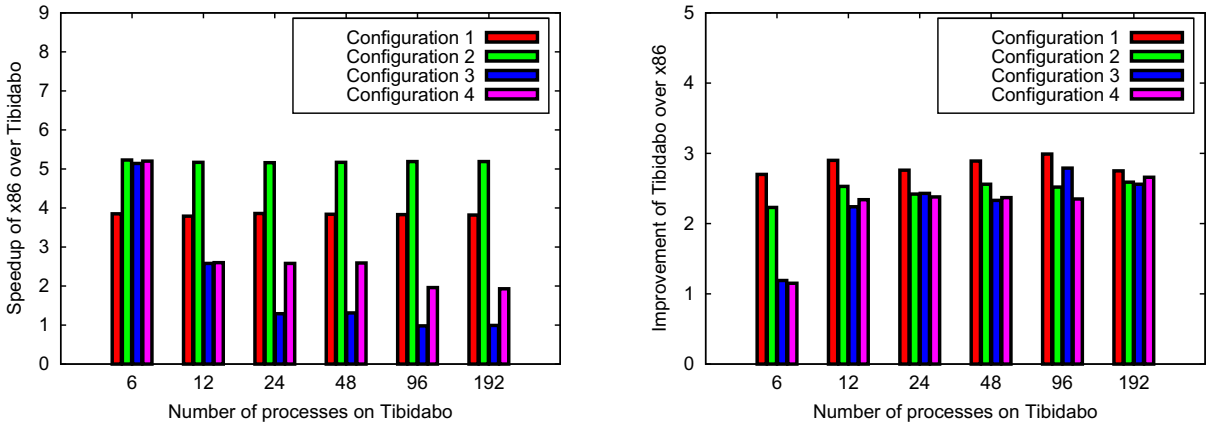

(b) Improvement Tibidabo over LiDOng (energy to solution)

**Fig. 4.** FEAST: Speedup in time to solution of x86 over ARM and improvement in energy to solution of ARM over x86.

but it takes roughly 15 times longer to solve this problem. Note however that this problem size does not require any off-node traffic on Nehalem, all MPI communication is performed via the shared memory mechanism of OpenMPI. On the other hand, for the largest problem under consideration, the fastest Configuration 2 results in a speedup of only four of x86 over ARM, while the latter is almost three times more energy-efficient. Even when using only the minimum number of x86 nodes to solve the problem (Configuration 3), Tibidabo only takes twice as long and still achieves 16% better energy efficiency. Overall, the data allow to pick smooth transition points, e.g., how much slower can we afford to be and how much energy would we save. Also, we see a strict separation (except for the smallest problem that, in all configurations, executes on a single LiDOng node) between the more strong-scaling oriented first two configurations and the more weak-scaling oriented second two configurations, as expected.

Similar trends are visible, and in part also more pronounced, for HONEI_LBM, see Fig. 5(a) and (b). We first note that the differences in energy efficiency improvement are smaller between the four configurations, a direct consequence of the comparatively small weak scaling problem sizes chosen for this application. Except for the smallest problem size, Tibidabo is always more energy-efficient. For Configuration 3 and larger problem sizes we observe almost no speedup on x86, but still more than a factor of 2.5 improvement in energy to solution by Tibidabo. Looking at the other extreme, i.e., Configuration 2, we see that the x86-based cluster is at most 5.5 times faster, but requires almost three times more energy to achieve this speedup.

