A Generalized Approach for Transferring Data-Types with Arbitrary Communication Libraries

Martial MICHEL
NIST, USA
RÉSÉDAS, France
martial.michel@nist.gov

Judith Ellen DEVANEY
NIST, USA
judith.devaney@nist.gov

Abstract

We present a generalized algorithm for implementing a communications library for dynamic data structures created with heterogeneous composed data types such as multiple C structs, and where the data-types may be nested and may contain pointers. This algorithm is divided into an absolute part that is the same for all instantiations, and a relative part that is specific to the communications mechanism used, such as PVM or MPI. We describe the algorithm in terms of our AutoMap/AutoLink implementation in C/MPI.

First, we will talk of the MPI case and of the AutoMap and AutoLink solutions (with ideas from version 3.0). Then we discuss what is to be followed in order to generalize the data-types transfer concepts presented in this article.

With this addition to AutoMap/AutoLink we can extend the functions provided from the current send and receive functions (blocking and non blocking) available for any data-types, to any kind of transfer function; from broadcast to reduce (as long as the reduce called process is "message aware"). This will also simplify the extension of this work to data-types load balancing.

1 Introduction

Data structures, including dynamic ones, are an integral part of effective computing. Yet message passing systems for parallel and distributed computation, such as Parallel Virtual Machine (PVM) and Message Passing Interface (MPI), do not provide the capability to send and receive dynamic data structures as part of their standard. If one created decision trees on multiple processors and wanted to send and receive these trees between processors for comparisons, one would have to develop specialized software to accomplish this for each message passing scheme used. This would involve flattening the data structure, sending it, receiving it, and then reconstituting it. In MPI the software to accomplish this is laborious to create. In contrast, languages such as Java provide a mechanism to flatten data structures, called serialization. One can send such serializations simply. However, Java does not have the amenities of MPI such as broadcast, or topologies. This work describes a general methodology that can be used to create serializations in C that can be used with any message passing system.

Note that we have already looked over the CORBA methods in [8].

2 Generalizing data-type transfer

2.1 Notation

First we define two names : complex data-types and dynamic data-types.

2.1.1 Complex data-types

Every composed data-type using C data-types (basic or constructed) without pointers (see figure 1).

Figure 1 Example of Complex data-type

```
typedef struct pen {
  char brand[20];
  int content;
  long unit_price;
}
```

2.1.2 Dynamic data-types

Every "complex" data-type using pointers to refer to other data-type(s) (see figure 2).

Figure 2 Example of Dynamic data-type

```
typedef struct penbox {
  char brand[20];
  pen *content[20];
}
```

2.2 Dynamic data-type transfer

Dynamic data-type transfer is an issue on all distributed systems, and if a message passage library provides means to transfer complex data-types, few provide their dynamic data-type counterpart.

In essence to be able to transfer dynamic data-types, there are a few basic steps that are to followed. Those steps are, for the sending part of the transfer process :

1. Graph traversal,
2. Address abstraction,
3. Data transfer.

And for the receiving part :

1. Data transfer,
2. Reverse address abstraction.

This provides a means for “data-type serialization”.

As we will show later in this article, the AutoLink algorithm matches (and specializes) those criteria. To fully understand this, we will present the MPI data-type problem that AutoMap and AutoLink solve.

3 AutoMap & AutoLink

3.1 MPI issues

Message passing is used widely on distributed memory parallel machines and clusters of computers. The MPI standard defines an easy way to work with such concepts, by providing support for :

- Point-to-point communication
- Collective operations
- Process groups
- Communication contexts
- Process topologies
- ...

The binding of the standard with the C language provides a way to address issues such as data-types, where in MPI this is done through opaque objects accessed via handles.

Still, the MPI library can only transfer data-types that it knows about, and the C implementation of MPI only knows about basic types in C. Extending the range of those data-types is allowed by creating user defined MPI data-types; a long, repetitive and complicated process, that can be described in six operations :

1. Set up an array defining the number of data of each kind that will be used (in the same order as the structure definition).
2. Set up an array that will contain the type specification for each element contained in the structure.
3. Set up an internal displacement array containing the memory offset of each field.
4. Give a name to the MPI data-type.
5. Build the new MPI type. Set the container of the MPI data-type .
6. Validate the type existence to be used with MPI.

Such complex data-types can only be sent and received once they are described to the MPI library, but in the case of dynamic data-types, it is left to the user to enforce the packing and unpacking required.

AutoMap and AutoLink are solutions to these needs using the MPI library;

- AutoMap creates user defined MPI data-types from C structs after reading them from a file.
- AutoLink gives the MPI user a means to transfer dynamic data-types simply from one MPI node to another.

