Journal of Parallel and Distributed Computing 155 (2021) 50–62

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

The cluster coffer: Teaching HPC on the road

Philipp Gschwandtner*, Alexander Hirsch, Peter Thoman, Peter Zangerl, Herbert Jordan, Thomas Fahringer

University of Innsbruck, Technikerstrasse 21a, Innsbruck, Austria

A R T I C L E   I N F O

Article history:
Received 30 October 2020
Received in revised form 24 February 2021
Accepted 28 April 2021
Available online 5 May 2021

MSC:
00-01
99-00

Keywords:
High performance computing
Teaching
Parallel programming
Portability
Public outreach

A B S T R A C T

Teaching parallel programming and HPC is a difficult task. There is a large number of sophisticated hardware and software components, each complex on their own and often showing non-intuitive interaction when used in combination. We consider education in HPC among the more difficult topics in computer science due to the fact that larger distributed memory systems are ubiquitous yet inaccessible and intangible to students. In this work, we present the Cluster Coffer, a miniature cluster computer based on 16 ARM compute boards that we believe is suitable for reducing the entry barrier to HPC in teaching and public outreach. We discuss our design goals for providing a portable, inexpensive system that is easy to maintain and repair. We outline the implementation path we took in terms of hardware and software, in order to provide others with the information required to reproduce and extend our work. Finally, we present two use cases for which the Cluster Coffer has been used multiple times, and will continue to be used in the upcoming years.

1. Introduction

High performance computing is an increasingly complex branch of computer science. The number of sophisticated software and hardware components as well as their complex interaction and coordination renders HPC a challenging topic, especially in teaching. Execution units, cores, caches, sockets, processor links, nodes, network links, and storage subsystems need to be understood and their capabilities and intricacies managed on multiple levels of the HPC hardware architecture hierarchy. Furthermore, a plethora of software tools and paradigms are available for interacting with these architecture aspects, including parallel programming models such as MPI [16], OpenMP [17], or SYCL [25], parallel algorithms, efficient data structures, optimizing compilers, manual code transformations, performance analysis and visualization tools or debuggers.

What makes HPC challenging from an educational perspective is the fact that access to many of the tools required is restricted. While there is a multitude of free implementations of parallel programming models, predominantly including OpenMP and MPI, HPC is a hardware-oriented field of study but access to HPC hardware is hard to come by. Multi-/many-core CPUs are readily available these days, but systems that consist of multiple CPUs or nodes, challenging distributed memory programming skills, are usually not feasibly available to students. This makes these systems intangible and often impedes HPC teaching efforts, as many characteristics of HPC hardware and software can only be shown in theory, with little practical application for students. For example, the effect of DVFS on not purely compute-bound HPC workloads in a distributed memory setting cannot be investigated in detail on commodity shared-memory hardware.

In order to mitigate this issue, it would be beneficial to have a miniature HPC system available that is low-cost and easy to maintain, yet representative of larger systems in its characteristics and use cases. To that end, we present the Cluster Coffer,[1] a mobile HPC platform consisting of 16 multi-core compute nodes interconnected via Gigabit Ethernet in a single robust metal carrying case. The goal of the Cluster Coffer and this paper is to

- show the feasibility of constructing small-scale but representative HPC systems that can be easily relocated to a given target audience,
- illustrate the benefits of using such a system to demonstrate all major HPC aspects in teaching and for public dissemination, especially using live interaction and application steering, and

* Corresponding author.
E-mail address: philipp.gschwandtner@uibk.ac.at (P. Gschwandtner).

[1] *Coffer*, besides its true meaning in English, is a play on words coming from the German word *Koffer*, which means “suitcase”.

https://doi.org/10.1016/j.jpdc.2021.04.013

- provide enough material and information for others to reproduce our work for their own use and build upon it.

This paper is structured as follows: Section 2 lists selected related work and puts our system in perspective. Section 3 discussed design principles while Section 4 and Section 5 present hardware and software implementation details, respectively. Our use cases, including one with live application steering by the audience, are presented and illustrated in Section 6, with Section 8 providing concluding remarks and future ideas.

2. Related work

Since the rise of multi-core 64 bit ARM CPUs, a great number of embedded computing boards emerged on the market, especially with the appearance of the Raspberry Pi line. These embedded boards are predestined for experimenting with computer science and also with HPC, and as such, many miniature cluster computers showed up on the landscape. They can be classified in multiple ways, including their intended use, portability, performance, or focus on specific aspects of computer science such as Cloud computing or feature sets such as power instrumentation. A comprehensive study of miniature clusters built from linking individual compute boards has already been created by Johnston et al. [9] and would exceed the scope of this work, given the vast amount of systems available due to inexpensive components and relatively mature software stacks. In contrast to that, the goal of this section is to give a small overview and outline the different perspectives and use cases of these clusters, while almost all systems contribute in the area of teaching and public outreach in (parallel) programming and partly also HPC.

One of the earliest systems and pioneer is Iridis-Pi, a cluster constructed in 2012 by Cox et al. [4] from 64 Raspberry Pi Model B nodes. It is enclosed in a housing made from LEGO bricks, which makes it less portable than our suitcase-based design. On the other hand, its 64 nodes allow for more fine-grained distributed memory scaling research. Due to its age, the cluster is limited to 700 MHz ARM1176JZF-S RISC processors, 256 MB RAM and a 100 MBit/s Ethernet network. Compared to our Cluster Coffer, besides its higher performance attributed to the newer architecture, it also offers per-node power instrumentation and Gigabit Ethernet. Similar approaches of low-cost housings such as LEGO include 3D-printed designs as used in the Raspberry Pi Cloud project [27], wooden panels [14], or even designs that simply omit the housing altogether [19].

Newer systems prominently include Wee Archie built by EPCC of the University of Edinburgh [6]. It features 18 Raspberry Pi 2 compute nodes in an acrylic glass enclosure and each board is equipped with small dot-matrix screens that display single-pixel bar charts holding e.g. CPU, memory, or storage load information per compute node. To the best of our knowledge, there is no per-node power instrumentation available on Wee Archie and there is no information on whether individual nodes can be switched off easily for resilience research. Beyond the cluster itself, EPCC offers a tutorial on how to build even smaller versions of Wee Archie.

Compared to these systems originating from educational institutions, there are also commercially available, semi-portable ARM-based clusters that are not mainly used for teaching or public outreach. Such systems include BitScope [13], a larger system consisting of 750 Raspberry Pi nodes in a total of 5 racks. These systems are used for testing scientific applications before moving to large-scale systems. Compared to such ARM-based clusters, our Cluster Coffer with its lower performance and infrastructure requirements aims at affordable education and public outreach rather than providing an intermediate HPC stage for scientists when moving to larger systems.

Finally, there is the option of using Cloud resources, Docker instances or simply remote access to real-world clusters for trying to achieve the same goals. However, these approaches share common disadvantages preventing their use for this purpose, since their lack of on-site physical access to all components involved reduces the engagement and attention level in our experience. Furthermore, for public outreach, it increases the entry threshold since the target audience often does not fully comprehend the workings behind the scenes when discussing e.g. Cloud computing. This is naturally one of the main goals of Cloud computing, but in this case it hinders teaching hardware-oriented parallel programming concepts. Virtualization also usually entails the absence of suitable power instrumentation and potentially introduces performance perturbation caused by the co-scheduling of virtual machines, which limits its use in teaching e.g. the concept of DVFS and performance/energy trade-offs.

To the best of our knowledge, ours is one of few works that offer finer-grained power instrumentation and the first work that is used for teaching and public outreach that engages the audience in live interaction with a simulation coming from a real physics application. We believe that this live interaction, real-world use-cases, and physical access to and visibility of all hardware components involved in the computation are crucial. Our personal experience so far, when presenting the Cluster Coffer, gathered over an aggregated total of several weeks' time, supports our hypothesis.

3. Motivation and design principles

In order to construct a successful substitute for real-world HPC systems and do so in a goal-driven fashion, there are several design principles that need to be established, both in hardware and in software.

3.1. Hardware

Our hardware design goals include that our system is representative of modern HPC systems. As such, it should hold at least several multi- or many-core nodes connected via a fast network interconnect. Offering several nodes each equipped with several cores ensures that the system can accept hybrid MPI+OpenMP-like application workloads. It should use a wide-spread CPU architecture supported by conventional modern compilers that are also common to larger HPC systems. The system should have at least one head node responsible for handling user login, compilation and application start-up.

Beyond classic HPC requirements, the system should be mobile and transportable by a single person, putting restrictions on size, weight, and handling. Its power-on process should be as hassle-free as possible and its power requirements be limited to a single commodity power cord and a thermal envelope that can be cooled reasonably. Given that power and energy consumption concerns have gained importance in HPC over the past years, we also need monitoring capabilities in that regard for e.g. multi-objective optimization projects. Finally, the system should be assembled from low-cost commodity hardware components to ensure technical and economical feasibility of the project.

3.2. Software

The system should also be representative of modern HPC systems in terms of software. As such, it should be running a standard Linux operating system, provide established compilers such as gcc and clang, and support OpenMP and MPI applications, given their ubiquity in HPC. Furthermore, to simplify application launches for distributed-memory environments such as MPI, storage should be network-mounted and all nodes should be accessible via SSH.

Table 1
Cluster Coffer hardware architecture.

