SCE Toolboxes for the Development of High-Level Parallel Applications^{*}

J. Fernández, M. Anguita, E. Ros, and J.L. Bernier

Departamento de Arquitectura y Tecnología de Computadores,
Universidad de Granada, 18071-Granada, Spain
{jfernand, manguita, eros, jlbernier}@atc.ugr.es
<http://atc.ugr.es>

Abstract. Users of Scientific Computing Environments (SCE) benefit from faster high-level software development at the cost of larger run time due to the interpreted environment. For time-consuming SCE applications, dividing the workload among several computers can be a cost-effective acceleration technique. Using our PVM and MPI toolboxes, MATLAB[®] and Octave users in a computer cluster can parallelize their interpreted applications using the native cluster programming paradigm — message-passing. Our toolboxes are complete interfaces to the corresponding libraries, support all the compatible datatypes in the base SCE and have been designed with performance and maintainability in mind. Although in this paper we focus on our new toolbox, MPITB for Octave, we describe the general design of these toolboxes and of the development aids offered to end users, mention some related work, mention speedup results obtained by some of our users and introduce speedup results for the NPB-EP benchmark for MPITB in both SCE's.

1 Introduction

GNU Octave [1] is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.

MATLAB [2] is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Decades of experience, a careful selection of algorithms and a number of improvements to accelerate the interpreted environment (JIT compiler, etc.), can make MATLAB faster for computationally intensive tasks than traditional programming languages such as C, C++, and FORTRAN, if the programmer is not so careful in the algorithm selection or implementation in terms of native machine instructions.

Both environments can be extended, either with scripts written in the interpreted language — the so-called "M-files", or external (FORTRAN, C, C++) programs using

^{*} This work was supported by the Spanish CICYT project SINTA-CC (grant TIN2004-01419) and the European FP-6 integrated project SENSOPAC (grant IST-028056).

the MATLAB API or Octave library — called "MEX-files" or "DLD functions", respectively. This latter choice lets us build an interface to a message-passing library.

Fig. 1 is an overview of how the end-user application would sit on top of the SCE (Octave, in the example shown) and our toolbox (MPITB in this case), being thus able to communicate with other SCE instances running in other cluster nodes.

Fig. 1. High-level overview diagram of Toolbox role. The Toolbox invokes message-passing calls and SCE API calls to intercommunicate SCE processes running on different cluster nodes.

PVM (Parallel Virtual Machine) [3] was built to enable users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. PVM can of course be used as message-passing mechanism in a dedicated cluster.

MPI (Message Passing Interface) [4,5] is a standard, and as such, it only defines the routines that should be implemented in a compliant implementation.

LAM/MPI (Local Area Multicomputer) [6,7] implements MPI-1.2 and large portions of the MPI-2.0 standard. In addition to high performance, LAM provides a number of usability features key to developing large scale MPI applications.

Our PVMTB and MPITB toolboxes interface the PVM and LAM/MPI libraries from within the MATLAB or Octave environments, thus letting SCE users parallelize their own applications by means of explicit message-passing. Since users are expected to modify their sequential code to adapt it to parallel execution, a number of utility M-files have also been developed on top of this direct, "low-level" library interface, to facilitate the parallelization process and reduce those modifications to a minimum.

To further clarify the idea, MEX or DLD interface is required, in order to be able to call PVM or LAM from C code. That cannot be accomplished from the interpreted language. Octave's *interpreted language* is compatible with MATLAB, but the MEX version of MPITB will simply *not load* as a valid DLD under Octave, let alone work correctly. Each version uses a different API to access SCE data and pass it from/to the message-passing library. For instance, to return without errors we would either code `*mxGetPr(plhs[0])=0` for MATLAB or `return octave_value(0)` for Octave.

Section 2 summarizes the design principles for these toolboxes, compares them to other related work, describes the utilities built on top of them and shows how to use these to ease the parallelization of sequential SCE code. Section 3 shows performance measurements for the NPB-EP benchmark. Section 4 describes speedup results from some of our users. Section 5 summarizes some conclusions from this work.

