Project no. 288570

**PARAPHRASE**

Strategic Research Partnership (STREP)

Parallel Patterns for Adaptive Heterogeneous Multicore Systems

**Application Requirements and Use Case Scenarios**

**D6.2**

Due date of deliverable: Mar. 31st, 2012; Version November 2012

*Start date of project:* October 1st, 2011

Type: Deliverable  
**WP number:** WP6  
**Task number:** T6.2

**Responsible institution:** SCCH  
*Editor and editor’s address:* Holger Schöner, Software Competence Center Hagenberg GmbH

---

<table>
<thead>
<tr>
<th>Project co-funded by the European Commission within the Seventh Framework Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dissemination Level</strong></td>
</tr>
<tr>
<td>PU</td>
</tr>
<tr>
<td>PP</td>
</tr>
<tr>
<td>RE</td>
</tr>
<tr>
<td>CO</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable describes the applications and use case scenarios selected by the partners that are active in WP6 (Use Cases, Requirements and Evaluation). Use case selection and evaluation criteria are given. Some selected and proposed core use cases are detailed, organized by the application area of a respective project partner. Further requirements, targeted hardware, and expected use case results are specified. Also, potential for the use of the generic patterns (deliverable D2.1) and application specific patterns (deliverable D2.3) are mentioned. The annex gives information about further use cases potentially relevant further on in the project or as dissemination and exploitation opportunities.

The parallel patterns and the use cases are expected to evolve together and mutually depend on each other, during the duration of the ParaPhrase project. Thus, the state of the use cases is expected to adapt to the evolution of the implemented parallel patterns and to further opportunities for use cases with high impact in the Erlang, industrial, and High Performance Computing communities.
Contents

Executive Summary ............................................................... 1

1 Introduction ........................................................................ 4
   1.1 Use Case Scenarios and Application Requirements ............... 4

2 Machine Learning Use Cases ............................................... 7
   2.1 Applications ............................................................... 7
   2.2 Use Cases .................................................................... 9
      2.2.1 Machine Learning in the Chemical Industry ............... 9
      2.2.2 Machine Learning in Constraint Programming .......... 9
   2.3 Algorithms, Relevant Patterns, Hardware and Expectations .... 10
      2.3.1 Waste Water Processing ....................................... 10
      2.3.2 Machine Learning in Constraint Programming .......... 11

3 Exploiting Data Parallelism on Heterogeneous Systems .......... 13
   3.1 Applications ............................................................... 13
   3.2 Use Cases .................................................................... 14
      3.2.1 Erlang and Parallelism .......................................... 15
      3.2.2 Matrix Operations in OpenCL ............................... 16
      3.2.3 Intensional Parallelism DSL ................................. 16
      3.2.4 Work-stealing Scheduler for Erlang ....................... 16
      3.2.5 Scalable OpenFlow Switching ............................... 17
   3.3 Algorithms, Relevant Patterns and Requirements ................ 17
      3.3.1 Erlang runtime system GPU support ....................... 18
      3.3.2 Parallel Patterns and Refactorings .......................... 18
      3.3.3 Orchestrational Patterns ..................................... 18

4 Scientific Use Cases .......................................................... 19
   4.1 Applications ............................................................... 19
   4.2 Use Cases ................................................................. 20
   4.3 Algorithms, Relevant Patterns and Requirements ................ 20
   4.4 Additional Requirements ............................................. 21

5 Requirements for Network Technology and Hardware ............ 22
6 Conclusion

A Additional Use Case Potential
   A.1 Further Machine Learning Use Cases
      A.1.1 Machine Learning Framework (MLF)
      A.1.2 Local Weather Prediction
1. Introduction

1.1 Use Case Scenarios and Application Requirements

The aim of this deliverable is to describe the applications and use case scenarios selected by the partners active in WP6 (Use Cases, Requirements and Evaluation). In the following chapters, the use case scenarios are detailed, organized by the application area of a respective project partner. Requirements resulting from these are specified, in addition to those already given in deliverable D6.1. Also, relevant generic patterns as described in deliverable D2.1 are mentioned. Additional application specific patterns are necessary for some use cases, and are specified in more detail in deliverable D2.3.

The partners contributing in WP6 and providing the use cases are mainly SCCH, HLRS, and ESL. SCCH is active at the interface between research and industrial partners, providing individual solutions and knowledge transfer in joint projects with partner companies. For ParaPhrase, computationally intensive use cases are selected from the Machine Learning domain, as described in Section 2. ESL as expert and solution and knowledge provider for the Erlang system and language selects use cases to enable the easy use of parallelization technology by Erlang programmers, and to implement a solution with high scalability demands, detailed in Section 3. HLRS provides massively parallel computation infrastructure to research and industrial customers, as well as services regarding the development of performant programs running on that infrastructure. In Section 4, they select use cases from typical applications, to serve as examples and knowledge base for further implementations using ParaPhrase technology. Section 5 gives further requirements on technology to consider for highly scalable distributed systems resulting from the interconnection hardware available in HPC systems.

The criteria for the choice of use cases are (i) to be representative for typical user scenarios with large amounts of data or high computational complexity; (ii) to cover a range of types of parallel activities – data parallel, streaming, divide and conquer; (iii) to exercise the range of standard patterns; (iv) to provide opportunities to discover domain specific patterns; (v) and to cover a range of hardware requirements (see Table 1.1 as well). The parallel patterns and the use cases are expected to evolve together and mutually depend on each other, during the duration of the ParaPhrase project. Thus, the state of the use cases as described in the following chapters is expected to adapt to the evolution of the implemented parallel patterns and to further opportunities for use cases with high impact in the Erlang,
Table 1.1: Selection criteria used for choosing use cases, and the desired variety of characteristics. Use cases are chosen such that the set of them covers as much of the characteristics of each criterion as possible.