Node	SoC	CPU (Cortex)	GPU	RAM	Network
1x NanoPC T4	Rockchip	2x A72, 1.8 GHz	Mali	4 GB	Gigabit
16x NanoPi M4	RK3399	4x A53, 1.4 GHz	T864	2 GB	Ethernet

Beyond these basic requirements, we aim for low maintenance overhead when installing, updating or modifying the software stack on the head node or any of the compute nodes, favoring a centralized maintenance approach. Finally, we need to be able to conveniently access non-functional metrics such as CPU load or power consumption for any of the nodes. These metrics should also be displayed in a graphical fashion that both allows to quickly ascertain the overall state of the Cluster Coffer at a glance, yet provide a visually intuitive way of providing information to non-experts.

3.3. Target audience and educational concept

Beyond hardware and software, there are also specific design goals regarding educational use that we aim for. In order to warrant development effort, the final system should be suitable for being used in teaching students from secondary schools up to undergraduate university levels. Furthermore, it should be highly engaging and interactive in order to attract as much attention as possible among the respective target audiences. This entails additional hardware requirements (directly visible and accessible components) and software (compelling, flexible teaching cases that can be varied in the level of detail of discussion; similar environment to real-life HPC systems, yet larger degree of freedom in environment settings). To further increase the merit of such a system, it should also be capable of presenting parallel and high-performance computing topics to a broader audience, including students from other fields of study and in public outreach. In order to maximize success, this aspect should also be interactive but lower-threshold, and we aim for content that is highly relevant and tailored to each respective target audience (widely used algorithms and benchmarks for teaching; commonly-known real-world problems such as weather prediction for broader audiences)

It can be argued and our personal experience over the past 10 years in open day events and education fairs has shown, that audience-specific, interactive showcases generally have a much higher rate of success at attracting attention compared to generic or pre-prepared material that is shown and discussed on screens or in hardcopy.

4. Hardware architecture

This section describes the hardware selection process and provides details on hardware characteristics that pertain to our use cases. For more detailed information on the individual components, any properties such as exact measurements or design drawings can be found in the technical documentation.[2] This documentation also offers a bill of materials at the end, which totals at less than 1800 EUR for all but minor components such as screws or cable ties.

4.1. Computational characteristics

In order to fulfill our design goal of a mobile, representative HPC cluster for demonstration purposes, we require a system of at least several compute nodes with a network interconnect, yet light enough to be easily portable. The size and weight of x86 hardware exceeds this limitation, aggravated by its need for stable mounting platforms, power requirements and cooling infrastructure. ARM-based architectures on the other hand are naturally better suited for this purpose and have demonstrated their viability in large-scale HPC even among the Top500, as shown by Fugaku, the no. 1 as of June 2020, powered by ARM-based Fujitsu A64FX processors [26]. The small form factor and small thermal footprint of modern ARM compute boards allows us to pack 16 such devices into an aluminum suitcase of approximately 32 l. Since we do not want to restrict our workloads to pure distributed-memory applications, we consider only multi-core boards with reasonable computational performance that are suitable for both shared-memory and distributed-memory parallel programming, yet inexpensive enough to be easily replaceable.

To that end, we choose NanoPi M4 boards with a Rockchip RK3399 System-on-Chip. It features ARM's big.LITTLE architecture with two Cortex-A72 cores clocked at up to 1.8 GHz and four Cortex-A53 cores clocked at 1.4 GHz and supports ARM's NEON instruction set for SIMD operation. The NanoPis are low-cost off-the-shelf products that can be easily replaced if necessary and form our compute nodes. Table 1 lists further characteristics of the hardware architecture. Even though they are also equipped with Mali T864 GPUs that support e.g. OpenCL, we do not employ the Cluster Coffer for any GPU-based computing at this time. Details with regard to the obtainable performance on our system are provided in Section 5.6.

The compute nodes are equipped with 2 GB RAM while the head node's NanoPC T4 provides 4 GB for compilation and management reasons. Local storage is available through per-node microSD cards (we use 16 GB), although — as Section 3.2 will describe in more detail — it is only used for enabling network boot and not involved in any storage operations after booting the Linux kernel. Local storage on the head node is provided through both its own microSD card and 16 GB of on-board eMMC 5.1 flash memory.

4.2. Frame, power, and cooling

Fig. 1 illustrates the entire compute node assembly without the encompassing suitcase or the head node, whereas Fig. 2 shows how individual nodes, coolers, and fans are attached. For clarity, these design illustrations do not show any wiring. The V-mount panels, shown in orange, each hold 4 compute nodes. Fig. 3 shows top and bottom photography of a V-mount, including the 40 mm fans to aid in cooling and the four toggle switches for controlling the power supply for each compute node. The switches allow us to do research in resilience by simulating failing nodes. Although the nodes can also be powered via USB-C, its cabling and connectors are comparatively expensive and inflexible regarding cable lengths. Instead, we opt for supplying power to the compute nodes via their GPIO header which is more easily accessible, versatile, and allows us to interface our own compute node power board (*ccpad*), one per node. These ccpads do not only act as a single-connector 5 V power supply but also provide in-band power instrumentation via an INA219 zero-drift bidirectional shunt-resistor-based current monitor accessible through I2C on each respective compute node. This enables us to expand the usage scenarios of our Cluster Coffer to power/energy tuning research and multi-objective optimization work. Similar shunt-based power instrumentation has been successfully demonstrated in related work in both ad-hoc [2] and

[2] https://github.com/uibk-dps-teaching/cluster-coffer/blob/master/docs/cluster-coffer.pdf.

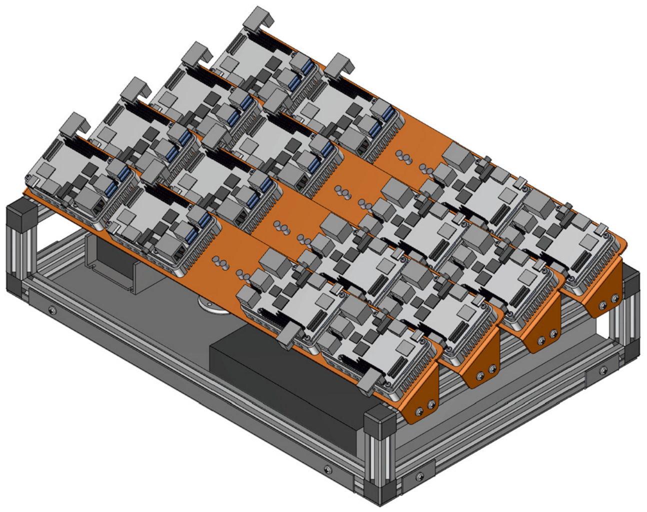

Fig. 1. Illustration of the entire compute node assembly, with the network switch and power supplies partially visible underneath.

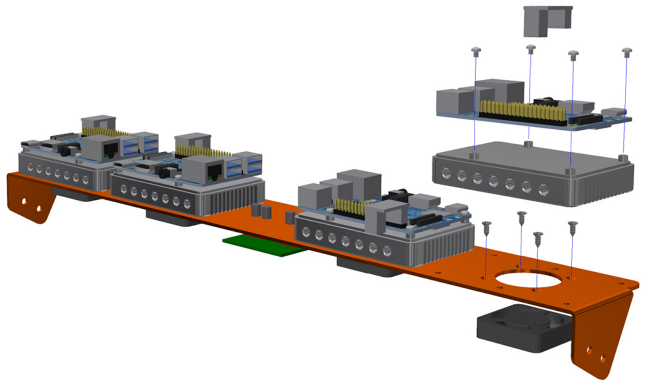

Fig. 2. Exploded view on compute node and cooling assembly. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

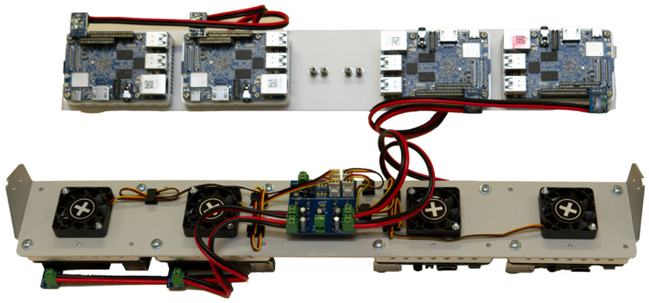

Fig. 3. Photo showing a fully assembled V-mount with compute boards, cooling assembly and cabling.

commercially available solutions [12]. In the Green500 methodology [21], this makes the Cluster Coffer a power measurement level 2 system, since we do have measurements for all compute nodes but need to rely on estimates on the remaining hardware such as the network switch or the fans to compute the overall power consumption.

Power is supplied to the Cluster Coffer through an external IEC C14 connector at the back of the aluminum suitcase, equipped with a switch and a fuse for safety reasons. Through this connector, 220 V AC power is supplied to two AC switching power supplies, with an output of 5 V 300 W and 12 V 48 W respectively, both housed underneath the compute nodes.

The 5 V rail supplies power to all compute nodes. Since adding a second power supply for redundancy would increase the weight, we opt for sacrificing redundancy rather than portability for an experimentally-focused cluster. Due to the lack of any connected USB devices or other external components, the overall power draw of the compute nodes does not exceed 200 W and therefore the power supply is amply dimensioned and not directly actively cooled. However, due to the fact that the compute node fans are located directly above, there is some limited air flow that helps to cool the entire assembly. Our preliminary experiments have not exhibited any overheating problems even under full load on all 16 compute nodes, with a top temperature of 45 degrees Celsius measured on the memory modules, which do not have a dedicated cooler. Note that we do not consider Cluster Coffer operation with the suitcase closed.

