Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

Intensional High Performance Computing

Pierre Kuonen[1], Gilbert Babin[2], Nabil Abdennadher[1], and Paul-Jean Cagnard[1]

[1] GRIP Research Group, Département d'informatique
Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland
{Pierre.Kuonen, Nabil.Abdennadher, Paul-Jean.Cagnard}@epfl.ch
[2] PARADIS Laboratory, Département d'informatique
Université Laval, Québec, Canada G1K 7P4
babin@ift.ulaval.ca

Abstract. In this paper, we describe how a metacomputing environment called *Web Operating System* (WOS™) together with a new programming paradigm called ParCeL-2 may be used to exploit available computing resources on a parallel/distributed environment. The main feature of the WOS™ is to manage contexts of execution (hardware, software, time, etc). The WOS™ fulfills users' requests while considering all possible execution contexts in order to provide the application with the best resources available. In the model presented, we assume that parallel/distributed HPC applications are written using ParCeL-2. The well defined computing model as well as the hierarchical syntactic structure of ParCeL-2 allow for an automatic adaptation, at execution time, of the size of the different parallel processes, depending on the context of execution. We have called this approach, derived from intensional logic : *intensional High Performance Computing (iHPC).*

1 Introduction

Until recently, the world of High Performance Computing (HPC) was mainly involved in solving huge numeric problems using matrix calculation (number-crunching). This fact may be attributed to two factors. First, traditional vector and parallel supercomputers were very expensive and especially well designed for the resolution of large numerical problems. Second, most potential users, having large enough budget for buying such machines, were people dealing with linear algebra problems as found in aeronautics, spatial and military industry, chemistry and fluid dynamics. As a consequence, the state-of-the-art in HPC has mainly relied on FORTRAN, inducing poor data structures and imposing poor software engineering approaches in the design of long programs. Application programmers were working in close relation with users of these applications and the main execution paradigm was the batch mode.

Since the beginning of the nineties, distributed computing is emerging and prototypes of parallel applications running on less costly local networks of workstations appeared. These applications were first realized using locally developed communication software and more recently using *de facto* standards such as the Parallel Virtual Machine (PVM) standard, thus popularising the message

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

passing parallel programming paradigm. This led to the now well established Message Passing Interface (MPI) standard. Following the development of Local Area Networks (LAN) during the eighties, Internet technology has popularized Wide Area Networks (WAN). Message passing programming, first localized on LAN and on massively parallel computers, is now moving to WAN. Habits of users are rapidly changing from a *program centric* vision to a *service centric* vision. Future users will require the realization of a given service in the most efficient environment presently available to them through a WAN.

This evolution in the user's needs has led to the creation of a new high performance computing paradigm called *metacomputing* [Buyya 99]. A major consequence of the emergence of this new paradigm in the world of HPC is the urgent need for new software environments to develop and execute HPC applications. Such environments should give access to existing and new parallel programming tools and allow for an efficient transparent remote execution of wide-area distributed HPC applications. Unfortunately, most of current tools available for High Performance Parallel/Distributed computing require that all the computing nodes be known in advance; each computer involved in the execution must be properly configured and the execution environment must usually know where the different processes of the parallel program will be executed. In this paper, we will show how a new parallel programming paradigm, called ParCeL-2 (Section 2), together with the WOS™ metacomputing environment (Section 3) leads to the concept of *intensional HPC* (Sections 4 and 5). We will show that intensional HPC (iHPC) is an elegant and powerful concept for the realization of HPC applications which are able, at execution time, to automatically adapt themselves to the context of the execution. This capability is of the highest importance for allowing HPC applications to benefit from the computing capacity offered by metacomputing environments since these environments are usually very dynamic and unstable.

2 ParCeL-2: a New Parallel and Distributed Programming Paradigm

More than a new parallel programming language, ParCeL-2 is a new parallel and distributed programming paradigm. Its objective is to provide a minimal set of new concepts to be added to a classical imperative programming language in order to allow an "intuitive" expression as well as an efficient execution of parallel and distributed applications. ParCeL-2 basically provides two main concepts:

- a well defined parallel computing model;
- a hierarchical syntactic structure.

