FlexMPI

Providing elastic and monitoring capabilites to MPI applications

USER MANUAL

Version 1.1

Avda. de La Universidad 30. 28911 Leganes. Spain.

Maintained by David E. Singh

First edition: 5 November 2017

Contents

Introduction

FLEX-MPI is a runtime system that extends the functionalities of the MPI library by providing dynamic load balancing and performance-aware malleability capabilities to MPI applications. Dynamic load balancing allows FLEX-MPI to dynamically adapt the workload assignments to the performance of the computing nodes that execute the parallel application. Performance-aware malleability improves the performance of applications by changing the number of processes at runtime.

Project goals The main goals of this project are to provide the following capabilities in a transparent way, without user intervention, to a MPI application:

- To automatically create or remove new processes and redistribute the data among the existing ones.
- To perform run-time monitoring of the parallel application by means of performance counters.
- To apply different polices that can be used in combination with the malleable capabilities to automatically adjust the application performance according to different criteria.
- To control the spawn or removal actions by means of external control commmands, produced by the user or third-party applications (like the system scheduler).

How to cite FlexMPI?. If you are using FlexMPI and you want to cite it, please, cite the following reference: Gonzalo Martin, David E. Singh, Maria-Cristina Marinescu and Jesus Carretero. Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration. Parallel Computing. Vol. 46, No. 0. Pages: 60-77. 2015.

Components

FlexMPI consists of different components: the Monitoring module uses both PAPI and RAPL interfaces to dynamically collect performance metrics from the MPI application. The approach relies on low-level PAPI interface to collect hardware events like the MIPS or FLOPS. In addition, RAPL interface is used to collect the energy consumption of each processor. The Dynamic Process Management performs the runtime addition and removal of MPI processes, as well as the inter-process communication whenever a reconfiguring action is carried out. The Load Balancing Module balances the workload. The Data Redistribution Module transparently transfers data between MPI processes when the workload is being balanced or when processes are being created or removed. The Computational Prediction Model and Power Prediction Model calculate, respectively, the application performance and energy consumption for new processor configurations that are being evaluated. Finally, the Reconfiguring Policy Model evaluates whether the application satisfies the user-given objectives. If so, it continues executing on the same processor configuration. Otherwise, it performs a reconfiguring action by adding or removing processes. A more detailed description can be found in [MSMC15].

The current distribution includes the FlexMPI library and several support programs that can be used for testing purposes. These components are:

- The FlexMPI runtime, located in `src`, `include`, and `lib` directories.
- Several configuration files, located in `configuration_files` directory, that describe the compute node characteristics of the platform. These files are internally used by FlexMPI.
- An use case of application integrated with FlexMPI, located in `examples` directory, that performs the Jacobi iterative method. This application can be used for evaluating the FlexMPI capabilities.
- An execution script, located in `scripts` directory, for executing the Jacobi code in a simple way.
- An example of external controller, located in `controller` directory, that communicates with the previous use case application, sending configuration commands and receiving monitoring data.

Installation

This section explains how to install FlexMPI as well all the required software necessary for the program execution. Currently, FlexMPI has only been tested on Linux platforms. Support for other operating systems is not guaranteed.

Prerequisite software installation

FlexMPI is written in C and uses different libraries to implement some of the runtime's functionalities. These components have to be installed in the platform before installing FlexMPI. The following list shows the prerequisite software:

- GNU gcc compiler. Compiles the program and libraries and generate the executable. Supported version 4.9 or above.
- GNU Make. Controls the generation of FlexMPI executable from the program's source files and libraries. Supported version 3.81 or above.
- MPI library. Provides support to communicate and synchronize the processes involved in the simulation execution. FlexMPI have been tested with MPICH version 3.2 or above.
- GLPK, the GNU Linear Programming Kit. This library is used to peform automatic performance optimizations. Supported version 4.47 or above.
- PAPI library [MBDH99]. It is used for accessing to the performance counters. Supported version 5.5.1.0 or above.
- Nping, a network packet generation tool. It is used for sending commands to FlexMPI from the command prompt. Supported version 0.7.01 or above.