The second, 12 V rail offered by the 48 W power supply (an LED driver in our case) powers all 16 compute node fans as well as the head node and its cooling fan. Since the head node is powered through its DC socket rather than the GPIO pins, power measurements are not available. However, since most cluster configurations prohibit running production workloads on head/login nodes by default, we do not consider this an issue. As the compute nodes do not power their fans themselves, the current draw of the fans is not covered by the compute node power instrumentation.

Finally, we also added a WS2812 LED strip to the Cluster Coffer, which runs on the inside of the bottom half of the suitcase and offers individually addressable RGB LEDs, controlled by an Arduino Nano ATmega328P micro-controller. The micro-controller is connected to the head node via an FTDI RS232 UART-USB interface. This serves two key purposes. First, it provides an eye catcher for younger audiences (and sometimes also older ones) in order to attract them to the Cluster Coffer and raise their interest in our research topic and in computer science in general. Its effectiveness in that regard has been proven at numerous public outreach events such as science or education fairs. Second, it does not merely display any color but actually shows a live visualization of the computational load of the Cluster Coffer, as Section 5.4 further describes.

Fig. 4 shows the fully assembled Cluster Coffer. The compute node assembly sits in the bottom part of the suitcase, the head node is mounted on the inside of the top cover. The blue Ethernet cable seen on the left is the external network connection for interacting with the system. The entire suitcase weighs 13.2 kg in its operational state.

4.3. Network

The area below the compute nodes houses the network switch (we chose a TP-Link TL-SG1024D 24-port Gigabit Ethernet switch), connecting all 16 compute nodes and the head node in a star topology using 25 cm and 50 cm Cat 7 cables. Its additional 7 unassigned ports leave ample room for extension for future components such as additional instrumentation devices and an external Ethernet cable that allows to interface the Cluster Coffer with the outside world. The maximum power draw of the switch is 15 W and it has a switching capacity of 48 GBit/s, which is more than enough for our use case.

5. Software architecture

This section details on the software environment of the Cluster Coffer, from the bootloader to HPC-specific packages. The software components described in this section are available publicly on GitHub,[3] to be used and built upon by other researchers and

[3] https://github.com/uibk-dps-teaching/cluster-coffer.

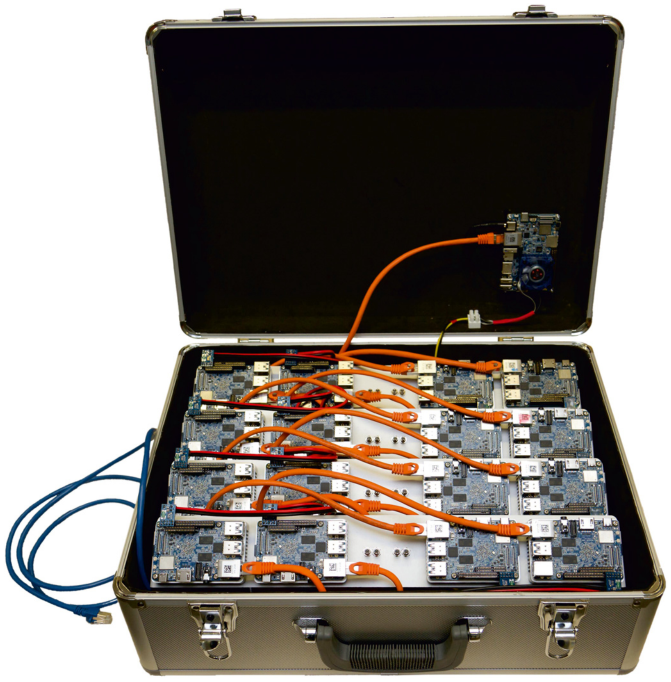

Fig. 4. The fully assembled Cluster Coffer. The head node is mounted on the top cover for direct access to its USB and HDMI ports. The blue Ethernet cable connects to the outside world.

instructors, and can be easily ported to similar architectures with minimum effort. The only exceptions are the bootloader in Section 5.1, the LED component in Section 5.4, and the power measurements provided via I2C. While they all are also published on GitHub, they are tailored to the specific hardware platform we use and likely need adjustment whenever the compute board models, LED controller, or power instrumentation implementation change.

5.1. Operating system and base software stack

In order to meet the software design requirements outlined in Section 3.2, we choose a standard Debian-based Linux operating system for both the head node and the compute nodes. It is stable, well-maintained and wide-spread, supports our hardware architecture, and provides access to a vast amount of software packages for cluster maintenance and parallel application development.

The compute nodes have local microSD card storage,[4] which — at the 16 GB card sizes that we use — could easily accommodate a Linux installation along with any packages and programs required for basic parallel systems operation. However, this would entail unnecessary work duplication when updating all compute nodes' software stacks, modifying their configuration, or simply adding new software packages. Instead, we choose to store only a small U-Boot [7] boot loader that network-boots the compute nodes and mounts `/dev/mmcblk2p2` on the head node as the root directory on the compute nodes (also referred to as `rootfs`). The compute nodes in turn use a temporary overlay file system to prevent any interference from simultaneous write operations to identical file paths on the NFS-mounted `rootfs`. Beyond the removal of duplicated work, this single point of information has the advantage of keeping the software stacks of all compute nodes automatically synchronized and reduces wear on the microSD cards since they do not need to be written to when the software stack changes or even when writing logs. Any persistent changes to the compute nodes' software stack can be performed on `rootfs` on the head node using e.g. `chroot`. This naturally does not offer persistent write capabilities to the compute nodes, as writes to the overlay file system are discarded upon shutdown. However, persistent writes to their root directory is not required during normal operation and application workloads are run from a different mount point. Any persistent node-specific settings, such as IP addresses or host names, can be achieved through DHCP and the nodes' unique MAC addresses, or by a single, unique identifier per compute node that can be included with the boot loader when initially flashing the microSD cards. In order to preserve the monotonicity of time, all compute nodes synchronize their clocks with an NTP server running on the head node (for which changes are indeed persistent, and which synchronizes its NTP server to the outside world using the external network connection upon booting).

5.2. Cluster-specific software environment

In addition to a commodity Linux OS, we require a specific software environment for proper Cluster Coffer operation and C/C++ development. This includes the installation of development packages on the head node in order to compile and debug programs (`gcc`, `cmake`, `valgrind`, `gdb`), an NFS server for serving the compute nodes with their root filesystem, or an MPI implementation on the compute nodes along with various additional packages such as ntp clients for clock synchronization.

Furthermore, any cluster system naturally requires persistent storage for providing all nodes with access to e.g. MPI application executables or input data. This is offered through a dedicated network-mounted directory `/share` and resembles common cluster user directories such as `$HOME` or `$SCRATCH`.

5.3. Interface to host

HPC systems often provide instrumentation that enables users to monitor the state of the system, such as the load of the cluster or its current power consumption. Frequently, this data is provided through additional interfaces to the outside world, besides SSH. Since the Cluster Coffer itself has no screen due to space, weight, and power constraints, we implemented a small framework that allows to exchange information between the Cluster Coffer and an optional *host* computer, e.g. a laptop. Developed during the H2020 AllScale project [11], this so-called *Dashboard* is a browser-based online performance visualization tool. It offers two-way communication by receiving and illustrating non-functional data from the Cluster Coffer such as computational, memory, or network load, as well as sending control information to the runtime system for performance steering. For compatible programs (those relying on the AllScale software framework [11]), it can affect scheduler decisions live, during the execution of a parallel application. Fig. 5 shows a screenshot of one of the Dashboard's views, with aggregated information displayed in gauges at the top, and more detailed per-node information below. On the left, a list of color-coded nodes is shown and nodes that are offline are printed in gray (nodes are considered offline if no data has been reported for them for a configurable amount of time, which we set to 2 seconds). Next to it, there are speed, efficiency, and power graphs that show data for every node. The two-dimensional plot on the bottom right visualizes the current data distribution within the AllScale runtime system [10] and is a means of observing the effectiveness and efficiency of active load balancing. Every rectangle corresponds to a certain data region of the same, two-dimensional domain of an application and its color matches the node colors on the left to illustrate in which

[4] we use Kingston SDC4/16GBSP class 4 microSDHC cards.

Table 2
Application-inspecific compute node metrics gathered for Dashboard visualization.