These two new concepts can be integrated in any existing sequential imperative language.

2.1 The ParCeL-2 Computing Model

The computing model of ParCeL-2 is inspired from the Bulk Synchronous Parallel (BSP) model [Valiant 90]. In BSP, a parallel program consists of a set of parallel processes each executing a sequence of *supersteps*. A superstep is composed of two phases: a computation phase and a communication phase. Supersteps are separated by synchronization barriers. The execution of a program using the BSP model can be represented as in Fig. 1.

During the computation phase of a superstep, a process executes computations which only manipulate data local to this process. These data can be local variables or data that have been received from another process. A process can send data to other processes in the course of a computation phase but the actual transmission of data happens at the end of each superstep. No data are exchanged during computation phases; this means that data sent from a given process P_1 to another process P_2 during superstep s, will only be available to process P_2 at the beginning of the next superstep, that is superstep $s + 1$. It can be observed that this computing model is intrinsically a distributed memory Multiple Instruction Multiple Data model (MIMD-DM).

ParCeL-2 extends the BSP model with several new features. The most important ones are:

- The specification of the communications allowed between processes (links). These allowed communications are typed and directed links;
- A more complex synchronisation mechanism between processes. As opposed, to the BSP definition, ParCeL-2 allows processes to have synchronisation periods that are an integer multiple of the execution environment's global clock period.

A more complete presentation of the computing model of ParCeL-2 can be found in [Cagnard 00].

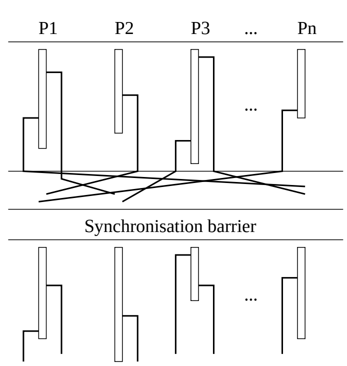

Fig. 1. Program execution in the BSP model

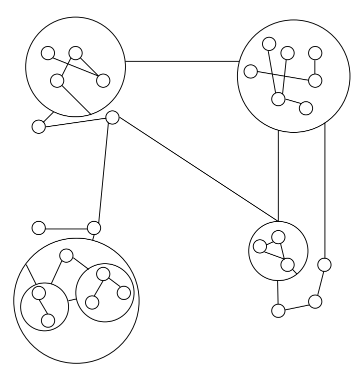

Fig. 2. Program execution on four processors.

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

2.2 The Hierarchical Syntactic Structure of ParCeL-2

A ParCeL-2 program can be represented as a directed graph where nodes symbolize processes and directed links, the communications allowed. A major difficulty when designing a parallel application is to determine the size of the nodes, i.e., the *grain of parallelism*. This problem is discussed in [Foster 94], where the author presents a design methodology for parallel programs. This method, called PCAM, is based on four steps:

- Partitionning;
- Communication;
- Agglomeration;
- Mapping.

The first two steps focus on parallelism and seek to discover algorithms that exhibit maximum parallelism. The third and fourth steps focus on performance issues. In particular, the agglomeration phase aims to determine the suitable size for the grain of parallelism.

Up to this point, we have seen a ParCeL-2 program as a flat structure where each process is at the same level. The main objective of the hierarchical syntactic structure of ParCeL-2 is to allow for aggregate processes. The ParCeL-2 paradigm is based on the assumption that, in most cases, the design of parallel applications can lead to programs composed of a large number of fine grained processes, that is, processes which execute only a small number of instructions. We call these small processes *cells*, by analogy to cellular automata. In ParCeL-2, aggregates of cells can be constructed in order to build abstract cells whose behavior is the result of the parallel execution of the cells they contain. In other words, a cell can be:

- a sequential process, called an *elementary cell*;
- an aggregate of cells, called a *complex cell*.

