Towards Software Performance Engineering for Multicore and Manycore Systems

Heiko Koziolek
Industrial Software Systems
ABB Corporate Research
Ladenburg, Germany
heiko.koziolek@de.abb.com

Steffen Becker
University of Paderborn
Paderborn, Germany
steffen.becker@upb.de

Jens Happe
SAP Research
Karlsruhe, Germany
jens.happe@sap.com

Petr Tuma
Faculty of Mathematics and Physics
Charles University Prague
Prague, Czech Republic
petr.tuma@d3s.mff.cuni.cz

Thijmen de Gooijer
Industrial Software Systems
ABB Corporate Research
Vasteras, Sweden
thijmen.de-gooijer@se.abb.com

ABSTRACT

In the era of multicore and manycore processors, a systematic engineering approach for software performance becomes more and more crucial to the success of modern software systems. This article argues for more software performance engineering research specifically for multicore and manycore systems, which will have a profound impact on software engineering practices.

1. INTRODUCTION

Despite rapidly improving hardware, many recent software systems are still suffering from performance problems, such as high response times or low throughputs [1]. Hardware is often not the limiting factor as powerful multicore and manycore processors are readily available on the market and modern software systems may run in huge data centers with virtually unlimited resources. Performance problems often stem from software architectures that are not designed to exploit the available hardware. Instead, these software architectures ignore the advances of distributed computing and multicore and manycore processors.

Systematic approaches for engineering software systems to achieve desired performance properties have been proposed [2, 3]. They advocate modeling software systems during early development stages, so that performance simulations can validate design decisions before investing implementation effort.

The advent of multicore processors results in new challenges for these systematic software performance engineering (SPE) methods. Modeling software running on thousands of cores requires rethinking of existing approaches [4]. While techniques and tools for parallelizing software are evolving [5], novel methods and tools need to be created to assist software architects in designing systems that can exploit the capabilities for parallel execution but do not overburden software developers during implementation [6].

This article summarizes the results of a Dagstuhl Workshop on Multicore Software Performance Engineering held in early 2012. More than 20 researchers and practitioners from different areas, such as software auto-tuning, compiler construction, performance modeling, benchmarking, etc. discussed open research challenges and sketched a corresponding research roadmap.

2. SOFTWARE PERFORMANCE ENGINEERING

2.1 Foundations

Connie Smith introduced 'Software Performance Engineering' (SPE) in the mid 90s [2]. SPE defined a new paradigm of considering performance throughout the software development process. The process starts with the definition of proper performance requirements, involves methods of architecture evaluation to identify problems early, and requires regular performance tests during implementation.

Architects that practice SPE use software models (e.g. UML models) plus additional performance related annotations (e.g. UML MARTE) to analyze system performance already in early design phases [7]. Existing tools transform annotated software models into performance models, such as Queuing Networks (QNs), Stochastic Process Algebras (SPAs), or Stochastic Petri Nets (SPNs). Analysis of performance models provides estimates of response times, throughput and resource utilization that allow architects to check whether a software design can fulfill given performance requirements. This approach avoids late redesigns which may become costly or even technically infeasible.

To follow SPE, developers should do regular performance tests to check response times, throughput and resource utilization of their software system under realistic conditions. Despite an efficient and scalable architecture, implementation details strongly affect performance properties of software systems. Constantly keeping an eye on a system's performance is essential to achieve high performance standards [8]. Ideally, performance tests are executed automatically together with regular continuous integration builds of the complete software system.

In the software model and the implementation, software architects and developers can apply different performance

patterns and validate their impact [2] via prediction or measurement. For example, they can check how the introduction of a cache improves the overall system throughput. Additionally, software architects and developers can detect and remove performance anti-patterns. For example, they may identify an overloaded database lock and investigate the impact of a different locking strategy.

2.2 State of SPE Research

Since the introduction of SPE, software tools supporting both modeling and measurement have evolved. For modeling and analysis, these are for example JMT[1] (for QNs), PEPA[2] (for SPAs) or QPME[3] (for SPNs). Additionally, integrated modeling and analysis environments, such as the Palladio Bench[4], exist. Many of these tools' providers have reported successful applications in prototypical industrial settings. For performance measurements on early prototypes or complete systems, there are for example automated load drivers, benchmarking tools, and monitoring frameworks (e.g. SoPeCo[5], Kieker[6]).

SPE, however, never focused on multicore software development explicitly [3]. As a consequence, many SPE approaches fail to produce accurate performance analysis results for modern multi-core systems. There are two reasons for this. First, software designs are changing and use more fine-grained parallelism that cannot be expressed in most SPE approaches. Second, often the bottleneck in multicore systems is not the processor but other shared resources such as caches or memory buses. This creates a need for new multicore models and measurement techniques. At the same time, the ability to use a large number of cores may be an opportunity for SPE tools and pave the way for more sophisticated solution techniques and simulations that were formerly deemed infeasible.

3. OVERVIEW OF MULTICORE SPE

The new field of 'Multicore Software Performance Engineering' (MCSPE) must enhance the SPE methodology to support and exploit the advent of systems that operate on processors with a large number of cores. One way to structure the issues faced by MCSPE is to analyze the different layers in a modern system, looking at how their performance is affected by the advent of multicore and manycore processors and what layer-specific techniques exist to deal with the issues. Fig. 1 depicts a simplified hierarchy of system layers, the following subsections look at the relevant performance issues in each layer, also listing the magnitude of observed effects and the existing modeling approaches.