2 Design Principles

Although the explanation and examples below are focused on our most recent MPITB for Octave, the same guidelines hold for the other toolboxes. The key implementation details are:

- **Loadable files:** For each MPI call interfaced to the Octave environment, an `.oct` file has been built, which includes the doc-string and invokes the call-pattern. The source for the loadable file includes at most just 2 files: the global include (`mpitb.h`) and possibly a function-specific include (`Send.h`, `Topo.h`, `Info.h`, etc).
- **Call patterns:** MPI calls are classified according to their input-output signature, and a *preprocessor macro* is designed for each class. This call-pattern macro takes the name of the function as argument, substituting it on function-call code.
- **Building blocks:** Following the criterion of not writing things twice, each piece of self-contained code that is used at least twice (by two call patterns or building blocks) is itself promoted to building block and assigned a *macro name* to be later substituted where appropriate. This feature makes MPITB maintenance tractable.
- **Include files:** build-blocks and call patterns are grouped for affinity, so that only the functions that use them have to preprocess them. A building block or call pattern shared by two or more such affine groups is moved to the global include file.
- **Pointers to data:** MPI requires pointers to buffers from/to where the transmitted info is read/written. For arrays, pointer and size are obtained using methods `data()` and `capacity()` common to all Octave array types. For scalars, C++ data encapsulation is circumvented, taking the object address as starting point from which to obtain the scalar's address. Shallow-copy mechanisms are also taken into account by invoking method `make_unique()` if the Octave variable is shared.
- **RHS value return:** Octave variables used as MPI buffers must appear in the right-hand-side (RHS) of the equal sign in MPITB function definitions; that is, we rather define `[info,stat]=MPI_Recv(buf,src,tag,comm)` than return `buf` as Octave left-hand-side (LHS) such as in `[buf,info,stat]=MPI_Recv(src,tag,comm)`. That would be a low-performance design decision, since the returned `buf` object should be constructed. Depending on `buf` type and size, that could be a very expensive operation, particularly in the context of a ping-pong test or an application with a heavy communication pattern. We instead force buffers to be Octave symbols (named variables), so the output value can be returned in the input variable itself.

According to our experience [8], these last two principles guarantee that performance will be superior to any toolbox lacking of them. This is particularly true for the mentioned object construction in LHS return, which is not required for MPI operation.

2.1 Related Work

Perhaps due to the different design principles used, other parallel Octave toolboxes no longer work, proving that the task is not trivial. The interface is elaborate except for a few of the simplest calls, and the design must keep an eye on performance and maintainability. We have surveyed the Octave mailing list in search for related work.

Table 1. Survey on parallel Octave prototypes

Package	based on	#cmds	date	status
PVMOCT	PVM	46	Feb. 1999	developed for octave-2.0.13 would require major edition
Octave-MPI	LAM/MPI	7	Nov.2000	prepatched for octave-2.1.31 would require major edition
P-Octave	LAM/MPI	18	Jun. 2001	works with octave-2.1.44 after minor source code edit
D-Octave	LAM/MPI MPICH	2+6	Mar. 2003	prepatched for octave-2.1.45 proof-of-concept only
MPITB	LAM/MPI	160	Apr. 2004	recompiled for 2.1.50-57-69-72 supported

Our MPITB interfaces all MPI-1.2 and some MPI-2.0 calls — 160 commands. It does not rely so heavily on the Octave version — no Octave patching is required, just minor code changes to adapt to changes in Octave API and internal representation of data, or to new Octave datatypes. The observed design principles convert MPITB maintenance in a feasible task. Our approach of pointers to data and RHS return is unique, so all other toolboxes would suffer from data copy and object creation costs.

2.2 Utilities

End-users are the ones knowing their own applications, and hence the most appropriate to design the message-passing protocol among the parallel SCE instances. However, many users benefit from some simple examples, either to ease the initial learning curve or to save development time by customizing very frequently used code.

This has been the origin for the startup files and protocol and instrumentation utilities. Again, although we concentrate here on our most recent MPITB for Octave, the same discussion is applicable to our other parallel toolboxes.