FLEXMPI INSTALLATION

This section describes how to install and compile FlexMPI. Once all the required software is properly installed, the installation process basically consists of two steps: (1) to extract the source files from the downloaded installation file and (2) to properly compile these files and generate the library and executables.

To complete the first step, you have to download the tarball file from the gitlab repository. Using a web browser, go to the following url:

`https://gitlab.arcos.inf.uc3m.es:8380/desingh/FlexMPI`

If you are a new user, you need to register first. Then, download the tar.gz file.

Then, you only have extract the runtime's files from the downloaded tarball file and rename the directory. In this document we assume that the extracted files are installed in the directory `FlexMPI`. The following listing shows an example for extracting the source files. Note that xxx is the hash key of the tar file. Note: **For the current configuration of the scripts, FlexMPI has to be installed in the user's home directory.**

```
# Extract the tarball
username@hostname:~/FlexMPI$  tar -zxvf FlexMPI-master-xxx.tar.gz
# Rename the directory
username@hostname:~$  mv FlexMPI-master-xxx     FlexMPI
```

The next step consists of generating the executable. By default, it is assumed that all the required software libraries have been installed in the local user LIB directory that is in the user's home directory (`~/LIBS`). In this case, it is only necessary to execute the `make` command to generate the executable. Otherwise, it is necessary to edit the `Makefile` files and include the paths to the related libraries.

```
# Compile the FlexMPI runtime
username@hostname:~/FlexMPI$  make
# Compile the example programs
username@hostname:~/FlexMPI/examples$  make
# Compile the controller demostrator
username@hostname:~/FlexMPI/controller$  make
# Provide permissions to the execution scripts
username@hostname:~/FlexMPI/scripts$  chmod 755 ./Execute*.sh
# Environment variables: assumming that the libraries are installed in $HOME/LIBS
username@hostname:~/FlexMPI$ export LD_LIBRARY_PATH=$HOME/LIBS/glpk/lib/:$HOME/FlexMPI/lib/
:$HOME/LIBS/mpich/lib/:$HOME/LIBS/papi/lib/:$LD_LIBRARY_PATH
```

FlexMPI execution

This sections describe how to execute FlexMPI. The current distribution includes an use case that performs the Jacobi iterative method integrated with FlexMPI. This section provides two examples for runnign this code. The first one is the simplest example with a stand-alone code. The second one includes a coordinated execution of the code with an external controller.

RUNNING A SIMPLE EXAMPLE

This section show how to run the example program in one compute node and how to communicate with it by means of the `nping` command. Two different command prompts are needed to run this example. The first one is used to execute the Jacobi application by means of the following command:

```
username@hostname:~/FlexMPI/scripts $   ./Execute1.sh 2 6666 6667 1
```

Where *Execute1.sh* is the execution script that runs the application. The first argument (2) is the initial number of processes, the second argument (6666) is the FlexMPI listening port for receiving commands, the third argument (6667) is the controller's listening port to receive the monitoring data (not applicable in this example because the controller is not used) and the last argument (1) is the application internal id.

The program output should be like the following one, where `compute-1` is the name of the compute node that is running the program.

```
username@hostname:~/FlexMPI/scripts $  ./Execute1.sh 2 6666 6667 1
Host compute-1 maxprocs is 12. ID is 0

[DEBUG] Initializing attribute successfully

[DEBUG] Setting detached state successfully

[DEBUG] Creating thread successfully
[1] Process spawned in compute-1 | Data loaded in 0.003427 secs.
[0] Process spawned in compute-1 | Data loaded in 0.006211 secs.
[0] Configuration: dim: 500 itmax: 10000 diff_tol: 0.000010 cpu_intensity: 100 com_intensity: 1 IO_intensity: 70

Policy is EMPI_Monitor_malleability_triggered
Iter: 100 FLOPs: 36709648321 MFLOPS:: 5046.735101 RTIME:: 7.273940 CTIME:: 0.016031 IOTime:: 0.000000 Size: 2
Policy is EMPI_Monitor_malleability_triggered
Iter: 200 FLOPs: 36302258290 MFLOPS:: 5041.587494 RTIME:: 7.200561 CTIME:: 0.000592 IOTime:: 0.000000 Size: 2
Policy is EMPI_Monitor_malleability_triggered
Iter: 300 FLOPs: 36302258270 MFLOPS:: 5041.247934 RTIME:: 7.201046 CTIME:: 0.000671 IOTime:: 0.000000 Size: 2
Policy is EMPI_Monitor_malleability_triggered
Iter: 400 FLOPs: 36302257996 MFLOPS:: 5041.667974 RTIME:: 7.200446 CTIME:: 0.000568 IOTime:: 0.000000 Size: 2
Policy is EMPI_Monitor_malleability_triggered
Iter: 500 FLOPs: 36302258291 MFLOPS:: 5041.320745 RTIME:: 7.200942 CTIME:: 0.000592 IOTime:: 0.000000 Size: 2
```

