Project no. 248828

ADVANCE

Strategic Research Partnership (STREP)
ASYNCHRONOUS AND **D**YNAMIC **V**IRTUALISATION THROUGH PERFORMANCE
ANALYSIS TO SUPPORT **C**ONCURRENCY **E**NGINEERING

Requirements Analysis

D1

Due date of deliverable: July 31st, 2010
Actual submission date: July 31st, 2010

Start date of project: February 1st, 2010

Type: Deliverable
WP number: WP2
Task number: WP2a

Responsible institution: HERTS
Editor & and editor's address: Raimund Kirner
University of Hertfordshire, College Lane
Hatfield, AL10 9AB, United Kingdom

Version 1.0 / Last edited by Raimund Kirner / January 31^{s}t, 2011

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	$\surd$
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	08/06/2010	Raimund Kirner	HERTS	Initial version
0.2	15/06/2010	Raimund Kirner	HERTS	all, feedback from HERTS
1.0	31/01/2011	Raimund Kirner	HERTS	Section 4, case studies

Reviewers:
Frank Penczek, Sven-Bodo Scholz, Alex Shafarenko, Bernd Scheuermann

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
WP2a	Requirements Analysis	HERTS*, USTAN, UvA, TWENTE

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

Within the **Advance** project we extent key technologies for the development and execution of concurrent programs to deploy the optimisation potential of runtime optimisations based on statistical program information.

In this document we list requirements of the runtime monitoring of data, the runtime optimisations, and the case studies provided by the industrial partners.

Contents

Chapter 1

Introduction

The efficient execution of concurrent application on many-core machines or clusters is a non-trivial problem, making it quite challenging to decide code optimisations and placements at compile time. The novel approach of the **Advance** project is to releave the compilation phase by extending the compilation phase into the runtime phase. This allows to exploit statistical information about the program behaviour collected at runtime to iteratively optimise the program code and the program execution.

The key technology to be used and extended in the **Advance** project consists of several technologies developed by the academic partners. A central concept of the application programming in **Advance**is the separation of concurrency specification and logic computation specification. This is done by the coordination language S-Net, which allows the concurrency specification of so-called boxes [4, 6, 1]. The implementation of the computation logic of these boxes is done by a conventional programming language. In **Advance**, we use SACas the implementation language for the boxes [3, 2], because its functional programming style nicely fits the requirement of S-Net that each box is stateless. Furthermore, the SAC compiler can also automatically deploy concurrent computations. To decouple the application programming from the specific features of different hardware platforms, we use SVP as a self-adaptive hardware virtualization layer [5].

Above technologies are quite well suited for the efficient execution of concurrent programs. Within **Advance** we address dynamic optimisation techniques based on statistical information to deploy their full potential.

In this document we collect a list of requirements to be considered for the work within **Advance**. We collect requirements from the tools of the academic partners as well of the case studies of the industrial partners.

Chapter 2

Information to be Monitored

The core work of **Advance** involves the use of statistical information about the program behaviour to perform program optimisations dynamically during runtime. To go for this approach, we first have to specify which information to be monitored by the runtime system.

At this stage it is important to note that the type of information to be monitored also depends on the actual implementation style of S-Net. With the current implementation of S-Net, boxes are mapped to one (or more) concurrent jobs.

There are also other implementation styles of S-Net possible. For example, in the so-called *Graph-Walker* approach, the data messages are the active entities to be mapped to concurrent jobs. In case we will switch to this alternative S-Net implementation style then we will also need to reconsider the list of information to be monitored at runtime.

But assuming that we still go with the current implementation style of S-Net, the monitoring of the following information is considered to be useful:

Average execution time of boxes (box latency)
: To measure the box execution times, it is necessary to ensure that the operating system does not interrupt the execution of a box. To achieve this, there is currently an activity to develop a new runtime layer for S-Net, which minimises task preemption.

Variation of box execution times (box jitter)
: The box jitter might be of interest to reason about potential optimisations. Further, if we will extend the approach to soft real-time scheduling, it might be also of interest to keep the maximum observed execution time.