<table>
<thead>
<tr>
<th>Selection Criterion</th>
<th>Desirable Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domains</td>
<td>Scientific, Industrial</td>
</tr>
<tr>
<td>Pattern coverage</td>
<td>Use of basic patterns: Farm, Pipeline, Map/Reduce, Use of high level patterns: Divide &amp; Conquer, Provision of interesting application specific patterns</td>
</tr>
<tr>
<td>Targeted hardware</td>
<td>Homogeneous multicore system, Heterogeneous with CPU and GPGPU cores, Distributed homogeneous and heterogeneous cluster</td>
</tr>
<tr>
<td>Refactoring potential</td>
<td>Parallelizing a sequential algorithm, Refactoring for exploring different semantically equivalent patterns with differing non-functional behaviour</td>
</tr>
<tr>
<td>Scaling potential</td>
<td>Embarrassingly parallel for scaling evaluation, large and small grained parallelism for evaluating GPGPU limits and communication overheads, stream and data parallelism</td>
</tr>
<tr>
<td>Implementation language</td>
<td>C++ / FastFlow, Erlang</td>
</tr>
</tbody>
</table>

The main body of this deliverable (Sections 2–4) describes in detail those use cases selected for implementation with and evaluation of the ParaPhrase technology during the course of the project. Should any of those turn out not to be feasible, the annex provides (in less detail) further potential use cases as backup. These additional use cases also present an opportunity for further dissemination and exploitation of ParaPhrase technology if the initial evaluation is found positive.

To have a high impact in use cases and in applications in general, the expectations wrt. ParaPhrase technology (and thus also on the evaluation of the use cases) are mainly concerned with ease of and clarity in parallelization. A clear definition of patterns and the refactoring support are key points in this respect. Very important is also the promise to support a wide variety of hardware. If an application has been ported to exploit parallelism, then it is a huge advantage, if this same (or via refactoring a similar version of the) code runs well on other hardware as well (e.g. multicore versus distributed cluster). This is a necessity to allow reuse of parallelized code and especially libraries. Of course, the expectations are also on performance gains. But the focus here, from a users perspective, is mainly on com-
Table 1.2: Expectations and evaluation criteria for the outcome of use case porting and for ParaPhrase benefits in general.

<table>
<thead>
<tr>
<th>Expectation/criterion</th>
<th>Description and evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of parallelization</td>
<td>Time spent for implementation of or porting to a parallelized algorithm,</td>
</tr>
<tr>
<td></td>
<td>Usability of refactoring,</td>
</tr>
<tr>
<td></td>
<td>Clarity of parallelized code</td>
</tr>
<tr>
<td>Flexibility of target hardware</td>
<td>Support for and efficient use of different target hardware (as appropriate for problem size</td>
</tr>
<tr>
<td></td>
<td>and characteristics)</td>
</tr>
<tr>
<td>Performance and Scalability</td>
<td>Speedup achieved wrt. sequential code,</td>
</tr>
<tr>
<td></td>
<td>Performance in the region of such obtainable by alternative parallelization frameworks,</td>
</tr>
<tr>
<td></td>
<td>Appropriate scheduling of workers on nodes/CPU-/GPU-cores</td>
</tr>
</tbody>
</table>

Comparison with a former sequential version of the code. One would still expect good scalability, but as long as ParaPhrase supports easy parallelization and is flexible wrt. target architectures, the performance need not be better than other existing parallelization approaches (although it should not be much worse either, of course). These expectations are summarized in Table 1.2. The evaluation of these criteria will be easiest regarding the performance, and the support for hardware flexibility, of course. Regarding the ease of parallelization, the evaluation will mostly depend on experience reports by WP6 partners, and comparison of code variants parallelized using ParaPhrase and alternative frameworks.
2. Machine Learning Use Cases

2.1 Applications

Software Competence Center Hagenberg GmbH (SCCH) is, among other areas, active in research and projects concerning data analysis and Machine Learning (ML), mainly for industrial applications. Its regions of activity, in particular Upper Austria, but Austria and selected regions in Europe as well, have strong industrial sectors. Several of the partners with whom SCCH is working together are significant players in these regions.

Automation of manufacturing and application processes is continually growing. With it, the potential for collection and usage of data for optimization of these processes also increases. Machine Learning is an important technology for performing such analysis and optimization. With growing opportunities and sizes of data sets the demands on analysis systems rise as well, and parallelization for computation clusters or for GPUs is an inviting direction for fulfilling these demands. Parallelization already is an active research area in Machine Learning [3, 12], but the main systems/libraries so far have no or limited capabilities in this respect. To meet the need for better performance of Machine Learning (ML) methods, SCCH in the context of the ParaPhrase project plans to parallelize existing ML methods used in projects with partners, as well as to extend the collection of available methods by new and parallel approaches.

Machine Learning can be utilized for different purposes, among them

**Understanding Industrial Processes** For many industrial processes, such as steel manufacturing and plastic or metal working, process knowledge is only partial. Many decisions and workflows are based on empirical knowledge and past experience. By analyzing data collected during normal operation or in extraordinary circumstances, hints about unknown dependencies can be generated and discussed with experts in the field. This can provide sensible directions for further research and generate new knowledge about the process. Such knowledge can later lead to optimized processes with less fault situations, better quality, higher production speed, or reduced production costs.

SCCH is active in this area and uses methods such as Decision Trees and Regression Trees, as well as visualization methods and introspection into
trained ML models for knowledge discovery. A relatively new area with active research is the detection of causal relations between variables in industrial data sets [23].