3.1 Hardware Layer

Starting from the bottom, the hardware layer is concerned with hardware features such as the processor throughput, the configuration of the processor cores, the architecture of the memory subsystem, or the device configuration. In the hardware layer, major MCSPE challenges stem from the fact that the individual cores do not execute in isolation, but instead influence each other through multiple shared resources.

With hardware thread support, one resource shared among threads are the processor execution units. This makes threads stall each other depending on the particular classes of executed instructions. Early work on thread scheduling has shown the impact on instruction throughput to be as high as 25% [9]. The effect has been modeled probabilistically in [10] and [11], where the authors predict thread instruction throughput from detailed thread behavior profiles.

Another significant performance effect is due to the sharing of various elements of the memory subsystem. Work on process scheduling [12] demonstrates over 50% impact on execution time and suggests complex interactions on shared caches, prefetchers, controllers and buses are to blame. These issues are addressed by models constructed for process scheduling purposes, which help distinguish better or worse schedules, but do not address absolute modeling accuracy. More accurate models for particular memory subsystem elements may exist, such as [13] or [14] for cache sharing, but these models again require detailed thread behavior profiles.

Yet another source of performance effects emphasized with MCSPE is the adaptive power management, which can change the speed or even power up and power down some cores depending on software demand and thermal conditions. Such management is likely to be an essential component of future chips [15] with significant performance impact – different combinations of management strategies and thermal margins were shown to change application throughput by factor of as much as four in [16]. Existing models of processor behavior that include thermal considerations rely on description of processor architecture and knowledge of how the executing software exercises the processor units [17].

The outlined issues become especially pronounced when systems with many cores are considered [4, 18]. Manufacturing general purpose processors with hundreds or thousands of cores may be feasible soon, and specialized manycore processors already exist in graphic processing units (GPUs). For example, the latest AMD graphic accelerator boards sport 32 compute units (CUs) totaling 2048 arithmetic units (ALUs). The computational power offered by these GPUs is already used, for instance, in complex physical simulations. Such computations are necessarily tailored to the complex heterogeneous architectures of the GPUs, especially where the data location constraints of the memory subsystem are concerned. The intricacies of such architectures are in fact still being investigated [19].

For MCSPE, the distribution of the work on the cores, the amount of communication resulting from this distribution, and the access patterns on shared resources have to be captured in the performance prediction models. System performance becomes a result of complex interactions through shared resources, which – although often well known – are currently either not accurately modeled, or modeled only in isolation and by models that require too detailed input information. Combinations of existing detailed models result in huge state spaces that are difficult to solve. New abstractions for representing large sets of cores, with good performance prediction accuracy, need to be researched [20].

3.2 Operating System & Virtualization Layer

On the operating system layer, the impact of process

[1] http://jmt.sourceforge.net/
[2] http://www.dcs.ed.ac.uk/pepa/tools/plugin/
[3] http://descartes.ipd.kit.edu/projects/qpme/
[4] http://www.palladio-simulator.com/
[5] http://www.sopeco.org/
[6] http://kieker-monitoring.net/

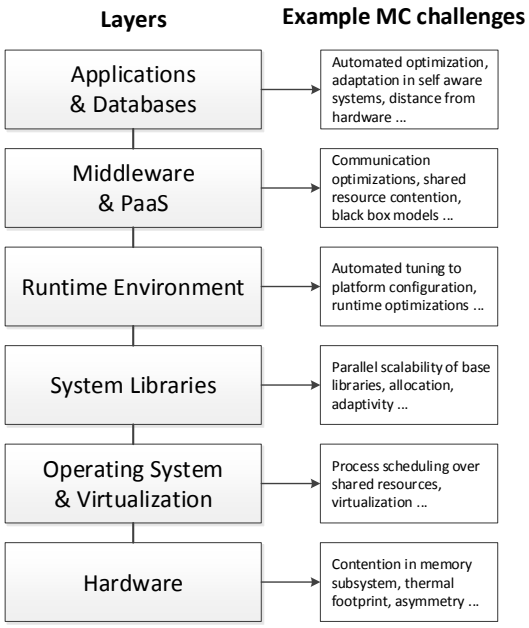

Figure 1: System layers and challenges related to multicore processors

scheduling on performance is of obvious interest. For general purpose operating system schedulers, the effects of load balancing and process priorities on process execution were investigated in [21] and later modeled in [22]. In addition to allocating execution time, operating system scheduling also has a major impact on how processes interact through shared resources at both hardware and software levels. Emerging schedulers provide mechanisms to optimize such interaction [11, 12], but this optimized scheduling behavior is yet to be captured by performance models, and can in fact disrupt probabilistic models that assume process independence.

An important feature that affects software performance in many modern systems is virtualization. Although the goal is to achieve performance isolation between virtualized systems [23], performance interference between virtual machines is still significant [24]. The model in [24] can predict performance interference between two virtual machines given detailed workload characteristics, however, in practice the interfering workloads may be unknown. Recently, black box models based on the Linear Parameter Varying (LPV) framework were applied to performance modeling and management of services in virtualized hosting environments [25], showing that it is possible to derive independent models for individual cores with acceptable runtime overhead.

Better understanding of multicore and manycore systems brings changes in the management of NUMA memory architectures. Where past operating system support relied mostly on static resource allocation, recent development heads towards automated memory migration [26, 27]. Automated memory migration has been shown to deliver speedup as high as 21% in [28].

