Fine Packet Size Tuning with AutoLink

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

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

Abstract

Using MPI to work with composed data-types, or data-types using pointers, is not a built-in operation. AutoMap and AutoLink are two tools that provide those functionalities to the MPI library, as a source-to-source compiler and as a subset of functions on top of MPI. AutoLink uses buffering by packets to transfer more than one node of the data-structure it is traversing, in a time/communication efficient way. This "Packet" needs to be adapted to the system (cpu, network) used to be the most suitable possible.

This article presents the "Fine Packet Tuning" tool developed to provide an easy way to do so.

1 Introduction

Parallel computing using distribution methods is made easier by using the Message Passing Interface (MPI) standard. Still, it is not easy to transfer data-types such as graphs or to create new data-types. An MPI solution called "MPI data-types tools" containing two tools, AutoMap and AutoLink, has been created by the National Institute of Standards and Technology for this purpose.

AutoMap creates the MPI data-types from the user's source code. AutoLink gives the user functions to transfer such data-types.

But AutoLink, using buffering by packet to hasten transfers, needs its Packet Size adapted to the system it is running on.

This paper provides an overview of the new addition to the AutoLink release, called the "Fine Packet Tuning" tool. Users of MPI desiring to work with AutoLink will obtain the best performance with the optimal packet size for their system.

First, we will present AutoMap and AutoLink. Then, we develop the concept used in the "Fine Packet Tuning" tool. Finally, we show an small example of use of the new tool.

2 AutoMap and AutoLink

2.1 AutoMap

AutoMap is a source-to-source compiler designed to read from user data-types definition files, C language `typedef` and `struct` entries. It generates a set of files containing MPI data-type definition and creation procedures.

AutoMap works by recognizing special directives, placed inside of C comments, that identify each data-type definition which is to be used as an MPI data-type. Sub-types (types that are used by the user type) do not have to be identified by directive; AutoMap will automatically generate requested types and sub-types. In addition, two special directives are used to tell AutoMap when to start and stop its parsing process.

The procedure created by AutoMap is defined inside the file `mpitypes.inc` that is to be included by the user main program file (on figure 1, it would be `userprog.c`). The procedure, `Build_MPI_Types()`, will generate new data-types to be sent by MPI commands (such as `MPI_Send` and `MPI_Recv`), where the MPI data-type name is the user data-type name prefixed by "`AM_`".

2.2 AutoLink

AutoLink is a library designed to allow users to send and receive "dynamic data-types" (data-structures using pointers to other data-structures, such as graphs) via MPI. It uses AutoMap (the generation process can be seen in figure 1) to parse the user data-types entries and provides some new high level functions to transfer those data-types.

These functions are :

- `AL_Send`, will send a dynamic data-structure (such as a graph) starting from its entry point, following each pointer link in a breadth first traversal, storing data into "Packets", and sending them.

- AL_Recv, will receive all the data sent by AL_Send, store the data, recreate the links between the fields of the data-structures, and give the user the entry point.

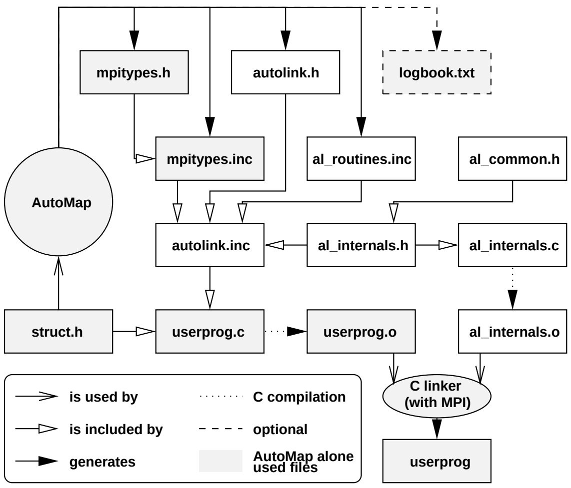

Figure 1. AutoMap and AutoLink files generation; `struct.h` is the user type definition file, `useprog.c` is the C source that uses the output of AutoMap/AutoLink

New AutoLink data-type nanmes are built from the user data-type names by prefixing them with "AL_".

Some parameters are left to the user to change or set so that he can fine tune them; examples include the AutoLink internal data-types sizes, the enabling of tracing/debugging/benchmarking modes for each MPI process, and specification of the default `PacketSize`.