An illustration of such a program is given in Fig. 2

The concept of agglomeration of cells has two advantages. First, it provides means for information hiding because the rest of the program does not see if a cell is an elementary or a complex cell. Second, it allows creating cells of size greater than the elementary cells and consequently, adjusting the grain of parallelism.

We can deduce, from the description above, that ParCeL-2 is not an object oriented model in the strict sense, since it does not provide any inheritance between cell types, for example. However, it provides several features from object oriented models that are useful in software engineering for high performance computing. These features are information hiding and aggregation, achieved through the hierarchical syntactic structure, and the fact that only local data can be modified by a given cell, access to data from other cells happens only through well determined interfaces, the communication channels.

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

2.3 Compiling and Executing ParCeL-2 Applications

From what has been said above about ParCeL-2 computing model, we note that one of the main characteristics of ParCeL-2 is that processes can either be elementary cells, thus very similar to traditional processes in parallel programs, or complex cells, themselves composed of elementary or complex cells. Hence, a ParCeL-2 program can be represented by a structure like a multilevel-oriented graph where each node corresponds to a cell (elementary or complex) and each edge corresponds to a link. As a consequence, the main program is also a cell. Thus, the global structure of a ParCeL-2 program is intrinsically a recursive structure; a program may be viewed at any level of abstraction desired: from the extremely low level where one sees only elementary cells, up to the extremely high level where there is a single complex cell, the main program. Moreover, due to the well-defined parallel computing model of ParCeL-2, a compiler can easily compile any cell in a very efficient sequential process. Consequently, a ParCeL-2 program can be compiled and executed in a number of different ways. The first extreme case consists of compiling the main program (the outermost cell) into a single large sequential program which can be efficiently executed on a single processor. The other extreme case is to compile all elementary cells of the program in a different sequential process. All the intermediate solutions are also possible. In other words, we are now able to choose the grain of parallelism at compilation-time instead of design-time. This feature is central for the realization of intensional HPC.

3 Metacomputing

A metacomputer is a set of computers sharing resources and acting together to solve a common problem given by the user [Buyya 99]. A metacomputer comprises many computers and terabytes of memory in a loose confederation, tied together by a network. The user has the illusion of a single powerful computer; he manipulates objects representing data resources, applications or physical devices. A metacomputer is a dynamic environment that has some informal pool of independant nodes, each relying on its own complete operating system, and which can join or leave the environment whenever it desires. In other words, a metacomputer is an extremely moving environment where the real target architecture for an application is only known at execution-time.

3.1 The Web Operating System

The Web Operating System (WOS™) [Kropf 99] was developed to provide a user with the possibility to submit a service request without any prior knowledge about the service (where it is available, at what cost, under which constraints) and to have the service request fulfilled within the user's desired parameters (time, cost, quality of service, etc.). Three features make the WOS™ a very attractive environment for metacomputing:

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

1. *Open Access.* Most of the metacomputing projects, such as Globus [Foster and Kesselman 97; The Globus Project], Legion [Grimshaw *et al.* 97; Lindahl *et al.* 98], and NetSolve [Casanova and Dongarra 97], require login privileges and a global catalog of resources. This may be interesting for small networks but could be impractical for large ones. Contrary to this, the WOS™ uses distributed databases, called warehouses, that allow open access and search procedures. The search engine takes into account the dynamic nature of the Web. The WOS™ is based on a demand-driven computation model: users' queries are only processed when needed and prior results are stored in the warehouses, where they can be accessed later on.
2. *Universality.* The WOS™ aims to supply users with adequate tools that allow the implementation of specific services not initially foreseen. In order to achieve this goal, a generic service protocol (WOSP), provided by the WOS™, allows the WOS™ node administrators to implement a set of services, called a *service class*, dedicated to specific users' needs. WOSP is in fact a generic protocol defined through a generic grammar [Babin *et al.* 98]. A specific instance of this generic grammar provides the communication support for a service class of WOS™. This specific instance is also referred to as a *version of WOSP*; its semantics depends directly on the service class it supports. In other words, knowing a specific version of WOSP is equivalent to understanding the semantics of the service class supported by that version. Several versions of WOSP can cohabit on the same WOS™ node.
3. *Intensionality.* The WOS™ manages contexts: hardware, software, time, place, etc. The basic nature of the WOS™ is to answer users' requests while considering all these contexts; the WOS™ node will provide the best resources available, as a function of the current context, which always changes.

