# On-The-Fly Computing: A Novel Paradigm for Individualized IT Services

Markus Happe, Friedhelm Meyer auf der Heide, Peter Kling, Marco Platzner, Christian Plessl
University of Paderborn, Germany
{markus.happe, fmadh, peter.kling, platzner, christian.plessl}@uni-paderborn.de

## Abstract

*In this paper we introduce "On-The-Fly Computing", our vision of future IT services that will be provided by assembling modular software components available on world-wide markets. After suitable components have been found, they are automatically integrated, configured and brought to execution in an On-The-Fly Compute Center. We envision that these future compute centers will continue to leverage three current trends in large scale computing which are an increasing amount of parallel processing, a trend to use heterogeneous computing resources, and—in the light of rising energy cost—energy-efficiency as a primary goal in the design and operation of computing systems. In this paper, we point out three research challenges and our current work in these areas.*

## 1. Introduction

Today we find ourselves on the brink of a new era in the development and implementation of services of information technology (IT). We are witnessing the beginnings of a shift away from the 40-year-old principle of either acquiring software by purchasing expensive, relatively inflexible standard solutions or relying on the even more expensive method of commissioning customized solutions from external software companies or in-house software departments. Service-oriented architectures provide methods for tailoring existing software components into customized IT services where the computing resources for executing these services can be commissioned on-demand only when necessary and only in the required form by using cloud and grid computing approaches. These initial advances towards a new way of providing IT services are the starting point for our research activities in *On-The-Fly Computing*.

*This work is partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) and by the Graduate School on Applied Network Science (GSANS).*

With On-The-Fly Computing we aim at developing techniques and processes for automatic (on-the-fly, OTF) configuration and provision of individual IT services out of a basis of services that are available on world-wide markets. Figure 1 shows an example OTF Computing scenario for the domain of optimization. A user demands an application software for a given optimization problem and contacts a so-called OTF service provider for optimization. Based on a proper specification of the desired application, the OTF service provider uses domain knowledge to decompose the application into smaller services such as data management, model generation, model solving, visualization and decision support. For each of these services, the OTF provider searches world-wide markets for available and matching services. After composing the found services to the overall application, one or more OTF computer centers are commissioned to execute the application. In addition to methods for specifying, searching, matching, configuring and provisioning of services, the OTF scenario involves developing methods for quality assurance, security, interaction and market developments.

At the University of Paderborn we investigate On-The-Fly Computing within a so-called Collaborative Research Centre (CRC), which in the German funding landscape is a large-scale and long-term fundamental research project aligning the work of some 40 researchers for up to 12 years. CRCs are highly competitive programs established at universities to enable researchers to pursue an ambitious and outstanding research program crossing the boundaries of disciplines, institutes, departments and faculties.

In this paper we first survey the vision and goals of the CRC On-The-Fly Computing and then discuss in more detail research challenges and approaches for the organization and operation of future On-The-Fly compute centers.