2.3 PacketSize

As explained before, AutoLink uses Packets. Its notion of Packet is that of a buffer; AutoLink —reducing the number of communications required to send the entire dynamic data-type— stores some elements of each required data-type in a "to send" buffer and only sends the buffer when it is full.

The `PacketSize` is given in bytes so that each buffer is close to MPI message sizes. The buffer size is calculated to possess a number of possible elements such as (if BE represents the number of elements to be stored in the Packet for this element) :

$$BE = \begin{cases} \texttt{PacketSize}\ div\ \text{ElementSize} & \text{if} \geq 1 \\ 1 & \text{otherwise} \end{cases}$$

Entry 1 Example of data-type definition

```
typedef struct _fields fields;

struct _fields {
  char    keyshortcut;
  char    title[36];
  fields* next;
};

typedef struct _menu {
  char    title[72];
  char    comment[256];
  fields* thefields;
} menu;
```

The user dynamic data-type `menu` in entry 1 is of size 660 (considering a char on 2 bytes[1] and a pointer on 4 bytes). If the user keeps the `PacketSize` at the AutoLink default value of 4096, the number of possible elements in the `Packet` would be 6, using an actual buffer size of 3960 bytes. In like manner for the struct `fields`, this would be 52 elements (of 78 bytes) defining a buffer of 4056 bytes.

During the send process, should the full data-type contain less than 52 `fields` (considering that `menu` is the entry data-type) it will be sent only after the full traversal is completed. Otherwise, it will be sent in more than one packet.

3 PacketSize tuning

3.1 Concept

To find the best `PacketSize`, a fine tuning test case has been added to the AutoLink web page. The test case is composed of four components :

1. Structure definition and C program to perform the benchmark log,
2. A log extractor to extract pertinent data
3. A multiple run analyzer
4. A script to automate the run

3.2 C program

The test case works with the data-type defined in entry 2, defining a single linked list of nodes containing 16 char (the tests were run on a system where a char is 1 byte and a pointer 4 bytes).

[1] to code JIS (Japanese Industrial Standard) characters for example

The test program will run with the MPI processes connected in a ring; the MPI process with rank 0 will start the ring, and finish it. The simplified version of the algorithm is described in entry 3 (on page 3) for rank 0, and entry 4 (on page 3) for all other ranks. Since we are using a ring topology, the previous rank of rank 0 is the last rank, and the next rank of the last rank is rank 0.

Since the algorithm uses specific data as `PacketSizeMIN`, `PacketSizeMAX`, `PacketSizeINC`, and `LinkedListSIZE`, it is possible to provide them via the command line as arguments to the program, for specific fine tuning.

The algorithm sends and receives the same content in a ring, the Linked List of `LinkedListSIZE` elements, starting with a `PacketSize` (in bytes) equal to `PacketSizeMIN`, up to `PacketSizeMAX` (default value is calculated as $((LinkedListSIZE + 1) * SizeOfOneElement)$, so that the last packet contains the entire Linked List), each time increasing the `PacketSize` by `PacketSizeINC`.

Entry 2 PacketSize tuning used data-type

```
typedef struct _AL_LL_Test AL_LL_Test;

struct _AL_LL_Test
{
  char content[16];
  AL_LL_Test *next;
};
```

Entry 3 Simplified algorithm for Fine PacketSize Tuning (rank 0)

```
|Fill Linked List of SIZE elements
|Initialize MPI for AutoLink
|PacketSize = PacketSizeMIN
|While PacketSize < PacketSizeMAX
| |MPI Send PacketSize to next rank
| |Set PacketSize
| |AL Send LinkedList to next rank
| |MPI Recv PacketSize from previous
rank
| |AL Recv LinkedList from previous
rank
| |PacketSize += PacketSizeINC
|Send Stop Value to next rank
|Recv Stop Value from previous rank
```

3.3 Extracting data

AutoLink provides the user with an internal log if asked. In this log —depending of the debug level— are stored in-

Entry 4 Simplified algorithm for Fine PacketSize Tuning (all ranks but rank 0)

```
|Initialize MPI for AutoLink
|While not received Stop Value from
prev. rank
| |MPI Recv PacketSize from previous
rank
| |Set PacketSize
| |AL Recv LinkedList from previous
rank
| |MPI Send PacketSize to next rank
| |AL Send LinkedList to next rank
|Send Stop Value to next rank
```

formation that a script can use to extract time relevant data, such as sending time and receiving time.

The Perl script used, extracts an output file per MPI rank number, the real time since the beginning of AutoLink, the `PacketSize`, and the time required to complete one send or receive operation.