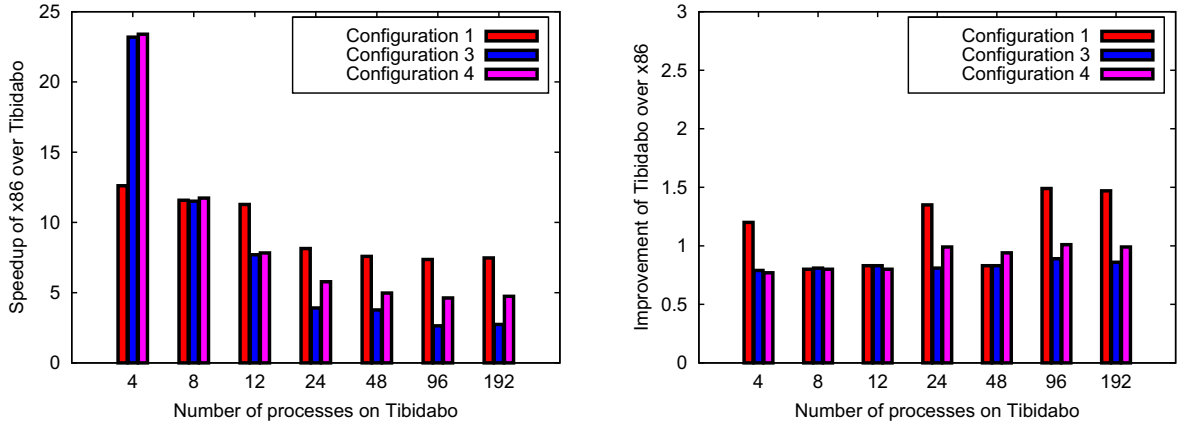
The differences between the two classes of mapping configurations are also less pronounced for SPECFEM3D_GLOBE, as depicted in Fig. 6(a) and (b). See also Table 2 for auxiliary data related to the mapping of the various configurations to cores and nodes. Since the speedup of x86 over ARM is on average more than twice higher than for the other two applications, Tibidabo is not always as energy-efficient as the Nehalem cluster. There are certain granularity effects stemming from limitations in partitioning of SPECFEM3D_GLOBE, e.g., Configuration 3 at the smallest problem size is twice faster than Config-




(a) Speedup LiDOng over Tibidabo (time per time step)

(b) Improvement Tibidabo over LiDOng (energy per time step)

**Fig. 5.** HONEI_LBM: Speedup in time per time step of x86 over ARM and improvement in energy per time step of ARM over x86.




(a) Speedup LiDOng over Tibidabo (time per time step)

(b) Improvement Tibidabo over LiDOng (energy per time step)

**Fig. 6.** SPEDFEM3D_GLOBE: Speedup in time per time step of x86 over ARM and improvement in energy per time step of ARM over x86.

uration 1 because it uses twice the amount of cores per node. When factoring these effects out, the prototypical ARM-based cluster is more efficient in terms of energy to solution when compared to the fastest possible solution on x86 using the same number of cores, but less favourable when compared to the most energy-efficient (and hence slowest) x86 mapping. The differences are always in the range of 20%, in one direction or the other. The reason for the reduced speedup is that in contrast to FEAST, SPECFEM3D_GLOBE is compute- rather than memory-bound for the most part on x86, and the difference in peak compute performance of the two architectures is much higher than the difference in peak memory bandwidth.

## 9. Conclusions and outlook

Due to various architectural improvements resulting in increased floating-point performance, processor designs originating from mobile computing are starting to be of interest in high performance computing. There are two main aspects that make them particularly worth considering as target architectures: First, in contrast to accelerator-like approaches, no fundamental rewrites of production codes are necessary because standard compilers and tool chains have been supported for a long time already. Second, they promise substantially lower power consumption, which in the exascale era will become the most important factor in the design of HPC systems.

The extreme energy efficiency compared to x86-based processors is achieved by a substantial reduction in per-core performance. Consequently, more cores (and thus, more nodes) have to be used to solve the same problem in a comparable amount of time, implying much stricter scalability requirements. But lower power does not automatically translate into lower energy, since the computations require either more time, more nodes, or both to finish. In this article, we have quantitatively evaluated this trade-off between time- and energy to solution for three representative applications to numerically solve PDE problems on a cluster of 96 ARM Cortex-A9 dualcore processors and on a 32-node dual-socket Intel Nehalem cluster that provides the same amount of total memory.