Note that it shows, for each 100 iterations: the iteration number (Iter), the absolute number of FLOPs (floating point operations) in the sampling interval (100 iterations) for the root process, the number of MFLOPS (floating point operations per second) for the root process, the user, communication and IO times of the sampling interval and the current number of processes (size value).

The second commmand prompt is used to send control commands to the application by means of `nping` command. For example, for creating two new processes, you should type the following command **replacing `compute-1` by the name of the compute node that you are using**[1].

```
username@hostname:~/FlexMPI/scripts $ nping --udp -p 6666 -c 1 compute-1 --data-string "6:lhost:2"
```

This command sends to the FlexMPI's listening port 6666 in `compute-1` node the command `"6:lhost:2"`. Command 6 performs a process spawn/destruction. `lhost` is the host name in which the processes are created/destroyed. `lhost` is an alias of the current compute node, you can see it especified in file corefile in `FlexMPI/configuration_files/corefiles` directory. Finally the last number is the number of processes modified. If the value is possitive, new processes are spawned, a negative number destroys existing processes in this compute node. **Note that it is not possible to destroy more processes than the initial number, 2 for this example.** The complete

[1] You can obtain this name by looking at the application output ("Process spawned in...") or executing the Linux command uname -n

list of commands can be seen in Section . The effect of this command can be seen in Jacobi program output.

```
Iter: 500 FLOPs: 36302258291 MFLOPS:: 5041.320745 RTIME:: 7.200942 CTIME:: 0.000592 IOTime:: 0.000000 Size: 2
Received packet from 10.0.40.12:52278 Data: 6:lhost:2

Command number is 6
Command: Create 2 processes in compute node: lhost
Sent 0 bytes as response
Policy is EMPI_Monitor_malleability_triggered
Spawn 2 at 43.300992
Iter: 600 FLOPs: 36302257986 MFLOPS:: 5041.678475 RTIME:: 7.200431 CTIME:: 0.000573 IOTime:: 0.000000 Size: 4
[2] Process spawned in compute-1 at 601 | Data received in 0.001189 secs.
[3] Process spawned in compute-1 at 601 | Data received in 0.001174 secs.
Policy is EMPI_Monitor_malleability_triggered
Iter: 700 FLOPs: 18272037517 MFLOPS:: 5119.567235 RTIME:: 3.569059 CTIME:: 0.074321 IOTime:: 0.000000 Size: 4
Policy is EMPI_Monitor_malleability_triggered
Iter: 800 FLOPs: 18214026665 MFLOPS:: 5104.331688 RTIME:: 3.568347 CTIME:: 0.061192 IOTime:: 0.000000 Size: 4
Policy is EMPI_Monitor_malleability_triggered
Iter: 900 FLOPs: 18213373846 MFLOPS:: 5103.804038 RTIME:: 3.568588 CTIME:: 0.060838 IOTime:: 0.000000 Size: 4
Policy is EMPI_Monitor_malleability_triggered
Iter: 1000 FLOPs: 18212326433 MFLOPS:: 5103.949614 RTIME:: 3.568281 CTIME:: 0.060708 IOTime:: 0.000000 Size: 4
```

We can observe that the number of processes is 4 after executing the command. The number of FLOPs decrease because part of the root process workload has been transferred to the new spaned processes. In case of running the program in a multicore node with at least four cores, the sample interval execution time (RTIME) should be reduced. You can remove one of the spawned processes by executing:

```
username@hostname:~/FlexMPI/scripts $ nping --udp -p 6666 -c 1 compute-4 --data-string "6:lhost:-1"
```