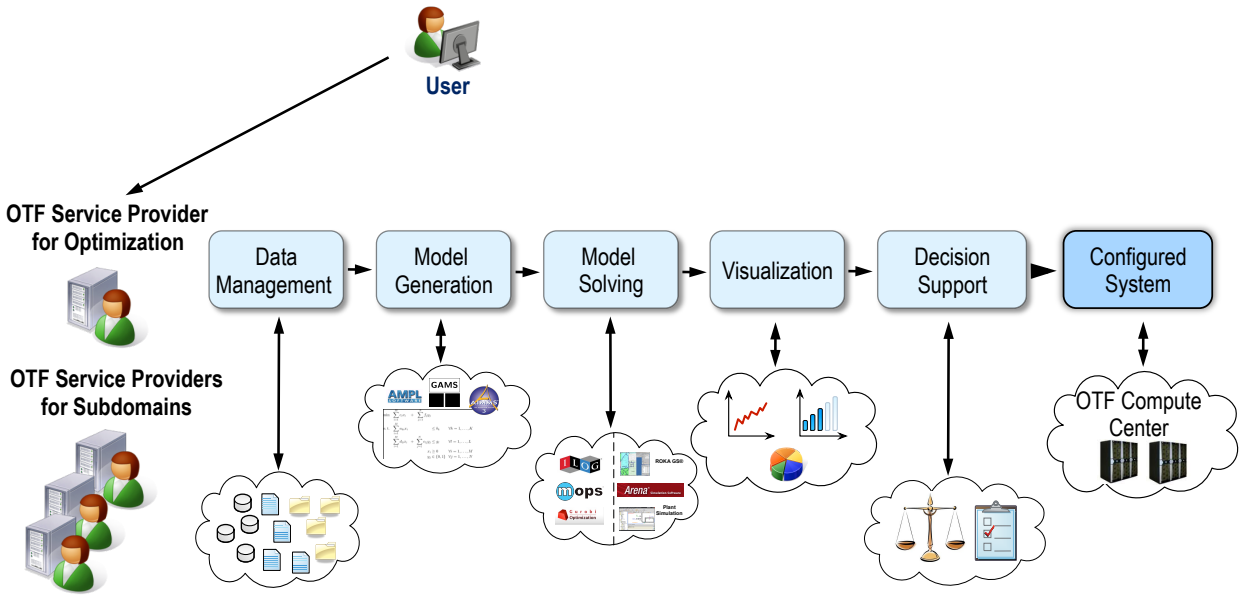


Figure 1. Exemplary OTF Computing scenario for the domain of optimization.

## 2. Vision and Goals of OTF Computing

The vision of the CRC On-The-Fly Computing[1] is that in future IT services will be individually and automatically configured and implemented by combining flexible services available on free world-wide markets. With this vision our CRC looks far into the future of IT development and usage, the transformation of which we are already experiencing today. In order to research the extent to which this vision can be realized, we will develop methods and techniques that enable an almost entirely automatic configuration, implementation and adaptation of IT services from services available on world-wide markets, guarantee the protection not only of the services acquired in this way, but also of the active participants in the markets, and support the organization and further development of these markets and the necessary interaction between those involved. Therefore, we also study ways of organizing markets whose participants maintain a lively service landscape by dedicated entrepreneurial action.

To reach these goals, computer science experts from diverse disciplines such as software technology, algorithmics, computer networks, computer systems architecture, security and cryptology are working hand-in-glove with economists who contribute their specific expertise on how to promote the organization and further development of the market. Furthermore, business information specialists contribute their expertise in operations research, which flows directly into an application domain for testing the methods and techniques developed in the CRC.

The research activities of CRC On-The-Fly computing are divided into three areas:

- *Algorithmic and economic foundations for organizing large dynamic markets*
  The basis for organizing the networks of participants and services in the markets are methods of distributed computing, since the sheer size and dynamics of these networks render central control impossible. For this reason, we are researching local methods that would allow the dynamics of the networks to be controlled and matched to the respective current requirements. Moreover, we are developing economic concepts that allow specific stimuli to be used as a means of controlling the behavior of the participants and, ultimately, ensuring the achievement of a globally successful market.
- *Modeling, composition and quality analysis for On-the-Fly Computing*
  Software technology methods that enable an exact description of services, and methods that are both

1. For more information see http://sfb901.uni-paderborn.de/

easy and extensive enough for these tasks, are a prerequisite for seeking, finding and configuring services. The achievement of high product quality in terms of functional and non-functional characteristics demands innovative methods of analysis and verification. Concepts from the fields of logic and heuristic search methods are required for the configuration of new services.

- *Reliable execution environments and application scenarios for On-The-Fly Computing*
  In this area we are concerned with the questions of robustness and security of markets and the processes of service provisioning, as well as the organization of highly heterogeneous compute centers, denoted as On-the-Fly Compute Centers. Furthermore, we also work towards an application case study to perform long-term trials of our innovative methods. In this case study we are concerned with possible methods for modeling, configuring and optimizing systems for supply and logistics networks.

## 3. On-The-Fly Compute Centers

The realization of the OTF computing vision requires a tight interaction of numerous processes that autonomously search, compose and validate software components to configure an individualized IT service. In this chapter, we focus on three challenges related to the execution of these composed services in OTF compute centers. We envision that future compute centers will continue to leverage three main trends in large scale computing which are a) an increasing amount of parallel processing at the node (cluster) and chip-level (multi-/many-core), b) an increasing amount of heterogeneity through the use of computing accelerators such as field-programmable gate arrays (FPGAs) or general-purpose graphics processing units (GPUs), and c) a further growing importance of energy consumption as a cost factor.