**Prediction and Model Predictive Control** Related to the understanding of technical and industrial processes are prediction, Soft Sensors and Model Predictive Control. Building on expert or automatically extracted knowledge, often manufacturing processes can be modeled to allow predictions about their state or outcome, or to replace expensive or problematic sensors. Predictions are done by using learned dependencies on some input parameters such as machine, material and tool properties, process settings, and environmental conditions like temperatures. Such predictive models can then be used to control the process by optimizing the process settings, to achieve better stability, better quality, or higher production speed.

Model Predictive Control applications are the topic of several cooperations of SCCH with customers in the areas of steel manufacturing, machine construction, and chemical production. Methods and parallelization potential are similar to the Understanding Industrial Processes application field, already described above.

**Fault Diagnosis** With machines and other systems becoming more and more complex, the task of identifying the sources of observed problems is getting harder as well. One way to deal with this situation is to create models of the components, their interactions, possible failures, and their consequences. A commonly used technique for doing so are Bayesian Networks [5,14,17,26].

Such models often are created using explicit (expert) knowledge about the process. Additionally, such knowledge can be augmented by relations extracted from actual data collected during operation and fault situations of the system.

Bayesian Networks are graphical models; the parallelization of algorithms working with them is non-trivial but interesting because of the diverse dependency structures in the graphs. A problem arising in the extraction of the relations is to determine causality, i.e. the direction of the dependencies [23]. Extraction of causal relationships from data is a very time consuming process (without heuristics it is combinatoric in the number of observed and hidden variables).

SCCH is involved in developing Fault Diagnosis applications for the areas of machine fault diagnosis, using graphical models and discovering causal relations, among other approaches.
2.2 Use Cases

The presented application areas and the available projects at SCCH provide several choices for selection of use cases. In the following, two are explicitly selected for further work in tasks T6.3 (porting and annotation of applications) and T6.4 (experimental evaluation). This selection is based on requirements from technical work packages, i.e. need for interesting parallel patterns, targeting of GPGPU as well as multicore machines, and provision of potential for refactoring. SCCH plans to use ParaPhrase technology in further projects; the potential for pattern, refactoring, and performance evaluation provided by these is presented in Annex A.1.

These use cases will be made available to project partners for evaluation and experimentation purposes. Data for benchmarking will be from real applications developed for partners of SCCH and available at least in anonymized form. Code will be based on mlpp, a library with all needed machine learning methods developed for this purpose at SCCH, and available in sequential and parallelized versions.

mlpp will be a C++ library implementing the machine learning methods needed for use cases, with original sequential and differently parallelized versions. Besides versions parallelized using ParaPhrase technology, there will also be some versions using other methods (potentially OpenMP, CUDA, GraphLab) to allow direct comparison of performance and code clarity between these.

2.2.1 Machine Learning in the Chemical Industry

SCCH is part of the K-Project PAC (Process Analytical Chemistry, www.k-pac.at), which includes company partners like Brau Union Österreich AG, Borealis AG, Dynea, OMV, Sandoz GmbH. As a first use case, SCCH will implement a causality detection framework for analyzing waste water processing data, provided by the PAC partner company Lenzing [16]. The data is collected in a plant processing water from industrial as well as commercial and residential neighborhoods. It contains time series data for 6000 features with information about contamination, throughput, chemical analyses and control parameters. Goal of the analysis is to find the (causal) dependency structure leading to the final processing outcome quality.

2.2.2 Machine Learning in Constraint Programming

A second selected use cases is a project with a large customer in the area of metal sheet processing. Material consumption for production of individual parts is optimized using algorithms from the area of Constraint Programming, solving a discrete optimization problem with constraints. The optimization of each part consists of a chain of optimization steps, each of these heuristically selected from a set of candidate steps. These methods are being improved and made more efficient at SCCH using Machine Learning to improve selection of appropriate candidates at
each step. For the Paraphrase use case, this selection and evaluation process is to be sped up by parallelization.

2.3 Algorithms, Relevant Patterns, Hardware and Expectations

This section summarizes the technical implications of the use cases selected in Section 2.2.

2.3.1 Waste Water Processing

The detection of the causal structure of dependencies among the features available in the waste water processing data is done using a combination of graphical lasso \[^{11}\] and granger causality \[^{2}\] methods. At the lowest level, this implies the use of linear algebra methods with according potential for either employment of data parallelization, or the use of highly optimized existing numerical libraries. On a higher level, the linear algebra operations are used by loops which allow stream parallel processing using a Farm pattern. In particular, coordinate descent, used for parameter optimization, allows a certain (data dependent) degree of parallelization \[^{6}\].

Considering the expected set of patterns to be used by this use case, and taking into account the hardware available at SCCH and partner sites, this application will target homogeneous or heterogeneous multicore machines. The lower level linear algebra parts seem well suited to be scheduled on GPGPU cores, while higher level parallelization can well be performed on multiple CPU cores. Distributed systems do not look promising, as there will be some communication necessary after each coordinate descent iteration, and the iterations will probably be too short to afford the communication costs involved in distributed systems. It might, on the other hand, still be interesting to test this, and to evaluate whether there is a problem size where e.g. InfiniBand interconnects (as opposed to standard Ethernet connections) might make distributed processing feasible. Refactoring will be useful for parallelizing the existing sequential version of the code.

Our expectations for this use case are primarily a relatively easy and flexible way of parallelizing sequential code. Flexibility is sought wrt. to supported target hardware, as we expect this code to be run at several sites with differing available hardware (number of CPU cores; availability of GPGPUs). Evaluation criteria will be developed further on in the project, but we expect to evaluate this criterion by comparison with alternative parallelization approaches (e.g. CUDA, OpenMP, use of numerical libraries).