Metric	Source
Power	`/sys/bus/i2c/drivers/ina2xx/3-0040/hwmon/hwmon0/power1_input`
CPU speed	`/sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq`
CPU load	`/proc/stat`
RAM load	`/proc/meminfo`
Network load	`/proc/net/dev`

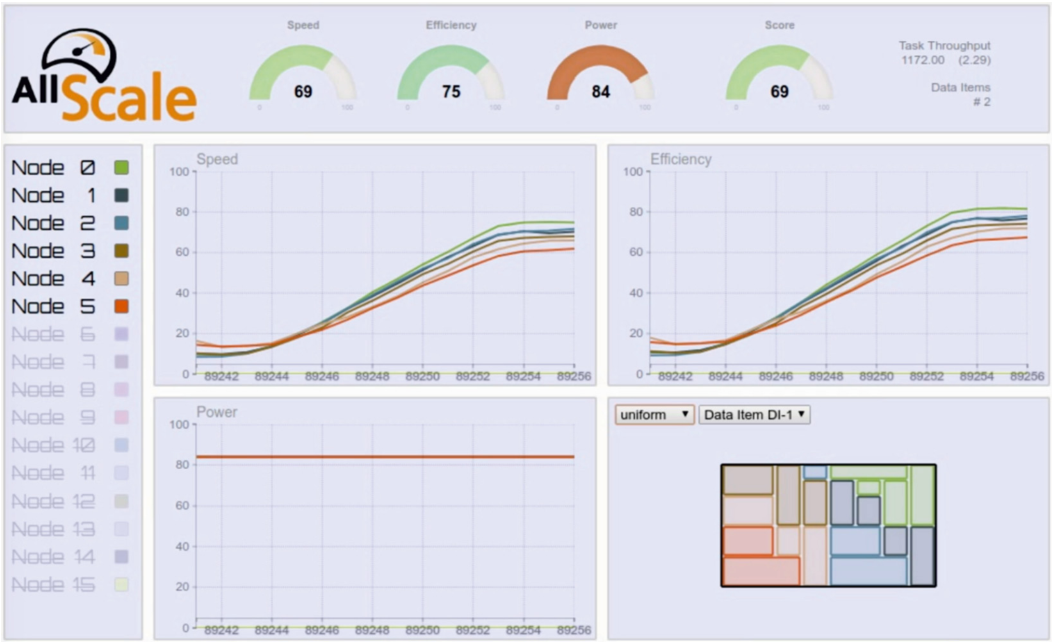

Fig. 5. One of several Dashboard pages. Aggregated information is shown on the top, individual nodes are listed on the left. The graphs show computing speed and efficiency for AllScale-type applications, along with the current data distribution and currently active scheduling policy.

node's main memory the data region currently resides. The active scheduler policy can be selected by choosing from a drop-down menu (set to the *uniform* policy in the screenshot). All information between the Dashboard and the Cluster Coffer is exchanged in JSON for compatibility and ease of debugging. The Dashboard web page is served by a lightweight standalone *Dashboard Server*[5] written in Go that can run either on the host computer or on the Cluster Coffer and performs the actual message exchange with the runtime system via TCP. For debugging purposes, the server can also generate random status data for the Dashboard.

Since not all applications executed on the Cluster Coffer are written using the AllScale software framework, we also a implemented standalone daemon that periodically provides non-functional data irrespective of any specific application being executed. For this use case, each compute node runs such an instance of the daemon for collecting its own data and forwards it to an aggregation daemon on the head node, which in turn merges the data and forwards it to the Dashboard server. When using these daemons, performance steering is naturally not available and communication is one-way only. Table 2 lists all collected metrics and data sources per compute node.

The open-source nature of the project and the use of modularized components and standards such as JSON facilitate modification and extension of this monitoring tool.

[5] https://github.com/allscale/allscale_dashboard/blob/master/server.go.

5.4. Status LEDs

As mentioned in Section 4.2, we also added a WS2812 LED strip to the Cluster Coffer. Beyond its effect of attracting audiences at public events, it can show a live visualization of the computational load of the Cluster Coffer.

The daemons that collect non-functional statistics forward this data, which also contains CPU load information, to the head node. The head node in turn sends this data to an Arduino ATmega328P micro-controller that controls the LED strip. Since the LEDs are individually addressable, we have the color of every LED correspond to the load of the compute nodes adjacent to it. Fig. 6 shows an illustration of this visualization with four selected load cases. The arrows of varying length illustrate the difference in speed of the brightness wave that progresses throughout the strip, i.e. higher computational load leads to higher speeds.

There is also a second mode that visualizes the state of the cluster by showing static brightness without any wave and switching off portions of the strip when corresponding nodes are offline.

This simple visual debugging tool has shown to be very effective in several aspects, e.g. when verifying correct MPI rank placement, being alerted to failing nodes, or even illustrating load imbalance, without having to consult the Dashboard. Since the CPU load data collected by the daemons is deliberately restricted to user load, the status LEDs also show inefficient program execution due to excessive use of or slow OS system calls.

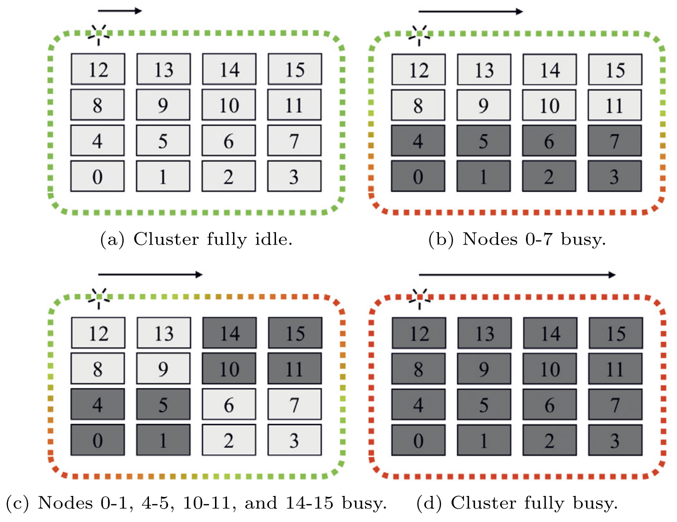

Fig. 6. LED load status visualization examples. The length of the arrow denotes the speed at which the wave is progressing.

Table 3
Major HPL parameters used for benchmarking.

Parameter	Setting
N	15000 (strong), 3500 (weak, single node)
NB	64
PMAP	0 (row-major)
PxQ	1x1, 1x2, 2x2, 2x4, 4x4

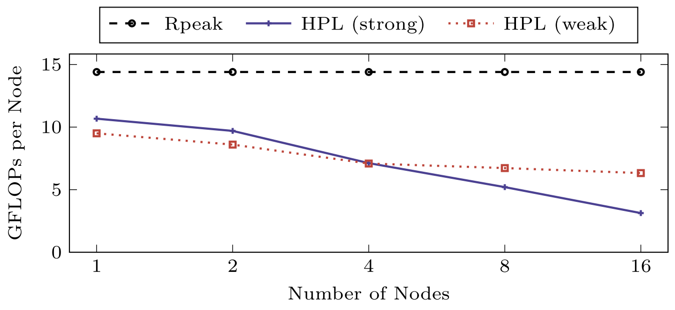

Fig. 7. HPL performance achieved in GFLOPS per node.

5.5. Setup process and booting

For the initial setup of the software images used by the Cluster Coffer, a small scripting framework is provided.[6] These scripts can be run from any Linux distribution (we use Debian) and build the three images that are required for setting up the cluster: the Linux image used by the head node, including all software packages for development and Cluster administration discussed in Section 5.2; the `rootfs` image stored on the head node to be used by the compute nodes; and the boot loader of the compute nodes for network-booting from `rootfs`. Subsequently, both the head node Linux image and the compute node boot loader are written to microSD cards, whereas the `rootfs` image needs to be copied to the head node's eMMC storage. The entire process takes less than an hour on modern desktop hardware and allows re-flashing any images in case of SD card breakdowns or head node software stack changes. Furthermore, all node configuration such as IP addresses, hostnames, NTP setup and SSH host/user keys are also set up by these scripts such that the Cluster Coffer can be booted without any additional work required. Further details on the inner workings of the scripting framework are given by a readme file in the repository and by reading the scripts themselves.

Since the compute nodes require their `rootfs` to be present on the head node, the head node needs to be switched on first when booting the cluster. After a grace period of approximately 20 seconds, the head node's services are up and running and allow the compute nodes to be switched on. Switching on all 16 compute nodes nearly simultaneously often induces high load on the head node and has, on occasion, entailed filesystem and network timeouts. For this reason, we recommend a staggered power-on procedure, leaving approximately 1-2 seconds between compute node power cycles.

5.6. Benchmarking

Although the Cluster Coffer is mainly aimed at literal portability, we still consider it an HPC system. Its Cortex-A72 cores are equipped with NEON, one of ARMs vector extensions, offering 128-bit wide registers. They can be used for multiply-accumulate operations on up to two double precision floating-point numbers, which leads to 4 FLOPS per clock cycle per core. At the nominal clock rate of 1.8 GHz, each CPU core provides a theoretical peak performance of 7.2 GFLOPS, while the entire Cluster Coffer offers an Rpeak of 230.4 GFLOPS. Naturally, this is slow in today's HPC world, given that even the last rank in the Top500 list of June 2020, Graham, offers 2.6 PFLOPS [24]. Nevertheless, the Cluster Coffer illustrates the vast performance improvements achieved through the decades, as it outperforms - on paper - the first rank in the June 1994 list, XP/S140 at Sandia Labs due to its lower Rpeak of 184 GFLOPS [23]. Moreover, while we did not find documentation on the power consumption of the XP/S140, we assume that it was at least in the order of several kilowatts, whereas our Cluster Coffer has an estimated overall maximum theoretical power consumption of approximately 300 W. Measurements using a Voltech PM1000+ industrial grade power meter show power consumption at the wall socket to stay well below 200 W for all experiments discussed in this paper.