The operating systems change to utilize the multicore and manycore systems more efficiently and predictably. For MCSPE, many details on the operating system layer should be hidden from the modeler, however, the complexity of the mechanisms involved requires that they are taken into account. Ideally, a casual modeler would simply plug models of operating system abstractions into more high level application models to make fast predictions, whereas a more demanding modeler would require facilities to tune the abstraction model features.

3.3 System Libraries Layer

The system libraries layer comprises generic functionality, such as file handling, memory management, sorting, and graphics routines. Many such libraries need to be optimized for multicore and manycore systems – attention is paid for example to memory allocation, which is an essential feature provided by system libraries to most applications. Work on memory allocation techniques that scale on multicore and manycore systems [29, 30, 31, 32] demonstrates significant performance impact. It has been reported to change the performance of a particular server benchmark by a factor of five [31], or even to cut by half the execution time of an application that itself rarely allocates memory [32]. Interestingly, where early research treats poor multicore and manycore scalability as an allocator design problem that can be remedied to achieve close to perfect scalability, more recent work suggests there may be an inherent trade off between achievable throughput, latency, and space efficiency [32].

Whether a design problem or an inherent property, performance effects related to scalability of system libraries on many cores are significant. Rather than assuming libraries whose performance scales with the number of cores, MCSPE therefore needs to model the performance effects, possibly using measurement-based models that might offer better cost-accuracy trade off than manually constructed models.

Certain libraries may also offer a degree of adaptivity when executing on multicore or manycore systems. This may consist of common autotuning, such as done by the Fastest Fourier Transform in the West (FFTW) library[7], or more complex decisions that trade e.g. response time for encoding quality in graphics processing. Since these functions are common in a large class of systems, MCSPE must again aim at providing standard model libraries for this layer, which so far do not exist.

For reasons including lack of operating system support, system libraries are also used to access specialized parallel hardware, especially GPUs. Sharing such hardware among applications and scheduling combinations of workloads has been shown to provide performance improvements over sequential workload execution [33]. MCSPE would need to treat such sharing as it does other workload scheduling mechanisms.

3.4 Runtime Environment Layer

Modern systems often feature a runtime environment layer, such as the Java Virtual Machine or the Common Language Runtime. This layer necessarily performs numerous decisions that affect multicore performance, such as sizing thread pools, implementing allocation policies or optimizing synchronization patterns. Facing the complex and diverse nature of the runtime environments, the developer community adopts mostly heuristic rather than systematic

[7] http://www.fftw.org/

approaches to performance tuning [34]. As far as MCSPE is concerned, distilling and capturing this heuristic experience may turn out to be more efficient than attempting to create a comprehensive performance model.

Particularly notable features of this layer are automatic memory management and just-in-time compilation. Automatic memory management is known to have significant performance impact especially in low memory conditions [35], however, models of garbage collection performance are rare [36]. Furthermore, it is difficult to determine actual memory consumption, itself an essential factor to garbage collection performance models [37]. The compiler construction research is currently aiming at techniques to automatically improve software for multicore environments, for example by parallelizing loop executions [38]. Once such techniques are commonly employed by compilers, MCSPE models will need to take thus gained speedup into account, e.g. by also introducing parallel actions in the system behavior model. Ideally, this can be realized by model transformations on the prediction model.

3.5 Middleware & PaaS Layer

The communication services provided by middleware or Platform-as-a-Service layer can take advantage of multicore and manycore systems, where application message routing can be optimized by tailoring it to the system architecture [39]. Existing SPE models do not yet reflect such optimizations [40].

A recent trend in SPE is the use of models at runtime [41, 25]. This is especially useful if the operational conditions of the software, for example the usage profile or the hardware environment, change at runtime. Models are constructed on demand to reflect the runtime conditions and then analyzed at runtime to prepare dynamic system reconfiguration. An example runtime optimization at this layer is the autoscaling feature of cloud service offerings [42].

MCSPE should provide guidelines for modeling important middleware parameters and provide benchmarks which specifically test middleware products for their multicore scalability. Middleware performance has been addressed in SPE by measurement-based performance exploration methods, e.g. [43], and by constructing model libraries, e.g. [44]. None of these provide explicit multicore or manycore support, adding such support would likely inflate the model parameter space and complicate the model calibration phase.

3.6 Application Layer

The highest system layer comprises software applications and databases. Databases are special kinds of software applications that traditionally have been subject to performance optimization. Query optimization, indexing, caching, partitioning and many other techniques are employed. These techniques aim at dividing the work most efficiently among the available cores. To this end, classical algorithms are revised to support multi-/manycore environments more effectively. For example, Horikawa [45] devised an approach to identify scalability bottlenecks in database management systems and demonstrated the approach on a 16-core system increasing the maximum achievable throughput by 60 percent. Unfortunately, there is still a lack of good database performance models that reflect the number of processor cores and do not suffer from hard assumptions [46].

For generic software applications, Tarvo and Reiss [47] were able to derive accurate performance models for multi-threaded programs. They used the models to determine configurations of the analyzed programs that allowed to best exploit the available multi-core hardware. The extension of this work to automate building performance models for arbitrary programs is planned.

SPE optimization methods have classically focused on the system design time assuming an abstract, constant system model [48]. For example, Li et al. [49] devised performance and cost models for SAP Enterprise Resource[8] planning systems. Their models can be used in multi-objective optimization to find an optimal tradeoff between performance and costs. de Gooijer [50] et al. used evolutionary algorithms to determine the optimal number of cores with respect to performance and costs for an industrial web-based architecture.