Startup Files

The default Octave run-command (`.octaverc`) can be replaced or modified to add the toolbox path, and to discriminate spawned SCE instances by checking for existence of the `LAMPARENT` environment variable. This use has evolved to a collection of startup files, each customized to a particular situation.

- **Path:** When starting multiple SCE instances via `mpirun`, the instances are expected to `MPI_Init` and `_Finalize` by themselves, without further support. This is most similar to a normal C-MPI application. The NPB-EP demo works this way.
- **Communicator Merge:** When starting slave instances from a master via the MPITB command `MPI_Comm_spawn`, the slaves' and master's communicators could be merged to simplify rank addressing. Slave `MPI_Init` and `_Intercomm_merge` are included in this startup. The master Octave is expected to `_Intercomm_merge` some time after `MPI_Comm_spawn`. The MPITB tutorial examples work this way.
- **Broadcast Protocol:** In many simple applications, slaves are sent just one and the same Octave command (an MPI string), which must be evaluated on all slaves. The

master Octave is expected to `MPI_Bcast` the required string. Slave `_Bcast` is included in this startup. MPITB demos *Pi*, *PingPong* and *Spawn* use this protocol.

- **NumCmds Protocol:** For more complex applications, users might want to send a fixed or undetermined number of commands, and possibly related variables, to slave SCE's; or perhaps the same command is repeated over and over on different data. A more complex startup is provided to cope with all these situations, as well as supporting scripts for the master. The MPITB demo *Wavelets* uses this protocol.

Protocol Utilities

The NumCmds protocol works as depicted in Fig. 2. The above mentioned supporting scripts are shown in the left SCE instance. The user would log in the headed cluster node, start an SCE (master) session, and define a cell-array of cluster hostnames.

Fig. 2. NumCmds protocol overview. *LAM_Init* boots *n* LAM daemons. *Octave_Spawn* starts *n* Octave instances on those cluster nodes. These instances are engaged inside *startup_NCmds* in a receive-unpack-eval loop. The loop is here fixed at *nc* epochs due to *NumCmds_Init* on the master side. On each epoch, *NumCmds_Send* is used in the master Octave to send a different command string and optional data to each slave (null string "" to repeat previous command).

n of these hosts are used by the *LAM_Init* and *Octave_Spawn* utilities to build the parallel Octave session. The slave instances are expected to source the *NumCmds* startup file, which expects an initial number *nc* from the master, with 0 meaning an undetermined number of commands —slaves will proceed until a `quit` command is received. Slaves then enter a receive-unpack-eval loop, executing on each iteration the command sent by the master instance.

Instrumentation Utilities

Since each user using our toolboxes will need to know how time is spent on the sequential SCE application, a number of simple to use routines are provided to generate, manipulate and graphically display the instrumentation data:

- `init_stamp`: Allocates (*n* annotations) and initializes the timestamp data structure.
- `time_stamp`: Annotates the current time together with a descriptive label.
- `save/load_instr`: Saves/loads the collected instrumentation data to/from a file.
- `look_stamp`: Lets the user graphically browse through the instrumentation data.

Showing the data in an intuitive way is vital to correctly parallelize applications.

3 NPB-EP Benchmark

The NAS (Numerical Aerodynamic Simulation) Program has been developing for two decades or so [9,10] a series of test codes aimed to benchmark highly parallel computers. They have tried to avoid both restricted kernels and full scale applications, arriving to a set of 5 kernels (EP, MG, CG, FT, IS) and 3 application benchmarks (LU, SP, BT), collectively known as NPB (NAS Parallel Benchmarks). Together, they mimic the computation and data movement characteristics of large scale CFD applications. These codes are well-known. Particularly the kernels have been ported to many programming languages, including Java, SAC, UPC and ZPL — see [11] for a review.

The EP (embarrassingly parallel) kernel measures floating point performance by tabulating statistics on pseudorandom data. It exhibits the simplest possible communication pattern among processes, hence the name. The MPITB version of this code uses the *path* startup file and is run using `mpirun`.