We have shown that indeed, substantial reductions in terms of energy to solution are possible at only moderate slowdowns compared to the x86-based cluster. The transitions between various time and energy targets allow to answer questions such as 'how much slower can the simulation afford to be for certain energy savings' and of course, the other way around. However, for one application that is compute-bound on x86 already, the difference between the peak floating point performance of the two architectures is too large to achieve a gain in energy efficiency.

Several important avenues for future work arise from our findings: First, the next generations of ARM processor designs have been announced to more than double floating point performance through SIMD features, higher clock rates and larger caches, at only very moderately increased power consumption. We thus believe that in the near future, *all* applications will be more energy-efficient on clusters built from these designs than on conventional ones. Since this improvement affects only the 'CPU' component of such clusters, the comparatively large static power consumption that we observed in our prototype cluster, i.e., power consumption for components and system integration that do not contribute to compute, will play a less dominant role. Once the GPU components of the SoC become properly programmable, their use will further boost energy efficiency, albeit at the loss of the programmability advantage outlined above. Finally, in view of the exascale era, the reliability and mean-time-between-failure of clusters built from ARM processors will have to be examined: In their original market of mobile computing, SoC built around ARM processors are designed for maximum reliability, but in HPC installations a great number of them needs to be included into a system, increasing the probability of failures. This problem is common to all exascale architectures (see e.g. [71]), not specific to ARM-based systems.

## Acknowledgments

The authors thank Jesús Labarta for initial exchange of ideas, Sven H. M. Buijssen for help with debugging and compilers, Thomas Rohkämper for setting up some of the FEAST coarse grids, Manh Ha Nguyen and Harald Servat for help with ParaVer, Harald Servat for detailed explanations of PAPI measurements, Brice Videau, Augustin Degomme and Jean-François Méhaut for discussions about tuning for ARM and on the Tibidabo cluster, Gabriele Carteni for substantial assistance in debugging performance and stability issues on Tibidabo, and Max Rietmann for discussion about Flush-To-Zero (FTZ) on current processors. We also thank the LiDOng team at TU Dortmund (Christian Becker, Jörg Gehrke, Sven Buijssen) for providing detailed energy cost data and installing the power measurement daemon. This work has been conducted during Dominik Göddeke's three-month visit to CNRS in Marseille, France, which was made possible by the Rudolf Chaudoire foundation and TU Dortmund through the Rudolf Chaudoire 2011 Award. Part of this work was done in the context of the European 'Mont-Blanc: European scalable and power efficient HPC platform based on low-power embedded technology' #288777 project of call FP7-ICT-2011-7; and the Tibidabo cluster is an outcome of the PRACE project (European Community funding under Grants RI-261557 and RI-283493). Markus Geveler has been partially supported by the German Ministry of Education and Research (BMBF), project 'SKALB' (01IH08003D) and the German Research Foundation (DFG), projects SFB 708/TB 1 and SPP 1423 (TU 102/32-2).

## References

[1] J. Koomey, Growth in Data Center Electricity use 2005 to 2010, Analytics Press, 2011.
[2] US environmental protection agency, Report to congress on server and data center energy efficiency, Energy star, Program, 2007.