Expectations regarding speedup are an almost linear scaling of the low level linear algebra code with the number of available CPU or GPGPU cores, up to the limits given by the problem size. I.e. basically speedup comparable to other existing linear algebra implementations. Regarding speedup due to parallelization
of higher level code, we expect strong problem dependency. For easy problems, the speedup should be linear as well (up to the number of available features in the data). On the other hand, the more dependency the features have on each other, the less the expected degree of sensible parallelization, to still ensure convergence of the optimization. Additionally, when approaching the parallelization limits, the added degree of parallelism is expected to be traded off with a higher number of iterations being necessary for convergence. This is expected due to theoretical considerations, regardless of the underlying parallelization framework.

2.3.2 Machine Learning in Constraint Programming

The resource optimization problem in Section 2.2.2 is solved using a stepwise algorithm enforcing an increasing number of constraints on a set of candidate solutions, interleaved with an optimization of the partial solutions. This algorithm allows to explore a variety of different parallelization approaches. In Deliverable D2.3 (table 1.1), a “computational object pool” pattern is mentioned (although not for the initial set of application specific patterns). Using this pattern, the pool would contain the partial solutions, from which in parallel several could be picked to be updated by further constraints and/or optimized solutions, with the resulting candidates being put back into the pool. The computation would terminate when a sufficiently optimized full solution is found.

Alternatively (until such a pattern becomes available), such an algorithm will be implemented using an iterated farm of pipelines, working on the stream of partial solution objects. This will encompass more programming overhead for management of the state of the “pool”, scheduling of partial solutions to work on, determining the state of the pool for termination check, but it will still be feasible. A refactoring (manual or later on possibly supported by WP4 technology) to use an iterated map pattern might also be interesting, although the performance might be inferior due to forced synchronization after each performed map.1

When the genetic algorithm patterns (Deliverables D2.1 and D2.2) become available, they might be used for implementation as well, by setting crossover functions to identity, and mutation function to perform a pipeline of constraint enforcement and solution optimization.

Target machines for parallelization of this use case are primarily homogeneous multicore or distributed cluster machines, with an emphasis on the multicore machines available at SCCH, UNIPI and RGU, in the beginning. GPGPU parallelization does not seem promising, as the individual tasks working on the partial solution objects will be different for the available objects. Distributed computing, on the other hand, seems promising, as the tasks can have considerable computation time, making the communication overhead worthwhile.

Additional potential for use of GPGPU parallelization might later on arise when use of Machine Learning to guide the selection of most promising trans-

1The code performing constraint enforcement and optimization is expected to take variable time depending on the object being worked on.
formations on individual candidate solutions will be considered. But this will not yet be considered in implementation of this application as a ParaPhrase use case.

This use case is primarily expected to show flexibility (wrt. to patterns and supported hardware) and ease of programming using ParaPhrase patterns, as well. Considering performance, a final absolute speedup of 40-50 should be reachable on some appropriate hardware. Initially, targeting dual-CPU hexacore nodes of the machines/clusters available at SCCH and RGU, we expect a speedup of about 8-12, i.e. nearly linear in number of available cores, but with some synchronization cost due to management of the pool.
3. **Exploiting Data Parallelism on Heterogeneous Systems**

3.1 **Applications**

ESL is involved in customer projects which evolve around building distributed systems with massive scalability requirements. Due to Erlang/OTPs distribution capabilities, Erlang is very well suited to building such systems compared to other systems programming languages.

The scaling patterns which Erlang/OTP offers out of the box cater for solving problems in the concurrent programming domain and can be leveraged to produce programs which scale to 1000s of nodes.

Although Erlang scales out well from node to node, it does not perform as well as it could for problems which require efficient in memory storage, shared memory and data parallelism.

ESL plans to introduce a package which allows Erlang programmers to leverage the performance of multicore and GPU aware machines in order to increase the performance of parallel applications which have intrinsic data parallelism.

Some fields of applications for massively scalable systems which ESL is targeting are

**Gaming** Many gaming companies have picked up Erlang as a tool for their backend servers due to its ability to scale nicely. Companies like MachineZone and Wooga are using Erlang extensively for this. The move towards more online features in games drives a need for better backend servers and this is ideally suited for Erlang. For games that require realistic physics there are many computations which must be parallelised and this fits with the objectives of **ParaPhrase**.

**Databases** A new breed of databases such as key-value datastores [7] or document-oriented databases have emerged in the last few years which provide better scale-out attributes for non-relational data compared to relational databases. Some of these new databases have been implemented in Erlang/OTP [13,25], each being good alternative data stores. In the future specialized databases will be developed according to a application’s use case. These databases are
expected to have the need for massive horizontal scalability in common, thus they would greatly benefit from the advances made in ParaPhrase.

**Data Processing** Erlang has been used to develop systems for various kinds of data processing such as data analysis [21], video streaming [15] and messaging systems [22]. Each of these applications have a need for coordinating tasks in a ever growing cluster of nodes. Furthermore these systems usually utilize sequential algorithms, while their parallel counterparts would greatly improve performance and scale-out characteristics of the overall systems. Therefore developers can make use of specialized parallel patterns and refactorings developed ParaPhrase to improve scalability of such systems.

