Grid Programming for Heterogeneous Environments - The Grid-Occam Project

Peter Tröger and Kai Köhne

Hasso-Plattner-Institute at University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3,
14482 Potsdam, Germany
{peter.troeger, kai.koehne}@hpi.uni-potsdam.de

Abstract. Today, the implementation and execution of a distributed parallel grid application confronts the developers with heterogeneous infrastructures and non-deterministic execution behavior. We present the current status of our Grid-Occam project, which introduces a well-founded parallel programming language to current distributed heterogeneous computing environments. We show how the Grid-Occam compiler and the pertinent runtime libraries can support developers for heterogeneous grid environments, while keeping the expressive clearness of the original Occam language. We give a short overview about according teaching activities and the BB-Grid testing environment.

1 Introduction

The last years showed a high impact of heterogeneous, distributed grid environments in both research and industry. The grid-related research provides new approaches and concepts regarding resource allocation, scheduling, and coordination. However, most grid environments still rely on low-level sequential programming paradigms. Programmers usually need to consider specifics of the target execution node, and must take care of infrastructure properties in their programming and deployment strategies. Due to the fact that distributed executions are usually difficult to predict and supervise, the detection of incorrect behavior caused by race conditions can take a lot of time in the development.

In the context of this situation, the Grid-Occam project[1] aims at the introduction of the traditional Occam programming language to current distributed grid environments. Within this paper, we introduce an extended version of the Occam language. It keeps compatibility to existing Occam programs and library code, while offering additional constructs and data types in order to consider specific demands of modern grid environments.

In our concept, Grid-Occam binaries are executed in modern virtual machines like Java or .NET, which ensures the independence from infrastructural conditions in a heterogeneous distributed system. Exchangeable runtime libraries encapsulate the communication and placement technologies and therefore allow the developer to concentrate on the implementation of her algorithm.

Due to the complete abstraction from runtime environment conditions, we see Grid-Occam both as a parallel programming language and as a coordination language for the distributed enactment of existing services. While this paper concentrates on our Java-based implementation of the Grid-Occam approach, we already demonstrated the feasibility of the concept with a .NET-based solution for the Rotor framework [24].

The remaining sections are structured the following way: Section 2 provides a short overview of state-of-the-art in grid programming. The following section 3 introduces our Grid-Occam idea, the according Occam language extensions and the design of the runtime libraries. Section 4 explains our evaluation efforts in according lectures and the BB-Grid testbed infrastructure. Section 5 concludes the paper.

[1] http://www.grid-occam.org/

2 Grid for End Users - State of the Art

Grid technology is currently becoming commonplace in both scientific and commercial environments. Existing grid middleware, such as the Globus toolkit [15] and Unicore [8], already solve the basic questions of running jobs, transfering large data sets and managing security in cross-organizational grid environments. The grid community provides a set of client libraries for such installations, e.g. GAT/SAGA [27], DRMAA [14] or COG [28]. Portal toolkits (e.g. GridSphere) make use of these libraries in order enable the creation of application-specific web frontends for a grid environment.

The Global Grid Forum (GGF) works on the standardization of the Grid middleware interfaces, in order to ensure interoperability across organizational borders. Adopted and largely accepted standards are usually aligned to the Open Grid Services Architecture [16], an architectural blueprint for service-oriented grid environments.

Developers of grid applications rely on well-known paradigms for the development of their distributed parallel application. Popular toolkits are based on grid-enabled message passing (MPICH-G2 [9]), master-worker paradigm (AMWAT [13]), remote procedure calls (GridRPC [19], NetSolve [5],) and service-oriented interfaces (WSRF [20][16]). The toolkits either concentrate on the support for high-performance or for high-throughput / parameter-sweep applications.

Several projects, such as GridLab [11], Cactus [12] and GrADS [10], establish their complex domain-specific infrastructures based on the Globus middleware services. However, the coordinated execution of parallel activities in an unpredictable and highly dynamic grid environment remains a major issue. Recent research works on supporting complex job workflow descriptions (Condor DAGMAN [6], K-Wf Grid , myGrid), which are usually based on petrinet descriptions for complex tasks.

3 The Grid-Occam Approach

Developers for computational applications usually want to concentrate on the aspects of parallelization for their algorithm, not on specific technical issues for a particular execution environment. The Grid-Occam approach aims at the introduction of an extended version of the proven Occam programming language to current grid environments. With respect to the largely heterogeneus and ever-changing runtime environments, we aim at a clear separation of algorithm-related and infrastructure-related issues for grid developers. We see a clear benefit in the first-level support of parallelism in the Occam programming language.