[3] S. Khan, P. Bouvry, T. Engel, Energy-efficient high-performance parallel and distributed computing, Journal of Supercomputing 60 (2) (2012) 163–164, http://dx.doi.org/10.1007/s11227-010-0485-0.
[4] S. Sharma, C.-H. Hsu, W.-C. Feng, Making a case for a Green500 list, in: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), 2006, pp. 1–8, http://dx.doi.org/10.1109/IPDPS.2006.1639600.
[5] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M.S. Mueller, W.E. Nagel, H. Nakashima, M.E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, K. Yelick, The international exascale software project roadmap, International Journal of High Performance Computing Applications 25 (1) (2011) 3–60, http://dx.doi.org/10.1177/1094342010391989.
[6] D.E. Keyes, Exaflop/s: the why and the how, Comptes Rendus Mécanique 339 (2–3) (2011) 70–77, http://dx.doi.org/10.1016/j.crme.2010.11.002.
[7] B. Schäppi, B. Przywara, F. Bellosa, T. Bogner, S. Weeren, R. Harrison, A. Anglade, Energy efficient servers in Europe – energy consumption, saving potentials and measures to support market development for energy efficient solutions, report, Intelligent energy Europe project, 2009.
[8] Lawrence Berkeley national laboratory, High-performance buildings for high-tech industries: data centers, 2006, <http://hightech.lbl.gov/datacenters.html>.
[9] ARM holdings plc, Annual report and accounts 2011, 2011.
[10] A. Gutierrez, R.G. Dreslinski, T.F. Wenisch, T. Mudge, A. Saidi, C. Emmons, N. Paver, Full-system analysis and characterization of interactive smartphone applications, in: Workload Characterization (IISWC'11), 2011, pp. 81–90, http://dx.doi.org/10.1109/IISWC.2011.6114205.
[11] K. Fürlinger, C. Klausecker, D. Kranzlmüller, Towards energy efficient parallel computing on consumer electronic devices, in: D. Kranzlmüller, A.M. Toja (Eds.), Information and Communication on Technology for the Fight against Global Warming (ICT-GLOW 2011), Lecture Notes in Computer Science, vol. 6868, Springer, 2011, pp. 1–9.
[12] A. Buttari, P. Luszczek, J. Kurzak, J.J. Dongarra, G. Bosilca, SCOP3: a rough guide to scientific computing on the PlayStation 3, Tech. Rep., Innovative computing laboratory, University of Tennessee Knoxville, UT-CS-07-595, 2007.
[13] G. Dasika, A. Sethia, T. Mudge, S. Mahlke, Low power scientific computing, in: Workshop on New Directions in Computer Architecture, IEEE/ACM International Symposium on Microarchitecture, 2009, pp. 7–8.
[14] W. Augustin, V. Heuveline, J.-P. Wei, Convey HC-1 hybrid core computer – the potential of FPGAs in numerical simulation, in: New Frontiers in High-performance and Hardware-aware, Computing (HipHaC'11), 2011, pp. 1–8.
[15] K. Fan, M. Kudlur, G. Dasika, S. Mahlke, Bridging the computation gap between programmable processors and hardwired accelerators, in: High Performance Computer Architecture (HPCA 2009), 2009, pp. 313–322, http://dx.doi.org/10.1109/HPCA.2009.4798266.
[16] M. Etinski, J. Corbalan, J. Labarta, M. Valero, Understanding the future of energy-performance trade-off via DVFS in HPC environments, Journal of Parallel and Distributed Computing 72 (4) (2012) 579–590, http://dx.doi.org/10.1016/j.jpdc.2012.01.006.
[17] J.-J. Kim, S.-Y. Lee, S.-M. Moon, S. Kim, Comparison of LLVM and GCC on the ARM platform, in: Embedded and Multimedia Computing (EMC 2010), 2010, pp. 1–6, http://dx.doi.org/10.1109/EMC.2010.5575631.
[18] D. Melnik, A. Belevantsev, D. Plotnikov, S. Lee, A case study: optimizing GCC on ARM for performance of libevas rasterization library, in: High-Performance and Embedded Architectures and Compilers (HiPEAC 2010) – Workshop on GCC Research Opportunities, 2010, pp. 34–46.
[19] K. Li, Energy efficient scheduling of parallel tasks on multiprocessor computers, The Journal of Supercomputing 60 (2) (2012) 223–247, http://dx.doi.org/10.1007/s11227-010-0416-0.
[20] Y.C. Lee, A. Zomaya, Energy efficient utilization of resources in cloud computing systems, The Journal of Supercomputing 60 (2) (2012) 268–280, http://dx.doi.org/10.1007/s11227-010-0421-3.