### 3.1. Challenge 1: Efficient usage of energy in future data centers

Over the last decade, energy usage has become a major design issue for data centers. Several studies show that the energy used for both the actual computation as well as the cooling constitutes a large part of the total operational costs (see, e.g., [1]). Our vision of OTF computing will not only increase the overall demand for computational power, but will also require efficient ways to cope with a large number of requests at short notice. In combination with improvements on the technical level, algorithmic research has great potential to improve both the performance and the (energy) efficiency of data centers. See for example [2] for a good insight on the role of algorithms to fully exploit the energy-saving mechanisms of modern systems. Two of the most prominent techniques for power saving are *dynamic speed scaling* (also known as *voltage scaling*) and *power-down*. While the former allows a system to save energy by adapting its speed to the current workload, the latter can be used to transition into a special sleep mode to conserve energy. Scheduling algorithms deployed in OTF compute centers – our vision of future data centers – must be able to make use of these and other techniques in a scenario where only limited (if any) information about the future workload is available (also referred to as the *online* scenario).

**Basics of Energy-efficient Scheduling.** Algorithmic research on energy-efficient scheduling algorithms was initiated by Yao et al. [3]. They considered a single-processor system where jobs (or tasks) arrive over time and must be assigned to the processor. Each job $j$ comes with its own release time $r_j$, deadline $d_j$, and workload $w_j$. Note that preemption is allowed in this model. In addition to the job assignment, the scheduler can use speed scaling to change the processor's speed at any point in time. Higher speeds consume higher energy, where the exact relation is modeled by the power function $P(s) = s^\alpha$ (energy consumption at speed $s$). Here, $\alpha > 1$ is a system depended constant. Typically, the power consumption of real-world CMOS based systems can be approximated by choosing $\alpha \in \{2, \dots, 5\}$ [4]. The objective is to finish all jobs until their deadline while minimizing the invested amount of energy. Here, the energy $E(S)$ of a schedule $S$ is power integrated over time: $E(S) = \int_{t=0}^{\infty} P(s(t))\,\mathrm{d}t$, where $s(t)$ is the system's speed at time $t$. It is essential to note that speed scaling drastically changes the problem's nature compared to classical (deadline) scheduling. For example, it is no longer infeasible to solve the problem optimally as long as one has knowledge of all jobs prior to their arrival (also referred to as the *offline* scenario). Yao et al. derived a polynomial time optimal offline algorithm as well as the two polynomial time online algorithms *Optimal Available* (OA) and *Average Rate* (AVR). Up to now, OA remains one of the most important algorithms in this area, as it is used as a basic building block in many scheduling strategies. Using an elegant amortized potential function argument, Bansal et al. [5] were able to show that OA's *competitiveness* is $\alpha^\alpha$. In other words, even without knowing the future OA is at most by a factor of $\alpha^\alpha$ worse compared to an optimal offline algorithm. In the past 15 years,

algorithmic research has considered many variations of Yao et al.'s basic model, including multiprocessor variants, systems featuring a sleep state, processors with a discrete set of possible speeds, throughput-based objectives instead of deadlines, and combinations of these. Albers [6] gives a comprehensive overview over the different models and known results.

**Relevant Models for OTF Compute Centers.** The focus of our models and algorithms for OTF compute centers is twofold: On the one hand, we want to capture the profitability aspect of data centers, ensuring that our schedules restrict the amount of energy invested into jobs of low value (for which customers pay only a small amount of money) and operate the data center as cost-effectively as possible. On the other hand, we have to deal with the online variant of the problem, where the scheduler has no knowledge about the future demand of computational resources (which may change frequently and arbitrarily in the OTF scenario).

One of our most basic models is the first to combine the two most prominent energy conservation techniques (speed scaling and power down) and profit-oriented scheduling models [7]. In this model, a job $j$ has an additional value $v_j$. In contrast to the model by Yao et al., we no longer need to finish all jobs until their deadline. However, when missing a deadline we suffer a loss equaling the job's value. Thus, instead of merely minimizing the invested energy, we want to minimize the invested energy plus the loss suffered due to missed deadlines. Moreover, our model takes into account the system's overhead (conserving the contents of registers, maintaining communication sockets, ...) by modeling the processor's power consumption at speed $s$ as $P(s) = s^\alpha + \beta$. Thus, even when running at speed zero the processor's energy consumption is $\beta > 0$. To avoid this overhead, the processor can enter a sleep state, in which it consumes zero power. Waking up from this sleep state causes an additional cost of $\gamma > 0$. Our major insight is that the combination of speed scaling, sleep state, and profitability increases the problem complexity substantially. While, separately, each of these aspects allows for a simple online algorithm with constant competitiveness, we can show that their combination causes any such algorithm to become arbitrarily bad. It turns out that the maximum value density $\delta_{\max}$ (i.e., the ratio between a job's value and its workload) is a parameter that is inherently connected to the necessary and sufficient competitive ratio achievable for our online scheduling problem.

