International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-3 Issue-4, September 2013

UNDERWARE - A Layer for Homogeneous Access to Heterogeneous Multi - Core Processors

Jan Haase

Abstract*— This paper proposes a new way to tackle the problem of using the existing resources of multi-core processors with different types of cores, like PowerPC, Intel, ARM, DSPs, FPGA areas, etc. Applications for these heterogeneous multi-cores are difficult to write, as most programmers and programming tools target one platform only and in many cases single-cores only. The way to accomplish this is by creating a new layer, which is located between the operating system and the underlying heterogeneous hardware. In analogy to the middleware between applications and the OS it is called "underware". The underware layer itself consists of two sublayers: The lower "adapter layer" builds a uniform interface for any underlying processor-core-configuration; above it lies the "scheduling layer", which automatically distributes applications to the corresponding core or cores (if parallel execution is possible) or even to several different idle cores if the application is available in different platform executables. Research on the approach described in this paper is yet in an early stage; the corresponding project "UNDERWARE" just started. Therefore no preliminary results can be given. However the concept is sound and it can therefore be expected to get a first prototype soon.

***IndexTerms*—Multi-core, heterogeneous, middleware, underware**

I. INTRODUCTION

The semiconductor industry supports the trend for increased parallelity by developing more and more processors containing multiple cores. The number of cores increases steadily; processors with 64 or 100 cores can be expected in the near future. Many of the new processors are not homogeneous (i.e. containing identical cores) anymore but contain apart from the usual standard processor cores (like Intel or PowerPC) additional cores like DSPs, GPGPUs, or even FPGA areas where soft cores like the Microblaze from Xilinx can be configured; the processors are thereby heterogeneous. The exploitation of the available resources is difficult in most cases: Many applications are developed for one processor architecture only and sometimes even for one processor only (and thus will support parallel execution only limited or not at all). The reason frequently lies in the complex development of parallel applications (in contrast to sequential applications). Furthermore, many software developers are familiar with only one processor architecture and only one operating system, leading to the fact that applications which truly support several processor architectures are rare. The approach described in this paper will remedy this situation by partly hiding the heterogeneity of different processor cores and pretending to be a homogeneous system of identical cores.

Manuscript published on 30 September 2013.
***Correspondence Author(s)**
Jan Haase, Institute of Computer Technology, Vienna University of Technology, Vienna, Austria.

This is achieved by inserting another logical layer between the cores and the operating system that will pass on the applications to their corresponding cores and furthermore the automatic distribution of application parts (e.g. threads). Similar to the middleware which is a layer between the operating system and the applications connecting several computers to form a cluster and presents a „single system image" to the applications, the proposed layer will present to some extent a „single processor image" and is called „underware". The goal is to be able to execute applications for different processor architectures on a heterogeneous system on the corresponding processor (or core), possibly in parallel using the parallelization information given by the programmer if there are other cores of the same architecture idle. Furthermore, applications are to be automatically distributed among different core architectures for self-organizing load balancing. The programmer therefore does not have to be concerned about on which architecture his application is finally executed. The proof of concept will consist of a prototypical heterogeneous system running several example applications.

II. RELATED WORK

Research in the broader field is performed by a number of institutes and companies throughout the world and is currently highly diversified, often focusing on detail problems of existing micro-architectures. Interesting work originated at Intel Corporation[1] focuses on heterogeneous, but partly overlapping instruction set architectures. Several interesting open research questions are pointed out, mainly about handling performance and instruction-set asymmetry and the difference in scheduling mechanisms [1]. A similar idea was presented in [2] with a focus on reconfigurable hardware. A specific use case for heterogeneous computing of movie decoding is presented in [3]. The basic idea was used in the works of [4] to improve reliability through task recomputation.

The idea of this paper however is to generate as well as make use of existing ideas from the *network and cluster layer* and to transpose them to the chip and system level. Multi-level simulations [5] are a well-covered field today. Two highly interesting projects offering ideas that might lead to synergistic effects are the portable virtual machine (PVM) [6] and the portable multithreaded PVM (PM-PVM) [7]. These projects distribute tasks between network nodes and/or homogeneous cores. The SDVM [8] (see next Section) is yet another project of this kind, providing several interesting approaches; it is the base for the work described in this paper. Other work originated in France deals with heterogeneous multi-core system on chips in conjunction with a lean OS. The authors are focusing on a high level system framework, which provides services as components to the application (developer) [9].