The cost of porting EP from the original FORTRAN version to MATLAB/Octave is mainly in the random number generator, adapted from the NAS-provided `randdp` double precision version. Particularly, the few MPI calls can be almost literally ported from the original FORTRAN bindings to the MPITB bindings. The `common` FORTRAN storage for the pseudo-randoms is ported to a `global` MATLAB variable. A number of performance-related details have been considered, including the use of real math in MATLAB (`reallog` & `realsqrt` instead of the normal complex versions `log` & `sqrt`) and the use of vectorization in Octave —not in MATLAB, where the JIT compiler on the iterative version is faster than the vectorized version.

Fig. 3 shows runtimes and relative speedups when EP is run in up to 8 computers, for all three versions, FORTRAN, MATLAB and Octave.

Fig. 3. Scalability test for the NPB-EP benchmark on an 8-node AMD Athlon 2400+ cluster. MATLAB run times in the left are one order of magnitude greater than FORTRAN ones, and two smaller than Octave. For all three versions, scalability is practically linear. The large computation time for Octave-2.1.69 made it difficult to measure it in a clean, zero-user situation.

4 End-Users Examples

Sebastián Dormido (UNED, Spain) has used PVMTB for MATLAB in order to implement DP (Dynamic Programming) in clusters of computers [12]. His group already had a large background on DP and MATLAB programming, but no previous knowledge on message-passing. They have developed several *parallel serial monadic DP* algorithms, some of them with very good scalability and speedup results, near to 12 using 15 slaves (80% efficiency). The referenced paper includes plots of these results.

Sébastien Goasguen (then at Purdue, IN, USA) used both PVMTB and MPITB for MATLAB in order to model quantum transport in nanoscale transistors [13,14]. He reduced simulation time from 3 days to 45 minutes (speedup 101x with 120 CPUs, 84% efficiency) after parallelizing the NanoMOS simulator. Incidentally, NanoMOS has been the subject for experiments on dynamic Grid filesystem sessions [15] where the whole MATLAB or just MPITB were updated.

Michael Creel (now at UAB, Spain) is an active developer of Octave, and has used MPITB for Octave in order to parallelize Monte Carlo simulations, Maximum Likelihood estimation and Kernel Regression [16]. He has included MPITB in his Parallel-Knoppix Linux distribution [17] with which he obtained speedups from 9 to 11 when using 12 computers (efficiency 75-92%). The code is available from the OctaveForge site (econometrics package [18]), so other users of this package can immediately benefit from cluster computing without previous knowledge of message-passing.

Morris Law (HKBU, Hong Kong) is the IT coordinator for the Science Faculty in HKBU. One of their clusters (TDG) ranked #300 in Top500 on 6/2003. Morris prepared some coursework related to MPITB for MATLAB [19]. MPITB has also been involved in research in HKBU [20].

5 Conclusions

With the popularization of clusters of computers, a processing power previously unavailable is now at the reach of normal users. But many users program their compute-intensive tasks in Scientific Computing Environments (SCE's). Our PVMTB and MPITB toolboxes make this processing power available to SCE users, at the cost of adapting their sequential applications to explicit message-passing. For some examples such as the NAS-EP benchmark this adaptation is straightforward, and a number of utility scripts have been provided to ease this adaptation for more complex cases.

Some of our end-users have successfully used the toolboxes for research, education and software development. Thanks to all them for their feedback; it has greatly improved the original versions and guided the development of the utility scripts.

References

1. Eaton, J. W.: GNU Octave Manual. Network Theory Ltd. (2002) ISBN: 0-9541617-2-6.
2. Moler, C. B.: Numerical Computing with MATLAB. SIAM (2004) ISBN: 0-89871-560-1.
3. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. The MIT Press (1994). ISBN: 0-262-57108-0.