Average fill-level of stream buffers
: The average fill-level of stream buffers is one indicator for potential performance improvements. If the buffer is almost allways empty, then the sender or one of its predecessors might be too slow. If the buffere is almost allways full, then the receiver or one of its successors might be too slow.

Throughput of messages
Besides the fill-level of stream buffers, the throughput of messages is a further indicator for potential performance improvements.

Message Attributes of Boxes
Monitoring the multiplicity of messages, and the distribution of the different message types produces of a box.

Average delay for message communication (message latency)
Message communication consists of two parts: the send and the receive of a message. In the current implementation these two activities are decoupled via a stream buffer. The average latency for send and receive will be determined without considering the blocking times. The message delay may vary greatly depending on whether the transfer is over the local data cache (i.e., sender and receiver running on the same core), the main memory (i.e., sender and receiver running on the same machine), or the network (i.e., sender and receiver running on different machines).

Jitter of messages
The jitter of message delay indicates timing variations that come from the executed code to send respectively receive a message. This information is, for example, important to reason about the real-time behaviour of the system.

Memory consumption of boxes
The memory consumption of boxes may be taken into account to detect memory shortness and to balance the memory needs among different machines.

Multiplicity of S-Net components
It might be interesting to monitor the multiplicity of S-Net boxes or subnetworks, for example, of components multiplied by sequential replication or of parallel composition.

Characteristica of the box implementations
For a given implementation language of S-Net boxes it can be useful to monitor language-specific characteristica of the message data. For example, for box implementations with SAC it might be useful to monitor the shape of data arrays.

Application-specific data
It is also important to be able to monitor application-specific data to allow specific optimizations. For example, a mechanism to monitor the value of tags S-Net in messages could be of interest.

The periods of sampled behaviour patterns on one hand side should be long enough to amortise the system interruption or delay due to the adaption of the

program. On the other hand side, the periods should be short enough to provide vivid optimisation cycles and to allow the deployment of short-term characteristics of program behaviour. Another challenge for the monitoring is the internal parallelism within a box. For example, the SAC compiler is able to generate concurrent code from a box implementation in SAC.

Chapter 3

Program Optimisations at Runtime

The statical information of program behaviour should be able to support the dynamic optimisation of programs at runtime. Typical program or system optimisations we want to support include:

Multiplication of resource demands
: One way to reduce performance bottlenecks in the S-Net program is to multiply S-Net boxes or subnets.

Addition of specialised components
: Besides the multiplication of resources, performance might be also improved by taking advantage of runtime information about the current program behaviour (e.g., temporally non-changing size of data arrays). A typical optimisation is *partial evaluation* to add a specialised component and the type system will insure that the matching message types are routed to the specialised component.

Relocation of boxes and sub-networks
: The relocation of S-Net boxes or sub-networks might be used, for example, to reduce the message communication delay. Relocation might be done among different cores or even among different machines. Relocation of boxes between different machines also requires that the provided hardware abstraction layer supports the different platforms of interest.

Above optimisation patterns are very generic. Within **Advance** we will develop techniques to apply this patterns at different scenarios.

Chapter 4

Characterisation of Case Studies

In the following we list some key requirements of the industrial case studies provided by the industrial partners. These requirements should help to identify useful improvements of the key technologies used within the project. Further, these requirements will guide the scientific work within the project with respect to monitoring and program optimisation. The following requirements on the case studies are documented:

Application domain
: What are the special implications of the application domain? For example, how much plays cost efficiency or security a role?

Hardware requirements
: What are the specific characteristics that the hardware has to provide in order to run the case study?

Real-time requirements
: Are there any real-time constraints of the case study? What are the tolerance ranges for this real-time requirements, i.e., what are the consequences of missing the deadline, given for different degrees of overruns?

Dependability requirements
: What are the required dependability attributes, like availability, reliability, safety, integrity, maintainability?

Additional platform requirements
: What kind of operating system and development tools are available to choose from in order to run the case study on the desired platform?

Additional software requirements
: What are the interfaces the case study has to work with? What type of legacy software should be supported?

The following lists the cases studies of industrial partners. The concrete requirements of these case studies and their detailed description are given in **deliverable D2**.