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Important topics focusing on node structures such as adaptive diversified scheduling are discussed in [10]. Other groups are already investigating some efficient architecture independent scheduling mechanism [11] showing promising first results. Further promising work constitutes research in flexible and adaptive heterogeneous multi core architectures [12] as well as in specialized parallel computation models for systems such as these [13]. What can be stated however is that most research in this domain is still in an early phase. The most promising approaches and results will be used partially in the scope of this project to provide further and better answers to open research questions in the field.

A. *The Self distributing virtual machine (SDVM)*

The SDVM [8] works as a middleware for computer clusters and manages the automatic distribution of chunks of application code and data to be executed over the cluster, focusing on execution performance by automatic parallelization. The cluster used can consist of different machines with different processing speeds and/or different operating systems (e.g. different Linux distributions, Windows, MacOS) and/or different instruction set architectures (e.g. Intel-processors, PowerPCs); binary code of code chunks is automatically compiled at run time if needed, thus it enables the user to work on heterogeneous clusters. The SDVM features automatic parallel scheduling based on data flow analysis and some self-x-abilities like

- self-optimization: applications are distributed automatically over the cluster and the order execution determined by automatic distributed, dynamic scheduling
- self-balancing: work is distributed depending on the current work load of the machines participating in the cluster; faster machines get more work, slower machines are less used
- self-configuration: the cluster may grow and shrink at runtime while the applications run on
- self-healing: crashes of single machines are mended by automatic stop and restart of the last check point of the running applications
- self-organization: the application code does not have to be present on all machines, as it is distributed, automatically, too

It is implemented as a daemon running on each machine in the cluster, forming a “site” each. Inside, it consists of several managers dealing with different tasks which are arranged into three internal layers (see Figure 1). A front end forms the user interface and enables the user to start and stop applications, monitor applications and sites, etc.

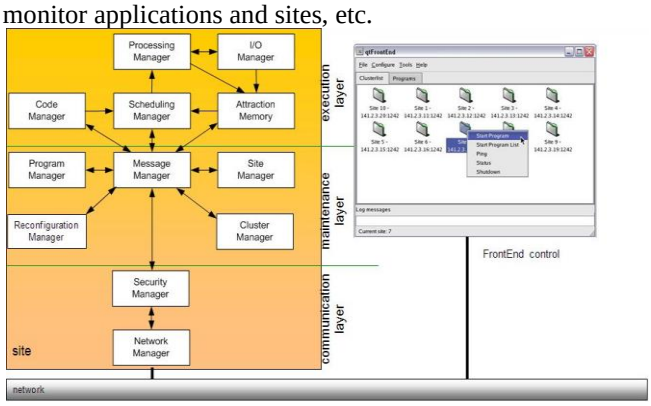

Figure 1. The layer structure and front end of the Self Distributing Virtual Machine (SDVM) [14]

As the SDVM was focusing only computer clusters, but showed, that the underlying concept can be used on even bigger networks (like the internet) or single machines with many processing devices (like multicore architectures), one more step was taken in the direction of embedded systems by implementing the SDVM with Reconfigurability (SDVM^R)[15,16]. In this project, the SDVM concept was taken to a single system in the form of an FPGA with existing processors (hard cores) and runtime-configured processors (soft cores). It was shown that automatic distribution on the single-system-level with multiple processing devices is feasible.

The approach described in this paper can be seen as the next step and will pave the way for the generalization of execution of any application on a system with any combination of processing devices.

III. THE APPROACH

In this Section the underware’s layer approach is presented.

A. *Model overview*

The overall model is shown in Figure 2. As can be seen the lowest level consists of the hardware CPU cores. Above that the new layer is going to be placed. It will consist of two sub parts, one is the adapter layer, which works as an information source for data about what capabilities a certain core offers, which instruction set it supports, what its current utilisation and local and shared memory usage is. The second part is the scheduling layer, which will show itself to the operating system as if it has more or less homogeneous cores to manage, which makes it easier for the OS to do its scheduling. It has to provide the possibility though to enable the OS layer or even the application layer to pin certain processes or threads to a specific core, if it is known a-priori that a certain application part runs most efficient (or only) on a certain core architecture.