The original Occam programming language [4] is based on Hoare's idea of "Communicating Sequential Processes" (CSP) [2], a process algebra for the description of parallel activities. The CSP ideas has influenced the design of languages like Ada [3], and libraries implementing the model are available for modern (sequential) programming languages, like C++ (Kent C++ CSP Library) and Java (JCSP [21, 1], Groovy [18]).

Due to the fact that the Occam language was directly derived from the CSP concepts, it still allows to perform formal proofs for the correctness of Occam algorithms and programs. With the application of Occam and CSP in modern distributed environments, these environments can benefit from the careful design of the language for the prevention of unwanted race-condition situations in parallel activities. Special properties of the specific runtime environment, like long-distance communication paths, heterogeneous execution platforms, and varying timing does not influence the behavioral semantic of an Occam application. For this reason, the language can act as high-level description of coordination aspects in a widely-distributed execution environment,

as well as providing an easy teaching platform for the programming concepts in distributed environments.

In order to achieve the distinction of the programming efforts and the runtime environment part, we introduce the notion of an *Occam runtime library*. This library is utilized by the compiled Grid-Occam application and is responsible for encapsulating all infrastructure-dependent parts from the overall program logic. In an ideal case, a programmer can adopt her already existing Occam program by simply exchanging this runtime library (e.g. thread runtime library vs. MPI runtime library), without any need for changes in the program code. New runtime libraries for different infrastructures can be developed without a need for changes in the Grid-Occam compiler itself.

Our concept supports the nested usage of runtime libraries with different granularity levels (see 3.2). An Occam process for a particular type of virtual processor, for example a cluster node, can itself be executed on a network of lower granularity virtual processors, for example with multiple threads.

The following sections provide an overview of our grid-enabled version of the Occam language and the according runtime library concept.

3.1 Programming Language

Occam was largely supported by the company INMOS for the transputer systems [17], which contained high-performance microprocessors that support parallel processing through on-chip hardware [25]. The Occam programming language, in its latest official version 2.1, is documented in a reference manual [4] by SGS-THOMPSON. It follows a set of simple principles, which are mostly derived from both the concepts of CSP and the hardware restrictions of the transputer platform. The Grid-Occam language therefore omits peripheral parts, and relaxes some of the limitations of the original Occam 2.1 standard [26]. We also carefully extended the language syntax, while paying attention that the purity, soundness and formal foundation of the language is not harmed.

Programs in Occam are build of processes. A *process* can be represented by an assignment, a channel input or a channel output.

A complex process can be formed as sequence (`SEQ`) of processes, as loop, as parallel, or as alternating execution of primitive or other complex processes. A sequence of component processes is indicated with an indentation of two spaces. Conditionals (`IF` statement) and selections (`CASE` statement) are defined as in other programming languages, loops are available with the `WHILE` keyword.

Occam distinguishes between procedures and functions (also called value processes). An Occam procedure defines name and formal parameters for a given process, while functions act as own unit and return one or more results in a call. Functions are forbidden to communicate or assign to free variables, which avoids side-effects in their execution.

A parallel process (`PAR`) lists up a number of processes which are executed in a concurrent manner. Like SEQ, the component processes are denoted by two spaces of indentation. The parallel statement simultaneously starts execution of all component processes. The entire statement is complete when the last component completes. The order of listed processes is irrelevant for the effect of the parallel. Since the parallel itself again is an Occam process, it can be used with all other Occam control structures (e.g. in conditionals). The nesting of parallels does not change semantics, but becomes relevant in case of an explicit mapping of Occam processes to physical processors.

Parallel processes communicate through unbuffered, unidirectional channels. A *channel* reflects a communication path of a specific type, where one process sends a value and the other

process receives it. A *channel input* statement (`channel ? variable`) receives a value from a channel and writes it, as an atomic operation, to a variable of the same type as the channel (*channel protocol*). A *channel output* action (`channel ! expression`) transmits the output of an expression to a channel, which is again reflected as one operation:

```
PAR
  SEQ
    keyboard ? var1
    to.compute ! var1
    from.compute ? var1
    display ! var1
  SEQ
    to.compute ? var2
    var2 := var2+1
    from.compute ! var2
```

In this example, two concurrent processes are defined. The first process takes an input from the special channel 'keyboard' and transmits it to the second process through channel 'to.compute'. The second process receives the value, performs the computation and puts back the result. The communication through a channel uses the rendezvous style - an output operation waits for an according input operation, and an input operation waits for the according output operation. After the transfer of the value, both parallel components continue to execute. The name of a channel may be used in a parallel only in one process for input and in another process for output, which ensures a consistent and predictable behavior of the resulting parallel application.