Still, the Top500 are ranked according to Rmax, not Rpeak, which is why we also benchmarked our system with HPL [18]. Table 3 lists the major settings chosen for our strong and weak scaling experiments. The maximum problem size of $N = 15000$ for strong scaling was chosen such that the data still fit in the memory of a single node, leaving only 12% of RAM available. For weak scaling, due to the comparatively limited amount of memory available and Linpack requiring $\frac{2}{3}N^3 + 2N^2$ operations for an $N \times N$ matrix, we scaled N linearly with the number of nodes (which increases the number of operations super-linearly) in order to try to find the highest Rmax value possible while still being able to run experiments for all numbers of nodes. Here, the maximum problem size for 16 nodes was $N = 56000$ or approx. 85% of the available RAM.

The block size $NB = 64$ was derived with an empirical study that shows smaller block sizes to worsen performance, and no measurable benefit for higher block sizes. The process grid of P and Q was chosen with our network topology in mind. For switched networks, HPL favors $P : Q$ ratios of $1 : k$ with k in [1..3], which lead to the grid selection described in the table. Beyond these benchmark-specific settings, we used GCC 8.3 for compiling with `-Ofast -mtune=cortex-a72` flags and linked against the BLAS implementation of ARMs Performance Libraries version 20.3, built with the `generic` microarchitecture setting, hence targeting ARMv8 CPUs with NEON capabilities. OpenMPI 4.0.5 provides us with the necessary MPI implementation.

[6] https://github.com/uibk-dps-teaching/cluster-coffer/tree/master/software.

Table 4
HPL raw performance data. Rmax and Rpeak are shown for every number of nodes tested, Time denotes the wall time.

#Nodes	Rmax (strong)	Rmax (weak)	Rpeak	Time (strong)	Time (weak)
1	10.67	9.50	14.40	210.99	3.01
2	19.39	17.20	28.80	116.04	13.31
4	28.48	28.35	57.60	79.02	92.85
8	41.63	53.80	115.20	54.06	272.09
16	50.07	101.21	230.40	44.94	1156.79

Table 5
HPCG weak scaling raw performance data including overall memory throughput for every number of nodes tested, Time denotes the wall time, Bandwidth the raw total memory bandwidth in GB/s.

#Nodes	GFLOPs	Time	Bandwidth
1	0.55	29.10	4.15
2	0.96	33.36	7.27
4	1.80	35.64	13.67
8	3.34	38.71	25.30
16	5.94	43.65	45.02

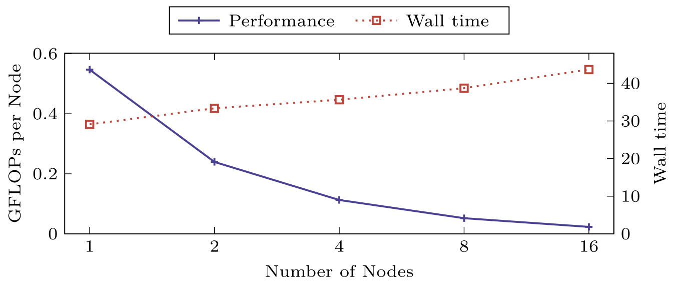

Fig. 8. HPCG performance achieved in GFLOPS per node and wall time in seconds.

Fig. 7 shows the performance data of these benchmark runs, with Table 4 providing the raw data. Since Rmax denotes achieved maximum values and not mean values, we did not conduct multiple experiment runs for each data point. Nevertheless, empirical evaluation of single data points indicates the variation to be less than 5%. The power consumption for the highest Rmax of 101.21 GFLOPs for all compute nodes was approximately 113 W, and the estimated power consumption of the entire cluster is approximately 200 W.

As the data shows, our Cluster Coffer would have ranked first in the Top500 in June 1993, outperforming CM-5/1024 [22] of the Los Alamos National Laboratory with its Rpeak of 59.7. However, besides the 27 years of progress in hardware research and development, it should be noted that software stacks were also improved over the years and HPL itself was updated several times since then.

However, since HPL resembles a subset of comparatively computationally-bound applications, the Top500 has also included HPCG [5] benchmark data for several years now. Compared to HPL, the overall performance of HPCG depends more on memory and node interconnect performance, hence better resembling many non-computationally-bound workloads. For this reason, we also benchmark our system with an ARM-optimized version of HPCG [20].

Similarly to HPL, we set the per-node problem size to $N = 96$ for each of the three dimensions in order to arrive at a RAM usage of 70%, thus fulfilling the benchmark's requirement of at least 25%. Also, due to this requirement, we only conduct weak scaling experiments. The build settings are identical to HPL (compiler version and flags, use of ARM performance libraries and NEON, OpenMPI version) except for using a single rank per node with two OpenMP threads to task both Cortex-A72 cores with work.

Table 5 lists the performance data of these HPCG runs. Note that while the runtimes are too short to meet the criteria for official results (at least 1800 seconds), they are sufficient for our purpose. The data shows a parallel efficiency of 68% for 16 nodes, which is expected given our commodity Gigabit node interconnect and nodes that are not optimized for fast memory hierarchy interaction. Fig. 8 illustrates the performance in GFLOPS per node on the left y-axis and wall time in seconds on the right y-axis.

6. Use cases

6.1. Student teaching

Teaching parallel programming and HPC is a challenging task. There are many intricacies on multiple levels in modern parallel hardware, including

- CPU architecture level: e.g. ISA, ILP, vectorization, hardware multithreading, on-chip interconnect
- memory hierarchy level: e.g. private/shared caches, replacement policies, NUMA topologies, hardware prefetching
- network level: e.g. network topologies, RDMA
- accelerators, if any: most of the CPU and memory aspects in a second, different

In addition, there are several software-focused aspects one must be aware of, such as choice of algorithm, task- and data parallelism and their decomposition, load balancing, temporal and spatial data locality, false sharing effects, or thread affinity. The increase in parallelism width (e.g. more NUMA-domains per node, NUMA domains within CPUs, growing vector register widths), the heterogeneity in both CPUs and accelerators, and the rise of new programming models and domain-specific languages and libraries makes mastering parallel programming on these systems a challenge.

We use the Cluster Coffer in teaching in order to better visualize the characteristics of HPC systems and to be able to provide a complementary system to x86 hardware commonly available to every student. Since we have direct control over our system, changes in the software stack or even hardware reconfigurations are made feasible - in contrast to production systems, for which computer science experiments requiring direct hardware access are often not possible for practical reasons.

One of these aspects is promoted by the power instrumentation system of the Cluster Coffer. It is highly suitable for teaching parallel program and hardware optimizations, leading to our first use case: illustrating the concept of multi-objective optimization at the example of frequency and voltage scaling (DVFS).

Fig. 9 shows data of the HPL benchmark run on two Cortex-A72 cores using a single process and the multi-threaded BLAS implementation of ARM's Performance Libraries with problem size of $N = 8000$. The benchmark was run repeatedly for different clock

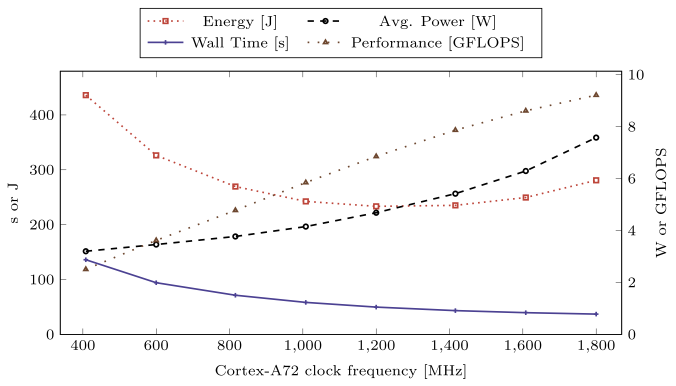

Fig. 9. Performance and power/energy characteristics of HPL with $N = 8000$ on two Cortex-A72 for various clock frequencies.

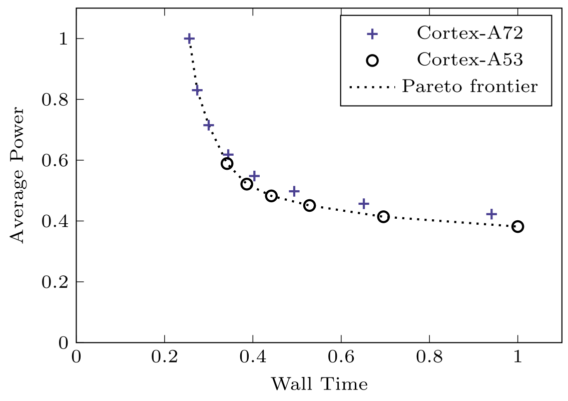

Fig. 10. Pareto set of optimal solutions on the trade-off between average power consumption and wall time for HPL with $N = 8000$ on two Cortex-A72 for various clock frequencies.

frequency settings as indicated on the y axis of the figure, while simultaneously measuring the power consumption of the compute node in question. As the figure shows, there is an expected increase in performance and decrease in wall time for increasing clock frequencies. However, what most students do not expect when initially exposed to the concept of DVFS, is the sweet spot of lowest energy consumption, which is neither the highest or the lowest frequency setting. When discussing these topics with students, we often find that students intuitively expect the most energy-optimal setting to be the lowest clock frequency. Subsequent examination reveals that they do not consider static power consumption overheads from the remaining Cortex-A53 cores or off-core entities such as caches or memory controllers that skew this data. Lowering the clock frequency below 1400 MHz — for the experiment configuration presented here — yields an execution time which is disproportionately long compared to the power consumption savings coming from the frequency and voltage reduction, due to these static overheads. In addition, this is an excellent teaching case for the roofline model [29], which deals compute-bound or memory-bound properties of workloads, or Amdahl's law with regard to the relative amount of a program that is parallel and its parallel efficiency, since these characteristics also influence the position of the sweet spot.