[21] E.K. Lee, I. Kulkarni, D. Pompili, M. Parashar, Proactive thermal management in green datacenters, The Journal of Supercomputing 60 (2) (2012) 165–195, http://dx.doi.org/10.1007/s11227-010-0453-8.
[22] D. Göddeke, R. Strzodka, S. Turek, Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations, International Journal of Parallel, Emergent and Distributed Systems 22 (4) (2007) 221–256, http://dx.doi.org/10.1080/17445760601122076.
[23] H. Anzt, V. Heuveline, B. Rocker, M. Castillo, J.C. Fernández, R. Mayo, E.S. Quintana-Ortí, Power consumption of mixed precision in the iterative solution of sparse linear systems, in: 19th International Parallel and Distributed Processing Symposium (IPDPS 2011) – Seventh Workshop on High-Performance, Power-Aware Computing (HPPAC 2011), 2011, pp. 829–836, http://dx.doi.org/10.1109/IPDPS.2011.226.
[24] H. Anzt, M. Castillo, J. Fernández, V. Heuveline, F. Igual, R. Mayo, E. Quintana-Ortí, Optimization of power consumption in the iterative solution of sparse linear systems on graphics processors, Computer Science – Research and Development (2011) 1–9, http://dx.doi.org/10.1007/s00450-011-0195-8.
[25] O. Azizi, A. Mahesri, B.C. Lee, S.J. Patel, M. Horowitz, Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis, SIGARCH Computer Architecture News 38 (3) (2010) 26–36, http://dx.doi.org/10.1145/1816038.1815967.
[26] A. Munir, S. Ranka, A. Gordon-Ross, High-performance energy-efficient multicore embedded computing, IEEE Transactions on Parallel and Distributed Systems 23 (4) (2012) 684–700, http://dx.doi.org/10.1109/TPDS.2011.214.
[27] W.J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R.C. Harting, V. Parikh, J. Park, D. Sheffield, Efficient Embedded Computing, Computer 41 (7) (2008) 27–32, http://dx.doi.org/10.1109/MC.2008.224.
[28] T. Mudge, U. Hölzle, Challenges and opportunities for extremely energy-efficient processors, IEEE Micro 30 (2010) 20–24, http://dx.doi.org/10.1109/MM.2010.61.
[29] ARM Ltd., Cortex-A series programmer's guide version 2.0, 2011.
[30] ARM Ltd., Cortex-A9 revision: r3p0 technical reference manual, 2011.
[31] NVIDIA corporation, The benefits of multiple cpu cores in mobile devices, 2010.
[32] ARM Ltd., Cortex-A9 floating-point unit revision: r3p0 technical reference manual, 2011.
[33] J.D. Owens, D.P. Luebke, N.K. Govindaraju, M.J. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, in: Eurographics 2005, State of the Art Reports, 2005, pp. 21–51.
[34] J.D. Owens, D.P. Luebke, N.K. Govindaraju, M.J. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, Computer Graphics Forum 26 (1) (2007) 80–113, http://dx.doi.org/10.1111/j.1467-8659.2007.01012.x.
[35] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, D. Glasco, GPUs and the future of parallel computing, IEEE Micro 31 (5) (2011) 7–17, http://dx.doi.org/10.1109/MM.2011.89.
[36] N. Rajovic, N. Puzovic, L. Vilanova, C. Villavieja, A. Ramirez, The low-power architecture approach towards exascale computing, in: Proceedings of the Second Workshop on Scalable Algorithms for Large-scale systems (ScalA'11), 2011, pp. 1–2, http://dx.doi.org/10.1145/2133173.2133175.
[37] S. Turek, D. Göddeke, C. Becker, S.H. Buijssen, H. Wobker, FEAST – realisation of hardware-oriented numerics for HPC simulations with finite elements, Concurrency and Computation: Practice and Experience 22 (6) (2010) 2247–2265, http://dx.doi.org/10.1002/cpe.1584.
[38] S. Turek, D. Göddeke, S.H. Buijssen, H. Wobker, Hardware-oriented multigrid finite element solvers on GPU-accelerated clusters, in: J. Kurzak, D.A. Bader, J.J. Dongarra (Eds.), Scientific Computing with Multicore and Accelerators, CRC Press, 2010, pp. 113–130. chap. 6.
[39] S. Turek, C. Becker, S. Kilian, Hardware–oriented numerics and concepts for PDE software, Future Generation Computer Systems 22 (1–2) (2004) 217–238, http://dx.doi.org/10.1016/j.future.2003.09.007.
[40] S. Turek, C. Becker, A. Runge, The FEAST indices – realistic evaluation of modern software components and processor technologies, Computers and Mathematics with Applications 41 (10) (2001) 1431–1464, http://dx.doi.org/10.1016/S0898-1221(01)00108-0.