3.4 Analyzing data

To choose an optimal `PacketSize` for one environment, analyzing the extracted data on a certain number of run is a simple method.

A Perl script integrated into the AutoLink release is designed to extract such data in two ways :

1. Showing the most suitable `PacketSize` for the sending and receiving part of each MPI rank.
2. Storing some reference usable files for each MPI rank for graphing the send or receive time as a function of the `PacketSize`.

We use up to four different analysis parameters :

1. Min, will extract the minimum value for each `PacketSize`
2. Max, will extract the maximum value for each `PacketSize`.
3. Mean, will calculate the average value taken for sending of receiving for each `PacketSize`.
4. Median, will take from the data the center value.

3.5 Automating the analysis

A shell script is provided with the AutoLink release, so that one can automate the fine packet tuning process; its simplified algorithm is detailed in entry 5.

There are also entries detailing the way the user wants the analysis performed with entries such as :

- number of processors to run on,
- number of runs to perform and process,
- command line to run the MPI program,
- arguments of the MPI program (mostly to be able to use the specific command line arguments),
- mode in which to run the data analysis script

Entry 5 Fine PacketSize tuning automation algorithm

```
Phase1:
|While more run to do
| |Execute the MPI program
| |Extract data for this run
| |Store it for future analysis

Phase2:
|Analyze all the run
```

4 Use study

4.1 Configuration

4.1.1 System used

This analysis was performed on 5 SGI dual 225 MHz R10000 processor Octanes with 384 Mb memory running IRIX 6.5. The systems were also used by other processes during the tests.

The network connecting the workstations is a 10/100 Mb Ethernet, using the same switch.

4.1.2 MPI version

The analysis ran using the SGI MPI, which implements the MPI 1.2 standards.

4.2 Test parameters

The test was run for 100 times on the 5 octane cluster using 1 processor per system, and using the following parameters for the MPI program :

- `LinkedListSIZE` = 25000
- `PacketSizeMIN` = 0
- `PacketSizeMAX` = ((`LinkedListSIZE + 1`) * SizeOfOneElement)
- `PacketSizeINC` = 1000

4.3 Results

The tests ran for more than 41 hours, generating 5 output files per run from which were extracted data to perform the phase 2 analysis.

The results for phase 2 analysis concern the best `PacketSize` for all modes on each processor; these are shown in Table 1. In it, one can see that since the system was not running these MPI processes alone, the values found for such concrete operators as `Min` and `Max` are not strong enough values to be used. `Mean` calculations are already more accurate, but still some values are not precise enough to allow a concrete use. `Median` presents some very precise values.

MPI rank		0	1	2	3	4
Min	Send	241000	488000	492000	487000	29000
	Recv	11000	13000	11000	8000	11000
Max	Send	11000	487000	453000	249000	275000
	Recv	48000	15000	17000	40000	31000
Mean	Send	14000	487000	13000	487000	14000
	Recv	14000	15000	15000	17000	17000
Median	Send	14000	8000	15000	8000	15000
	Recv	7000	11000	8000	8000	8000

Table 1. Result for 100 runs on 5 MPI ranks

It is easy to see the median curve gives the most reliable measurement after seeing the figure 2 showing the time spent sending the entire Linked List (we are not showing the entire `PacketSize` range —that goes up to 500,000 bytes— for after 50,000 the values are quite similar).

Note also that the —minimal— value for `PacketSize` equal to 0 (similar to sending each element just after traversing it) is 7.43 seconds.

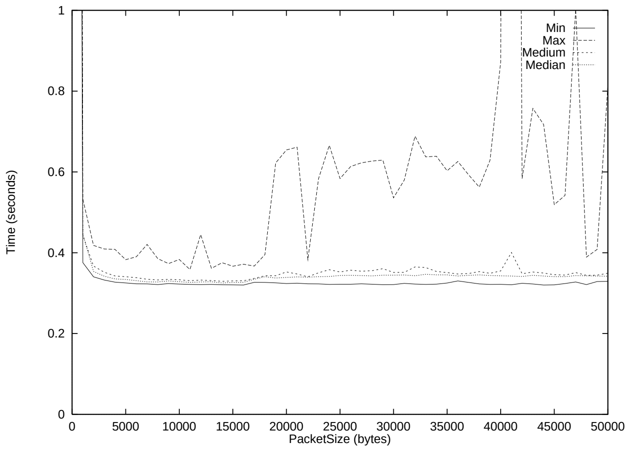

Figure 2. Rank 0, Send

Figure 3, presenting the median send plot for all ranks, indicates that a value for `PacketSize` between 5000 and 15000 would suit this environment.