Implementation time optimization has been carried out by software developers manually by search for bottlenecks in the code and then tuning the performance [3]. Developers are assisted in implementation time optimization by tools, such as Intel's Parallel Studio[9], but they are typically not integrated with higher level, architectural performance models [5]. Sometimes, the discovered bottlenecks may be out of developer control due to distance from hardware.

3.7 Summary

Multicore technologies have a complex impact on each described layer. For some layers, there are performance models of decent accuracy, which, however, are often themselves complex, making them difficult to solve. For other layers or particular techniques used on these layers, there are hardly any accurate performance models. A direct approach to whole system modeling, where the individual models would be combined, is therefore not likely to succeed. Some constituent models do not exist yet, and even if they did, it would be hard to maintain, solve, and interpret their combination efficiently.

4. CHALLENGES AND RESEARCH DIRECTIONS

In this section, we focus on selected challenges in MCSPE which raised special interest during the Dagstuhl workshop this article is based on. For each category, we first discuss the challenges and then suggested research directions. We have combined the research suggestions into a MCSPE research roadmap in Fig. 2. Each of the remaining subsections describes one lane of Fig. 2. For each research direction, we also estimate how difficult it will be to address the highlighted challenges based on the existing research results.

4.1 MCSPE Modelling

4.1.1 Challenges

As elaborated in Section 3, multicore and manycore has a profound impact on different system layers. The performance influences and interdependencies in a multicore world are much more complex than in a single-core environment. For example, memory accesses and caching are often safely disregarded in today's modeling notations, but they gain in importance for multicore systems.

[8]http://www54.sap.com/pc/bp/erp.html

[9]http://software.intel.com/en-us/intel-parallel-studio-xe

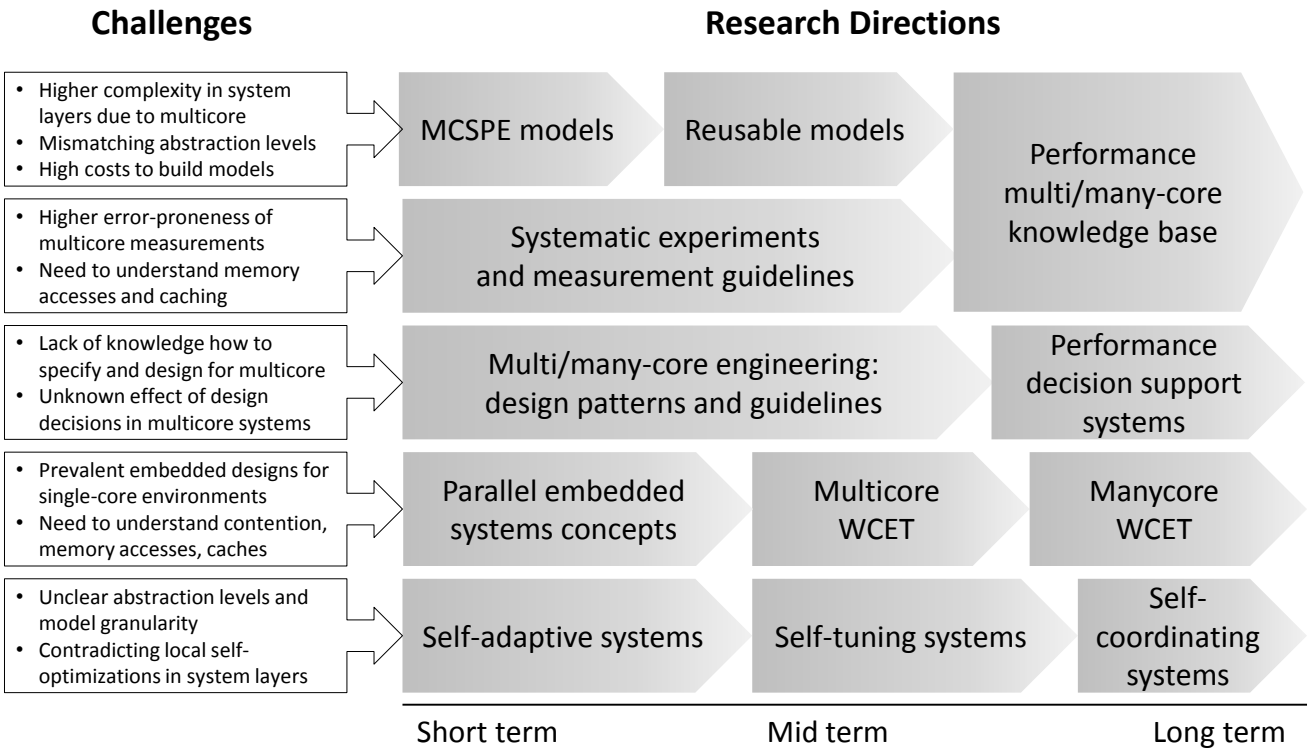

Figure 2: A potential roadmap for multicore software performance engineering research

Many modeling approaches tackle the multicore trend on a single, isolated system level, disregarding other levels. For example, detailed hardware models may exist for certain multicore processors, but they are difficult to combine with the scheduler models on the operating system layers or the database models on the application layer. There is a mismatch of abstraction levels that complicates building valid predictive models combining different layers.

In general, it is still expensive to build predictive performance models [50], and this is even more true for multicore models, which are more complicated than classical single core performance models. Besides the need for well-educated and experienced performance experts, who are able to find suitable abstractions and have a feeling for the important performance issues in a system, it is also necessary to reduce the overheads for manually creating models based on measurements.