[41] T.A. Davis, A column pre-ordering strategy for the unsymmetric-pattern multifrontal method, ACM Transactions on Mathematical Software 30 (2) (2004) 165–195, http://dx.doi.org/10.1145/992200.992205.
[42] M. Schäfer, S. Turek, Benchmark computations of laminar flow around a cylinder, in: E.H. Hirschel (Ed.), Flow Simulation with High-Performance Computers II, Notes on Numerical Fluid Mechanics, vol. 52, Vieweg, 1996, pp. 547–566.
[43] J.G. Zhou, Lattice Boltzmann Methods for Shallow Water Flows, Springer, 2004.
[44] M. Geveler, D. Ribbrock, S. Mallach, D. Göddeke, S. Turek, A simulation suite for lattice-Boltzmann based real-time CFD applications exploiting multi-level parallelism on modern multi- and many-core architectures, Journal of Computational Science 2 (2011) 113–123, http://dx.doi.org/10.1016/j.jocs.2011.01.008.
[45] L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michéa, N. Le Goff, A. Snavely, J. Tromp, High-frequency simulations of global seismic wave propagation using SPECFEM3DGLOBE on 62 thousand processor cores, in: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008, pp. 1–11, http://dx.doi.org/10.1145/1413370.1413432 (article #60, Gordon Bell Prize finalist article).
[46] D. Komatitsch, S. Tsuboi, J. Tromp, The spectral-element method in seismology, in: A. Levander, G. Nolet (Eds.), Seismic Earth: Array Analysis of Broadband Seismograms, Geophysical Monograph, vol. 157, American Geophysical Union, 2005, pp. 205–228.
[47] J. Tromp, D. Komatitsch, Q. Liu, Spectral-Element and Adjoint Methods in Seismology, Communications in Computational Physics 3 (1) (2008) 1–32.
[48] L.C. Wilcox, G. Stadler, C. Burstedde, O. Ghattas, A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media, Journal of Computational Physics 229 (24) (2010) 9373–9396, http://dx.doi.org/10.1016/j.jcp. 2010.09.008.
[49] C.A. Acosta Minolia, D.A. Kopriva, Discontinuous Galerkin spectral element approximations on moving meshes, Journal of Computational Physics 230 (5) (2011) 1876–1902, http://dx.doi.org/10.1016/j.jcp. 2010.11.038.
[50] S.P. Oliveira, G. Seriani, Effect of element distortion on the numerical dispersion of spectral element methods, Communications in Computational Physics 9 (4) (2011) 937–958.
[51] Y. Maday, A.T. Patera, Spectral-element methods for the incompressible Navier–Stokes equations, in: State of the Art Survey in, Computational Mechanics, 1989, pp. 71–143.
[52] M.O. Deville, P.F. Fischer, E.H. Mund, High-Order Methods for Incompressible Fluid Flow, Cambridge University Press, 2002.
[53] G. Cohen, Higher-Order Numerical Methods for Transient Wave Equations, Springer, 2002.
[54] A.M. Dziewoński, D.L. Anderson, Preliminary reference Earth model, Physics of the Earth and Planetary Interiors 25 (1981) 297–356.
[55] D. Komatitsch, S. Tsuboi, C. Ji, J. Tromp, A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth simulator, in: SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003, pp. 4–11, http://dx.doi.org/10.1109/SC.2003.10023 (Gordon Bell Prize winner article).
