Managing true parallelism in ADA through PVM

Françoise Baude Nathalie Furmento
Daniel Lafaye de Micheaux *
I3S URA 1376 - Université de Nice Sophia-Antipolis
650 Route des Colles, B.P. 145
06903 Sophia-Antipolis Cedex
baude furmento ldm @essi.essi.fr

1 Introduction

There is a large community of developers using ADA, especially for computation intensive scientific problems. ADA is well known for providing the user with concurrency expression features, but the concurrent ADA tasks very seldom yield truly parallel tasks (except for a few cases where the compiler distributes tasks on shared-memory multiprocessors as in [Lun89]). On the one hand, ADA is a very attractive language from the software engineering point of view, but it lacks true parallelism expression and easily portable parallel run-times. On the other hand, PVM is implemented on more and more platforms, thus providing a standard and portable run-time for parallel and distributed programs.

The underlying idea of this paper is to build an ADA package which gives ADA programs easy access to the PVM primitives. We briefly describe this ADA interface to PVM (more details are in [Fur94]), then we discuss its use in a master/slave application.

2 The ADA to PVM interface

An ADA package has been implemented which associates to each PVM 3.3 primitive an ADA primitive with the same name, and proposes additional ones. All the C primitives defined by PVM have been implemented using the `pragma` interface: managment of the virtual machine (`start_pvmd, add_host, ...`), of tasks (`my_tid, spawn, ...`), of groups (`join_group, group_size, ...`), of errors (`perror`), of communications (`init_send, notify, ...`). However, these primitives have been created in the spirit of ADA and using the opportunities offered by this language but not by C. Here are some aspects of the implementation.

Genericity For the elementary communication functions, a generic package has been implemented, enabling easy definition of communications of any data type: the user of this package has just to instantiate it onto its required data type.

Overloading The same procedure name `send` can be used to send a byte, an integer, an array or a record, in contrast with the multiple procedure names in the PVM version (`pvm_pkbyte, pvm_pkint, ...`).

Reliability The use of enumerated data types is a way to better verify control parameter validity. In other respects, the definition of 19 different exceptions in the package permits rigorous managment of the errors in a more elegant way than available in C through function return codes.

*This work was supported by the Groupement PRS of the French Centre National de la Recherche Scientifique

Legibility In order to lighten the use of PVM primitives, it has proved to be useful to modify some primitive interfaces. For example, only one `send` primitive replaces the `pvm_pk*` and `pvm_send`: an optional boolean parameter indicates whether the data send has to be differed ($\approx$ `pvm_pk*`) or not ($\approx$ `pvm_pk*` followed by `pvm_send`), the immediate send being the default.

Opening It is possible to integrate in a single application executable files calling PVM stemming from different languages. For example, a master process can be written in ADA while the slave processes are written either in C or in FORTRAN.

3 Programming an application on a parallel architecture

The previous library has been used to write a parallel program to reconstruct astronomy images. The goal is to restore a clear image of an astronomic object from a large number (approximately 500) of real photos of this object. The photos f_k (128 by 128 matrices of reals) are strongly perturbed by multiple local variations of the refractive index in the atmosphere. To eliminate these parasitic effects, the right thing to do is to compute for each photo:

- its spectrum F_k (matrix of complex numbers resulting from a bidimensional Fourier Transformation) and its module $|F_k|$
- its bispectrum B_k (4 dimensional array of complex numbers defined by $B_k(\mathbf{u},\mathbf{v}) = F_k(\mathbf{u})\, F_k(\mathbf{v})\, F_k^*(\mathbf{u}+\mathbf{v})$, where u and v are 2 dimensional indexes)

in order to obtain for the set of photos the module of the mean spectrum $|\overline{F}| = \frac{1}{n}\sum_{i=1}^{n}|F_k|$ and the mean bispectrum $\overline{B} = \frac{1}{n}\sum_{i=1}^{n} B_k$. Then, we can quickly reconstruct the clear image $\hat{f}$ from $|\overline{F}|$ and $\overline{B}$.