4. MPI Forum: MPI: A Message-Passing Interface Standard. *Int. J. Supercomput. Appl. High Perform. Comput.* Vol.8, no.3/4 (1994) 159–416. See also the MPI Forum Documents: MPI 2.0 standard (2003) University of Tennessee, Knoxville. Web <http://www.mpi-forum.org/>.
5. Gropp, W., Lusk, E., Skjellum, A.: *Using MPI: Portable Parallel Programming with the Message Passing Interface*. 2nd Edition. The MIT Press (1999) ISBN: 0262571323.
6. Burns, G., Daoud, R., Vaigl, J.: LAM: an Open Cluster Environment for MPI. *Proceedings of Supercomputing symposium* (1994), 379–386
7. Squyres, J., Lumsdaine, A.: A Component Architecture for LAM/MPI. *Proceedings of the 10th European PVM/MPI Users' Group Meeting, Lecture Notes in Computer Science*, Vol.2840 (2003) 379–387.
8. Fernández, J., Cañas, A., Díaz, A.F., González, J., Ortega, J., Prieto, A.: Performance of Message-Passing MATLAB Toolboxes, *Proceedings of the VECPAR 2002, Lecture Notes on Computer Science*, Vol.2565 (2003), 228–241. Toolboxes URL <http://atc.ugr.es/~javier>.
9. Bailey, D. et al: The NAS Parallel Benchmarks. RNR Technical Report RNR-94-007 (1994)
10. Bailey, D. et al: The NAS Parallel Benchmarks 2.0. Report NAS-95-020 (1995). Reports and software available from <http://www.nas.nasa.gov/Software/NPB/>.
11. Buss, B.J.: Comparison of serial and parallel implementations of Benchmark codes in MATLAB, Octave and FORTRAN. M.Sc. Thesis, Ohio State University (2005). Thesis and software available from <http://www.ece.osu.edu/~bussb/research/>.
12. Dormido C., S., de Madrid, A.P., Dormido B., S.: Parallel Dynamic Programming on Clusters of Workstations, *IEEE Transactions on Parallel and Distributed Systems*, vol.16, no.9 (2005), 785–798
13. Goasguen, S., Venugopal, R., Lundstrom, M.S.: Modeling Transport in Nanoscale Silicon and Molecular Devices on Parallel Machines, *Proceedings of the 3rd IEEE Conference on Nanotechnology* (2003), Vol.1, 398–401. DOI 10.1109/NANO.2003.1231802.
14. Goasguen, S.; Butt, A.R.; Colby, K.D.; Lundstrom, M.S.: Parallelization of the nano-scale device simulator nanoMOS-2.0 using a 100 nodes linux cluster, *Proceedings of the 2nd IEEE Conference on Nanotechnology* (2002) 409–412. DOI 10.1109/ NANO. 2002. 1032277
15. Zhao, M., Chadha, V., Figueiredo, R.J.: Supporting Application-Tailored Grid File System Sessions with WSRF-Based Services, *Procs of the 14th IEEE Int. Symp. on High Perf. Distributed Computing HPDC-14* (2005), 24–33. DOI 10.1109/HPDC.2005.1520930.
16. Creel, M.: *User-Friendly Parallel Computations with Econometric Examples*, Computational Economics, Springer (2005) 26 (2): 107–128. DOI 10.1007/s10614-005-6868-2
17. Creel, M.: Parallel-Knoppix Linux, <http://pareto.uab.es/mcreel/ParallelKnoppix/>.
18. Creel, M.: Econometrics Octave package at OctaveForge, <http://octave.sf.net/>. See package index at <http://octave.sourceforge.net/index/extra.html#Econometrics>.
19. Law, M.: MATLAB Laboratory for MPITB (2003), Coursework resource for MATH-2160 HKBU, available from <http://www.math.hkbu.edu.hk/math2160/materials/MPITB.pdf>. See also Guest Lecture on Cluster Computing (2005), Coursework resource for COMP-3320, available <http://www.comp.hkbu.edu.hk/~jng/comp3320/ClusterLectures-2005.ppt>.
20. Wang, C.L.: Grid Computing research in Hong Kong, 1st Workshop on Grid Tech. & Apps. (2004), <http://www.cs.hku.hk/~clwang/talk/WoGTA04-Taiwan-CLWang-FNL.pdf>.