[56] D. Komatitsch, J. Labarta, D. Michéa, A simulation of seismic wave propagation at high resolution in the inner core of the Earth on 2166 processors of MareNostrum, in: J. Palma, P. Amestoy, M. Daydé, M. Mattoso, J. Lopes (Eds.), High Performance Computing for Computational Science – VECPAR 2008, Lecture Notes in Computer Science, vol. 5336, Springer, 2008, pp. 364–377.
[57] D. Komatitsch, G. Erlebacher, D. Göddeke, D. Michéa, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, Journal of Computational Physics 229 (20) (2010) 7692–7714, http://dx.doi.org/10.1016/j.jcp. 2010.06.024.
[58] D. Komatitsch, Fluid-solid coupling on a cluster of GPU graphics cards for seismic wave propagation, Comptes Rendus Mécanique 339 (2011) 125–135, http://dx.doi.org/10.1016/j.crme.2010.11.007.
[59] F.A. Dahlen, J. Tromp, Theoretical Global Seismology, Princeton University Press, 1998.
[60] D. Komatitsch, L.P. Vinnik, S. Chevrot, SHdiff/SVdiff splitting in an isotropic Earth, Journal of Geophyical Research 115 (B7) (2010) B07312, http://dx.doi.org/10.1029/2009JB006795.
[61] V. Akçelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E.J. Kim, J. Lopez, D. O'Hallaron, T. Tu, J. Urbanic, High-resolution forward and inverse earthquake modeling on terascale computers, in: SC'03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, 2003, pp. 52–72, http://dx.doi.org/10.1145/1048935.1050202 (Gordon Bell Prize winner article).
[62] A. Fichtner, Full Seismic Waveform Modelling and Inversion, Springer, 2010.
[63] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, J. Tromp, Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes, Geophysics Journal International 186 (2) (2011) 721–739, http://dx.doi.org/10.1111/j.1365-246X.2011.05044.x.
[64] S.R.L. Seco, SECOCQ7-MXM, 2012, <http://www.seco.com/en/item/secocq7-mxm/>.
[65] ARM Ltd., Cortex-A9 processor, 2011, <http://www.arm.com/products/processors/cortex-a/cortex-a9.php>.
[66] Micron Technology, Inc., DDR2 SDRAM system-power calculator, 2004, <http://www.micron.com//media/Documents/Products/Power0Calculator/4298ddr2powercalcweb.xls>.
[67] Intel Corporation, Intel82574 Gigabit ethernet controller family: datasheet, 2012, <http://developer.intel.com/content/www/us/en/ethernet-controllers/82574l-gbe-controller-datasheet.html>.
[68] Standard Microsystems Corp. (SMSC), USB 2.0 Hub and 10/100 ethernet controller, 2012, <http://www.smsc.com/media/DownloadsPublic/DataSheets/9514.pdf>.
[69] S. Browne, J. Dongarra, N. Garner, G. Ho, P. Mucci, A portable programming interface for performance evaluation on modern processors, International Journal of High Performance Computing Applications 14 (3) (2000) 189–204, http://dx.doi.org/10.1177/109434200001400303.
[70] P. Reviriego, J. Hernandez, D. Larrabeiti, J. Maestro, Performance evaluation of energy efficient ethernet, IEEE Communications Letters, 1089-7798 13 (9) (2009) 697–699.
[71] L.A. Bautista-Gomez, D. Komatitsch, N. Maruyama, S. Tsuboi, F. Cappello, S. Matsuoka, T. Nakamura, FTI: high performance fault tolerance interface for hybrid systems, in: SC'11: Proceedings of the 2011 ACM/IEEE Conference on Supercomputing, 2011, pp. 32:1–32:12, http://dx.doi.org/10.1145/2063384.2063427 (article #32).