The application output should be like:

```
Iter: 1000 FLOPs: 18212326433 MFLOPS:: 5103.949614 RTIME:: 3.568281 CTIME:: 0.060708 IOTime:: 0.000000 Size: 4
Received packet from 10.0.40.12:37057 Data: 6:lhost:-1

Command number is 6
Command: Create -1 processes in compute node: lhost
Sent 0 bytes as response
Policy is EMPI_Monitor_malleability_triggered
Remove 1 at 61.685445
Process [3] removed from compute-1
Iter: 1100 FLOPs: 18407621625 MFLOPS:: 5116.241127 RTIME:: 3.597880 CTIME:: 0.061216 IOTime:: 0.000000 Size: 3
Policy is EMPI_Monitor_malleability_triggered
Iter: 1200 FLOPs: 24929281854 MFLOPS:: 5189.151770 RTIME:: 4.804115 CTIME:: 0.215792 IOTime:: 0.000000 Size: 3
Policy is EMPI_Monitor_malleability_triggered
Iter: 1300 FLOPs: 25095791927 MFLOPS:: 5251.143398 RTIME:: 4.779110 CTIME:: 0.230193 IOTime:: 0.000000 Size: 3
```

RUNNING FLEXMPI WITH AN EXTERNAL CONTROLLER

We have developed an external program called controller that commmunicates with FlexMPI to send commands and receive monitoring information. The controller reads the file `workload` located in `controller` directory. This file has one line per application that is being executed (each application is independent). In each line, the first string is the application name (jio corresponds to the Jacobi code), then there is a list of duplets `{node_name,n_procs}`, where `node_name` is the name of the compute node and `n_procs` is the initial number of processes executed in the system. For instance, for the following workload directory two Jacobi applications are executed in the local node.

```
jio:compute-1:4
jio:compute-1:2
```

The first application runs with four processes and the second one with two. The controller produces in the `controller/execscripts` directory as many execution scripts as lines in the workload file. Each execution script runs the Jacobi code with the required configuration. Note that these scripts are automatically created and executed by the controller.

To execute it, we need to type in a commmand prompt.

```
username@hostname:~/FlexMPI/controller $  ./controller
```

Using the previous workload file, it will automatically execute two Jacobi applications. The applications' output is located in the `controller/logs` directory.

To communicate with each application, you need to type FlexMPI commands using the following syntax: `"app_id command"`, where app_id is the application id (the first one in the workload file has id 0, the second one has id 1) and command is the FlexMPI command.

For instance, the following example uses first the commands `0 4:on` and `1 4:on` to activate the monitoring component of FlexMPI for each application. A new thread is created in the application that periodically collects and sends to the controller different monitoring data. On the controller, another thread is created (one per application that has the monitoring activated). This thread collects and displays the data sent by FlexMPI. Then, it uses commmand `0 6:lhost:2` for creating two new processes in the first application. You can observe the last line of the monitoring data that the application size is increased.

```
************************************************************
FlexMPI program controller 1.05
************************************************************

— Initializing

### Application maleability enabled

Reading the workload ...
@@@@ App [0], root node: localhost
Creating the rankfile 1 ...
Creating the execution script 1 ...
Executing the application 1 ...
@@@@ App [1], root node: localhost
Creating the rankfile 2 ...
Creating the execution script 2 ...
Executing the application 2 ...

— Creating the listerner threads
Initializing attribute successfully
Setting detached state successfully
Creating listener thread successfully
Initializing attribute successfully
Setting detached state successfully
Creating listener thread successfully

— Displaying the application workload
Application 0, Port1 (listener): 6666 Port2 (Sender): 6667
localhost 4
Application 1, Port1 (listener): 6668 Port2 (Sender): 6669
localhost 2
Please type a command [appId command]:

— Waiting for new input commands

0 4:on
Message: 4:on. Size: 500 bytes sent to app1.
0 data from 127.0.0.1:33418 –> [compute-1] rtime 0 ptime 0 ctime 0.000000 Mflops 1 PAPI_SP_OPS 0
PAPI_TOT_CYC 0 Ratio=-nan iotime 0.000000 size 4

1 4:on
Message: 4:on. Size: 500 bytes sent to app2.
1 data from 127.0.0.1:33233 –> [compute-1] rtime 0 ptime 0 ctime 0.000000 Mflops 1 PAPI_SP_OPS 0
PAPI_TOT_CYC 0 Ratio=-nan iotime 0.000000 size 2


0 6:lhost:2
Message: 6:lhost:2. Size: 500 bytes sent to app1.
0 data from 127.0.0.1:33418 –> [compute-1] rtime 3597220 ptime 3602790 ctime 0.005926 Mflops 20196
PAPI_SP_OPS 0 PAPI_TOT_CYC 24414653641 Ratio=inf iotime 0.000000 size 4

1 data from 127.0.0.1:33233 –> [compute-1] rtime 7199452 ptime 7200341 ctime 0.001092 Mflops
10087 PAPI_SP_OPS 0 PAPI_TOT_CYC 24398573397 Ratio=inf iotime 0.000000 size 2

0 data from 127.0.0.1:33418 –> [compute-1] rtime 2399344 ptime 2463459 ctime 0.064228 Mflops
30845 PAPI_SP_OPS 0 PAPI_TOT_CYC 24975784510 Ratio=inf iotime 0.000000 size 6
```