For our extended Grid-Occam language, we introduced the concept of channel direction specifiers, as proposed by Welch et.al. [23]. It allows the syntactical differentiation of channel endpoints. Native Occam channels are unidirectional, having an input and an output endpoint. In our enhanced version of Occam, the direction is being made explicit by appending a question or explanation mark to the channel name.

We extended the language with a convenient syntax to define bidirectional channels. A channel declared as `BCHAN OF INT i` allows an easy realization of the often needed request response pattern. It can be easily shown that this channel can be mapped to the definition and usage of two different unidirectional channels, and therefore does not compromise the foundations of Occam.

Occam supports several primitive data types, named data types, literals, arrays and records. We decided to add support for the definition of opaque data types (`OPAQUE TYPE X:`), which cannot be manipulated within the Occam language, but can be transferred between coordinated external processes through Occam channels. This language extension, together with a functional enhancement of the Occam channel implementation and the support for bidirectional communication, allows the formulation of grid service workflows as Occam program. The program can read the output of one grid service channel output, and uses it as input for the next grid service channel.

Another minor extension of the Occam type system is the introduction of a `STRING` data type, representing Unicode strings of the underlying execution environment. Occam 2.1 represented strings as array of bytes.

The original Occam compiler for transputer allowed to formulate inline placement instructions for parallel processes. While the description format changed with the different Occam versions, the all where related to the concept of virtual process topology [7]. Our Grid-Occam

compiler and runtime libraries support this annotation of processes, in order to allow the formulation of an optimized placement strategy for the application developer.

3.2 Runtime Libraries

The Grid-Occam tool chain consists of the Grid-Occam language compiler, together with exchangeable Occam runtime libraries that encapsulate all infrastructure-dependent parts of the program execution. We identified 4 classes of runtime libraries, which differ in their level of process granularity - threads, processes, cluster nodes, and grid nodes. Every level of granularity has its own concepts of intra-process communication (e.g. MPI in cluster environments) and virtual processors (e.g. MPI rank number).

Runtime libraries of different granularity levels can be used in a nested way (see figure 1). An Occam process for a particular type of virtual processor, for example a cluster node, can itself be executed on a network of higher granularity virtual processors, for example with multiple threads.

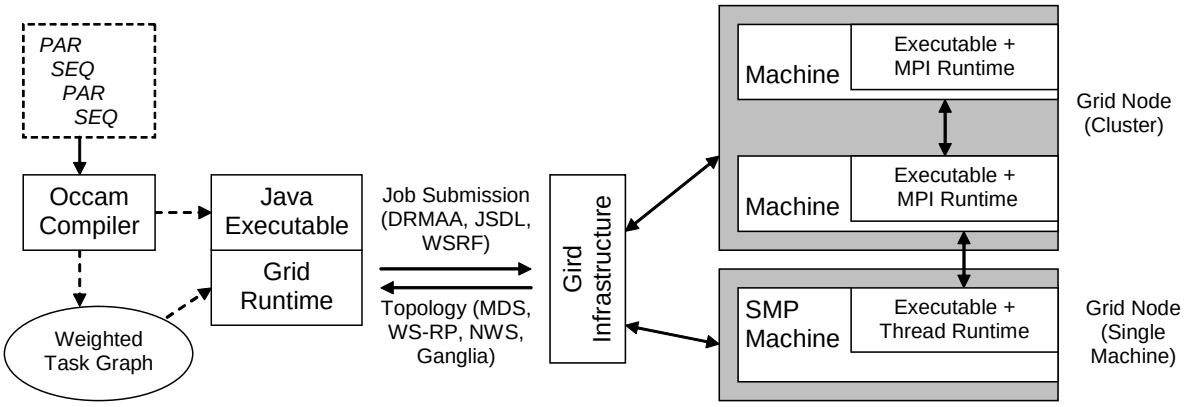

Fig. 1. Overview of the Grid-Occam architecture

Our concept of Grid-Occam also includes the generation of data-flow information by the compiler, which can be used by the runtime libraries for resource scheduling and placement issues. First basic implementations in the actual Grid-Occam compiler export a XML-based description of parallel processes and their interconnection through static channels. With increasing communication efforts in the runtime granularity levels, the best-possible consideration of network interconnection characteristics becomes more and more crucial. The task-graph information can act as helpful source of information for placement and resource binding decision in the runtime library.

Currently we have running versions of thread, remoting and MPI runtime libraries for the discontinued .NET compiler. For the Java compiler, different versions of thread, RMI, and MPI-based runtime libraries are also available.

... Gestaltung RT Interface ...