**Theorem 1** (see [7])**.** *The competitive ratio of any algorithm whose scheduling decisions are solely based on the current system state and job properties is at most* $\Theta(\delta_{max})$*. Moreover, there is such an algorithm that achieves competitiveness* $\alpha^\alpha + 2e\alpha + \Theta(\delta_{max})$.

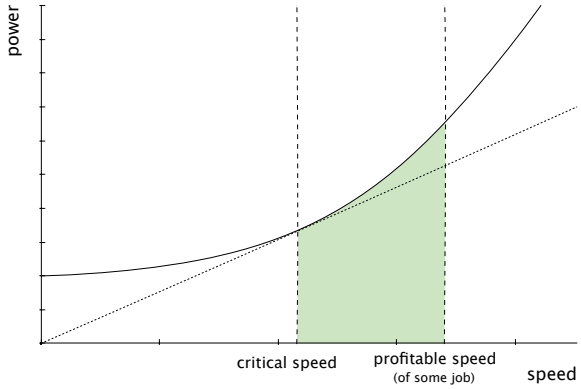


Figure 2. Our algorithm tries to use job speeds that essentially stay in the shaded interval.

The proposed algorithm uses a greedy sleeping policy, suspending the processor whenever the energy consumed while idle reaches $\gamma$. Moreover, it features a minimal *critical speed* (scheduling jobs with a slower speed is guaranteed to incur a loss) and a job-dependent maximum *profitable speed* (see Figure 2).

Note that the above model is not yet suited to model the computing resources of a large scale data center, as it considers only a single processor. In our current work, we overcome this weakness and extend the profitability and speed scaling aspect to multiple processors. More exactly, we achieve substantial improvements over a result by Chan et al. [8]. They showed that given job values and *one* speed-scalable processor (no sleep state), a competitiveness of $\alpha^\alpha + 2e\alpha$ is achievable. Using a new and significantly different technique based on work by Gupta et al. [9], we derive an algorithm for multiple processors and with an improved competitiveness of $\alpha^\alpha$. Our technique is based on primal-dual methods known from linear programing applied to a convex programming formulation of our scheduling problem.

**Current Research.** Currently, our research concentrates on models and algorithms for more heterogeneous architectures and more sophisticated preemption mechanisms. As homogeneous parallel systems start to reach their limits in scalability (especially due to communication and bandwidth bottlenecks), more and more experts see heterogeneous systems as the next major step in the evolution of computer architec-

tures. Thus, researchers experiment with heterogeneous chips, featuring not only many hundreds or even thousands of cores, but different types of cores (CPUs, GPUs, FPGAs, ...). Such architectures pose many exciting new algorithmic problems. For example, one can show that scheduling heterogeneous multiprocessors is not only a different but an algorithmically more challenging problem than scheduling in homogeneous systems [10]. Another very interesting challenge is to take into account the variable and non-negligible costs of transmodal migration (see Section 3.2). Due to the more complex migration process, these costs can make up a significant portion of the scheduling costs. As an example, consider the migration from a software thread on the CPU to an FPGA, which not only involves storing and loading of thread contexts but also reprogramming of the FPGA.

## 3.2. Challenge 2: Transmodal migration

Numerous studies have shown that the use of heterogeneous computing resources can significantly improve the performance or energy-efficiency of computers. Hence, heterogeneous computing resources are increasingly deployed in compute centers. In current operating systems, the management of heterogeneous resources is static and controlled by the application. That is, the applications either assume or detect the presence of a particular computing resource type and use this resource exclusively during their execution. From the OTF compute center provider's perspective, this static, exclusive and application-controlled binding of tasks and heterogeneous resources can result in suboptimal behavior. For examples, tasks may be blocked while waiting for a currently unavailable heterogeneous resource. Also, if functionally equivalent implementations for different heterogeneous computing resources are available, migrating computations between different resource may be beneficial from a performance, energy-efficiency, cost, or utilization point of view. While migrating computations between computing resources is considered state-of-the-art if the resources are homogeneous instruction set architectures, moving tasks at runtime between different resource types is an open research question. In particular, moving tasks between instruction set architectures such as CPUs and non-instruction-based accelerators such as FPGAs, which we denote as *transmodal migration*, is an open research problem since the architectures use completely different execution models and context representations.