RUNNING FLEXMPI IN A LARGER NUMBER OF COMPUTE NODES

This section shows how to execute FlexMPI-based applications in a generic number of compute nodes. For the sake of simplicity we will only describe the configuration steps running a stand-alone application (without the controller). To do so, it is necessary to set up the following configuration files:

- In `configuration_files/corefiles` the corefile has one line per compute node. File `"corefile.demo2"` includes an example a configuration file with four compute nodes. The first line of the file has de following contents: `"compute1:12:comp1:100:5170"` where compute1 is the compute node name (obtained by uname -n), 12 is the number of cores of the compure node, comp1 is the alias (used with FlexMPI command 6), 100 is the operational cost and 5170 is the relative compute power (used for load balancing).
- In `configuration_files/nodefiles` the nodefile has one line per compute node. File `"nodefile.demo2"` shows the file contents for the previous example, where each lines corresponts to the previous corefile, including the compute node name and number of cores.
- In `controller/rankfiles` contains the MPI rankfile. File `"rankfile.demo2"` shows the rankfile contents for the considered example.

Once you have updated these files with the compute nodes that you are using, you can execute Jacobi application by means of the following scripts:

```
username@hostname:~/FlexMPI/scripts $  ./Execute3.sh 2 6666 6667 1
```

Now, you can try to spawn processes in more than one compute node. For instance:

```
username@hostname:~/FlexMPI/scripts $ nping --udp -p 6666 -c 1 compute-4 --data-string "6:node1:2:node2:1:node3:3"
```