In addition, we use this data to teach and illustrate the non-linear relationship that power consumption exhibits with clock frequency. This is caused by the definition of dynamic power consumption, which is $P = C * F * V^2 * \alpha$ where C is the electrical capacitance (a fixed property of the hardware), F denotes the frequency, V denotes the voltage, and α is the so-called switching factor, a property of the workload (in essence the percentage of transistors that change state at every clock cycle). We employ this data, illustrations and the equation above as the initial motivation in our parallel programming courses, and to explain the need and rise for increased parallelism and multi-/many-core CPUs. Feedback from students in our courses has shown that live visualization of e.g. benchmarks running on our Cluster Coffer greatly increases their interest in the topic, compared to only presenting the background in a theoretical fashion.

In contrast to the raw data of Fig. 9, Fig. 10 presents the (normalized) trade-off between wall time and power consumption, where every point corresponds to a clock frequency setting. In this figure, we also include measurements done on the four slower but more energy-efficient Cortex-A53 cores, which support the same clock frequencies except for 1.6 GHz and 1.8 GHz. The dotted line shows the so-called the Pareto-frontier, which consists of the set of points that are considered Pareto-optimal [3], meaning there is no point that outperforms a point on the Pareto frontier in both objectives. Given common static power consumption overheads in hardware and application workloads that are not fully computationally bound, it is comparatively easy to obtain this trade-off between power and time. This makes it an excellent teaching case for students to explore this trade-off themselves with their own applications implemented during course homework or to motivate the existence of energy-aware scheduling on large-scale systems such as the SuperMUC supercomputer [1].

To further illustrate the effect of DVFS in distributed-memory HPC environments, we also run the HPCG benchmark of Section 5.6 with all frequency settings on the Cortex-A72 cores on varying numbers of nodes. Table 6 shows the results of these experiments as heatmaps for both overall performance as well as per-node power. Several effects can be observed here. First, both the performance and power consumption are naturally decreasing for decreasing frequencies. Similarly to HPL, a sweet spot can be found, e.g. for 16 nodes at a frequency setting of 1416 MHz, we reduce power consumption by 21.4% but performance only by 9.2% compared to the nominal setting of 1800 MHz. However, also communication overhead is visible in the power data, as the per-node power consumption decreases when increasing the number of nodes, caused by stalled cores that are waiting for message passing operations to complete, even though our implementation of HPCG uses non-blocking communication. This effect is strongest for the setting of 1608 MHz, where per-node power is reduced by 7.1% for 16 nodes compared to a single node. This stall time has also been referred to as *slack time* in related work and has been used for energy optimization by reducing the clock frequency of cores that are busy-waiting in MPI wait states [8]. On our Cluster Coffer, this effect is essentially eliminated at the lowest setting of 408 MHz, with any differences well within the margin for measurement errors. Here, the latency of non-blocking communication is fully hidden by the slow computation and data processing, hence reducing core stall to a minimum.

These experiments show the capability of the Cluster Coffer to produce data suitable for teaching the aforementioned concepts of scalability, (non-)compute-boundness and the effect of DVFS on HPC workloads. Nevertheless, these experiments rely on the availability of fine-grained power measurements and exclusive node access, preventing use of Cloud resources and many cluster systems.

In order to ensure a productive teaching environment, we recommend teams of 2-3 students each that either work in parallel on individual nodes for shared-memory experiments or take turns using e.g. a job submission system to work in distributed memory without mutual measurement perturbation. Given that practical courses involving programming exercises are often limited to a maximum of 25-35 students per group, platforms such as the Cluster Coffer can also accommodate larger numbers of students by working with one group at a time.

Table 6
Performance and per-node power consumption data for HPCG for several core frequencies. The first row specifies the number of nodes, the first column specifies the frequency setting in MHz.

	1	2	4	8	16
1,800	0.55	0.96	1.8	3.34	5.94
1,608	0.52	0.89	1.71	3.15	5.64
1,416	0.48	0.84	1.6	2.99	5.39
1,200	0.43	0.76	1.45	2.73	4.93
1,008	0.37	0.66	1.29	2.44	4.5
816	0.32	0.57	1.1	2.1	3.91
600	0.24	0.44	0.86	1.64	3.12
408	0.17	0.31	0.6	1.13	2.25

(a) Compute performance (GFLOPS).

	1	2	4	8	16
1,800	5.63	5.53	5.48	5.38	5.31
1,608	4.96	4.86	4.78	4.76	4.6
1,416	4.46	4.41	4.36	4.31	4.18
1,200	4.04	3.98	3.96	3.87	3.8
1,008	3.69	3.63	3.59	3.58	3.52
816	3.41	3.37	3.36	3.33	3.3
600	3.13	3.12	3.09	3.09	3.07
408	2.88	2.88	2.86	2.84	2.86

(b) Average power consumption (W).

Algorithm 1 High-level view of image capture and data visualization on the host PC.

```
 1: while true do
 2:     $image \leftarrow$ POLLCAMERA()
 3:     VISUALIZE($image$)
 4:     $motion \leftarrow$ EXTRACTMOTION($image$)
 5:     if $motion \neq \emptyset$ then
 6:         VISUALIZE($motion$)
 7:         SENDTOCLUSTER($motion$)
 8:     $particles \leftarrow$ RECEIVEFROMCLUSTER()
 9:     VISUALIZE($particles$)
10: end
```

Algorithm 2 High-level view of simulation running on cluster.

```
 1: $particles \leftarrow \emptyset$
 2: while true do
 3:     $motion \leftarrow$ RECEIVEFROMPC()
 4:     if $motion \neq \emptyset$ then
 5:         $newParticles \leftarrow$ GENERATEPARTICLES($motion$)
 6:         $particles \leftarrow particles \cup newParticles$
 7:     SIMULATETIMESTEP()
 8:     if $particles \neq \emptyset$ then
 9:         SENDTOPC($particles$)
10: end
```

6.2. Public outreach

The second main use case of our Cluster Coffer is to engage with the general public during science fairs, education fairs, or even just open day events at our institution. For this purpose, we do not want to rely on comparatively sophisticated research such as multi-objective optimization for performance and energy, but rather aim to demonstrate the basic principles of parallel programming, work- and data decomposition, and how HPC impacts people's everyday lives.

To that end, we selected one of the AllScale project pilot applications, a port of iPiC3D [15], which is a Particle-in-Cell code for space weather applications. It is used for simulating the interaction of solar wind (and more specifically solar storms) with the Earth's magnetosphere. Solar storms can cause damage in today's electric and electronic systems, such as the nine-hour power outage in Québec in March 1989 [28]. For this reason, we consider solar storms a good choice to motivate and justify the need for HPC and its expenses to the general public, as the effect of solar storms can neither be investigated analytically nor experimentally on demand.

Nevertheless, there are two caveats: first, solar wind is still an abstract topic that many among the general public do not know about; second, the simulation usually works with static input data, which might lead to interesting visualizations, but does not engage the audience in lively interaction. As a consequence, we modified iPiC3D to accept live input data coming from a camera, enabling the audience to directly influence the simulation state and hence the computational load on the cluster, and watch the functional and non-functional visualization effects.

Algorithms 1 and 2 outline the setup of this use case, with a visualization provided in Fig. 11. People's movements are captured using a camera connected to a host PC. Since there are usually many people at fairs, moving in the background and possibly causing perturbation in our input data, we use a Microsoft Kinect. It provides a depth sensor that allows us to consider only information in a finite difference of a few meters, removing any

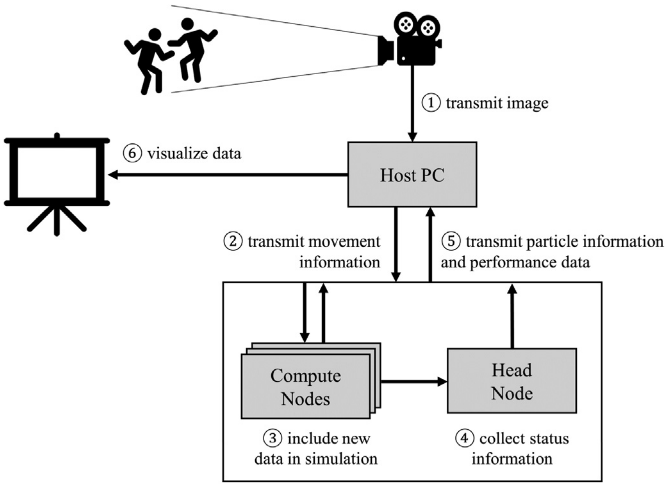
① transmit image
⑥ visualize data
Host PC
② transmit movement information
⑤ transmit particle information and performance data
Compute Nodes
Head Node
③ include new data in simulation
④ collect status information

Fig. 11. Overview of the entire public outreach setup.

information beyond that and making the result visualization much more clear. Fig. 12 shows a visualization of these images that are captured on the host. The image data is analyzed on the host PC and motion information is extracted using a small OpenCV-based program. If any motion was detected, the motion information is forwarded via TCP to the Cluster Coffer, which generates particles from this data using the recorded position and direction of movement. These new particles are then included with the ones already present in the simulation from previous time steps. Also, the speed of the movement is used to initialize the particle's energy. The Cluster Coffer then runs one simulation step, gathering the new particle positions, if any, and sending them to the host PC. The host PC in turn displays both the captured images from the Kinect camera (Fig. 12) as well as a 3D visualization of the particles received from the Cluster Coffer (Fig. 13). Fig. 14 shows the corresponding Dashboard visualization when the Cluster Coffer is fully loaded. All three visualizations of Figs. 12 to 14 are shown