**OpenFlow** OpenFlow [19] is a specification for software-defined networking, allowing networks to be reprogrammed at runtime via open APIs. This approach was originally meant to allow researchers to easily setup and alter networking experiments, being able to combine proprietary hardware with a open protocol. However more recently OpenFlow has become a viable option for commercial deployments since it acts as a unified API across proprietary hardware, thus users won’t buy into a particular hardware vendor’s protocols. This move has been underlined by the recent addition of the de-facto standard OpenFlow switching component Open vSwitch [8] into the Linux Kernel [9].

### 3.2 Use Cases

Once work was conducted within WP5 it was decided that two distinct models of computation were required, one which caters for concurrent programming and another which deals with data parallelism and the parallel programming domain.

ESL have been exploring the possibility of providing a framework which harnesses OpenCL within Erlang in order to offload spatially parallel computations to GPUs, the Parallela hardware and to CPUs.

The first part of this framework is an Erlang API to OpenCL which can be leveraged from the rest of the ParaPhrase project who are currently doing work in Erlang. This framework can be used as the base for work conducted within WP2.

The second is a DSL which allows for computations involving matrix and vectorial operations to be expressed in a simple manner, and to provide offloading to devices which support these operations.

The devices which are being targetted by this framework are the GPU and the Parallela architecture, the distinct benefits one may leverage from either will be elucidated with benchmarks.

In the long term ESL is very interested in developing improvements which directly benefit Erlang developers in the industry, such as scheduler and memory allocator extensions which provide better efficiency on different classes of problems.
The extensions we are looking at will be designed for the new and coming multi-core platforms, which hopefully have benefits over a scheduler that has evolved from single core to handle multi-core.

The ultimate goal is to provide an extended Erlang VM where the scheduler and runtime are optimised towards both concurrency & parallel oriented applications.

Since this is new territory for ESL in terms of the potential business area, we may find other opportunities as the use cases are implemented. The current work being done is in providing a way to add parallel constructs to Erlang which result in space efficient code so that parallelism is leveraged on par with concurrency within the Erlang VM.

3.2.1 Erlang and Parallelism

The Erlang memory heap consists of processes along with message data. Typically when solving a data parallel problem in Erlang such as matrix multiplication, processes are created which operate upon a portion of the heap, and fill some resulting heap with the result of a primitive functional operation, such as multiplication being applied to it.

The scheduler will automatically handle the parallelism in this case by load-balancing the operations across cores. Erlangs current model can be used in this manner, but it results in much redundant memory usage and processes being scheduled according to all other processes in the system rather than treating these operations independently.

If the memory which is currently being utilised to represent the matrix is converted to a space efficient one which can be plugged into an offloading unit, the offloading unit can then apply spatially parallel operations upon the data without incurring any of the overheads one would get whilst executing the code within the Erlang runtime system.

The scheduler in Erlang has two major drawbacks which limit its interoperation with external components on the one hand, and bottleneck parallel applications which depend upon short lived fine-grained processes on the other.

- The native function interface provided in Erlang causes the Scheduler to block
- The scheduler expects processes to be long-lived and utilises an expensive load balancing algorithm in order to migrate processes across cores

The tradeoffs made within the scheduler work exceedingly well for concurrent oriented applications such as web servers which require processes to stay alive for a long time, but do not cater well for the set of problems which require either data parallelism or short-lived tasks which compute the result of a computation.

Ideally we would like to have an efficient data representation for vectors & matrices such as an unboxed array of floats, and apply a spatially parallel operation upon this efficient representation in isolation.
3.2.2 Matrix Operations in OpenCL

When modelling physics in games it is typical to make use of matrix operations provided by some unit of hardware such as the GPU in order to leverage the efficiency that can be obtained by doing so.

The aforementioned DSL will allow the programmer to compile an expression which has primitive operations which map 1 to 1 to a GPU operation.

In order to measure the applicability of the DSL we propose two use cases:

- An LOD based algorithm which renders a voxel based cuboid in real-time
- A primitive 3D environment where physics is applied to the voxels

This problem is a good fit for OpenCL since a streaming algorithm which progressively renders the mesh can be developed.

Aside from generating OpenCL code from the DSL, offloading of computations to the GPU and the Parallela hardware will be investigated in order to obtain better performance.

3.2.3 Intensional Parallelism DSL

The DSL for OpenCL is a subset of an outer intensional programming language. This model was chosen in order to express matrices and their operations in a user friendly manner and is based on the work of Plaice and Beck [24].

An example of the DSL we are working on along with an example of how it can be used to operate on a data cube can be found in [10] section 5.5.

Building on the offloading work done with the OpenCL DSL, we will investigate how to integrate offloading to the GPU in a sensible manner. The intention is to minimise the amount of low-level configurations that the developer has to do, so that the scheduler has some heuristics for offloading that will work well in most cases.

As part of the tool set around this parallel DSL we will create a mechanism for integrating the DSL with Erlang, so that it easy for the developer to convert from one memory representation to the other.

Multi-dimensional programming is not the easiest model to get into — it has a rather steep learning curve — so we will identify what sort of supporting tools are needed in order to help people adopt this new way of programming. Whether we have the time and resources to implement these tools cannot be judged at this point in time.

Part of this work involves investigating the use of a scheduler designed for parallel programming problems such as the Cilk scheduler and comparing it to Erlangs.

3.2.4 Work-stealing Scheduler for Erlang

Once we have implemented our DSL for multi-dimensional intensional programming and its scheduler as well as integrated it with Erlang we will be ready to look
into how to implement a new runtime for Erlang which runs on a work-stealing scheduler developed at ESL which is optimised for short-lived processes.

It is not obvious how this should be done, nor trivial, so this is a stretch goal for our use cases.

Erlang’s scheduler was not created to exploit multi-core, it just happens that the process model fits multi-core quite well, but the scheduler is undergoing constant improvements in order to improve the ability to scale as more cores become available. There is also no notion of offloading to GPU or in other ways deal with heterogeneous architectures.