4 Evaluation

4.1 Example applications

Our work on the Grid-Occam toolkit is constantly accompanied by practical experiments with small simulation examples. Beside the still growing set of small compiler test cases, we try to

identify interesting simulation problems which can give a better indication of Occam's feasability for distributed parallel applications.

One of the larger example programs simulates macroscopic properties of a low energy ion scattering (LEIS) experiment. Helium ions are scattered off a target surface with a grazing incidence angle. After the scattering process the helium particle is neutralized and excited. Photons are emitted from the excited helium atoms. The position of the photon emission is calculated by the application and stored in a two-dimensional array. A light intensity map is generated by calculating a larger set of these independant processes. (see also [22]).[2].

We experienced that most sequential parts of the program were portable in a relatively mechanical way. Most efforts where spend for the parallization of the underlying algorithm, especially with respect to the Occam channel concept.

The feasibility of Occam as coordination language was tested with the parallelization of an existing Python program for Cornways *Game of Life* algorithm. *ToDo: Kai - Case Study Dokument verwursten*

4.2 Grid-Occam in the Curriculum

The Grid-Occam research project is accompanied by ongoing lectures in master studies curriculum. Students are enabled to apply basic concepts of distributed environments on their own version of a Grid-Occam runtime library.

The first lecture in 2004 concentrated on the development of a .NET based implementation. Five groups of 2-3 students designed an Occam-to-CSharp compiler, together with a runtime library based on .NET Remoting technology. The lecture enabled the students to work practically with compiler toolkits (LEX, YACC, Coco/R, Kimwitu, ANTLR), as well to study the design and implementation of compilers for parallel applications. The architectural and implementation work concluded in the .NET Grid-Occam toolkit, which was presented on the second Microsoft Rotor RFP workshop.

The Grid-Occam lecture in 2005 focused on Java as the target execution platform for compiled Grid-Occam binaries. The most elaborate compiler, as presented in this paper, now acts as base for the overall research project. Other results are an Eclipse plugin for easier Occam development, as well as several optimized runtime libraries for RMI and MPI environments.

4.3 The BB-Grid Testing Environment

Practical experiments with the compiler prototype are performed in the BB-Grid testbed, an open infrastructure established by four institutions in Berlin and Brandenburg (BTU Cottbus, TU Berlin, University of Potsdam, Hasso-Plattner-Institute). The BB-Grid project allows an easy mutual sharing of heterogeneous resources for research and teaching purposes.

The participants of the BB-grid alliance aim at the integration of their existing compute resources (around 60 SMP nodes), in order to establish a real-word environment for research and teaching in the area of Grid Computing. The research activities of the regarding partners cover the areas of cluster management, fault-tolerant execution environments, distributed programming and predictable resource usage through best-effort SLA fulfillment.

Primary goals of the collaboration are easy access to mutual resources, as well as a hands-on testing environment for distributed computing. Beside the Grid-Occam tests, the established infrastructure is currently also used in multiple teaching activities (lectures, projects). New institutions and partners are invited to join the BB-Grid project for a cost-neutral mutual usage of distributed and heterogeneous computing resources. We established a decentralized Globus-based infrastructure (GT4), where the authorization of users remains at each partner's site.

[2] We would like to thank Dipl.-Phys. Michael Dirska for the explanation text

5 Conclusion

This paper presented our Grid-Occam architecture and tools, which aim at the introduction of an extended version of the proven Occam programming language to current distributed environments. The provisioning of parallelism as first-level language construct, together with a careful extension of the language semantics, leads to a powerful toolkit for the development of distributed parallel applications.

The Grid-Occam compiler relies on a flexible abstraction of the execution infrastructure. It allows an Occam application to be executed on different kinds of runtime environments, without modification of the compiler or the application.

The Occam language rules and restrictions, mainly reasoned by the formal foundation on CSP, enable the compiler to check for possible race-conditions at compile-time. Beside the given advantages, interviews and demonstrations for possible user groups showed a broad interest in the 'revival' of a well-known language for today's distributed systems.

Our ongoing activities will focus on two major issues: We are working on the further improvement of the Grid-Occam compiler in order to achieve a complete support of all Occam language features. This includes also the generation and consideration of task graphs for parallel Occam code.

The work on runtime libraries will concentrate on the further improvement of the MPI versions and the initial demonstration of a Grid runtime library. In the context of the "Adaptive Services Grid" [3] EU project, the results will be used for the the implementation of complex resource coordination tasks with Grid-Occam applications.