About the implementation The computation of $|\overline{F}|$ and $|B|$, which warrants parallelization, can be put in the general form of a combining computation: computation of $y = g(x_1) * \ldots * g(x_k) * \ldots * g(x_n)$ where

- x_k is an element of a set `set_items`
- y and $g(x_k)$ belong to a set `set_results`
- g is a mapping from `set_items` into `set_results`
- $*$ is a binary operator defined on `set_results`

In our application,

- x_k is a photo: $x_k = f_k$
- $g(x_k)$ is composed of the spectrum module and the bispectrum: $g(x_k) = (|f_k|, B_k)$
- $*$ is the sum of two results (sum of spectrum modules, and sum of bispectra)

This general situation can be easily implemented in ADA interfacing PVM by a generic package which provides a computation implying a master and p slaves, and assuming $*$ is a commutative and associative operator (which is the case here). The master process runs in a loop where it waits for any slave requiring some work. In response to this request, the master sends to the slave a group of elements to treat. For each element x_k belonging to the received group, the slave first processes it (computation of $g(x_k)$) then combines (using $*$) the result with previous locally computed results. Once all elements to be treated have been distributed, the master asks the slaves to send back their respective results. It combines these results in order to obtain the final result.

Remarks regarding the implementation

1. In order to limit the exchanges between the master and the p slaves, in the case where an element of `set_items` is large (here, a photo), it has been agreed that the master sends identifiers, instead of items to the slaves (a photo number)
2. Sending many grouped identifiers from the master to a slave is a way to
   - make an equitable distribution of the work on the p slave processors (p groups of n/p identifiers) in the case of a single-user homogeneous network
   - or, make a dynamic load-balancing in the case of a heterogeneous network, according to the processor type or availability (groups of a few elements)

4 Some performance evaluation

Performances have been evaluated on a set of SPARC stations connected by Ethernet.

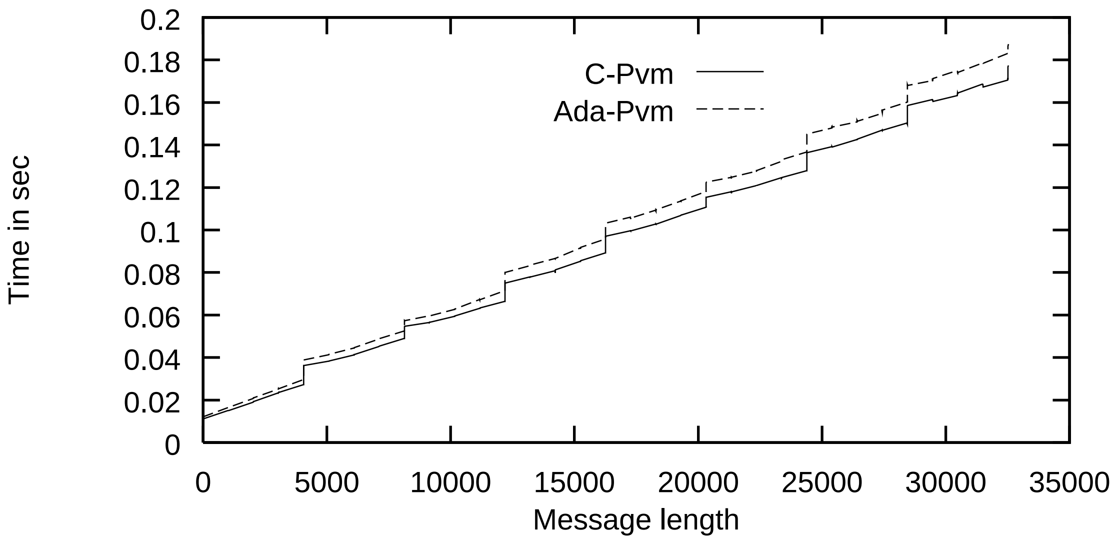

Figure 1: Performance comparison of 2 exchanges between PVM tasks

Communication only We have measured the speed of communication for different message lengths, and have compared them to the result using the original C primitives. It appears that the two linear outlines, with steps due to packet length of 4064, are almost identical, with less than a $5 - 10\%$ increase for ADA (cf. figure 1).