Our goal is to start with a scheduler that was designed to work well on multi-core as well as heterogeneous architectures and see if we can map the semantics of Erlang to that. Then we will be able to execute the concurrent and parallel computations on the same runtime, which will ensure that the resources of the platform are used optimally in terms of what the running program needs. With the parallel DSL there might be situations where the responsiveness of the system is impeded because there is no control between the concurrent Erlang side and the parallel DSL side. That kind of coordination can most easily be done if all the code is executing on the same runtime.

3.2.5 Scalable OpenFlow Switching

A OpenFlow switch, as described by the specification, provides two main functionalities. (1) It is responsible for routing network packages according to a up-to-date routing configuration. (2) It provides a API to OpenFlow controllers, allowing the modification of its configuration at runtime. When scaling up hardware into which a OpenFlow switch runs, each of these functions needs to scale as well to provide consistent routing performance even during periods of frequent reconfiguration and high load. ESL is developing a OpenFlow switch in collaboration with a client which addresses the following scalability issues:

- Configuration changes need to be atomic
- Configuration changes need to be performant even for large scale
- Configuration changes may not impact routing performance

These issue become more apparent the more parallelization is supported by the underlying hardware. The resulting switch will be vertically scalable, allowing its use in network setups for actively transformed high-demand networks.

3.3 Algorithms, Relevant Patterns and Requirements

The use cases described above require enhancements made through ParaPhrase to make Erlang/OTP truly viable for massive horizontal scalability as needed for
gaming and other applications in need of parallel computations. This section describes the main requirements in enough detail such that one can the individual work packages can get a initial idea of each use case’s direction.

3.3.1 Erlang runtime system GPU support

Depending on the learnings from the intensional parallel DSL we will either implement our own runtime for Erlang or suggest changes to the Erlang runtime system to support parallelism better.

The Erlang runtime system already allows a developer to transparently utilize all CPU cores available on a given node. This functionality should be extended such that a hardware abstraction API is used to offload computation to a GPU available to the runtime system. It is expected that the same API could be used in the future to add support for other specialized processing units. Furthermore the runtime system needs to be able to reason about which computation to offload, either fully automatic or through hints given by the developer. Either way a computation needs to be able to run on any processing unit in order to provide a useful foundation for fault-tolerance.

3.3.2 Parallel Patterns and Refactorings

While Erlang does provide high-level abstractions for distribution of computation throughout a cluster, it is still much easier for a developer to reason about a algorithm with sequential semantics. Therefor a developer should be able to express a computation using sequential semantics according to high-level patterns, then use automatic or semi-automatic refactorings to transform the implementation into one optimized for parallel execution on a given cluster configuration. This will greatly improve developer productivity as well as maintainability of implementations. The patterns themselves should take into account on what kind of hardware and cluster size the computation will be run to determine a efficient coordination of tasks.

3.3.3 Orchestrational Patterns

When developing algorithms which need to satisfy strong requirements for atomicity and consistency, it is easier to reason about the impact of certain computations by describing their relationship. Therefor one can easily create chains of actions. Patterns which allow a developer to express such connections between actions are needed, such that the pattern provides the ability the parallelize the given computations without sacrifizing the constraints laid out by the developer. Such patterns would serve a orchestration facility for high-complex tasks which would normally become a performance bottleneck in a otherwise parallelized system.
4. Scientific Use Cases

HLRS has a long standing history in High Performance Computing (HPC). It provides its customers from research and industry beside powerful compute resources not available otherwise also the knowledge in the usage of these complex systems. This knowledge is the result of active research in many scientific and mathematical areas.

4.1 Applications

Even today Scientific and mathematical applications in High Performance Computing (HPC) must be able to run on systems of 100,000 processing elements or more. Efficient usage of the compute resources is a difficult task, which needs to deal with issues as taking advantage of the latest instruction sets (e.g. AVX or FMA4), load balancing on the thread, process or node level, error handling or the processing of very large datasets.

In the future the already complex systems will have to become more power efficient. This trend is observable today in the move towards the usage of accelerators like GPGPUs: 3 of the current 5 fastest supercomputers in the TOP500 list are already equipped with GPGPUs. This increases the complexity of the HPC systems further making programming even harder. This complexity is more and more becoming a problem for the application programmers which would like to focus onto their science and not onto the specific implementation of parallel patterns around their algorithms.

Here comes in the ParaPhrase approach: Many of these applications use the same parallelization strategies which can be expressed by ParaPhrase patterns. The usage of these generalized patterns will not only ease and speed up the programming process, but will also help with its dynamic runtime scheduling system to reduce load balancing issues and hardware failures by dynamic adaption of the compute resources as well. This approach will also result in better portability of source code across the highly tuned HPC-systems.

As examples we will use the field of Molecular Dynamics simulations. This field is strongly linked to the history of HPC and play an important role in many research areas. MD is used, for instance, in chemical engineering to predict properties of toxic or explosive substances. In Biochemistry it is used to determine the structure and mechanics of complex proteins at the nanometer-scale serving as
for cell1 in cells do
  process_cell(cell1)
  for cell2 in cell1.neighbours do
    process_cell_pair(cell1, cell2);
  end for
end for

Listing 1 Basic structure of a cell algorithm with neighbour interactions.

a microscope with very high time resolution. But the basic concepts can also be applied to problems in Astronomy or other computational sciences.