**Basics of Transmodal Migrations.** The preemption and migration of tasks across the hardware/software boundary was investigated, for example, by Götz et al. [11]. They proposed a framework that generates templates for hardware and software tasks based on a task structure description. These templates are then filled out by the task designer, including a specification of viable migration points. Mignolet et al. [12] implemented a comparable thread migration technique by integrating checkpoints at which task interruption and context transfer is possible. They generate both hardware and software task representations from the same description and map inter-task communication to a message-passing mechanism. Koch et al. [13] modeled the migration of finite state machine-based hardware tasks to software and vice versa on predefined morph points, where equivalent context representations exist in hardware and software together with a mapping morph function. They automatically insert the state access logic necessary to access a hardware task's state by a custom tool chain. A formal investigation of hardware/software state equivalence, motivated by the migration of applications between software and FPGAs, was presented by Blumer et al. [14]. The authors developed a taxonomy of migration realms consisting of processors modeled as finite state machines and describe the migration of processes within a migration realm. Pellizzoni and Caccamo [15] proposed a real-time computing architecture that supports quality-of-service (QoS) adaptation on a hybrid CPU/FPGA system featuring seamless migration of (periodic) tasks between hardware and software. A hardware task runs for the entirety of its period. Thus, at most one task at a time is allocated to a hardware region. Since there is no support for task preemption, migration can occur only at the start of a period. Additionally, a task exists in multiple configurations (variants) for different QoS constraints, typically a software version and several differently optimized hardware implementations. Tasks are migrated in response to newly arriving tasks, changing QoS constraints, or differing workloads.

Related work on transmodal migration has demonstrated basic migration techniques using rather simple case studies. It is not obvious how well the presented approaches will scale for larger application scenarios. Furthermore, related work has not tackled the integration of migration techniques into programming models or the operating system level, e.g., by extending standard Linux.

**Relevant Programming Models for OTF Compute Centers.** In our research we are studying how new pro-

gramming models or specializations of current models can support transmodal migration and how we can exploit transmodal migration to optimize the operation of the compute center. We will base our work on *ReconOS* [16], our own multithreaded programming model and execution environment for software and reconfigurable hardware. ReconOS promotes passive hardware accelerators, which are fully controlled by software, to active so-called *hardware threads*, which can independently initiate operating system (OS) calls, access shared memory, synchronize and communicate with other threads in the system. Hardware threads are configured into specific reconfigurable areas on the FPGA and can be replaced by other hardware threads over time. A hardware thread is partitioned into three components: (highly-parallel) user logic, an operating system synchronization state machine (OSFSM), and, a local memory. The user logic comprises the accelerator data paths of the thread and implements the main computations. The OSFSM controls the accelerator's data paths using handshake signals and provides access to shared resources such as the main memory, message boxes, and, semaphores using dedicated library functions. The local memory buffers data blocks for an efficient processing.

The hardware thread is connected to a master CPU using an operating system interface (OSIF). Software delegate threads represent the hardware threads on the master CPU, which hosts a monolithic operating system such as Linux. Whenever a hardware thread performs an operating system call, the command and the parameters are forwarded to the corresponding delegate, which carries out the system call on behalf of the hardware thread. The OSFSM of the hardware thread is blocked until the delegate returns the result of the OS call. Software threads being executed on worker processors can access the operating system in a similar manner with the difference that the thread does not have to be partitioned into multiple parts. Due to the delegate approach the thread modality is fully transparent to the host operating system, i.e., the host OS does not need to know whether a thread is executed on the master CPU, a worker CPU or in a reconfigurable hardware slot. Memory read and write operations are handled transparently using a dedicated memory subsystem. Depending on the host OS ReconOS also supports virtual memory for hardware threads, see [17]. Figure 3 depicts the architecture of a ReconOS systems with two CPUs and two reconfigurable hardware slots.

To optimally utilize the reconfigurable hardware accelerators, ReconOS supports hardware multitasking using a cooperative approach, see [18]. The operating

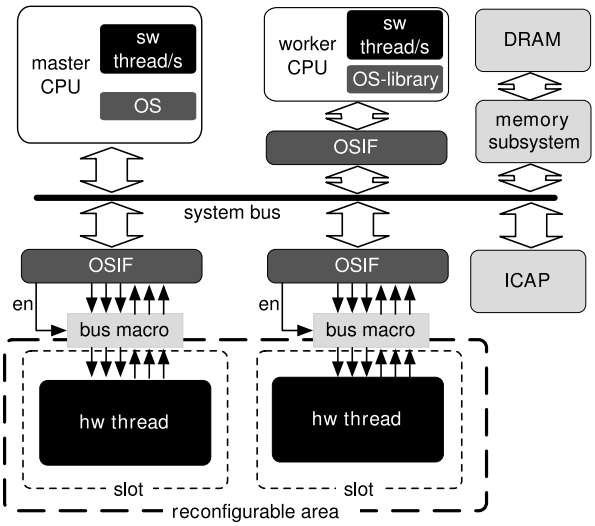


Figure 3. ReconOS architecture with one master CPU, one worker CPU and two reconfigurable hardware slots.