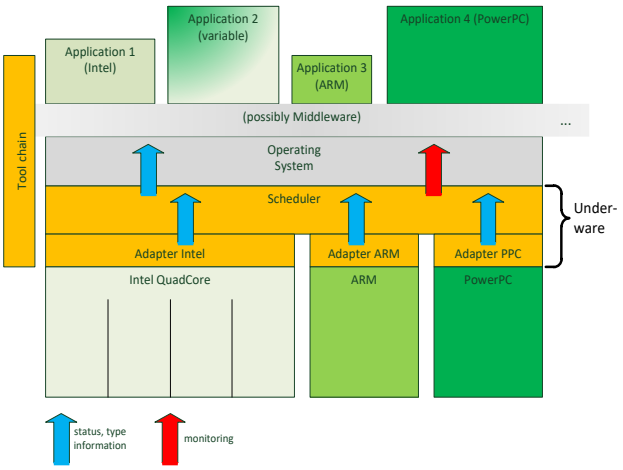

Figure 2. Example of the proposed layer structure (in orange) for a system with Intel Quad Core, ARM, and PowerPC processors. Additionally, a tool chain for easy use is to be implemented.

The overall model is shown in Figure 2. As can be seen the lowest level consists of the hardware CPU cores. Above that the new layer is going to be placed.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

It will consist of two sub parts, one is the adapter layer, which works as an information source for data about what capabilities a certain core offers, which instruction set it supports, what its current utilisation and local and shared memory usage is. The second part is the scheduling layer, which will show itself to the operating system as if it has more or less homogeneous cores to manage, which makes it easier for the OS to do its scheduling. It has to provide the possibility though to enable the OS layer or even the application layer to pin certain processes or threads to a specific core, if it is known a-priori that a certain application part runs most efficient (or only) on a certain core architecture.

The operating system, which is located above, together with possibly some kind of middleware layer for connection between systems (e.g. in a computer cluster), has to provide correctly compiled machine code for the cores and abstractions for the system-call interfaces to the application layer. The OS should be able to do JIT (just in time) (re)compilation of source code parts if the machine code is not yet available for the core architecture the task should be run on. The topmost layer in the model is the application layer. The application programmer should generally not have to care about the architecture the applications are to be executed on, as the lower levels should provide the needed abstractions.

A generalized tool-chain, which should not be bound to a specific core layout, is going to help the developer to handle more easily the complexity of heterogeneous platforms. It is going to span the whole range from the underware (below the OS level) up to the application program interface (API). It will possibly consist of software drivers for the OS, and libraries and services for the application developer and also of the hardware drivers built into the Underware.

B. Model structure

The structure of the model as well as scheduling behaviour is shown in Figure 3. The parts to be developed are depicted in orange, namely the *adapter layer* for each processor architecture, the *scheduling layer*, and the accompanying software development tool chain. The following Sections describe these aspects (alongside important others) in greater detail.

1) Adaption Layer

The adaption layer is a hardware centric layer which provides a standardized API to the scheduling layer for each underlying processor architecture. In this, it can be compared to a software driver for hardware components, which offers a similar functionality to the operating system.

It provides vital information to the scheduling layer and the OS. It provides information about the instruction set architecture, the utilization of the cores, the number of "subcores" (e.g. 4 in a participating quad core processor) and information about local memory. It might be partly responsible for data conversions in inter core communications.

Due to its very processor-optimized functionality, it might be conceivable for future processor designs to build the adaption layer as a standardized hardware module inside (homogeneous or heterogeneous) multi- or many-core processors.

2) Scheduling Layer

This layer is responsible to give the OS an image of a system with homogeneous cores, while still allowing core/architecture selection and pinning. It is responsible for the clever distribution of the workload to the various kinds of cores. It also ensures that the correct binary executable of applications is sent to the corresponding core. If this binary executable does not yet exist, it requests a (re)compilation. The scheduling algorithm might even be self-learning in the sense that it uses profiling data for its scheduling decisions. The scheduling decisions take into account the information it receives from the adaption layer, particularly current utilization, core types, special abilities, etc.

In future developments, it might be possible to implement portions of the scheduling layer as hardware components of the system, e.g. the scheduling algorithms or the management parts.