4.2 Use Cases

Many scientific applications rely on cell-based algorithms. The underlying concept is a decomposition of the simulation domain, network, etc. into smaller parts, i.e. cells, which interact only relatively weakly. The weak nature of these interactions across cells is exploited to reduce computational complexity of the algorithm. For example in the link-cell algorithm used in MD simulations [1] each cell contains the molecules belonging to the subvolume represented by the cell. In the case of short range potentials this partitioning allows an efficient way to calculate interactions with neighbouring molecules: Selecting the potential’s cutoff radius as the cell dimensions, interaction partners are either found in the own cell or in it’s direct neighbours. This reduces the computational costs from $O(n^2)$ to $O(n)$.

The calculation of all molecule pair interactions is now done by iterating over all cells in the domain. For each cell the interaction of molecules within itself and with molecules within its neighbour cells are evaluated. The interactions between two molecules can be further divided into the evaluation of different potential functions. Common examples are the Lennard-Jones, Coulomb, etc. potentials.

While the cell based algorithms are generally known in the specific scientific user community, the programmers implementing them are in most cases not specialists in HPC. So programmers waste time porting their applications to the HPC facilities, or even worse, use slow programs. The refactoring tools which will be provided by the ParaPhrase project will help these users with optimizing their codes. Porting to different platforms will be possible with less effort than it is now.

4.3 Algorithms, Relevant Patterns and Requirements

Listing 1 shows the basic structure of a cell algorithm which can be parallelized with different strategies. It can be build up from the farm and streaming paradigms.

The farm pattern will visit each cell and invoke all necessary calculations on it. For our MD example these are the interactions within the cell and its neighbours. The calculations within the cell and those necessary for the neighbours can then be
represented by the stream pattern, which ensures that no concurrent data updates in
the central cell occur. In the MD simulation one step in the stream would process
the molecule pairs within the involved one or two cells.

Special care must be taken when data needs to be modified in both interacting
cells, as is done in MD to exploit Newton’s 3rd Law because there may occur
concurrent data accesses and updates. This problem can be solved using a coloring
scheme where only cells with non-interleaving neighbour cell areas are evaluated at
the same time [20]. This can be represented as a *stream of farms* where each stream
step processes a *farm* for a different color. A more general and better approach
could be the development and usage of a pattern including *mutual exclusion* based
on a static or dynamic stencil.

Another pattern which is used in nearly every scientific or mathematical appli-
cation is the *reduction*. The vast of information generated during a program run
are combined to some meaningful variables. For example it makes not much sense
in a MD simulation to study the path of each single molecule. Instead the average
values temperature, pressure or density are studied.

### 4.4 Additional Requirements

While we have general *farm* and *reduction* patterns the “colored farm” would be
a general pattern applicable to a wide range of applications. This colored farm
pattern will allow the easy parallelisation of all kind of stencil operations working
on datasets.
5. Requirements for Network Technology and Hardware

Some additional requirements for ParaPhrase technology regarding High Performance Computing (HPC) use cases can be deduced by considering the architectures used for HPC. Mellanox as a leading provider of interconnection technology offers important solutions to the problem for efficient communication among thousands of processing elements. For systems where these are available, ParaPhrase technology should take advantage of these. Especially for HPC applications (cf. Section 4) such interconnection technology is important, and existing solutions are often tuned to take advantage of it. To be competitive, ParaPhrase has to support it as well, in appropriate environments.

The individual technologies relevant and offered by Mellanox are:

• Transport Offloads – Mellanox InfiniBand solutions are the only solutions that do not rely on the host CPUs to drive the network communications, and thus allow more CPU cycles to be dedicated to the applications. Mellanox transport offload technology allows the MPI processes to communicate utilizing PIO or Work-Requests options, in a full hardware-based reliable manner and be protected and isolated from other processes by the adapter hardware.

• Fabric Collectives Accelerations (FCA) with CORE-Direct hardware engine - Collective communications, which have a crucial impact on the application’s scalability, are frequently used by scientific simulation codes like broadcasts for sending around initial input data, reductions for consolidating data from multiple sources and barriers for global synchronization. Any collective communication executes some global communication operation by coupling all processes in a given group. This behavior tends to have the most significant negative impact on the application’s scalability. In addition, the explicit and implicit communication coupling, used in high-performance implementations of collective algorithms, tends to magnify the effects of system-noise on application performance further hampering application scalability. Mellanox adapters and switches address the collective communication scalability problem by offloading the MPI collective communications to the network. This solution provides the mechanism needed to support computation and communication overlap, allowing the communication to
progress asynchronously in hardware while at the same time computations are processed by the CPU. It also provides a means to reduce the effects of system noise and application skew on application scalability. FCA includes floating point units at the network to address the data manipulation of the collectives operations. In order to ensure maximum scalability, Mellanox switches support the highest numbers of multicast groups, 300% higher than any other switch solution. Since FCA utilizes the interconnect hardware engines, it reduces the collectives runtime, increases the CPU availability to the application and allows overlap of communications and computations with asynchronous collective operations.

- **GPUDirect** - The system architecture of a GPU-CPU server requires the CPU to initiate and manage memory transfers between the GPU and the InfiniBand network. GPUDirect enables GPUs to transfer data to pinned system memory that Mellanox InfiniBand ConnectX HCAs can access directly and transfer over the network (with MPI communications). Mellanox HCAs are the solution that can read and write data directly from the GPU pinned memory (in the host memory) with zero buffer copies and no host CPU involvement due to the complete transport offload implementation and the native RDMA support. RDMA technology is the key technology for GPUDirect which reduced MPI communication for GPU workloads by 30%. Applications performance results demonstrate performance include from 20 to 35%.
6. Conclusion

The previous chapters describe the use cases selected by ParaPhrase partners for the three main application areas, Machine Learning, Erlang parallelization support, and High Performance Computing. Given these application areas and the expertise of the three main contributing partners, a wide field is covered, of different use cases, requirements with respect to ParaPhrase technology, and ways of dissemination of the results by actual usage of the technology.