4.1 The BioID Case Study – Biometric test and optimisation framework

BioID is a software for multi modal biometric authentication, combining several human characteristics such as face, voice, and iris patterns. The fusion approach achieves reliable authentication even though a limited amount of biometric user data is available to the system for enrolment and verification, increasing user friendliness and recognition accuracy compared to a biometric system that is based on a single modality only.

The **Advance** hardware abstraction feature is very promising for the biometric test and optimisation framework, so it can be expected that this feature can be demonstrated with BioID's application. Another challenge of the optimisation framework is that the optimal mapping and the optimal mapping parameters are likely to change in the course of an optimisation run. If certain algorithmic parameters (not to be confused with mapping parameters) are varied for the purpose of optimisation, this can strongly influence the optimal hardware mapping and the optimal setting of parallel execution parameters like the thread count or data partitioning sizes, as these are dependent on algorithmic parameters. So, the **Advance** feature of dynamic re-configuration and re-mapping on the fly during execution of the application can be demonstrated here.

Another important requirement for the described optimisation problems is a maximisation of the throughput. As the application is not interactive, latency is not an issue. The biometric framework therefore could demonstrate the definition of a cost function that is used by the **Advance** run-time system for optimising the mapping.

A detailed description of the BioID case study and its requirements is given in the **Advance** deliverable D2.

4.2 The PHILIPS Case Study – Low-latency image processing for interventional X-ray

Within medical usage of X-rays, one can distinguish between diagnostic, interventional, and therapeutic usages. In interventional usage, x-rays are used to provide a constant stream of X-ray images (so called fluoroscopy images). It allows the physician to see anatomical structures and medical devices while he/she manipulates them (e.g., catheters, guide-wires, needles, prostheses, etc.). Such usage is characterised by:

- Low x-ray dose
- Eye-Hand coordination
- Operator fatigue
- Data-dependent latency tasks
- Managing performance
- Safety
- Control over performance
- Predictability of performance

Philips wishes to use image processing pipelines of "boxes" (algorithms) with a significantly data-dependent latency. To facilitate scheduling they came up with the idea of using special "predictor" boxes that look at image arrays and work out scalar metrics that correlate with the latency of the processing boxes. They also have a huge library of images (hundreds of thousands) which enables statistical evaluation of the predictors vis a vis appropriately instrumented boxes.

A detailed description of the PHILIPS case study and its requirements is given in the **Advance** deliverable D2.

4.3 The SAP Case Study – Vehicle Scheduling and Routing Optimizer

The *Vehicle Scheduling and Routing Optimiser* is a specific optimiser in the *Advanced Planner and Optimiser (APO)*, which helps users to solve the *Vehicle Scheduling and Routing Problem (VSRP)*, aiming to minimise the costs while satisfying the demands of customers and fulfilling all problem constraints.

Since 2001, SAP has developed and continuously improved an optimization algorithm (VSR-optimizer) for the vehicle scheduling and routing problem. Because of the $\mathcal{NP} - hard$ complexity, it takes usually several hours to provide a satisfactory solution for customer. Customers are interesting in the quality of the solution and the wait time, so only the criterion throughput and latency are related in this application.

The VSR-optimizer might benefit from the parallel programming models of ADVANCE: 1. It can take advantage of the parallelism from S-Net / SAC and gain speedup for performance; 2. Several customer scenarios might be run at the same time within the optimizer by using S-Net; 3. Concurrency engineering may bring some algorithmic improvement to the VSR-optimizer.

The goal of this case study is:

- Evaluate S-Net / SAC of ADVANCE by business application (Vehicle Scheduling and Routing Problem)

- Implement part of VSR-optimizer with S-Net / SAC
- Improve the VSR optimization algorithm to be suitable for ADVANCE
- Compare the new implementation with the old one

In the future the technology developed by ADVANCE could be used by the VSR-optimizer and be integrated in SAP system. It might help customers from all over the world to reduce costs of transportation, distribution and logistics.

A detailed description of the SAP case study and its requirements is given in the **Advance** deliverable D2.