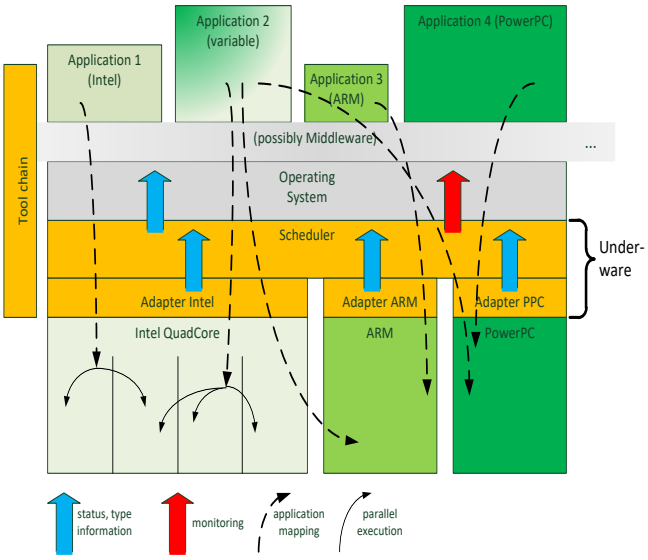

Figure 3. Example of the proposed model structure for a system with Intel Quad Core, ARM, and PowerPC processors. Applications are mapped to their corresponding processor (shown in equal colors) or to several processors, if possible (see application 2). Multi core processors are used accordingly (see the dotted lines).

3) OS Integration

This part consists mostly of software drivers for the operating system. For the prototype, the focus lies on Linux 2.6.x, since the source code is open and modifications can easily be done. The kernel has to cooperate with the scheduling layer and has to request the correct object code for the core a task is going to be scheduled to run on. For instance, it will need to copy with different calling conventions of the core architectures and it will have to abstract the system call interface to support cross architecture kernel function invocation. It will be a mixture between kernel level changes and user space library modifications. Some of the concepts created by the QEMU-system emulator virtual machines might be useful in this part of the work.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

4) Development Tool-Chain

This part targets tools for the developer to easily develop distributed/parallel applications for heterogeneous multi core architectures and possibly even to develop applications for clusters of those systems. It will consist of libraries and system services and some modifications to a C compiler (either GCC or CLANG), to support architecture selection/pinning for certain code paths via code annotations in the source code of the "user" applications.

5) Software Design: Methodological Aspects

Multithreading an application—the task of converting sequential program code into multi-threaded code in order to make use of multiple processor cores in shared-memory multiprocessor systems—to improve performance is more often than not a non-trivial and therefore time-consuming activity, especially in heterogeneous architectures. Automatic parallelization usually refers to generating a multi-threaded version of a sequential program without further user interaction. Today's parallelizing compilers rely on more or less simple control flow and data flow analysis tools as well as performance metrics. A loop statement is analyzed regarding data dependencies between iterations and performance estimates of parallelization. If required conditions are met, multiple iterations of the loop can be executed in parallel.

While automatic approaches can help reduce software development time drastically in some cases, they are highly disputed by a broad community of developers, arguing that von-Neumann-inspired languages often lack major prerequisites of large-scale automatic parallelization like implicit thread-safety, performance evaluation capabilities and standardized declarative patterns . All this holds true for both homogeneous and heterogeneous micro-architectures. In heterogeneous architectures however complexity of software parallelism multiplies with the number of different core architectures available to the software developer.

It is not the aim of the underware to provide a solution to the problem of automatic parallelization, but to reduce the inherit complexity of developing optimized multi-threaded software for heterogeneous micro-architectures to a level comparable—in the best case equal—with that of homogeneous micro-architectures. This provides software developers with the capabilities of making use both of already existing software design methodologies for multi-threaded applications in general as well as of the potential results of current and on-going research in the field of automatic parallelization on homogeneous architectures.

Therefore the aim of the project is to take steps towards providing the software developer community with a well-structured and standardized software design methodology utilizing a defined tool-chain that lets developers make use of task parallelization in a general—probably partly automated—way regardless of the underlying micro-architecture (abstracting away architectural details) while still gaining efficiency (e.g. performance, power consumption, heat distribution etc.), but offers still all means necessary to develop highly-optimized software (e.g. the developer can make use of dedicated cores in a meaningful manner in specified heterogeneous target architectures).

IV. CONCLUSION

The basic principles and concepts of the new "underware" approach for easy homogeneous access to all resources of a heterogeneous multi-core system was presented. Due to the early stage of the project no results are given yet but can be expected soon.