3.2 The Web Operating System and High Performance Computing

A version of WOSP, HP-WOSP [Abdennadher *et al.* 00], has been defined specifically to configure and execute HPC applications in the WOS™ environment. Specifically, it supports the communication requirements for HPC applications, which are:

- To locate potential computation nodes with the appropriate set of resources (hardware and software) and to reserve these resources. This is called the configuration stage;
- To launch the execution of the parallel program. This is called the execution stage.

4 Intensional HPC

The Intensional HPC (iHPC) approach is the integration of intensional logic, HPC, and metacomputing. Let us look at these three perspectives of iHPC in more details.

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

Intensional logic is based on the notion that an expression is always evaluated within a certain context [Paquet 99]. For instance, the expression "how is the weather?" will yield very different answers, depending on where and when it is evaluated. In most natural languages, such ambiguities are easily processed because most conversations are done within a specific context. It is not that easily handled in computer science, however. Intensional logic, and its specialized versions modal logic and temporal logic, provide the tools to manage such context-dependant expressions. In intensional logic, this context is represented as a multidimentional space, where many, possibly decomposable dimensions constrain the evaluation of an expression. This means that the expression might have an arbitrary large number of values, each one depending on a set of dimension values. Clearly, one cannot compute all the values of an expression, based on all its dimensions. Some computing "trick" must therefore be used. This "trick" is called *eduction* [Swoboda and Wadge 00]. Simply put, the eduction model of computation states that a value should only be computed when required. That value should be stored, so it can be reused instead of recomputed. For us, iHPC can only exist if all the concepts of intensional logic and eduction are applied.

From an HPC perspective, iHPC involves many changes in the way of developing HPC applications, which are usually parallel applications. For iHPC to be achieved, a parallel application should be described in such a way that all implementations could be extracted from the same design. An implementation is in fact a specialization of a design where all decisions about the specific (implementation and execution) constraints are made. The design description method selected should allow for multiple dimensions of constraints to be represented within the same design. Furthermore, an implementation should be automatically produced by setting all the constraints. Therefore, we need a compiler that can take as input a multidimensional design and all the values for the different constraints. We call such a compiler an *intensional compiler*.

However, to truly be intensional, and therefore to fully take advantage of eduction, an iHPC environment must wait until the last minute to compile the necessary pieces of code. This should occur during the configuration stage of the parallel execution. This is where metacomputing comes into play. Metacomputing tools can be used to evaluate the user's constraints for the execution of a parallel application and to identify a set of computation nodes that can run the parallel application within the user's constraints. The selection of nodes is done in parallel with the selection of the compilation parameters (design and execution contraints) to suit the user's needs. This selection is dynamic and should also use eduction. Once all suitable nodes have been identified and that a proper implementation was constructed, the application can be executed.

To summarize, an environment can only be called an iHPC environment when all the following requirements are met:

- The environment supports the execution of HPC applications;
- Intensional logic transcends all components of the environment:
  - Eduction is used as an execution model (dynamic selection of computation nodes);

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

- Eduction is used as a compilation model (intensional compiler).

Other researchers have thought of using intensional logic to perform high performance computing. The GLU (Granular LUCID) parallel programming environment was developed for parallel computing [Jagannathan and Dodd 96; Jagannathan *et al.* 97]. The GLU environment provides a "collage" of C functions using LUCID. Yet, it does not provide an intensional HPC environment, since the grains are fixed (they correspond to the C functions) and the environment does not use an intensional compiler.