That produces the following output:

```
Host compute-1 maxprocs is 12. ID is 0
Host compute-2 maxprocs is 12. ID is 1
Host compute-3 maxprocs is 12. ID is 2

[DEBUG] Initializing attribute successfully

[DEBUG] Setting detached state successfully

[DEBUG] Creating thread successfully
[1] Process spawned in compute-1 | Data loaded in 0.003422 secs.
[0] Process spawned in compute-1 | Data loaded in 0.006392 secs.
[0] Configuration: dim: 500 itmax: 10000 diff_tol: 0.000010 cpu_intensity: 100 com_intensity: 1 IO_intensity: 70

Policy is EMPI_Monitor_malleability_triggered
Iter: 100 FLOPS: 36703503582 SMFLOS:: 5045.011583 RTIME:: 7.275207 CTIME:: 0.014139 IOTime:: 0.000000 Size: 2
Policy is EMPI_Monitor_malleability_triggered
Iter: 200 FLOPS: 36303023005 SMFLOS:: 5040.310519 RTIME:: 7.202537 CTIME:: 0.000996 IOTime:: 0.000000 Size: 2
Received packet from 10.0.40.12:38308 Data: 6:node1:2:node2:1:node3:3

Command number is 6
Command: Create 2 processes in compute node: node1
Command: Create 1 processes in compute node: node2
Command: Create 3 processes in compute node: node3
Sent 0 bytes as response
Policy is EMPI_Monitor_malleability_triggered
Spawn 6 at 7.227459
Host list: compute-1:2,compute-2:1,compute-3:3
[5] Process spawned in compute-3 at 301 | Data received in 0.001871 secs.
[4] Process spawned in compute-2 at 301 | Data received in 0.001887 secs.
[7] Process spawned in compute-3 at 301 | Data received in 0.001881 secs.
[6] Process spawned in compute-3 at 301 | Data received in 0.001887 secs.
Spawn cost:: process_creation= 0.924374 data_redistribution= 0.002459 LoadBalance_computation= 0.000001
Iter: 300 FLOPS: 36367660864 SMFLOS:: 5049.052803 RTIME:: 7.202868 CTIME:: 0.023683 IOTime:: 0.000000 Size: 8
[3] Process spawned in compute-1 at 301 | Data received in 0.002502 secs.
[2] Process spawned in compute-1 at 301 | Data received in 0.002637 secs.
Policy is EMPI_Monitor_malleability_triggered
Iter: 400 FLOPS: 9857804592 SMFLOS:: 5616.160250 RTIME:: 1.755257 CTIME:: 0.245347 IOTime:: 0.000000 Size: 8
```

FlexMPI commands

Table 1 shows the list of FlexMPI input commmands. These commands are also summarized next:

- The first command allows to setup malleability policies based on different optimization criteria. This option requires of tunning several internal FlexMPI parameters that are not described in the current version of this manual.
- The second command performs a load balancing action at the end of the sampling interval. This action takes into account both the computational power of the existing processors and the performance of each application process. The load balancing involves redistributing the partitioned application arrays and vectors. This is done automatically, in a user-transparent fashion.
- The third command is designed for testing purposes. It displays on the application's output, several performance values.
- Command number four activated/deactivates the FlexMPI monitoring service. It requires of a external application (like the controller) that is listening to the receiver port, and receives the FlexMPI's messages. In the default configuration, the controller has to be executed in the same node as the one running the application root process. However, it is possible to specify a generic controller node by changing the FlexMPI configuration.
- The fifth command terminates the application. This is done by FlexMPI in a asynchronous manner.
- Command six is used to spawn or remove application processes. The syntax is a sequence of valid compute nodes with the increment of processes. Possitive and negative values correspond to spawned and removed processes, respectively. It is not possible to combine in the same commmand different signs of values. That is, for each commmand, all the values have to be either possitive or negative. Compute nodes with zero increment of processes can be included in the list or skipped from it.
- FlexMPI monitors two user-defined performance counters. Commmand seven permits to specify the counter names. These names are the ones used by PAPI library. For instance, command `"7:PAPI_STL_ICY:PAPI_TOT_INS:"` sets the firts counter to *Cycles with no instruction issue* and the second one to *Number of instructions completed*. You need to make sure that the compute node architecture supports the specified counters.
- Commmand eight binds the application processes to the compute cores. This is done by providing a secuence of process numbers (that corresponds to the rank value)

and the core id (related to the compute node where the application is running). For instance, for a 8-process application running in a single multicore node, the following sequence: `"8:0:0:1:0:2:0:3:0:4:1:5:1:6:1:7:1"`, binds the first four processes to the first core (core 0) and the remaining four to the second core (core 1).

Table 1: FlexMPI command list.

Command No.	Description	Syntax
1	Change the malleability policy	–
2	Performs load balancing	2:
3	Displays the performance values	3:
4	Turns on/off the monitoring service	4:on: or 4:off:
5	Terminates the application	5:
6	Spawns/removes processes	6:node1:p1:node2:p2:....:noden:pn
7	Changes the monitoring performance counters	7:perf_counter1:perf_counter2:
8	Performs core binding	8:p1:c1:p2:c2:...:pn:cn

Bibliography

[MBDH99] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[MSMC15] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero. Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration. *Parallel Computing*, 46(0):60 – 77, 2015.