3.2 AutoMap

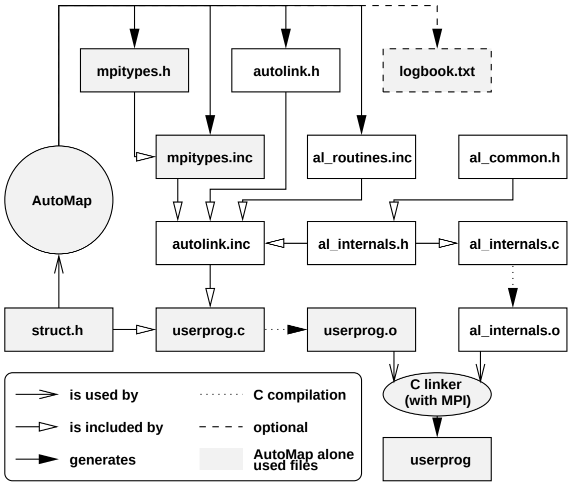
AutoMap[1, 3] is designed to simplify the MPI user’s task when creating complex data-types. It is a source-to-source compiler designed to read from user C data-types definition files `typedef` and `struct` entries, recognizing special directives (placed inside of C comments) and generating a set of files containing MPI data-type definition and creation procedures.

3.3 AutoLink

AutoLink[3] is a library extension to MPI, designed to allow users to transfer dynamic data-types via MPI (the marshalling/unmarshalling process of CORBA[8]). It requires AutoMap to parse the user data-type entries and provides high level functions to transfer them. The file generation process is shown in figure 3.

The public transfer operations are based on the MPI functions of the same name, and are preceded by `AL_`[1].

Figure 3 AutoMap and AutoLink files generation; `struct.h` is the user type definition file, `useprog.c` is the C source that uses the output of AutoMap/AutoLink



AutoLink works with Packets (elements of the same data-types put together in order to fasten the transfer process), and its transfer functions can be described easily by :

Send Starts from the user data-type entry point, follows each pointer link in a breadth first traversal, stores data into "Packets", and sends them.

Recv Receives all the data, stores it, recreates the links between the fields of the data-structures, and return the entry point to the user.

AutoLink performance is influenced by the use of PACKETs. This is studied in [7].

4 Transfer operations

AutoLink transfer operations are based on send and receive algorithms that are generic enough so that the basic concept and order presented can be used both in blocking and non blocking operations (even if the lower level code needs to be different to process the requests generated by non blocking operations).

[1] `AL_Send` for example

4.1 Internal data-types

To fully understand the simplified algorithms to be presented hereafter, one must understand the specific data-types developed for AutoLink.

NEXT Elements to traverse.

mark Elements already traversed

ADDRESS Addresses –memory representation– of elements.

LINKS Information required to perform the reverse address conversion of recreated elements.

PACKET Used to "pack" elements of the same data-type together before transfer.

4.2 Initial message

The new algorithm for AutoLink reduces the number of communications from two messages for each partial send to one initial message and one for each partial send. This is done by separating computation and transfer while in the previous version of the algorithm, both overlapped.

The initial message contains two pieces of information :

1. Number of elements to be sent; total number of elements to be send for each data-type.
2. Initial element, so that we can recreate the data-type by knowing which element is the leading element of the reconstructed dynamic data-type.

4.3 Send Algorithm

The send simplified algorithm is given in figure 4. Lines preceded by "@" are found in the AutoMap generated code, and entries preceded by "*" are MPI specialized.

4.4 Reception Algorithm

It could be found in figure 5, and follow the same rules that are defined for figure 4.

5 Communication system dependencies

Some functions of the algorithm need to be specialized in order to work properly with a given communications system such as MPI or PVM.

There are three types of items that need consideration :

1. data-type specification. In MPI, composed data-types may be created, and this simplifies sending and receiving.

Figure 4 Send Algorithm

```
|Add entry node in ADDRESS and NEXT,
|  and MARK it
|While there is a element in NEXT
|   |Get Current Node (CN) from NEXT
|   |@For each children of CN
|   |@If the child in not MARKed
|   |@Then |@If the child exists
|   |@     |@Then |@Add child in
|   |      |      |  ADDRESS and NEXT,
|   |      |      |  and MARK it
|   |Go to next in NEXT
|*Send initial message
|For each (=i) data-type
|   |For each (=j) element of ADDRESS[i]
|   |   |Add ADDRESS[i][j] (Current Node)
|   |   |  in PACKET (will work on copy)
|   |   |@For each child of CN
|   |   |   |@If the child does not exist
|   |   |   |@Then |@Add ``NO CHILD'' to
|   |   |   |      |  LINKS
|   |   |   |@Else |@Add child's MARK to
|   |   |   |      |  LINKS
|   |   |*If PACKET is full, send it
```

2. data-type representation between heterogeneous machines. For example, MPI has predefined data-types such as `MPI_Integer` that are guaranteed to be transfered successfully between machines.

3. method to aggregate data into a packet. For example, PVM has a Pack/Unpack construct. MPI does also, but this method is considered inefficient and a higher level data-type may be constructed.