4.1.2 Research Directions

To tackle these challenges several steps are necessary (Fig. 2, top lane). In step 1, novel MCSPE models need to be researched. For example, it is conceivable to integrate higher-level concurrency patterns, such as map/reduce, into todays models to avoid the need to model parallelization using low-level abstractions, such as control flow forks and locking of passive resources [51, 52]. For manycore processors, it might be necessary to find different predictive abstractions as the complexity of such technologies is likely overwhelming to be modeled with classical abstractions.

Future SPE models must be capable of valid predictions in multi- and manycore scenarios incorporating the effects from the different system layers. There is a need to integrate models and avoid the mismatch of abstraction levels. There also might be multiple models provided with single software components, but on different granularity levels (e.g., one for scenarios in the range of milliseconds and one for scenarios in the range of seconds). Software architects could choose an adequate model depending on the context, e.g., early/late design time or short/long performance scenario.

In step 2, MCSPE models must be made reusable across products, companies, and application domains. This would address an important roadblock for broader adoption of SPE practices in industry. Ideally, MCSPE models must be cost-effective to create even by non-performance experts. Similar to the SPE software execution models, they should be written in a notation familiar to the developer or architect (e.g. UML).

Ultimately, in step 3, a multicore/manycore performance knowledge base could evolve [3], where performance experts, software architects, and developers share models, measurements, experience, and pitfalls to avoid specifically in multicore settings. This could support an engineering approach towards performance of software systems and overcome todays error-prone craftsmanship approaches. Such knowledge bases could reside inside companies, between collaborating organizations, or even specific technical domains (e.g., enterprise Java, real-time embedded) and application domains (e.g., banking, 3D graphics).

We estimate the difficulty of the modeling research directions to be medium to high. Some initial models, e.g., for multicore scheduling or database systems, already exist. It

is conceivable that such effort will be extended and linked with other system model layers. However, it is still hard to find good abstractions across all system layers that respect the complex interdependencies.

4.2 MCSPE Measurements

4.2.1 Challenges

Measuring, comparing, and evaluating software performance is a challenging and error-prone task. Experiments that fail to address the complex interactions in multicore and manycore environments have been shown to deliver potentially misleading results [53], and while some techniques emerge to address these issues, they are still too cumbersome to be deployed in production environments [54].

Making correct performance observations is but one challenge of measurements in multicore and manycore systems. When the performance impact of workload interaction is to be modeled, various characteristics of the workload need to be measured – for example, to extrapolate how an application will perform on a platform with a different number of cores, it may be important to understand its instruction mix composition or its memory access pattern. The data from such measurements can be used to calibrate models that allow assessing the performance on a different platform, another usage profile, or more data. Current measurement techniques still wrestle with the issues of setup costs and overhead disruptions when collecting such data.

Yet another set of challenges to measurement arises in the cloud environments, where novel metrics are adopted to characterize elasticity, and where the closed nature of the platform and the presence of competing workloads further complicates measurements [55].

Measurements also need to play a role in discovering and characterizing new interactions. Clearly, not all performance relevant interactions in multicore and manycore systems can be anticipated, and it is up to explorative measurements, possibly coupled with expected behavior models, to discover when and what interactions occur.

4.2.2 Research Directions

Following the roadmap sketch in Fig. 2, step 1 calls for methods and tools that use systematic experiments and measurements to help developers and architects understand the performance properties of their applications in multicore and manycore settings. They can systematically search for performance problems or identify parts in their application where parallelization yields the largest performance improvement. There is a need to define performance measurements in a goal-driven manner, focusing on specific multicore aspects to answer specific questions.

Portable MCSPE models require measurements to populate the model with parameters reflecting particular concurrent usage profiles on particular platforms, collected for example by systematically exploring parameter spaces or by relying on measurements on multiple reference platforms. Besides documented systematic experiment setups, multicore measurement guidelines can evolve. Best practices and common mistakes in multicore performance measurements need to be communicated. Knowledge from this work can contribute to the multicore/manycore performance knowledge base described earlier.

Judging by the gradual emergence of sophisticated measurement tools, the measurement research directions appears relatively less difficult – the challenge lies not in the research topics themselves, but in the need to accompany the research with robust tool development that will provide shared experimental environments. The diversity of platforms where measurements need to be taken, together with the necessarily tight coupling between the platforms and the tools, can make this a significant effort.

4.3 MCSPE Design Support

4.3.1 Challenges

Multi- and manycore processors affect all phases of the software development process and also the way software is developed [5]. Performance and scalability *requirements* become more important to guide design, development and testing. Especially the design of software architectures has to take into account effects of data volume and load to support partitioning of data, definition of independent tasks running on individual cores, and to define proper synchronization points. It is often hard for developers and architects to identify those parts of an application where concurrency yields the largest benefit. Future designs and implementations must be created with more emphasis on concurrency awareness.

The effect of design decisions on the overall performance of an application is difficult to anticipate given multicore technologies. Parallelizing existing software is difficult and not a matter of local changes, instead it requires a structured approach [56]. There is a lack of proper tool support for these tasks.

4.3.2 Research Directions

Fig. 2 shows two steps for future research. In step 1, both developers and architects require adequate support to increase their performance awareness due to increased concurrency and to assess the trade-off between complexity and performance. Existing multi-/manycore design patterns and design guidelines should be tested and need to be tailored for domain specific support [51]. For example, design patterns such as Map-Reduce [57] can help developers to systematically parallelize data processing thus leading to increased concurrency. These guidelines will support architects and developers alike in building scalable architectures that make effective use of the available resources.