4.4 The SCCH Case Study

The SCCH case study addresses the automation of high-speed quality inspection problems. Especially the inspection of endless material like foils or industrial textile fabrics poses the challenge of guaranteeing high-data throughput while processing cost-intensive algorithms. Advanced cost-intensive algorithms become potentially necessary due to a complex phenomenology of textures and defects and a large variation of products in terms of surface characteristics and shape.

This requires a) the application of powerful algorithms from image processing as well as machine learning, b) the use of high-performance computational hardware like GPU or multi-core systems, and c) the exploitation of parallelization potentials. Particularly, in this case study straightforward parallelization concepts are to be applied to standard filter operations based on fixed sized matrices like anisotropic diffusion filter. Moreover, also the parallelization of non-standard classification algorithms based on support vector machines with specially designed kernels should be a focus of applied research.

For the foil inspection application under consideration the inspection systems have to cope with a scanning speed of $5m/s$ and a data throughput of $80MB/s$ per camera, on quad-core computers equipped with NVIDIA graphics cards. The performance requirements are determined by the production throughput and the actual inspection scanning speed; overruns and delays in processing time would cause either the risk of producing bad quality or a slowdown of the production process, thus unacceptable costs which have to be avoided. Such inspection systems are expected to run 24 hours a day which requires high degree of availability and reliability. Moreover, such systems should be easily maintainable, particularly when the system configuration parameters like production speed or the characteristics of the produced materials are changing. Still the MS WINDOWS operating system is dominating in this context due to the availability of appropriate drivers for the hardware periphery like cameras, illumination or synchronization devices. But, in the meanwhile UNIX based platforms are gaining more and more acceptance by hardware providers as well as costumers. For the purpose of this case

study SCCH installed a LINUX cluster to develop SAC and S-NET implementations and to perform benchmarking tests. For engineering such inspection solutions SCCH uses its own-developed C++ code added by modules from open source packages like OPENCV, WEKA and libSVM.

A detailed description of the SCCH case study and its requirements is given in the **Advance** deliverable D2.

Chapter 5

Conclusion

In this document we listed some key requirements to plan and coordinate the research within the **Advance** project. As the deployment of statistical runtime information plays a crucial role in the project, we have identified several aspects of program execution to be monitored during runtime. Further, we have identified some patterns of program optimisations we want to deploy at different contexts of the program execution. The concretization of these program optimisations is subject to the work to be done within **Advance**. Finally, we have collected a rough first picture about the industrial case studies to be used within the project to demonstrate the feasibility of the technology to be developed within **Advance**.

Bibliography

[1] C. Grelck and A. Shafarenko. Report on S-Net: A Typed Stream Processing Language, Part I: Foundations, Record Types and Networks. Technical report, University of Hertfordshire, Department of Computer Science, Compiler Technology and Computer Architecture Group, Hatfield, England, United Kingdom, 2006.

[2] Clemens Grelck. Shared memory multiprocessor support for functional array processing in SAC. *Journal of Functional Programming*, 15(3):353–401, 2005.

[3] Clemens Grelck and Sven-Bodo Scholz. SAC: A functional array language for efficient multithreaded execution. *International Journal of Parallel Programming*, 34(4):383–427, 2006.

[4] Clemens Grelck, Sven-Bodo Scholz, and Alex Shafarenko. A Gentle Introduction to S-Net: Typed Stream Processing and Declarative Coordination of Asynchronous Components. *Parallel Processing Letters*, 18(2):221–237, 2008.

[5] Chris Jesshope, Mike Lankamp, Michiel van Tol, Thomas Bernard, and Raphael Poss. Svp model reference (the blue book). `https://mac-chris.science.uva.nl/csa/notes/www/book2.html`, 2009.

[6] A. Shafarenko, S.B. Scholz, and C. Grelck. Streaming networks for coordinating data-parallel programs. In I. Virbitskaite and A. Voronkov, editors, *Perspectives of System Informatics, 6th International Andrei Ershov Memorial Conference (PSI'06), Novosibirsk, Russia*, volume 4378 of *Lecture Notes in Computer Science*, pages 441–445. Springer-Verlag, Berlin, Heidelberg, New York, 2007.