5 Towards an Intensional HPC Environment

We argue that the combination of ParCeL-2 and the WOS™, in particular the HP-WOSP service class, constitutes an iHPC environment. To demonstrate this, we will focus on the problem of choosing the grain of parallelism. The correct size of the grain depends on the characteristics of the parallel architecture which will execute the program. In other words, it depends on the context of the execution. If that context is a metacomputing environment, the exact characteristics of the target architecture are only known at execution-time. As a consequence, the size of the grain should be fixed only at execution-time if we want to adopt the iHPC philosophy.

Let us suppose that we have developed a service (an application) using the ParCeL-2 programming language. This service is represented by its resource needs: CPU and network performance, particular software resources, etc. Let us also suppose that we make this service available in the WOS™ environment. When a WOS™ node receives a request for the execution of this service, it can act in two different ways. First it can decide that it has enough resources to execute the service locally. In such a case, it would like to execute a fully sequential version of this service. Otherwise (*i.e.*, no WOS™ node can provide the resources needed by the service), the WOS™ node can decide to split the program (the main cell) into its components in order to execute the service in parallel. Therefore, it will transform the received request into several requests, that is, one for each cell (elementary or complex) which composes the main program. In so doing, the WOS™ node becomes a client which requests for the execution of several services. Since "children" cells are less complex than the father cell, there is a higher probability to find a WOS™ node providing the requested resources. The same reasoning can be recursively applied for each service request sent by the current WOS™ node, until all the requests are eventually applied to elementary cells. In other words, a ParCeL-2 application does not represent only one service, but rather all the services corresponding to all the possible decompositions of the ParCeL-2 application (Section 2.3). In general, it is not reasonable to generate all these services when making a ParCeL-2 application available on the WOS™. A more efficient solution consists in using an intensional compiler. When a node receives a request and decides to run the service locally, it will look whether or not it possesses the sequential version of the service. If not, it will compile it and keep this sequential version for future use.

At the end of this configuration phase, the main service is seen by the system as a set of sub-services, each of them is assigned to a WOS™ node and represents an elementary or complex ParCeL-2 cell. At this stage of our research we assume that this assignement is static and therefore no process migration or fault tolerancy policy are considered. The above description shows that, at least for the problem of choosing the grain of parallelism, the association of the WOS™ and ParCeL-2 creates a iHPC environment. This follows from the requirements elicited in Section 4:

1. This combination supports the execution of HPC applications:
   - ParCeL-2 is a design and programming tool for HPC applications using a BSP model of computation and an MIMD parallel programming model;
   - The WOS™ provides a specialized service class, materialized through HP-WOSP, to configure and run HPC applications.
2. This association uses an eduction approach to configure and execute an HPC application:
   - The WOS™ provides the mechanisms to dynamically select and configure the nodes that will be used for the execution;
   - This dynamic selection also involves the dynamic identification of the grains of parallelisms, which can be identified in the ParCeL-2 model of the application.
3. The combination of the WOS™ and ParCeL-2 can support an intensional compilation approach:
   - A parallel application made with ParCeL-2 yields multiple possible implementations, which vary based on the grain of parallelism, the links established, and the actual resources available;
   - The WOS™ is used to supply the parameters required by an intensional compiler to only build the required executables.

6 Conclusion

In this paper, we have shown that the WOS™, together with the ParCeL-2 programming language, leads to the creation of an environment which exhibits the characteristics of an intensional High Performance Computing (iHPC) environment. Specifically, we showed that the approach is intensional for the determination of the grain of parallelism and the selection of the nodes that will execute these grains. Further investigations are required to validate the concepts presented in this paper. We are currently implementing an HPC version of the WOS™ (HP-WOS) that will allow us to compare the performances of our approach with other the performance of already available tools such GLOBUS or NetSolve.