A subtle but important impact of distributing performance relevant information through patterns and guidelines is the improved communication between the authors and the users of the documented artifacts (e.g. libraries). This will help remedy the current situation, where the performance relevant information often has to be derived by the users through experimental measurements and reverse engineering.

Eventually, in step 2, a performance decision support system could arise that software architects use during requirements engineering, architecture design, and implementation. Developers need to be informed about the effect of their decisions on the overall performance of their application. For example, feedback on the speedup achieved by their code on different processors [58] can guide decisions and focus development efforts. Architecture trade-offs between multicore performance and other quality attributes, such as reliability, maintainability, and security need to be made visible by

decision support system in order to guide well-rationalized design decisions.

We estimate the difficulty of the design support research directions to be low, because the initial work is ongoing and first books and pattern collections for multi-core design already exist. Based on such patterns, models enriched with measurements are currently investigated, leading to systematic design decision support in the future. However, the complexity of the various interactions among cores in a many-core system still leads to gaps between the models and realistic measurements.

4.4 MCSPE in Embedded/Real-time Systems

4.4.1 Challenges

Multi- and manycore systems will also change the way embedded systems are designed [59, 60, 61]. Due to the cost efficient production of multi-core CPUs, their use in embedded systems will increase. Today, many embedded system are based on single-core architectures, and their migration to multi-core architectures will require a pervasive paradigm shift in design, implementation, engineering, and testing. In parallel systems which run on multicore CPUs, this requires new analysis methods that cope with sources of uncertainty introduced through contention over processor caches or memory buses.

Deterministic functional and timing behavior is often crucial for realtime systems and there is a need to determine worst-case execution times (WCET) for the certification of safety-critical systems. A significant challenge exists in reworking WCET estimation technology, as much of its theoretical foundation assumes single-core systems and both the formal methods and current attempts at WCET for multicore have not yet been shown to scale to realistic examples (e.g., [62]).

4.4.2 Research Directions

Based on the existing and on-going work for engineering general purpose multicore systems, concepts for parallel embedded and real-time systems will have to be developed. The concepts will have to take into account domain specific non-functional requirements such as safety and be developed for different (older) languages and embedded/real-time software engineering tools.

There is a big potential and need for research work for tackling the worst-case execution time (WCET) problem [63] for embedded and/or real-time systems in the multicore era. Scheduling algorithms, worst-case executing time predictions, or soft-real time analyses need to take the new uncertainty due to multicore hardware into account to gain reliable results. As shown in Fig. 2, in a first step multicore WCET estimation need to be tackled, whereas future approaches may also develop approaches for manycore WCET estimation.

Software engineering concepts for parallel embedded systems will be relatively straightforward to construct based on the experience gained with general purpose systems. WCET estimates for multi- and manycore systems on the other hand seem to be a harder problem. The resource contention in these systems is only one of many new uncertainties that these systems introduce to a domain in which CPU caches often were disabled to trade performance for predictability. As a consequence, the difficulty of the embedded real-time challenges appear to be high. Both models as well as tools need to be evolved for full MCSPE support.

4.5 MCSPE in Self-Adapting Systems

4.5.1 Challenges

One particular approach to the development of multi-core software systems and the migration of existing legacy code to multi-core systems is the use of self-adaptation [64]. Examples for this include automated allocation of processes on cores, auto-tuning of data partitioning and thread use, as well as an adaptive middleware and application layer. Automated allocation of processes decides on the use of single cores operated at higher frequencies with increased heat production versus the use of parallel processing on multiple cores at lower temperatures. Auto-tuners inspect executing programs with parallel parts and try to adapt the number of parallel threads to the system's number of cores, cache structure or bus layout. Adaptive middleware and application layers try to allocate at runtime an optimal number of cores to dynamic workloads.

Several issues arise for MCSPE. First, it is necessary to answer the question which of the details of each of the self-adaptation layers need to be included into the performance analysis model to get sufficient accuracy. Second, the overall system may turn into a stack of self-adaptive layers. In current approaches, each layer tries to self-optimize independent of other layers. However, this may result in one layer's optimization to negatively impact the optimization on a different layer. Here, a federated self-adaption in which each layer's adaptation coordinated with other layers could improve the overall, global result. In this setting, MCSPE could help to give insights into the data and models needed to achieve this federated self-adaptation.

4.5.2 Research Directions

Three steps could be envisioned to structure this research (Fig. 2, bottom lane). Self-adaptive systems need to be supported by MCSPE through modeling and measurement techniques that account for the performance effects. The next step is to use the techniques and their performance predictions and evolve them into self-tuning systems that are capable of automatically learning the impacts of optimizations on different abstraction levels and eventually optimizing overall system performance by self-coordination in the last step.

We estimate the difficulty of the self-adaptation research directions to be high. The reason is that new models for MCSPE need to be created, which also demands new analysis techniques. As self-adaption often leads to transient phases in the behavior of software systems (i.e., phases in which a system adapts), the demand for transient analyses arises. This requirement conflicts with the wide-spread use of steady-state average-response-time analyses dominating MCSPE today. Hence, new analysis techniques need to be developed and tested, making full support for self-adaptation in MCSPE difficult.

5. CONCLUSIONS

Classical software performance engineering from the single core era has brought forward methods, tools, and development guides. All these techniques help to avoid performance problems, thereby saving cost and preserving company rep-

utation. Now multicore processors have become a pervasive technology and MCSPE is a major concern. This article has outlined the challenges that came with the advent of multicore and structured the field of multicore software performance engineering. From the identified challenges it becomes clear that many classical SPE techniques need to evolve to become applicable in MCSPE. To support this evolution, our article points out open issues which require further research and proposes an MCSPE research roadmap.