References

1. Briant Vinter and Peter H. Welch. Cluster Computing and JSCP Networking. In James Pascoe and Peter Welch and Roger Loader and Vaidy Sunderam, editor, *Communicating Process Architectures*, volume 60 of *Concurrent Systems Engineering Series*, Amsterdam, 2002. IOS Press.
2. C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.
3. C. J. Fidge. A Formal Definition of Priority in CSP. *ACM Transactions on Programming Languages and Systems*, 15(4):681–705, September 1993.
4. C.A.R. Hoare. *Occam 2 Reference Manual: Inmos Limited*. Prentice-Hall, 1988.
5. H. Casanova and J. Dongarra. Netsolve: A network-enabled server for solving computational science problems, 1997.
6. Condor Team. *Condor Manual*. University of Wisconsin-Madison, 2004.
7. D. Loveman. High Performance Fortran. *IEEE Parallel & Distributed Technology*, pages 25–42, February 1993.
8. Dietmar W. Erwin and David F. Snelling. UNICORE: A Grid Computing Environment.
9. I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC'98*. ACM Press, 1998.
10. Francine Berman and Andrew Chien and Keith Cooper and Jack Dongarra and Ian Foster and Dennis Gannon and Lennart Johnsson and Ken Kennedy and Carl Kesselman and John Mellor-Crumme and Dan Reed and Linda Torczon and Rich Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15(4):327–344, 2001.
11. G. Allen and K. Davis and K. Dolkas and N. Doulamis and T. Goodale and T. Kielmann and A. Merzky and J. Nabrzyski and J. Pukacki and T. Radke and M. Russell and E. Seidel and J. Shalf and I. Taylor. Enabling Applications on the Grid: A GridLab Overview, 2003.
12. Gabrielle Allen and Werner Benger and Thomas Dramlitsch and Tom Goodale and Hans-Christian Hege and Gerd Lanfermann and Andre Merzky and Thomas Radke and Edward Seidel and John Shalf. Cactus Tools for Grid Applications. *Cluster Computing*, 4(3):179–188, 2001.
13. Gary Shao. Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources, 2001.
14. Hrabri Rajic and Roger Brobst and Waiman Chan and Fritz Ferstl and Jeff Gardiner and Andreas Haas and Bill Nitzberg and John Tollefsrud. Distributed Resource Management Application API Specification 1.0. http://forge.ggf.org/projects/drmaa-wg/, 2004.
15. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[3] http://www.asg-platform.org/

16. I. Foster and C. Kesselman and J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, 2002.
17. Ian Graham and Tim King. *The Transputer Handbook*. Prentice Hall, January 1991.
18. Jon Kerridge and Ken Barclay and John Savage. Groovy Parallel! A Return of the Spirit of occam? In Jan Broenik et al, editor, *Communicating Process Architectures*, volume 63 of *Concurrent Systems Engineering Series*, Amsterdam, September 2005. IOS Press.
19. K. Seymour and H. Nakada and S. Matsuoka and J. Dongarra and C. Lee and H. Casanova. An Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *3rd International Workshop on Grid Computing*, November 2002.
20. Karl Czajkowski and Donald F. Ferguson and Ian Foster and Jeffrey Frey and Steve Graham and Igor Sedukhin and David Snelling and Steve Tuecke and William Vambenepe. The WS-Resource Framework. http://globus.org/wsrf/specs/ws-wsrf.pdf, 05 2004.
21. Neil Brown and Peter Welch. An Introduction to the Kent C++CSP Library. In Jan F Broenink and Gerald H Hilderink, editor, *Communicating Process Architectures*, volume 61 of *Concurrent Systems Engineering Series*, Amsterdam, The Netherlands, September 2003. IOS Press.
22. A. Nrmann, M. Dirska, J. Manske, G. Lubinski, M. Schleberger, and R. Hoekstra. Spin-sensitive electron capture into excited states as a probe to investigate magnetic surfaces. *Surface Science*, 398(1-2):84–90, 1998.
23. Peter H. Welch and others. The KRoC Home Page, October 24, 2005.
24. Peter Tröger and Martin von Löwis and Andreas Polze. The Grid-Occam Project. In *GSEM 2004 - Grid Services Engineering and Management, First International Conference*, pages 151–164, Erfurt,Germany, September 2004. Springer, Lecture Notes in Computer Science.
25. Ram Meenakshisundaram. Transputer Information Home Page. http://www.classiccmp.org/transputer/.
26. SGS-THOMPSON Microelectronics Limited. *Occam 2.1 Reference Manual*, May 1995.
27. Tom Goodale and Keith Jackson and Stephen Pickles. Simple API for Grid Applications (SAGA) Working Group. Global Grid Forum Recommendation Document Draft.
28. Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.