Design of Departmental Metacomputing ML

Frédéric Gava

LACL, University Paris XII, Créteil, France
gava@univ-paris12.fr

Abstract. Bulk Synchronous Parallel ML or BSML is a functional data-parallel language for programming bulk synchronous parallel (BSP) algorithms. The execution time can be estimated and dead-locks and indeterminism are avoided. For large scale applications, more than one parallel machine is needed. We consider here the design and cost-model of a BSML-like language devoted to the programming of such applications: Departmental Metacomputing ML or DMML.

Introduction. Bulk-Synchronous Parallel ML (BSML) is an extension of ML for programming Bulk Synchronous Parallel (BSP) algorithms as functional programs. BSP computing [4] is a parallel programming model introduced by Valiant to offer a high degree of abstraction like PRAM models. Such algorithms offer portable, predictable and scalable performances on a wide variety of architectures. BSML expresses them with a small set of primitives taken from the *confluent* BSλ-calculus. Those operations are implemented as a parallel library (http://bsmllib.free.fr) for the functional programming language Objective Caml (http://www.ocaml.org).

In recent years there has been a trend towards using *a set* of parallel machines in particular SMP clusters for these kinds of problems. Programming this kind of *supercomputers* is still difficult and libraries, languages and formal cost models are needed. Computing with these "cluster of clusters" is usually called *departmental metacomputing*.

BSML is not suited for departmental metacomputing, in particular to the heterogeneous nature of computing resources and networks. In fact the BSP model itself is not well suited to these two-tiered architectures. This paper describes our first work on the design of a model and a functional programming language for departmental metacomputing.

Bulk Synchronous Parallelism. A BSP computer contains a set of processor-memory pairs, a communication network allowing inter-processor delivery of messages and a global synchronization unit which executes collective requests for a synchronization barrier. Its performance is characterized by 3 parameters expressed as multiples of the local processing speed: p is the number of processor-memory pairs, l is the time required for a global synchronization and g is the time for collectively delivering a 1-relation (communication phase where every processor receives/sends at most one word). The network can deliver an

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 50–53, 2004.

h-relation in time gh for any arity h. A BSP program is executed as a sequence of *super-steps*, each one divided into three successive disjoint phases. In the first phase each processor uses its local data to perform sequential computations and to request data transfers to/from other nodes. In the second phase the network delivers the requested data transfers and in the third phase a global synchronization barrier occurs, making the transferred data available for the next super-step. The execution time of a super-step s is thus the sum of the maximal local processing time, of the data delivery time and of the global synchronization time:

$$\text{Time}(s) = \max_{i:processor} w_i^{(s)} + \max_{i:processor} h_i^{(s)} * g + l$$

where $w_i^{(s)}$= local processing time and h_i is the number of words transmitted (or received) on processor i during super-step s. The execution time of a BSP program is therefore the sum of each super-steps.

There are some arguments against using BSP for metacomputing. First the global synchronization barrier is claimed to be expensive and the BSP model too restrictive. For example, divide-and-conquer parallel algorithms are a class of algorithms which seem to be difficult to write using the BSP model. This problem can be overcome at the programming language level without modifying the BSP model. The main problem is that this model does no take into account the different capacity of the parallel machines and of the networks. So bulk synchronous parallelism does not seem to be suitable for metacomputing.

The global synchronization barrier could be remove. [3] introduces MPM which is a model directly inspired by the BSP model. It proposes to replace the notion of super-step by the notion of m-step defined as: at each m-step, each process performs a sequential computation phase then a communication phase. During this communication phase the processes exchange the data they need for the next m-step. The model uses the set of "incoming partners" for each process at each m-step. Execution time for a program is thus bounded the number of m-steps of the program. The MPM model takes into account that a process only synchronizes with each of its incoming partners and is therefore more accurate.

The heterogeneous nature of networks has been investigated. [2] proposes a two-levels Hierarchical BSP Model for a supercomputing without subset synchronization and two levels of communications. A BSP^2 computer consists of a number of *uniformly* BSP units, connected by a communication network. The execution of a BSP^2 program proceeds in `hyper-steps` separated by global synchronizations. On each hyper-step each BSP unit performs a complete BSP computation (some super-steps) and communicates some data with other BSP units. However, the authors noted that none of the algorithms they have analyzed showed any significant benefit from this approach and the experiments do not follow the model. The failure of the BSP^2 model comes from two main reasons: first the BSP units are generally different in practice and second the time required for the synchronization of all the BSP units is too expensive.

DMM: Departmental Metacomputing Model. To preserve the work made on BSP algorithms and to deal with the different architectures of each parallel

computer we propose a two-tiered model: the BSP model for each parallel unit and the MPM model for coordinating this heterogeneous set of BSP units. Thus, a DMM computer is characterized by the followings parameters: P the number of parallel computers, L the latency of the global communication network, G the time to exchange one word between two units, $\mathcal{P} = \{p_0, \ldots, p_{P-1}\}$ the list of the number of processes for each BSP unit, $\mathcal{L} = \{l_0, \ldots, l_{P-1}\}$ the times required for a global synchronization for each BSP unit and $\mathcal{G} = \{g_0, \ldots, g_{P-1}\}$ the time for collectively delivering a 1-relation on each BSP unit. We supposed that $\forall i\,(l_i \ll L)$ and $(g_i \ll G)$.