This selection of use cases reflects the current status of work in WP6; some factors will lead to the continuous adaptation of the selected use cases. For one, they are dependent on available technology, most notably initial patterns (planned for milestone MS16 in month 12) and run time system for an appropriate parallel architecture (MS2 in month 19). And another factor is the dynamic nature of practical projects with customers; depending on the progress of such projects selected as use cases and for dissemination, and depending on new opportunities for projects with high impact, these use cases will be subject to adaptation or change.

Regarding the continuation of work in WP6, the availability of deliverable D2.1 allows to begin making concepts about porting of the first selected applications. This will include the transfer of knowledge from the technical work package partners, and later also the development of concepts for performance analysis and comparison with traditional sequential and parallel approaches. Actual implementation of the use cases can start with the availability of the first parallel patterns (or at least their exact interfaces in Erlang and/or C++).
A. Additional Use Case Potential

This annex collects further information about potential use cases, which do not belong to the core set. It is provided to indicate the range of applications which can benefit from ParaPhrase technology in future, and which will be part of the dissemination and/or exploitation activities.

A.1 Further Machine Learning Use Cases

A.1.1 Machine Learning Framework (MLF)

For the Upper Austrian company uni software plus SCCH is developing the Machine Learning Framework MLF [18], an extension for the software package Wolfram Mathematica to enable efficient use of machine learning methods and corresponding visualizations. uni software plus is distributor of Wolfram Mathematica and MLF. MLF is used worldwide, and SCCH has used and is using it in several data analysis projects with customers.

It is planned, to incorporate parallelized methods relevant for the previously described application areas and developed in ParaPhrase into MLF, and to parallelize existing methods of MLF. This way, also current and future customers of SCCH and uni software plus can benefit from ParaPhrase technology and accelerated learning.

In addition, uni software plus plans to restructure and modernize the MLF, allowing access to the implemented methods not only from Mathematica, but also from other languages, most notably C++ and python. This will further increase the range of potential users of parallelized machine learning methods.

Methods with potential for parallelization could be the following:

**Linear Algebra** A large class of Machine Learning algorithms is heavily relying on linear algebra. Such methods have a high potential for parallelization, and would well be suited for GPGPU computation, because of their data parallel computation structure.

In particular, algorithms computing sparse models are being considered. Sparseness is a principle applied in many Machine Learning algorithms to reduce the number of parameters in models, and their complexity. It is used also for computational reasons, for speedup of the application of learned models. But the main reason to use sparseness is that models using little
complexity to model given dependencies are often more robust wrt. performance on previously unseen data, than those using a higher degree of complexity. Enforcing sparse solutions to a modeling problem often makes optimization approaches more complex. Because of changes in the characteristics of the optimized cost function, the optimization algorithms to use often have to be adapted or changed as well. This results in the need for acceleration of sparse model learning.

Because of the data parallel nature of linear algebra algorithms and their application in Machine Learning we expect to make heavy use of the patterns Map, Reduce, and Farm.

**Decision Trees and fuzzyfied variants** These are trained on a dataset to predict a target feature from input features. Given the training dataset, they recursively split the contained samples in two or more branches, resulting in a tree of splits and their criteria. The split criteria are learned to optimize the predictive performance of the model. This algorithm could well be implemented using the Divide & Conquer high level pattern described in Deliverable D2.1.

**Graphical Models** Graphical Models, especially Bayesian Networks [14] allow a natural integration of expert knowledge and learned dependencies into a unified model. Because of their flexibility, they are applied to problem settings ranging from standard classification to fault diagnosis and explanation. Both learning of the structure and parameters, as well as applications of the learned models to perform inference can be computationally demanding tasks, and benefit from parallelization.

These models involve linear algebra computations (see above). Additionally on a higher level, we expect the Data Dependency pattern from Section 2.2 of Deliverable D6.1 would be highly relevant, because of the graphical nature with variable dependency structures of Bayesian networks and most of the algorithms working on them.

### A.1.2 Local Weather Prediction

BlueSky [4] is an Upper Austrian company offering individualized weather forecasts and services. In cooperation with them, SCCH has developed a system which adapts global weather forecasts to specific locations using machine learning and historic local weather information. Because of the high number of prediction locations and high computational demands due to the amount of data, SCCH plans to apply parallelization for multicore machines and clusters of such to different components of this system, namely model selection, feature selection, sparse model learning. Furthermore, similar methods are planned to be applied in future to projects involving prediction of energy needs dependent on weather (e.g. for heating in households), and prediction of available energy (e.g. from renewable energy sources).

More and more solar and wind energy units are being deployed, which are also used as sensor networks to collect data for future improvement of deployment,
operation, and maintenance of such units. SCCH is cooperating with a manufacturer and maintainer of solar units, who has access to a large collection of such data. Topics relevant in this cooperation are the prediction of power output from solar units during the next hours or days (taking local weather into account), using their data to improve local weather predictions, and performing fault diagnosis and prediction for the deployed units.

Parallelization in this use case would be useful on two levels. On the higher level, a parallel learning of several independent models allows coarse grained parallelization, by using a Farm of workers pattern. This would be possible for learning models for different locations in parallel, for performing architecture selection on different machine learning models in parallel, for selection of the best set of features to use in learning. On the lower level, the individual machine learning methods used (and compared in architecture selection) can themselves be parallelized using patterns appropriate for them (see description of algorithms in Section A.1.1).
Bibliography


29