Computation only At this time of writing, the FFT algorithm written in ADA, developed under the Meridian ADA tool, spends 50% more time than the one written in C. We have been unable to improve these performances even using all compilation and link optimization options, so we plan to try other ADA compilers.

The whole application Figure 2 gives the duration of parallel computation of $|\overline{F}|$ and $\overline{B}$ for different values of the number of photos and processors. The efficiency for $p \in \{2, 4, 8\}$ on 16 photos varies between 0.9 and 1. These results show the efficiency of the technique used and the practical interest in solving big computational problems using several tens of stations usually unloaded by night. On a real parallel machine (such as the IBM SP), the efficiency should be much better because communications don't need to be mutually exclusive as in the case on Ethernet.

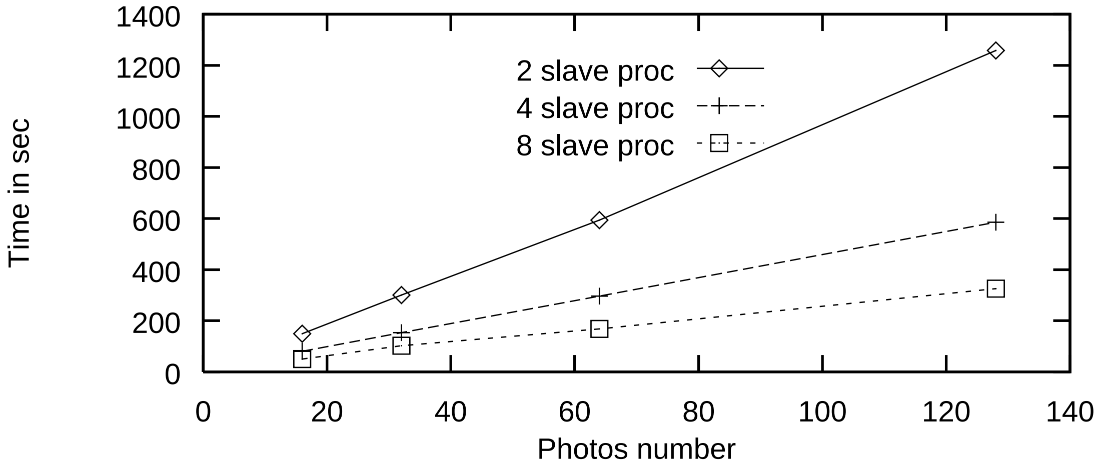

Figure 2: Time versus number of photos with one curve per number of processors

5 Conclusion

We have provided a tool to enable writing truly parallel ADA programs, easily portable on a wide range of parallel platforms. In fact, PVM has been implemented as well on networks of workstations and on parallel machines. In the parallel architecture domain, there is a trend towards standard processors connected by high speed networks. Thus, standard sequential ADA compilers should also be available on the processors of the new generation parallel machines, and consequently ADA applications could easily take advantage of the parallel hardware thanks to our interface to PVM.

We have presented our ADA-to-PVM package (soon available by anonymous `ftp` and soon ported onto an IBM parallel SP-2) and its high performances compared with the PVM library written in C. We have also demonstrated how the use of the ADA language (instead of the C language) is a means of writing parallel generic programs. In this way, the developer (or the student) can focus on the algorithmic structure of his/her application and is less "disturbed" by the syntaxic or data type details.

Relaxing the constraint of dynamic groups of tasks, thus assuming static groups, we are currently working on a more efficient use and implementation of global communications and combining operations, like gather, scatter and reduce: we can give them a generic feature using ADA, and, (first in the context of Ethernet, then for others targets), there is an opportunity to overlap computation phases when exchanging data between PVM tasks [Des92].

References

[Des92] F. Desprez. *Procédures de base pour le calcul scientifique sur machines parallèlles à mémoire distribuée.* PhD thesis, INPG, Grenoble, 1992.

[Fur94] N. Furmento. Accès à PVM par l'intermédiaire du langage ADA dans un objectif de programmation conviviale d'applications parallèles. Rapport de DEA I3S, 1994.

[Lun89] Lars Lundberg. A parallel Ada System on an Experimental Multiprocessor. *Software – Practice and Experience*, 19:787–800, 1989.