Embedding Data Parallelism in Sequential Object Oriented Languages (Position Paper)

J.-M. JÉZÉQUEL
F. GUIDEC

I.R.I.S.A. Campus de Beaulieu
F-35042 Rennes Cedex, FRANCE
E-mail: guidec@irisa.fr
Tel: +33 99 84 71 91 ; Fax: +33 99 38 38 32

The spreading of Distributed Memory Parallel Computers (DMPCs) is hampered by the fact that writing or porting application programs to such architectures is a difficult, time-consuming and error-prone task. Nowadays software environments for commercially available DMPCs mainly consist of libraries of routines to handle communications between processes. We believe that the reuse of carefully designed software components could help to manage the complexity of concurrent programming.

Our approach aims at embedding the scalable data parallelism programming model in an OOL. This approach is orthogonal to those where objects can be made active and invocations of methods result in actual message passing communications (sometimes referred to as functional parallelism, because the definition of processes is determined by the decomposition into sub-tasks). We believe that functional parallelism does not allow an easy and efficient mapping of large scale computing algorithms onto the very high number of processors that are made available in a DMPC.

We focus on the data parallelism model associated with a SPMD mode of execution (Single Program Multiple Data). This approach seems to be rather natural in an OO context, since object oriented programming usually focuses on data rather than on functions. Furthermore, the SPMD mode of execution appears as an attractive one because it offers the conceptual simplicity of the sequential instruction flow, while exploiting the fact that most of the problems running on DMPCs involve large amounts of data in order to generate usefull parallelism.

Each processor of the DMPC runs the same program, which corresponds to the initial sequential program written by the application programmer, on its own data partition. Our goal is to completely hide the parallelism to the user, who still views his program as a sequential one: the parallelism is automatically derived from the distribution of data over the DMPC, thus leading to a regular and scalable kind of parallelism.

We implemented these ideas in EPEE (Eiffel Parallel Execution Environ-

ment): data distribution and parallelism are totally embedded in standard language structures (classes) using nothing but already existing language constructions. EPEE is based on Eiffel because this language offers all the concepts we need, using a clearly defined syntax and semantics. However our approach is not strongly dependent on Eiffel; it could be implemented in any OOL featuring strong encapsulation, static type checking, multiple inheritance, dynamic binding and genericity.

We distinguish two levels of programming in EPEE: the *client* level and the *designer* level. Our aim is that at the client level, nothing but performance improvements appear when running an application program on a DMPC. We would like these performance improvements to be proportional to the number of processors of the DMPC (linear speed-up), which would guarantee scalability.

The designer of a parallelized class is responsible for implementing general data distribution and parallelisation rules, thus ensuring portability, efficiency and scalability, while preserving a "sequential-like" interface for the user. To achieve this goal, the designer selects interesting classes to be data parallelized, *i.e.* classes aggregating large amounts of data, such as classes based on *arrays, sets, trees, lists...* For each of these classes, one or more distribution policies are to be chosen and data access methods redefined accordingly, using the communication tools and distribution abstractions provided in the EPEE *distributed aggregate* class. This class abstracts the common parts of all distributed objects: it describes an abstract aggregate of generic data that can be spread across a DMPC, together with a set of methods to access its data transparently, to redistribute the aggregate, to perform a method on each of its sub-elements (partitions), and to compute any associative function on the aggregate. These methods are defined in an abstract way so that they can be reused and/or customized in classes inheriting from the distributed aggregate class. A parallelized class is therefore a class that is both a descendant of an original sequential aggregate model (*matrix, tree...*) and of the *distributed aggregate* class.

An EPEE prototype is available for Intel iPSC computers (iPSC/2 or iPSC/860 and very soon for Paragon XP/S) and networks of workstations above TCP/IP. We validated our approach through an experimentation with an implementation of distributed matrices using EPEE, and got interesting results.