A Data Parallel Java Client-Server Architecture for Data Field Computations over $\mathbb{Z}^n$

Jean-Louis Giavitto, Dominique De Vito, Jean-Paul Sansonnet

LRI u.r.a. 410 du CNRS, Bâtiment 490 – Université de Paris-Sud,
F-91405 Orsay Cedex, France.
email: {giavitto|devito}@lri.fr

Abstract. We describe `FieldBroker`, a software architecture, dedicated to data parallel computations on fields over $\mathbb{Z}^n$. Fields are a natural extension of the parallel array data structure. From the application point of view, field operations are processed by a field server, leading to a client/server architecture. Requests are translated successively in three languages corresponding to a tower of three virtual machines processing respectively mappings on $\mathbb{Z}^n$, sets of arrays and flat vectors in core memory. The server is itself designed as a master/multithreaded-slaves program. The aim of `FieldBroker` is to mutually incorporate approaches found in distributed computing, functional programming and the data parallel paradigm. It provides a testbed for experiments with language constructs, evaluation mechanisms, on-the-fly optimizations, load-balancing strategies and data field implementations.

1 Introduction

Collections, Data Fields and Data Parallelism. The data parallel paradigm relies on the concept of *collection*: it is an aggregate of data *handled as a whole* [6]. A *data field* is a theoretically well founded abstract view of a collection as a function from a finite index set to a value domain. Higher order functions or intensional operations on these mappings correspond to data parallel operations: point-wise applied operation (map), reduction (fold), etc. Data fields enable to represent irregular data by using a suitable index set. Another attractive advantage of the data field approach, in addition to its generality and abstraction, is that many ambiguities and semantical problems of "imperative" data parallelism can be avoided in the declarative framework of data fields.

A Distributed Paradigm for Data Parallelism. Data parallelism was motivated to satisfy the increasing needs of computing power in scientific applications. Thus, the main target of data parallel languages has been supercomputers and the privileged linguistic framework was Fortran (cf. HPF). Several factors urge to reconsider this traditional framework:

– Advances in network protocols and bandwidths have made practical the development of high performance applications whose processing is distributed over several supercomputers (metacomputing).

– The widening of parallel programming application domains (e.g. data mining, virtual reality, generalization of numerical simulations) urges to use cheaper computing resources, like NOWs (networks of workstations).
– Development in parallel compilation and run-time environments have made possible the integration of data parallelism and control parallelism, e.g. to hide the communication latency with the multithreaded execution of independent computations.
– New algorithms exhibit more and more a dynamic behavior and perform on irregular data. Consequently, new applications depend more and more on the facilities provided by a run-time (dynamic management of resources, etc.).
– Challenging applications consist of multiple heterogeneous modules interacting with each other to solve an overall design problem. New software architectures are needed to support the development of such applications.

All these points require the development of portable, robust, high-performance, dynamically adaptable, architecture neutral applications on multiple platforms in heterogeneous, distributed networks.

Many of theses attributes can be cited as descriptive characteristics of distributed applications. So, it is not surprising that distributed computing concepts and tools, which precisely face this kind of problems, become an attractive framework for supporting data parallel applications. In this perspective, we propose `FieldBroker`, a client server architecture dedicated to data parallel computations on data field over $\mathbb{Z}^n$. Data field operations in an application are requests processed by the `FieldBroker` server.

`FieldBroker` has been developed to provide an underlying virtual machine to the $8_{1/2}$ language [5] and to compute recursive definitions of group based fields [2]. However, `FieldBroker` aims also to investigate the viability of client server computing for data parallel numerical and scientific applications, and the extent to which this paradigm can integrate efficiently a functional approach of the data parallel programming model. This combination naturally leads to an environment for dynamic computation and collaborative computing. This environment provides and facilitates interaction and collaboration between users, processes and resources. It also provides a testbed for experiments with language constructs, evaluation mechanisms, on-the-fly optimizations, load-balancing strategies and data field implementations.

2 A Distributed Software Architecture for Scientific Computation

The software architecture of the data field server is illustrated by Fig. 1 right. Three layers are distinguished. They correspond to three virtual machines:

– The **server** handles requests on *functions over* $\mathbb{Z}^n$. It is responsible for parallelization and synchronization between requests from one client and between different clients.

– The **master** handles operations between sets of arrays. This layer is responsible for various high-level optimizations on data field expressions. It also decides the load balancing strategy and synchronizes the computations of the slaves.
– The **slaves** implement sequential computations over contiguous data in memory (vectors). They are driven by the master requests. Master requests are of two kinds: computations to perform on the slave's data or communications (send data to other slaves; receives are implicit). Computations and communications are multithreaded in order to hide communication latency.