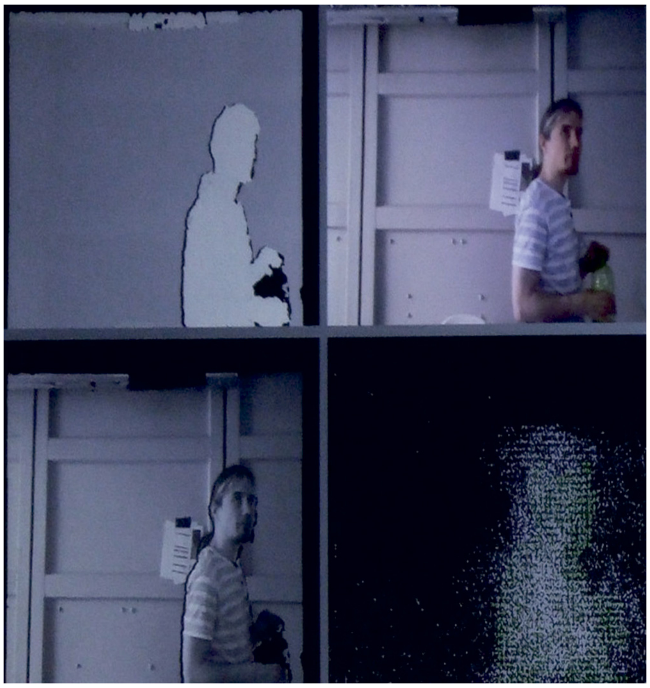

Fig. 12. Visualization of the images captured by the Kinect camera with a person passing by. The top left shows the image captured by the depth sensor, the top right the image captured by the normal sensor. The bottom left is a masked version of this image (in this case the entire scene is in range and hence visible) and the bottom right shows the captured motion information.

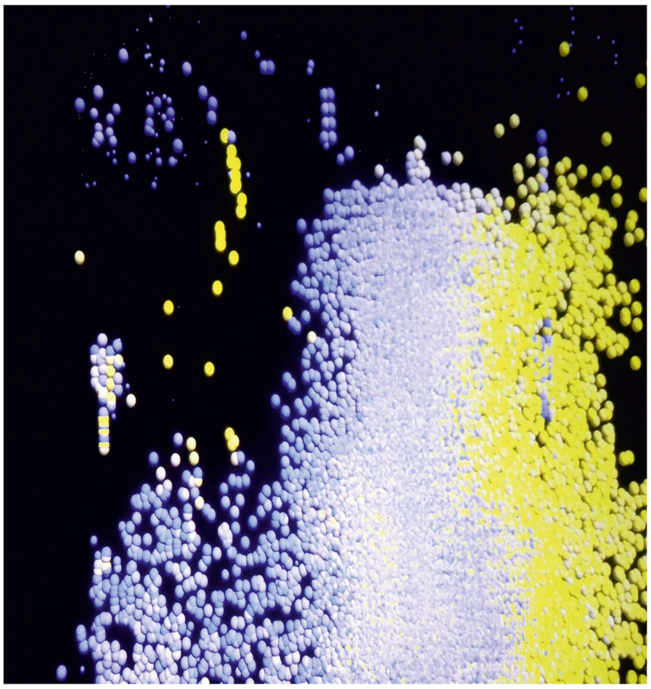

Fig. 13. Visualization of the particles coming from the iPiC3D simulation running on the Cluster Coffer. The state of the simulation corresponds to the input as illustrated in Fig. 12. Particles with higher energy are shown in yellow, those with lower energy are shown in blue.

live to the audience in order to explain the flow of information and to maximize interactivity. When possible, an additional screen shows artist's visualizations of solar winds in photos or videos for illustration of the physics involved.

The number of simulation updates per second is mainly limited by the interconnect between host PC and the Cluster Coffer, and depends on the number of particles, with approximately $5 * 10^4$ particles per second saturating the head node's NIC performance, network bandwidth and latency — more than sufficient for our needs. In order to optimize the bandwidth usage, we use single-precision data for the data exchange from the host PC to the Cluster Coffer. Also, we remove any data irrelevant for visualization when transferring particle information back to the PC. Furthermore, for visualization clarity, particles are equipped with a time-to-live field that is reduced every simulation step, and after a finite number of steps they are removed from the simulation. While this naturally does not correctly represent the physical processes involved, it serves its main purpose of clear illustration and interaction.

We have successfully demonstrated the Cluster Coffer at multiple public outreach events since 2018, including institution-wide open day events, university-wide and public education and science fairs, pre-scientific work courses with pupils, or general networking events, all at varying locations — which emphasizes the usefulness of a mobile solution such as ours. All these events were carried out with great success and highly favorable feedback from the respective audiences.

7. Cost analysis

Costs include upfront and maintenance costs for both hardware and software. The hardware totals at less than 1800 EUR for purchasing all but minor components such as screws or cable ties. While there was limited effort involved in system design (less than one person month of a full-time Master student well versed in CAD design and construction), the components can be ordered online at minimal cost by re-using our blueprints provided on GitHub. Full assembly from scratch takes one person approximately 1-2 days, the software setup is highly automated and takes 2-3 h, whereas its development required approximately one person month. Running costs are minimal, as the system takes no special effort to maintain once it has been set up (comparable to any other Linux system), and power costs are negligible, given its maximum theoretical power consumption of 300 W and a measured maximum of less than 200 W (comparable to a moderately powerful desktop computer). On-site set up time for a public outreach event including host PC, webcam, screens, etc. is approximately 30 min.

In terms of stability and hardware repairs, the system is deliberately composed of easily exchangable commodity hardware components that can be replaced at minimal cost. However, since its first use, we encountered only a single hardware component failure, namely an SD card.

8. Conclusion

In this work, we have demonstrated the feasibility of constructing a portable HPC system for education and public outreach, the Cluster Coffer. We outlined our perspective on this topic, which already gave rise to numerous miniature clusters, and detailed on the design process and its key elements. In addition, we demonstrated two use cases of our cluster for teaching students and low-threshold research dissemination, for which it will continue to serve us for the next years. With its modular design and feasible commodity components, we do not expect any long outages or expensive repairs, and our experience so far has shown the system to attract a lot of attention among both students and the general public.

Future work includes performance optimization of the software stack, as the benchmark data presented clearly shows there is room for improvement with respect to the theoretical peak performance. Furthermore, we intend to include the Cluster Coffer in

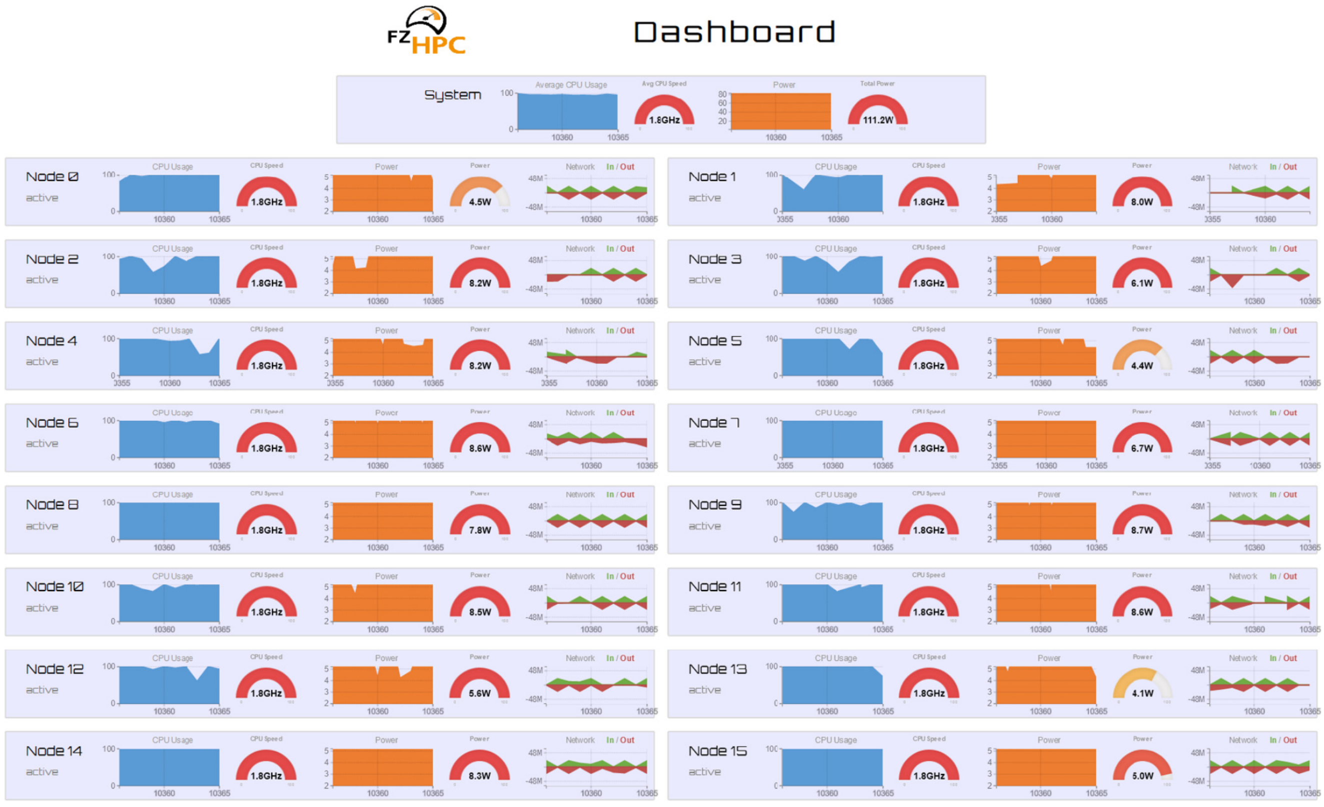

Fig. 14. Dashboard visualization of the cluster under full load when running iPiC3D with a large number of particles. The individual graphs show, from left to right: CPU load, CPU clock frequency, power consumption, current power consumption, and network load in (green) and out (red).