system and the hardware threads work together in order to find adequate preemption points, where the thread context is well-defined and minimal, and in order to transfer the thread context. By applying cooperative multitasking, the overhead in FPGA logic is drastically reduced compared to a system applying preemptive hardware multitasking. Cooperative multitasking can be seen as a compromise between non-preemptive and preemptive multitasking allowing for infrequent thread preemptions at the cost of an acceptable hardware overhead. In ReconOS, the preemption points are usually combined with OS calls where the hardware thread is blocked.

**Current Research.** Our current research focus is set on the transmodal thread migration between software and hardware threads. Transmodal thread migration between CPUs and hardware slots impose many challenges. The toughest challenge is the translation of the context from software to hardware, or vice versa. As one possibility to allow for transmodal thread migration we will extend the cooperative multitasking approach from hardware threads to hybrid threads, which can be executed in hardware and in software. As a first approach, we will rely on migration points that are defined and programmed by the application designer manually. Then, we will investigate if such migration points can be automatically extracted from high-level descriptions such as Kahn process networks or synchronous data flow graphs. We intend to ex-

perimentally demonstrate our transmodal migration techniques using a multithreaded video object tracking case study, which has already been investigated for performance management in previous work [19]. Finally, ReconOS which was initially developed for managing hybrid multi-cores in embedded system-on-chips (SoCs) will be adapted to heterogeneous high-performance compute nodes. Such nodes comprise a high-end multi-core processor and several FPGA accelerator cards and will be typical for OTF computer centers.

### 3.3. Challenge 3: Just-in-time acceleration

Transmodal migration assumes that tasks are available in functionally equivalent implementations for different resource types. Depending on the scenario, this may not be a valid assumption. For example, certain software components may be available only in binary form for a specific resource type. In other cases, the source code for the component may be available, but only for one particular resource type. The OTF compute center provider however still has an strong incentive to use the available heterogeneous compute resources to increase resource utilization and to profit from performance and energy efficiency benefits of heterogeneous computing, which can reach up to several orders of magnitude, see for example [20], [21].

**Basics of Just-in-time acceleration.** We aim at solving this challenge by using just-in-time (JIT) compilation techniques, which analyze binary applications during execution, identify the runtime intensive parts (hotspots), and translate these hotspots to specifications for application-specific accelerators implemented on heterogeneous resources. Depending on the binary format, decompilation techniques are required to reconstruct high-level program information from the executable application. While just-in-time compilation and binary optimization methods have worked for instruction set processors, the application of this idea to the domain customizable accelerators, in particular, FPGAs is a novel and ambitious idea. While the general feasibility of translating software binaries to circuits, denoted as binary synthesis, has been demonstrated [22], [23], the application of these methods during runtime has so far only be rudimentarily studied for restricted embedded systems. Seminal work in this area has been conducted by Vahid, Stitt and Lysecky [24], [25]. These works are restricted to very simple applications that can be accelerated with custom instructions. Further, a custom system-on-chip (SoC) system comprising a purely combinational FPGA architecture is assumed as target platform. While these works rely on costly decompilation methods to reconstruct the application's structure, Bispo and Cardoso study in recent work [26], whether this information can also be extracted from monitoring the instruction trace. They also consider an embedded SoC system, but assume that the accelerator is mapped to a coarse-grained reconfigurable array. In contrast to Vahid et al., Bispo and Cardoso also consider the acceleration of complicated loop structures, which is a precondition for fully exploiting the potential of FPGAs. The works of Clark et al. put the emphasis less on the practical implementation, but on the exploration of the design space for systems with just-in-time hardware acceleration. They primarily focussed on CPUs with reconfigurable instruction sets [27], in later work they also studied programmable loop accelerators [28].

While related work has proven the general feasibility of the approach for embedded systems with custom SoC architectures it is an open research question, whether the approach is viable in a compute center context, which relies on high-performance computer architectures, commercially available heterogeneous accelerators, and server operating systems. Hence, the objective of our research is to study the potential and the fundamental and practical limitations of the approach in the OTF compute center context.