Our software architectures corresponds to a three levels language tower. Each language specifies the communications between two levels of the architecture and describes a data structure and the corresponding operations. Three languages are used, going from the more abstract $\mathcal{L}_0$ (client view on a field) to $\mathcal{L}_1$ and to the more concrete $\mathcal{L}_2$ (in core memory view on a field). The server-master and the slave programs are implemented in Java. The rationale of this design decision is to support portability and dynamic extensibility. The expected benefits of this software architecture are the following:

– **Accessibility and client independence:** requests for the data field computation are issued by a client through an API. However, because the slave is a Java program, Java applets can be easily used to communicate with the server. So, an interactive access could be provided through a web client at no further cost. In this case, the server appears as a data field desk calculator.
– **Autonomous services:** the server lifetime is not linked to the client lifetime. Thus, implementing persistence, sharing and checkpointing will be much easier with this architecture than with a monolithic SPMD program.
– **Multi-client interactions:** this architecture enables applications composition by pipelining, data sharing, etc.

The figure 1 illustrates that $\mathcal{L}_0$ terms are successively translated into $\mathcal{L}_1$ and $\mathcal{L}_2$ and that $\mathcal{L}_2$ terms are dispatched to the slaves to achieve the data parallel final processing. More details about these languages can be found in [1].

3 Conclusion

The aim of our first ongoing implementation is to evaluate the functionalities provided by such an architecture. At this stage, we have not pay attention to its performance which is certainly disappointing. A promising way to tackle this drawback consists in the use of *just-in-time* Java compiler that are able to translate Java bytecode into executable machine-dependent code.

`FieldBroker` integrates concepts and technics that have been developed separately. For example, relationships between the definition of functions and data fields are investigated in [4]. A proposal for an implementation is described in [3] but focuses mainly on the management of the definition domain of data fields.

One specific feature of `FieldBroker` is the use of heterogeneous representations, i.e. extensional and intensional data fields, to simplify field expressions.

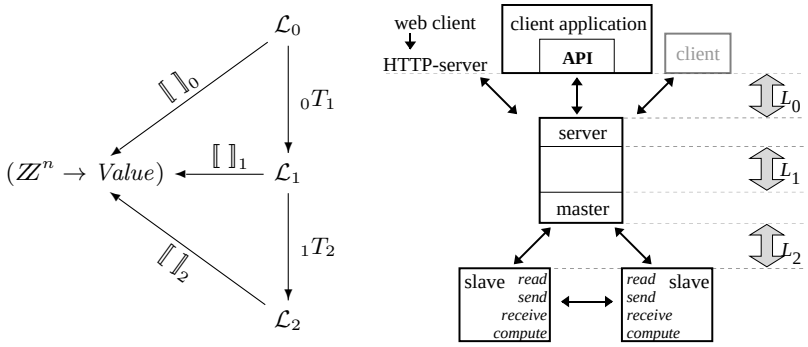

Fig. 1. ***Left:*** *Relationships between field algebras* $\mathcal{L}_0, \mathcal{L}_1$ *and* $\mathcal{L}_2$. ***Right:*** *A client/server-master/multithreaded-slaves architecture for the data parallel evaluation of data field requests.* The software architecture described on the right implements the field algebras sketched on the left. Functions ${}_iT_{i+1}$ are phases of the evaluation. The functions $[\![\ ]\!]_i$ are the semantic functions that map an expression to the denoted element of $\mathbb{Z}^n \to Value$. They are defined such that the diagram commutes, that is $[\![e_i]\!]_i = [\![{}_iT_{i+1}(e_i)]\!]_{i+1}$ is true for $i \in \{0, 1\}$ and $e_i \in \mathcal{L}_i$. This property ensures the soundness of the evaluation process.

Clearly, the algebraic framework is the right one to reason about the mixing of multiple representations.

References

1. J.-L. Giavitto and D. De Vito. Data field computations on a data parallel Java client-server distributed architecture. Technical Report 1167, Laboratoire de Recherche en Informatique, Apr. 1998. 9 pages.
2. J.-L. Giavitto, O. Michel, and J.-P. Sansonnet. Group based fields. *Parallel Symbolic Languages and Systems (International Workshop PSLS'95)*, vol. 1068 of *LNCS*, pages 209–215, Beaune (France), 2-4 October 1995. Springer-Verlag.
3. J. Halén, P. Hammarlund, and B. Lisper. An experimental implementation of a higly abstract model of data parallel programming. Technical Report TRITA-IT 9702, Royal Institute of Technology, Sweden, March 1997.
4. B. Lisper. On the relation between functional and data-parallel programming languages. In *Proc. of the 6th. Int. Conf. on Functional Languages and Computer Architectures*. ACM, ACM Press, June 1993.
5. O. Michel. Introducing dynamicity in the data-parallel language 81/2. *EuroPar'96 Parallel Processing*, vol. 1123 of *LNCS*, pages 678–686. Springer-Verlag, Aug. 1996.
6. J. M. Sipelstein and G. Blelloch. Collection-oriented languages. *Proceedings of the IEEE*, 79(4):504–523, Apr. 1991.