REFERENCES

1. Tong Li; Brett, P.; Knauerhase, R.; Koufaty, D.; Reddy, D.; Hahn, S.; , "Operating system support for overlapping-ISA heterogeneous multi-core architectures," High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, pp.1-12, 9-14 Jan. 2010
2. Werner, S.; Oey, O.; Gohringer, D.; Hubner, M.; Becker, J., "Virtualized on-chip distributed computing for heterogeneous reconfigurable multi-core systems," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012 , vol., no., pp.280,283, 12-16 March 2012
3. Yu-Sheng Lu; Chin-Feng Lai; Yueh-Min Huang, "Parallelization of DVFS-enabled H.264/AVC Decoder on Heterogeneous Multi-core Platform," Computer, Consumer and Control (IS3C), 2012 International Symposium on , vol., no., pp.157,160, 4-6 June 2012
4. Nimer, B.; Koc, H., "Improving reliability through task recomputation in heterogeneous multi-core embedded systems," Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on , vol., no., pp.72,77, 9-11 May 2013
5. Jan Haase, Mario Lang, Christoph Grimm, "Mixed-Level Simulations of Wireless Sensor Networks", Proceedings of Forum on Specification & Design Languages (FDL), IET Digital Library, 2010
6. Pernice, M.; , "PVM: Parallel Virtual Machine-A User's Guide and Tutorial for Networked Parallel Computing [Book Rev," Parallel & Distributed Technology: Systems & Applications, IEEE, vol.4, no.1, pp.84, Spring 1996
7. Santos, C.M.P.; Ande, J.S.; , "PM-PVM: A portable multithreaded PVM", Parallel and Distributed Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings , pp.191-195, 12-16 Apr 1999
8. J.Haase, F.Eschmann, B.Klauer, and K.Waldschmidt, "The SDVM: A Self Distributing Virtual Machine for computer clusters", In Organic and Pervasive Computing -- ARCS 2004, International Conference on Architecture of Computing Systems, volume 2981 of Lecture Notes in Computer Science. Springer Verlag, 2004
9. Guerin, X.; Petrot, F.; , "A System Framework for the Design of Embedded Software Targeting Heterogeneous Multi-core SoCs," Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on , pp.153-160, 7-9 July 2009
10. Venkateswaran, N.; Saravanan, K.P.; Nachiappan, N.C.; Vasudevan, A.; Subramaniam, B.; Mukundarajan, R.; , "Custom Built Heterogeneous Multi-core Architectures (CUBEMACH): Breaking the conventions," Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on , pp.1-15, 19-23 April 2010
11. Jian Chen; John, L.K.; , "Efficient program scheduling for heterogeneous multi-core processors," Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE , pp.927-930, 26-31 July 2009
12. Pericas, M.; Cristal, A.; Cazorla, F.J.; Gonzalez, R.; Jimenez, D.A.; Valero, M.; , "A Flexible Heterogeneous Multi-Core Architecture," Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on , pp.13-24, 15-19 Sept. 2007
13. Li, Liang; Zhang, Xingjun; Feng, Jinghua; Dong, Xiaoshe; , "mPlogP: A Parallel Computation Model for Heterogeneous Multi-core Computer," Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pp.679-684, 17-20 May 2010

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

14. J.Haase, F.Eschmann, and K.Waldschmidt, "The SDVM - an Approach for Future Adaptive Computer Clusters", In 10th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 05), Denver, Colorado, USA, Apr. 2005
15. Jan Haase, Andreas Hofmann, Klaus Waldschmidt: "A Self Distributing Virtual Machine for Adaptive Multicore Environments", International Journal of Parallel Programming, DOI: 10.1007/s10766-009-0119-4, Springer, ISSN 0885-7458
16. Andreas Hofmann, Jan Haase, Klaus Waldschmidt: "SDVM^R - Managing Heterogeneity in Space and Time on Multicore-SoCs", Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2010), Anaheim, USA

AUTHOR PROFILE

Jan Haase received the Diploma degree in computer sciences at Goethe University, Frankfurt, Germany. He then became a Research Assistant at the Department of Technical Informatics, Frankfurt University. There he focused on the field of computer architectures, dynamic and distributed parallel computation, middlewares, and embedded systems, resulting in his Ph.D. thesis on the scalable, dataflow-driven virtual machine (SDVM). Since 2007, he is a Senior Researcher and Project Leader at Vienna University of Technology. His research interests include hardware/software co-design, design methodologies for heterogeneous (AMS+HW/SW) embedded systems, wireless sensor networks, power consumption optimization, and automatic parallelization. He is author and coauthor of more than 90 reviewed publications. Dr. Haase is currently appointed IEEE Region 8 Conference Coordinator. He is an Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS and of the IEEE Industrial Electronics Tech News and served and serves as Program (Co-) Chair or Member in several program committees of international conferences like IECON, AFRICON, DATE, DAC, FDL, CSNDSP, ICIEA, Austrochip, and smaller workshops.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)