In our previous work [29], we have studied under what conditions the time overhead of just-in-time generation of FPGA accelerators can be amortized when targeting a reconfigurable instruction set processor implemented with current off-the-shelve FPGA technology and standard tool flows. Our studies have revealed a strong dependence on the kind of applications. For comparatively simple and regular applications from the embedded computing domain, the time overhead will be amortized on average after the application has been executed for two hours, due to the high average acceleration factor obtained for this application class. After this time, the execution of the application on the FPGA accelerator will result in a net performance benefit over the execution on the CPUs. For general purpose applications we have found that the custom instruction approach is too limiting, since it results in rather low acceleration, which increases the time until the overheads have been amortized to many hours or even days. This may still be practical for jobs in the OTF compute centers that are long-running or executed over and over again, but it limits the applicability of the approach.

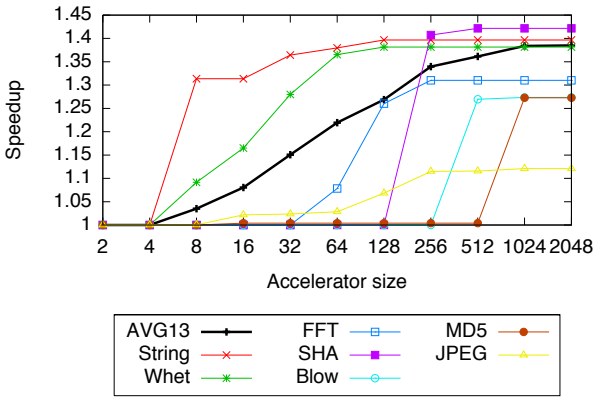


Figure 4. Evaluation of the achievable speedup for a reconfigurable accelerator as a function of the maximum capacity of the accelerator for different benchmarks (AVG13 is the average over all 13 benchmarks.)

**Current Research.** Currently, we are developing methods that allow us to estimate the achievable performance when offloading computations to heterogeneous computing resources. To this end, we build on our previous work [30] that uses a combination of analytical models (for modeling the latency and bandwidths between CPU and the heterogeneous computing resources), static and dynamic program analysis (for modeling the control and data flow in the application), and design space exploration for CPU/accelerator partitioning (for determining the optimal distribution of functionality between different computing resources). This evaluation infrastructure will allow us to explore the acceleration potential for a given application and concrete system parameters. Further, it will enable us to explore effect of computer system parameters, such as memory latencies or I/O bus bandwidths, to design computer systems with accelerators, to extrapolate the performance of future computing system, and to understand the sensitivity of these parameters.

Figure 4 shows an exemplary result from such an evaluation. In this simulation, we have studied the impact of limited hardware accelerator resources on the speedup, which was limited to a factor of 2 in this experiment. We can observe a distinct saturation behavior in the speedup for all applications. If the capacity of the accelerator is below a benchmark-specific threshold no speedups can be obtained because the performance-critical hotspots do not fit to the accelerator. If the capacity exceeds this threshold the speedups quickly reach a plateau, where further increasing the capacity does not lead to higher speedups.

## 4. Conclusion

In this paper we have introduced "On-The-Fly Computing", our vision of future IT services which is investigated in a large-scale research project at the University of Paderborn. The OTF computing vision poses a number of different challenges and requires not only a large but also a multi-disciplinary team of researchers to work on it. We have briefly pointed out the involved disciplines from computer science and economics and the main working areas. The second part of the paper has then focused on OTF computer centers. OTF compute centers will be pivotal for the OTF computing scenario since they finally execute the assembled software compositions under dynamically agreed on service level agreements. We have discussed three challenges for OTF compute centers, novel scheduling techniques for energy-efficiency, transmodal migration of functions across the software/hardware boundary to optimize energy-efficiency or performance and, finally, just-in-time acceleration techniques that make code available only in binary form or only for specific resource types amenable to acceleration on heterogeneous compute nodes.

## References

[1] L. A. Barroso, "The price of performance," *ACM Queue*, vol. 3, no. 7, pp. 48–53, 2005.

[2] S. Albers, "Energy-efficient algorithms," *Communications of the ACM*, vol. 53, no. 5, pp. 86–96, 2010.

[3] F. F. Yao, A. J. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, 1995, pp. 374–382.

[4] T. Mudge, "Power: A first-class architectural design constraint," *IEEE Computer*, vol. 34, pp. 52–58, 2001.

[5] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, pp. 1–39, 2007.

[6] S. Albers, "Algorithms for dynamic speed scaling," in *Proc. Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), T. Schwentick and C. Dürr, Eds., vol. 9. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 1–11.