Figure 4, presenting the median receive plot for all ranks, confirms it : a value for the `PacketSize` between 5000 and 15000 best fits this particular run environment.

The default AutoLink `PacketSize` value of 4096 was not the best value for this environment.

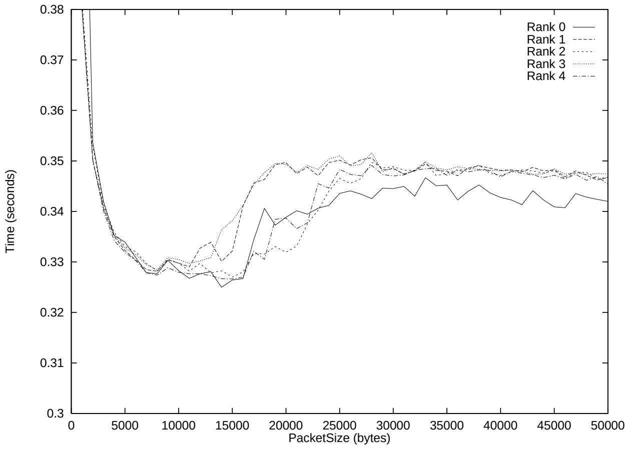

Figure 3. Median values for Send on all ranks

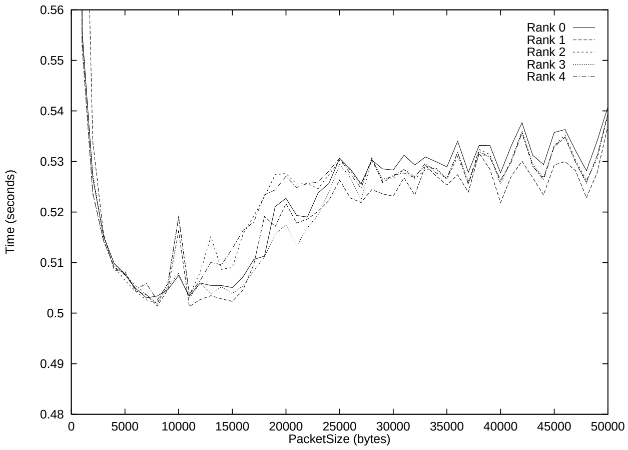

Figure 4. Median values for Receive on all ranks

A second run was done, on the same workstations, 100 times, changing the MPI program settings to :

- `LinkedListSIZE` = 25000
- `PacketSizeMIN` = 5000
- `PacketSizeMAX` = 15000
- `PacketSizeINC` = 200

The median results for all ranks for send and receive show that the value 8000 for `PacketSize` gives the most suitable value for this environment. This value should be used as the new default value (it can be changed in the `al_common.h` file, by setting the value in the `Makefile.base` provided with AutoLink, or by using the AutoLink command `AL_SetPacketSize()`).

5 Conclusion

In this article, we presented AutoMap and AutoLink, two MPI data-types tools, designed to help MPI programmers wotk with user defined data-structures. We also presented the "Fine Packet Tuning" tool that one can use with AutoLink to specify a `PacketSize` that will fit best the environment used. We saw that in some cases, the default `PacketSize` provided for AutoLink is not the best, that changing it would improve the performance of the tool.

New transfer methods are being added to the AutoLink tools and further improvements will give users a powerful tool to work with data-types.

HTTP references

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

References

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

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

[3] William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, MA, 1994.

[4] 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.

[5] Judith Ellen Devaney, Martial Michel, Jasper Peeters, and Koen Vrielink, "AutoLink: An MPI C Library For Sending and Receiving Dynamic Data Structures," Tech. Rep., NIST, April 1997, `http://www.itl.nist.gov/div895/sasg/parallel/`.

[6] Judith Ellen Devaney, Martial Michel, Jasper Peeters, and Eric Baland, "AutoMap: A Software Tool for the Automatic Creation of MPI Data Structures From User Code," Tech. Rep., NIST, April 1997, `http://www.itl.nist.gov/div895/sasg/parallel/`.

[7] Delphine Stéphanie Goujon, Martial Michel, Jasper Peeters, and Judith Ellen Devaney, "Automap and autolink : Tools for communicating complex and dynamic data-structures using mpi," *Lectures Notes in Computer Science*, vol. 1362, 1998, Presented at CANPC'98.

[8] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language, second edition*, Prentice Hall PTR, Englewood Cliffs, NJ, 1988.

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