We proposes to replace the notion of hyper-step by the notion of d-step defined as: at each d-step, each BSP unit performs a BSP computation phase then a communication phase to deal with the values of other BSP units. Each processor within a unit parallel accesses to the outer world and so its messages passed on the local network, then on the global network and then to the destination unit local network. $\Omega_{d,i}$ is defined as the set of j such as the BSP unit j received messages from BSP unit i during d-step d. $\Phi_{d,i}$ is inductively defined as:

$$\begin{cases} \Phi_{1,i} = \max\limits_{j \in \Omega_{1,i}} (W_{(1,j)}, W_{(1,i)}) + \sum\limits_{j \in \Omega_{1,i}} ((g_i + G + g_j) \times h^j_{1,i}) + (g_i \times h^1_i + l_i) + L) \\ \Phi_{d,i} = \max\limits_{j \in \Omega_{d,i}} (\Phi_{d-1,j} + W_{(1,j)}, \Phi_{d-1,i} + W_{(1,i)}) + \sum\limits_{j \in \Omega_{d,i}} ((g_i + G + g_j) \times h^j_{d,i}) \\ \qquad\qquad + (g_i \times h^d_i + l_i) + L) \end{cases}$$

where $h^j_{d,i}$ denotes the number of words received by BSP unit i from the BSP unit j ("incoming partner") during the d-step d for $i \in \{0, \ldots, P-1\}$ and h^d_i is the maximum number of words exchanged within the BSP unit i during the d-step d and $W_{(d,j)}$ is the sum of the super-steps of a BSP units j.

Thus $\Psi = \max\{\Phi_{R,j}/j \in \{0, 1, \ldots, P-1\}\}$ bounds execution time for a complete program, where R is the number of d-steps of the program. The DMM model takes into account that a BSP unit only synchronizes with each of its incoming partner and is therefore more accurate than the BSP^2 model. Moreover more algorithms for irregular problems could be analyzed efficiently.

Departmental Metacomputing ML. Rather than a full Departmental Metacomputing ML (DMML) language we provide a library for Objective Caml. It is based on the elements given in figure 1. It gives access to the DMM parameters of the underling architecture: `dm_p()` gives the static number of BSP units, `(dm_bsp_p i)` the static number of processes of the BSP unit i. There are also two abstract polymorphic types: `'a par` (resp. `'a dpar`) which represents the type of p_i-wide parallel vectors (resp. P-wide departmental vectors) objects of type `'a`, one per process (resp. per BSP unit).

The DMML parallel constructs operates on parallel (resp. departmental) vectors. Those vectors are created by `mkpar` and `mkdpt` so that `(mkpar f)` stores `(f i)` on process i and `(mkdpt f)` stores `(f i)` on BSP units i. Asynchronous phases are programmed with `mkdpt`, `mkpar`, `apply` and `applydpt`:

`apply (mkpar f) (mkpar e)` stores `(f i)(e i)` on process i
`applydpt(mkdpt f)(mkdpt e)` stores `(f i)(e i)` on BSP units i.

```
dm_p: unit->int                              dm_bsp_p: int->int
mkpar: (int -> 'a) -> 'a par                 mkdpt: (int -> 'a) -> 'a dpar
apply: ('a -> 'b) par -> 'a par -> 'b par
applydpt: ('a -> 'b) dpar -> 'a dpar -> 'b dpar
put: (int->'a option) par -> (int->'a option) par}
get: (int->int->bool)par dpar->'a par dpar->(int->int->'a option)par dpar
```

Fig. 1. DMML Library

The communication phases are expressed by `put` and `get`. Consider: `get(mkdpt(fun i->mkpar(fun j->`$\mathtt{f}_{i,j}$`)))(mkdpt(fun a->mkpar(fun b->`$\mathtt{v}_{a,b}$`)))` For a process j of the BSP unit i, to receive a value $\mathtt{v}_{a,b}$ from the process b of the BSP unit a (it is an incoming partner), the function $\mathtt{f}_{i,j}$ at process j of the BSP unit i must be such that ($\mathtt{f}_{i,j}$ a b) evaluates to `true`. To receive no value, ($\mathtt{f}_{i,j}$ a b) must evaluate to `false`. Our expression evaluates to a departmental vector containing parallel vector of functions $\mathtt{f'}_{i,j}$ of delivered messages on every process of every BSP unit. At process j of the BSP unit i, (`f'`$_{i,j}$ a b) evaluates to `None` if process j of the BSP unit i received no message from process b of the BSP unit a or evaluates to `Some` $\mathtt{v}_{a,b}$ if received a value of the process b of the BSP unit a (it is an incoming partner).

Conclusions and Future Works. The BSP model has been used for the design of great variety of parallel algorithms. It also allowed the production of reliable and portable codes with predictable efficiency. However, additional complexity introduced by metacomputing forces a revision of the model. We have considered how to extend the BSP model hierarchically using the MPM model and we have also described a new functional parallel library which follows this new model. Several directions will be followed for future work: parallel implementation of DMML as a library using Madeleine [1], design and validation of new algorithms for this model, comparison with other hierarchical models [5].

Acknowledgments. This work is supported by a grant from the French Ministry of Research and the ACI Grid program, under the project CARAML.

References

1. O. Aumage, L. Bougé, and al. Madeleine II: A Portable and Efficient Communication Library. *Parallel Computing*, 28(4):607–626, 2002.
2. J. M. R. Martin and A. Tiskin. BSP modelling a two-tiered parallel architectures. In B. M. Cook, editor, *WoTUG'99*, pages 47–55, 1999.
3. J. L. Roda and al. Predicting the execution time of Message Passing Models. *Concurrency: Practice and Experience*, 11(9):461–477, 1999.
4. D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Scientific Programming*, 6(3), 1997.
5. T. L. Williams and R. J. Parsons. The Heterogeneous BSP Model. *LNCS*, volume 2000 ,page 112–118, Springer Verlag, 2000.