[7] A. Cord-Landwehr, P. Kling, and F. Mallmann-Trenn, "Slow down & sleep for profit in online deadline scheduling," in *Proc. Mediterranean Conf.erence on Algorithms (MedAlg)*, ser. LNCS, G. Even and D. Rawitz, Eds., vol. 7659. Springer, 2012, pp. 218–231.

[8] H.-L. Chan, T.-W. Lam, and R. Li, "Tradeoff between energy and throughput for online deadline scheduling," in *Proc. Int. Workshop on Approximation and Online Algorithms (WAOA)*. Springer, 2010, pp. 59–70.

[9] A. Gupta, R. Krishnaswamy, and K. Pruhs, "Online primal-dual for non-linear optimization with applications to speed scaling," in *Proc. Workshop on Approximation and Online Algorithms (WAOA)*, 2012.

[10] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs, "Scheduling heterogeneous processors isn't as easy as you think," in *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*. SIAM, 2012, pp. 1242–1253.

[11] M. Götz, F. Dittmann, and T. Xie, "Dynamic relocation of hybrid tasks: Strategies and methodologies," *Microprocessors and Microsystems*, vol. 33, no. 1, pp. 81 – 90, 2009.

[12] J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-On-Chip," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2003, pp. 986 – 991.

[13] D. Koch, C. Haubelt, T. Streichert, and J. Teich, "Modeling and synthesis of hardware-software morphing," in *ISCAS*. IEEE, 2007, pp. 2746–2749.

[14] A. D. Blumer, H. S. Mortveit, and C. D. Patterson, "Formal modeling of process migration," in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2007, pp. 104–110.

[15] R. Pellizzoni and M. Caccamo, "Hybrid hardware-software architecture for reconfigurable real-time systems," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symp.*, 2008.

[16] E. Lübbers and M. Platzner, "ReconOS: Multithreaded Programming for Reconfigurable Computers," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 1, October 2009.

[17] A. Agne, E. Lübbers, and M. Platzner, "Memory virtualization for multithreaded reconfigurable hardware," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*. IEEE, 2011.

[18] E. Lübbers and M. Platzner, "Cooperative Multithreading in Dynamically Reconfigurable Systems," in *IEEE Int. Conf. on Field Programmable Logic and Applications (FPL)*. IEEE, 2009.

[19] M. Happe, E. Lübbers, and M. Platzner, "A Self-adaptive Heterogeneous Multi-core Architecture for Embedded Real-time Video Object Tracking," *Journal of Real-Time Image Processing*, pp. 1–16, 2011.

[20] M. B. Gokhale and P. S. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer, 2005.

[21] W. mei W. Hwu, *GPU Computing Gems Emerald Edition*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[22] G. Mittal, D. Zaretsky, X. Tang, and P. Banerjee, "An overview of a compiler for mapping software binaries to hardware," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 11, pp. 1177–1190, Nov. 2007.

[23] G. Stitt and F. Vahid, "Binary synthesis," *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–30, 2007.

[24] F. Vahid, G. Stitt, and R. Lysecky, "Warp processing: Dynamic translation of binaries to FPGA circuits," *IEEE Computer*, vol. 41, no. 7, pp. 40–46, Jul. 2008.

[25] R. Lysecky and F. Vahid, "Design and implementation of a MicroBlaze-based processor," *ACM Trans. on Embedded Computing Systems*, vol. 8, no. 3, pp. 1–22, 2009.

[26] J. Bispo and J. M. P. Cardoso, "On identifying and optimizing instruction sequences for dynamic compilation," in *Int. Conf. on Field Programmable Technology (ICFPT)*. IEEE Computer Society, Dec. 2010, pp. 437–440.

[27] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner, "An architecture framework for transparent instruction set customization in embedded processors," in *Proc. Int. Symp. on Computer Architecture (ISCA)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 272–283.

[28] N. Clark, A. Hormati, and S. Mahlke, "VEAL: Virtualized execution accelerator for loops," in *Proc. Int. Symp. on Computer Architecture (ISCA)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 389–400.

[29] M. Grad and C. Plessl, "On the feasibility and limitations of just-in-time instruction set extension for FPGA-based reconfigurable processors," *Int. Journal of Reconfigurable Computing (IJRC)*, 2012.

[30] T. Kenter, M. Platzner, C. Plessl, and M. Kauschke, "Performance estimation framework for automated exploration of CPU-accelerator architectures," in *Proc. Int. Symp. on Field-Programmable Gate Arrays (FPGA)*. New York, NY, USA: ACM, Feb. 2011, pp. 177–180.