Although we only consider a single dimension in this paper, namely the grain of parallelism, we are investigating the possibility to extend our approach to the selection of the memory model (distributed or shared). This will add an extra selection criteria for the nodes, i.e, another dimension in our iHPC environment.

Pierre Kuonen, Gilbert Babin, Nabil Abdennadher, and Paul-Jean Cagnard. "Intensional High Performance Computing." in Workshop on Distributed Communities on the Web (DCW 2000). Ste-Foy, Québec, Canada. Peter Kropf, Gilbert Babin, John Plaice, and Herwig Unger (eds.). Lecture Notes in Computer Science. No. 1830. Springer Verlag. June 2000. pp. 161-170.

We also ignored other run-time issues, usually associated with high performance computing, namely, load distribution (static and dynamic), inter-process communications, and synchronization. The GLU environment might provide us with interesting approaches which could be integrated into our vision of intensional HPC.

References

[Abdennadher *et al.* 00] Abdennadher, N., G. Babin, P. Kropf and P. Kuonen, "A dynamically configurable environment for high performance computing," in *High performance computing 2000 (hpc 2000)*, Washinton, DC, USA, April 2000.

[Babin *et al.* 98] Babin, G., P. Kropf and H. Unger, "A two-level communication protocol for a Web Operating System (WOS^{TM})," in *IEEE Euromicro Workshop on Network Computing*, pp. 939–944, Västerås, Sweden, August 1998.

[Buyya 99] Buyya, R., *High performance cluster computing : Architectures and systems*, vol. 1, Prentice Hall PTR, Upper Saddle River, N.J., USA, 1999.

[Cagnard 00] Cagnard, P.-J., "The parallel cellular programming model," in *8^{th} euromicro workshop on parallel and distributed processing (euro-pdp 2000)*, pp. 283–289, Rhodes, Greece, IEEE Computer Society, January 2000.

[Casanova and Dongarra 97] Casanova, H. and J. Dongarra, "NetSolve: A network server for solving computational science problems," *International Journal of Supercomputer Applications and High Performance Computing*, 3(11), pp. 212–223, 1997.

[Foster 94] Foster, I., *Designing and building parallel programs, concepts and tools for parallel software engineering*, Addison Wesley, 1994.

[Foster and Kesselman 97] Foster, I. and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, 1997.

[Grimshaw *et al.* 97] Grimshaw, A., W. Wulf, J. French, A. Weaver and P. Reynolds, "The Legion vision of a Worldwide Virtual Computer," *CACM*, 40(1), January 1997.

[Jagannathan and Dodd 96] Jagannathan, R. and C. Dodd, *GLU programmer's guide*, Technical report, SRI International, Menlo Park, CA, USA, 1996.

[Jagannathan *et al.* 97] Jagannathan, R., C. Dodd and I. Agi, "GLU: A high-level system for granular data-parallel programming," *Concurrency: Pratice and Experience*, (1), pp. 63–83, 1997.

[Kropf 99] Kropf, P., "Overview of the WOS project," in *1999 Advanced Simulation Technologies Conferences (ASTC 1999)*, San Diego, CA, USA, April 1999.

[Lindahl *et al.* 98] Lindahl, G., A. Grimshaw, A. Ferrari and K. Holcomb, *Metacomputing — what's in it for me ?*, http://legion.virginia.edu/papers/why.pdf, last visited on Jan. 20, 2000.

[Paquet 99] Paquet, J., *Intensional scientific programming*, Phd Thesis, Faculté des études supérieures, Université Laval, Québec, Canada, 1999.

[Swoboda and Wadge 00] Swoboda, P. and W.W. Wadge, "Vmake, ISE and IRCS: General tools for the intensionalization of software systems," in *Intensional Programming II*, World-Scientific, Singapore, 2000.

[The Globus Project] The Globus Project, *The Globus project*, http://www.globus.org, last visited on Jan. 20, 2000.

[Valiant 90] Valiant, L. G., "A bridging model for parallel computation," *Communications of the ACM*, 33(8), pp. 103–111, August 1990.