Acknowledgements

We are grateful to the team at Schloss Dagstuhl for hosting the seminar that led to this paper. We thank all participants for their contributions and the engaging discussions.

6. REFERENCES

[1] http://highscalability.com/, last checked 2013-01-10.

[2] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[3] M. Woodside, G. Franks, and D. C. Petriu, "The Future of Software Performance Engineering," in *Proc. Future of Software Engineering (FOSE'07)*. IEEE Computer Society, 2007, pp. 171–187.

[4] W. Hwu, S. Ryoo, S.-Z. Ueng, J. Kelm, I. Gelado, S. Stone, R. Kidd, S. Baghsorkhi, A. Mahesri, S. Tsao, N. Navarro, S. Lumetta, M. Frank, and S. Patel, "Implicitly parallel programming models for thousand-core microprocessors," in *Proc. 44th ACM/IEEE Design Automation Conference (DAC '07)*, june 2007, pp. 754 –759.

[5] H. Vandierendonck and T. Mens, "Techniques and tools for parallelizing software," *IEEE Softw.*, vol. 29, no. 2, pp. 22–25, 2012.

[6] C. A. Schaefer, V. Pankratius, and W. F. Tichy, "Engineering parallel applications with tunable architectures," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. ACM, 2010, pp. 405–414.

[7] D. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from uml designs," *Software and Systems Modeling*, vol. 6, no. 2, pp. 163–184, 2007.

[8] http://velocityconf.com/, last checked 2013-01-10.

[9] A. Snavely and D. M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor," in *Proceedings of ASPLOS 2000*, 2000.

[10] X. E. Chen and T. M. Aamodt, "A first-order fine-grained multithreaded throughput model," in *Proceedings of HPCA 2009*. IEEE, 2009, pp. 329–340.

[11] S. Eyerman and L. Eeckhout, "Probabilistic Job Symbiosis Modeling for SMT Processor Scheduling," 2010.

[12] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)*. ACM, 2010, pp. 129–142.

[13] D. Eklov, D. Black-Schaffer, and E. Hagersten, "Fast Modeling of Shared Caches in Multicore Systems." ACM, 2011, pp. 147–157.

[14] V. Babka, P. Libič, T. Martinec, and P. Tůma, "On the accuracy of cache sharing models," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 21–32. [Online]. Available: http://doi.acm.org/10.1145/2188286.2188294

[15] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Commun. ACM*, vol. 56, no. 2, pp. 93–102, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408797

[16] J. Srinivasan and S. V. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications," in *Proceedings of ICS 2003*. ACM, 2003.

[17] Y. Li, D. Brooks, Z. Hu, and K. Skadron, "Performance, Energy, and Thermal Considerations for SMT and CMP Architectures," in *Proceedings of HPCA 2005*. IEEE, 2005.

[18] A. Vajda, "Debugging and performance analysis of many-core programs," in *Programming Many-Core Chips*. Springer US, 2011, pp. 117–126.

[19] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture through Microbenchmarking," in *Proceedings of ISPASS 2010*, 2010.

[20] V. Babka and P. Tuma, "Can linear approximation improve performance prediction ?" in *Computer Performance Engineering*, ser. Lecture Notes in Computer Science, N. Thomas, Ed., vol. 6977. Springer Berlin Heidelberg, 2011, pp. 250–264.

[21] J. Happe, H. Groenda, and R. H. Reussner, "Performance Evaluation of Scheduling Policies in Symmetric Multiprocessing Environments," in *Proceedings of the 17th IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'09)*, 2009. [Online]. Available: http://sdqweb.ipd.uka.de/publications/pdfs/happe2009b.pdf

[22] J. Happe, H. Groenda, M. Hauck, and R. H. Reussner, "A prediction model for software performance in symmetric multiprocessing environments," in *Proceedings of the 2010 Seventh International Conference on the Quantitative Evaluation of Systems*, ser. QEST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 59–68. [Online]. Available: http://dx.doi.org/10.1109/QEST.2010.15

[23] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proceedings of MIDDLEWARE 2006*. Springer, 2006, pp. 342–362.

[24] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," in *Proceedings of ISPASS 2007*, 2007.

[25] D. Ardagna, M. Tanelli, M. Lovera, and L. Zhang, "Black-box performance models for virtualized web service applications," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 153–164. [Online]. Available:

http://doi.acm.org/10.1145/1712605.1712630

[26] A. Arcangeli, "AutoNUMA Linux Kernel Patch Set," http://lwn.net/Articles/488686.

[27] P. Zijlstra, "SchedNUMA Linux Kernel Patch Set," http://lwn.net/Articles/486850.

[28] J. A. Lorenzo, J. C. Pichel, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, "A Flexible and Dynamic Page Migration Infrastructure based on Hardware Counters," *Journal of Supercomputing*, vol. 65, no. 2, 2013.

[29] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, "Hoard: A Scalable Memory Allocator for Multithreaded Applications," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 117–128, 2000.

[30] M. Michael, "Scalable Lock-Free Dynamic Memory Allocation," in *Proceedings of PLDI 2004*, 2004.

[31] S. Schneider, C. D. Antonopoulos, and D. S. Nikolopoulos, "Scalable Locality-Conscious Multithreaded Memory Allocation," in *Proceedings of ISMM 2006*, 2006.