6 Generalizing the AutoLink solution

When looked at from a general point of view, the specific algorithms developed for AutoLink are in the essence :

- Send :
  1. Graph traversal (marked)
  2. Address conversion (absolute to relative)
  3. Data transfer (using packets)
- Receive :
  1. Data reception (and memory creation)
  2. Graph links recreation (reverse address conversion)

Figure 5 Receive Algorithm

```
|*Receive Initial Message
|While there are PACKETs to receive
|   |*Receive PACKET
|   |For each element of PACKET
|   |   |@Memory create the element
|   |   |Add created element information
|   |   |  in ADDRESS
|@For each element in LINKS
|   |@If ``NO CHILD''
|   |@Then |@Set child to no child
|   |@Else |@Set child to ADDRESS's
|   |      |  reference
|@Result is Initial Element with
|  recreated links
```

Also, the only specialized part required for the AutoLink algorithm to work properly is to have AutoMap generate a few specialized functions for traversal and reconstruction of those user created data-types (C structs).

The generalization splits the send and receive functions in two parts : communication (eg MPI) specific and communication non specific, so that one can use the non specific part to create a genuine “message” that he can then transfer using methods of the communication mechanism chosen, and simply use on the other end the reverse message creation process to recreate the data-types in memory.

To do so, we would have to modify the algorithm so that there are only two data-type specific functions (communication system independent) : message aggregation and message expansion. The first one would generate a message by packing together the multiple nodes of the dynamic data-type to be sent. The second, would perform the reverse message generation process, and memory reconstruct the received message. It may be required to have a part that makes the data-type specific functions “communication system aware”; so that there is an actual mapping between the created message and the communication system.

The actual data transfer part (communication system dependent) would then be left open to the user discretion, understanding that in the case of the MPI standard, a simple MPI send/receive.

Then one will not be limited by the functions provided by AutoLink in the sense that as of now only the send and receive functions (blocking and non blocking) would be available to use on the data-type but any kind of transfer function from broadcast to reduce (as long as the reduce called process is “message aware”).

7 Conclusion

The software evolution required here is for future versions of the “MPI data-types tools” project, making it possible to evolve into a “data-types transfer tools”.

We will not speak of future implementations issues yet for those ideas are still to be looked into more closely, but this makes it possible for us to think of other possibilities regarding issues such as load balancing of data-types through nodes of distributed systems.

HTTP references

- MPI data-type tools :
  `http://www.nist.gov/itl/div895/savg/auto/`
- NIST :
  `http://www.nist.gov/`
- RÉSÉDAS :
  `http://www.loria.fr/equipes/resedas/`
- SAVG :
  `http://www.nist.gov/itl/div895/savg/`

References

[1] J. E. Devaney, M. Michel, J. Peeters, and E. Baland. AutoMap: A Software Tool for the Automatic Creation of MPI Data Structures From User Code. Technical report, NIST, April 1997. `http://www.itl.nist.gov/div895/savg/parallel/`.

[2] J. E. Devaney, M. Michel, J. Peeters, and K. Vrielink. AutoLink: An MPI C Library For Sending and Receiving Dynamic Data Structures. Technical report, NIST, April 1997. `http://www.itl.nist.gov/div895/savg/parallel/`.

[3] D. S. Goujon, M. Michel, J. Peeters, and J. E. Devaney. Automap and autolink : Tools for communicating complex and dynamic data-structures using mpi. *Lectures Notes in Computer Science*, 1362:98, 1998. Presented at CANPC’98.

[4] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, MA, 1994.

[5] B. W. Kernighan and D. M. Ritchie. *The C Programming Language, second edition*. Prentice Hall PTR, Englewood Cliffs, NJ, 1988.

[6] Message Passing Interface Forum. *MPI : A Message-Passing Interface Standard*.

[7] M. Michel and J. E. Devaney. Fine packet size tuning with autolink. *Proceedings of the 1999 ICPP Workshops*, page 295, 1999. Presented at IWPC’99.

[8] M. Michel, A. Schaff, and J. E. Devaney. Managing data-types : the corba approach and automap/autolink, an mpi solution. *Proceedings of the third MPI Developer’s and User’s Conference*, page 143, 1999. Presented at MPIDC’99.

[9] MPI: A Message Passing Interface Standard. HTML document, 1994. `http://www.mcs.anl.gov/Projects/mpi/index.html`.

[10] K. H. J. Vrielink, E. C. Baland, and J. E. Devaney. AutoLink: An MPI Library for Sending and Receiving Dynamic Data Structures. In *International Conference on Parallel Computing*. University of Minnesota Supercomputer Institute, october 3-4, 1996.

Disclaimer

Certain commercial products may be identified in order to adequately specify or describe the subject matter of this work. In no case does such identification imply recommendation or endorsement by the NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, nor does it imply that the products identified are necessarily the best available for the purpose.

License statement regarding AutoMap and AutoLink

This software was developed at the NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain.

AutoMap and AutoLink are experimental systems. NIST assumes no responsibility whatsoever for their use by other parties, and makes no guarantees, expressed or implied, about their quality, reliability, or any other characteristic.

We would appreciate acknowledgement if the software is used.