GPU programming courses that teach OpenCL and SYCL by working on the Mali GPUs. While the Cluster Coffer was already used in reaching out to pupils during events hosted at our university, we intend to extend our efforts to on-site events in schools, given the portability of the system. Finally, we are considering the option of creating a small curriculum around the use of this system, offering a pre-defined set of exercises with expected learning outcomes that could be re-used by similarly instrumented systems.

CRediT authorship contribution statement

Philipp Gschwandtner: Conceptualization, Writing - original draft and editing, Visualization, Software, Supervision. **Alexander Hirsch:** Hardware. **Peter Thoman:** Software, Visualization. **Peter Zangerl:** Software, Visualization. **Herbert Jordan:** Conceptualization. **Thomas Fahringer:** Funding acquisition.

Declaration of competing interest

There are no known conflicts of interest.

References

[1] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, T. Wilde, A case study of energy aware scheduling on SuperMUC, in: J.M. Kunkel, T. Ludwig, H.W. Meuer (Eds.), Supercomputing, Springer International Publishing, Cham, 2014, pp. 394–409.

[2] D. Bedard, M.Y. Lim, R. Fowler, A. Porterfield, PowerMon: fine-grained and integrated power monitoring for commodity computer systems, in: Proceedings of the IEEE SoutheastCon 2010, SoutheastCon, 2010, pp. 479–484.

[3] Y. Censor, Pareto optimality in multiobjective problems, Discrete Appl. Math. 4 (1) (1977) 41–59.

[4] S.J. Cox, J.T. Cox, R.P. Boardman, S.J. Johnston, M. Scott, N.S. O'brien, Iridis-Pi: a low-cost, compact demonstration cluster, Clust. Comput. 17 (2) (2014) 349–358.

[5] J. Dongarra, P. Luszczek, M.A. Heroux, K. Ye, HPCG benchmark, https://www.hpcg-benchmark.org/, 2021.

[6] U. Edinburgh, Wee archie, https://www.epcc.ed.ac.uk/discover-and-learn/resources-and-activities/what-is-a-supercomputer/wee-archie, 2020.

[7] D.S. Engineering, Das U-Boot – The Universal Boot Loader, https://www.denx.de/wiki/U-Boot, 2020.

[8] V.W. Freeh, N. Kappiah, D.K. Lowenthal, T.K. Bletsch, Just-in-time dynamic voltage scaling: exploiting inter-node slack to save energy in MPI programs, J. Parallel Distrib. Comput. 68 (9) (2008) 1175–1185.

[9] S.J. Johnston, P.J. Basford, C.S. Perkins, H. Herry, F.P. Tso, D. Pezaros, R.D. Mullins, E. Yoneki, S.J. Cox, J. Singer, Commodity single board computer clusters and their applications, Future Gener. Comput. Syst. 89 (2018) 201–212, https://doi.org/10.1016/j.future.2018.06.048, http://www.sciencedirect.com/science/article/pii/S0167739X18301833.

[10] H. Jordan, T. Heller, P. Gschwandtner, P. Zangerl, P. Thoman, D. Fey, T. Fahringer, The allscale runtime application model, in: 2018 IEEE International Conference on Cluster Computing, CLUSTER, 2018, pp. 445–455.

[11] H. Jordan, P. Gschwandtner, P. Thoman, P. Zangerl, A. Hirsch, T. Fahringer, T. Heller, D. Fey, The allscale framework architecture, Parallel Comput. (2020) 102648, https://doi.org/10.1016/j.parco.2020.102648, http://www.sciencedirect.com/science/article/pii/S0167819120300417.

[12] J.H. Laros, P. Pokorny, D. DeBonis, PowerInsight - a commodity power measurement capability, in: 2013 International Green Computing Conference Proceedings, 2013, pp. 1–6.

[13] M.P. Ltd, Scalable clusters make HPC R&D easy as Raspberry Pi, http://bitscope.com/blog/FM/?p=GF13L, 2020.

[14] A. Marinos, What would you do with a 120-Raspberry Pi Cluster?, https://www.balena.io/blog/what-would-you-do-with-a-120-raspberry-pi-cluster/, 2014.

[15] S. Markidis, G. Lapenta Rizwan-uddin, Multi-scale simulations of plasma with iPIC3D, Math. Comput. Simul. 80 (7) (2010) 1509–1519, https://doi.org/10.1016/j.matcom.2009.08.038, multiscale modeling of moving interfaces in materials, http://www.sciencedirect.com/science/article/pii/S0378475409002444.

[16] F. Nielsen, Introduction to HPC With MPI for Data Science, 2016.

[17] OpenMP Architecture Review Board, OpenMP application program interface version 5.0, https://www.openmp.org/spec-html/5.0/openmp.html, May 2018.

[18] A. Petitet, R.C. Whaley, J. Dongarra, A. Cleary, HPL - a portable implementation of the high-performance linpack benchmark for distributed-memory computers, https://www.netlib.org/benchmark/hpl/, 2018.

[19] A.M. Pfalzgraf, J.A. Driscoll, A low-cost computer cluster for high-performance computing education, in: IEEE International Conference on Electro/Information Technology, IEEE, 2014, pp. 362–366.

[20] D. Ruiz, HPCG for arm, https://github.com/ARM-software/HPCG_for_Arm, 2020.
[21] T.R. Scogland, C.P. Steffen, T. Wilde, F. Parent, S. Coghlan, N. Bates, W.-c. Feng, E. Strohmaier, A power-measurement methodology for large-scale, high-performance computing, in: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ICPE '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 149–159.
[22] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, CM-5/1024, https://www.top500.org/system/166997/, 1993.
[23] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, XP/S140, https://www.top500.org/system/172538/, 1994.
[24] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, Graham, https://www.top500.org/system/179047/, 2020.
[25] Khronos OpenCL Working Group, SYCL Specification 1.2.1, Tech. rep., Khronos OpenCL Working Group, 2017.
[26] Top500, Supercomputer Fugaku, https://www.top500.org/system/179807/, 2020.
[27] F.P. Tso, D.R. White, S. Jouet, J. Singer, D.P. Pezaros, The Glasgow Raspberry Pi Cloud: a scale model for cloud computing infrastructures, in: 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, 2013, pp. 108–112.
[28] Wikipedia, March 1989 geomagnetic storm, https://en.wikipedia.org/wiki/March_1989_geomagnetic_storm, 2020.
[29] S. Williams, A. Waterman, D. Patterson, Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76, https://doi.org/10.1145/1498765.1498785.

Philipp Gschwandtner is a senior scientist at the Research Center HPC at the University of Innsbruck, Austria. He completed his PhD in 2017 with a thesis on performance and energy analysis and optimization of parallel programs. His main research interests lie in high performance computing and parallel programming, including adjacent topics such as scientific computing, program optimization, API design, runtime systems, and portability.

Alexander Hirsch is a PhD student at the department of computer science at the University of Innsbruck, Austria. Completing his master in 2017, he implemented a Haskell framework for static program analysis and rapid prototyping. Since then he focused on embedded parallel hardware and its optimizations for both academic and industry use cases.

Peter Thoman is an assistant professor of computer science at the University of Innsbruck, Austria. His research focus has been to improve the effective performance and programmability of complex, highly parallel systems. He has created and contributed to various APIs for HPC and GPGPU platforms, such as the Celerity high-level single-source platform for GPU Clusters. He also investigates runtime systems, as well as compiler-based tools for performance optimization as well as improved developer support.

Peter Zangerl is a former PhD student at the University of Innsbruck, Austria. During his studies at the Distributed and Parallel Systems group, he was part of the Insieme research compiler and runtime developer team. He focuses on both compiler and runtime development, with a strong research interest in compiler analysis for guided runtime parameter tuning.

Herbert Jordan is a former PostDoc at the University of Innsbruck, Austria. After completing his PhD in 2014 on the Insieme optimizing compiler infrastructure he became the Scientific Coordinator of the EU H2020 project AllScale, aimed at exposing nested recursive parallelism for distributed memory. His main research interests are in programming language and API design, static program analysis, code transformations, and performance optimizations.

Thomas Fahringer is a professor of computer science at the University of Innsbruck since 2003. His research focuses on supporting researchers in science and engineering by developing, analyzing, and optimizing parallel and distributed applications. Fahringer was involved in numerous national and international research projects including 15 EU funded projects, published 5 books, 40 journal and magazine articles, and more than 200 reviewed conference papers. He is a member of the IEEE and ACM.