[32] S. Kahan and P. Konecny, "MAMA: A Memory Allocator for Multithreaded Architectures," in *Proceedings of PPOPP 2006*, 2006.

[33] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar, "Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework," in *Proceedings of HPDC 2011*, 2011.

[34] E. Andreasson, "JVM Performance Optimization Series," http://www.javaworld.com/javaworld/jw-08-2012/120821-jvm-performance-optimization-overview.html.

[35] S. M. Blackburn, P. Cheng, and K. S. McKinley, "Myths and realities: The performance impact of garbage collection," in *Proceedings of SIGMETRICS 2004*. ACM, 2004.

[36] D. Vengerov, "Modeling, analysis and throughput optimization of a generational garbage collector," in *Proceedings of ISMM 2009*, 2009.

[37] G. S. Nick Mitchell, "The causes of bloat, the limits of health," in *Proceedings of OOPSLA 2007*, 2007.

[38] C. Dave and R. Eigenmann, "Automatically tuning parallel and parallelized programs," in *Proceedings of the 22nd international conference on Languages and Compilers for Parallel Computing*, ser. LCPC'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 126–139.

[39] S. Ramos, G. L. Taboada, J. T. Roberto R. Exposito, and R. Doallo, "Design of scalable java communication middleware for multi-core systems," *The Computer Journal*, 2013. [Online]. Available: http://dx.doi.org/10.1093/comjnl/bxs122

[40] K. Sachs, S. Kounev, and A. Buchmann, "Performance Modeling and Analysis of Message-Oriented Event-Driven Systems," *Software and Systems Modeling*, 2012.

[41] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in software architecture," in *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, ser. QoSA '08. Springer, 2008, pp. 1–27.

[42] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proceedings of SOCC 2011*. ACM, 2011, pp. 5:1–5:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038921

[43] Y. Liu, I. Gorton, L. Bass, C. Hoang, and S. Abanmi, "Mems: a method for evaluating middleware architectures," in *Proc. 2nd International Conference on the Quality of Software Architectures (QoSA'06)*, ser. LNCS. Springer, 2006, pp. 9–26.

[44] J. Happe, S. Becker, C. Rathfelder, H. Friedrich, and R. H. Reussner, "Parametric Performance Completions for Model-Driven Performance Prediction," *Performance Evaluation*, vol. 67, no. 8, pp. 694–716, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.peva.2009.07.006

[45] T. Horikawa, "An approach for scalability-bottleneck solution: identification and elimination of scalability bottlenecks in a dbms," in *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/1958746.1958756

[46] R. Osman and W. J. Knottenbelt, "Database system performance evaluation models: A survey," *Perform. Eval.*, vol. 69, no. 10, pp. 471–493, Oct. 2012.

[47] A. Tarvo and S. P. Reiss, "Using computer simulation to predict the performance of multithreaded programs," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/2188286.2188320

[48] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *Proc. 1st Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW'10)*. ACM, January 2010, pp. 105–116.

[49] H. Li, G. Casale, and T. Ellahi, "Sla-driven planning and optimization of enterprise applications," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, ser. WOSP/SIPEW '10. New York, NY, USA: ACM, 2010, pp. 117–128. [Online]. Available: http://doi.acm.org/10.1145/1712605.1712625

[50] T. de Gooijer, A. Jansen, H. Koziolek, and A. Koziolek, "An industrial case study of performance and cost design space exploration," in *Proc. 3rd Int. Conf. on Performance Engineering (ICPE'12)*. ACM, April 2012, pp. 205–216.

[51] J. Zheng and K. E. Harper, "Concurrency design patterns, software quality attributes and their tactics," in *Proceedings of the 3rd International Workshop on Multicore Software Engineering*, ser. IWMSE '10. New York, NY, USA: ACM, 2010, pp. 40–47. [Online]. Available: http://doi.acm.org/10.1145/1808954.1808964

[52] H. Koziolek, "Performance evaluation of component-based software systems: A survey,"

Perform. Eval., vol. 67, no. 8, pp. 634–658, 2010.

[53] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing Wrong Data Without Doing Anything Obviously Wrong," in *Proceedings of ASPLOS 2009*, 2009.

[54] C. Curtsinger and E. D. Berger, "Stabilizer: Statistically Sound Performance Evaluation," in *Proceedings of ASPLOS 2013*, 2013.

[55] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in *Proceedings of CCGRID 2009*. IEEE, 2009, pp. 472–477. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.40

[56] K. E. Harper, J. Zheng, and S. Mahate, "Experiences in initiating concurrency software research efforts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810316

[57] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[58] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: an efficient multithreaded runtime system," in *Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPOPP '95. ACM, 1995, pp. 207–216.

[59] J. Yan and W. Zhang, "WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*. IEEE, 2008, pp. 80–89.

[60] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," in *Proceedings of Embedded Real Time Software and Systems (ERTS'10)*, 2010, pp. 36–42.

[61] M. Oriol, M. Wahler, R. Steiger, S. Stoeter, E. Vardar, H. Koziolek, and A. Kumar, "FASA: a scalable software framework for distributed control systems," in *Proc. 3rd Int. ACM SIGSOFT Symposium on Architecting Critical Systems (ISARCS'12)*. ACM, June 2012, pp. 51–60.

[62] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson, "Towards wcet analysis of multicore architectures using uppaal," in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*. Österreichische Computer Gesellschaft, July 2010, pp. 103–113.

[63] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1347375.1347389

[64] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science, vol. 5525. Springer, 2009.