Runtime Configuration for MPC - User Manual

Sogeti High Tech

June 5, 2013

Contents

1 Introduction **1**

2 The configuration file structure **1**

3 Define a configuration file **1**
- 3.1 Defining profiles . . . 1
  - 3.1.1 Step 1: Define a default profile . . . 1
  - 3.1.2 Step 2: Define a profile for debugging . . . 2
- 3.2 Mapping profiles . . . 2

4 Configuration test **2**
- 4.1 Simple execution without parameters . . . 3
- 4.2 Execution in debugging mode . . . 3
  - 4.2.1 Using the environment variable MPC_DEBUG . . . 3
  - 4.2.2 Using the option –profiles of mpcrun . . . 3
  - 4.2.3 Using the environment variable MPC_USER_PROFILE . . . 3
- 4.3 Simple execution with parameters . . . 3

5 Network configuration **4**

6 mpc_print_config **5**

1 Introduction

Since MPC 2.5.0, a configuration system has been introduced through the module `MPC_Config`: it enables the user to setup some parameters at the runtime when running his binary with MPC.

This manual will explain to an user how he can use and modify the configuration for running his applications with `mpcrun`.

2 The configuration file structure

In MPC 2.5.0, some variables can be configured during the runtime. To do this, a configuration file defines all the variables and their values, and MPC will load it at the execution.

An example of a configuration file can be found in the `$PREFIX/share/mpc` folder, where `$PREFIX` matches to the install directory of MPC.

The configuration file is written in XML, and two parts make up it:

1. A first one listing all the available profiles and their associated parameters;
2. An other one defining which profiles to apply at runtime depending on the values of environment variables.

3 Define a configuration file

3.1 Defining profiles

In this manual, and to illustrate the configuration system, we will define two profiles: the default one and an other for debugging.

3.1.1 Step 1: Define a default profile

A configuration file must have a default profile (see XML just beyond): it initializes all the variables of the configuration system with their default values.

Example of `default` profile

```
<profile>
  <name>default</name>
  <modules>
    <launcher>
      <nb_node>4</nb_node>
      <nb_processor>4</nb_processor>
      <share_node>false</share_node>
    </launcher>
    <allocator>
      <numa>true</numa>
      <debug>false</debug>
    </allocator>
  </modules>
<profile>
```

This simple example defines three variables for a module called `launcher`, and two for `allocator`.

The parameters inside a module can be of different types (matching to C types) :

- Numbers such as integers, doubles, etc.;
- String;
- Size (i.e. `50 MB`, `10 PB`, etc.);
- Boolean (`true` or `false`);
- Enum;

- Function Pointer.

All the parametrizable variables can be listed by executing the command `man mpc_config`: it gives information (type, default value, description) for each variable.

3.1.2 Step 2: Define a profile for debugging

To debugging applications, some parameters need to be initialized with different values than the default profile ones. So the user can defined a new profile as beyond:

Example of `debug` profile

```
<profile>
  <name>debug</name>
  <modules>
    <launcher>
      <nb_node>2</nb_node>
      <nb_processor>2</nb_processor>
    </launcher>
    <allocator>
      <debug>true</debug>
    </allocator>
  </modules>
<profile>
```

While the default profile must have `default` as name, the name of this new profile has no importance: to be coherent with our example, we call it `debug`. Selecting this profile will overwrite the default values with those ones.

3.2 Mapping profiles

One way to select the profiles to apply during execution is to define mapping in the configuration file. The XML code beyond will enable the user to select the `debug` profile depending on the value of the environment variable `MPC_DEBUG`.

Example of mapping to apply `debug` profile

```
<mapping>
  <selectors>
    <env name=''MPC_DEBUG''>true</env>
  </selectors>
  <profiles>
    <profile>debug</profile>
  </profiles>
<mapping>
```

The user can define multiple mappings which will be surrounded with `<mappings>...</mappings>`. By default, the `default` profile is applied which means that if the user does not defined any mapping, the variables into the configuration will be initialized.

4 Configuration test

MPC will use many configuration files that it will in the following order:

1. `$PREFIX/share/mpc/config.xml`;
2. `$PREFIX/share/mpc/config.xml.example` if the previous file does not exist;
3. the file given by the parameter `--config` of `mpcrun` if present.

As an example, let's assume an executable called `test`: its goal is to simply print the values of the variables which are in the configuration system.

4.1 Simple execution without parameters

If the binary `test` is simply called executing the command `./test`, the ouput is:

Output of a simple execution

```
launcher:
    nb_node: 4
    nb_processor: 4
    share_node: false

allocator:
    numa: true
    debug: false
```

If nothing is precised, the `default` profile will be loaded at execution.

4.2 Execution in debugging mode

4.2.1 Using the environment variable MPC_DEBUG

If the user executes the command `MPC_DEBUG=true ./test`, this will enable the `debug` profile:

1. the values of the `default` are first loaded;
2. all the variables redefined in the `debug` profile are overwritten, and all the others keep their previous values.

The output of this command is:

Output when enabling the debug mode

```
launcher:
    nb_node: 2
    nb_processor: 2
    share_node: false

allocator:
    numa: true
    debug: true
```

4.2.2 Using the option –profiles of mpcrun

The user can also precise the profiles he wants to apply using the option `--profiles` of `mpcrun`.

The command `mpcrun --profiles=debug ./test` produces the same output as in §4.2.1.

4.2.3 Using the environment variable MPC_USER_PROFILE

The user can also precise the profiles he wants to apply using the environment variable `MPC_USER_PROFILE`.

The command `MPC_USER_PROFILE=debug ./test` produces the same output as in §4.2.1.

4.3 Simple execution with parameters

If an argument is passed to the executable, it will overwrite the values put in configuration for the associated variable. For example, if the argument `-n=3` sets the number of nodes, the command `./test -n=3` outputs:

Simple execution with parameters

```
launcher:
    nb_node: 3
    nb_processor: 2
    share_node: false

allocator:
    numa: true
    debug: false
```

5 Network configuration

The network that will be used during the execution can also be parametrized into the configuration file between the tags `<networks>...</networks>`. For the moment, MPC 2.5.0 only supports Infiniband and TCP.

The first step is to defined all the network configurations available:

Define network configurations

```
<configs>
  <config>
    <name>ib1</name>
    <driver>
      <infiniband>
        <param1>4</param1>
        <param2>300</param>
      </infiniband>
    </driver>
  </config>
  <config>
    <name>ib2</name>
    <driver>
      <infiniband>
        <param1>2</param1>
        <param2>1500</param>
      </infiniband>
    </driver>
  </config>
  <config>
    <name>tcp1</name>
    <driver>
      <tcp>
        <fake_param>0</fake_param>
      </tcp>
    </driver>
  </config>
</configs>
```

Next step is to defined the different rails which will use the previous network configurations. A rail consists in a specific topology to use, and one of the defined configurations.

Define network configurations

```
<rails>
  <rail>
    <name>rail_ib1</name>
    <device>0</device>
    <topology>ondemand</topology>
    <config>ib1</config>
  </rail>
  <rail>
    <name>rail_ib2</name>
    <device>0</device>
    <topology>fully</topology>
    <config>1500</config>
  </rail>
  <rail>
    <name>rail_itcp1</name>
    <device>0</device>
    <topology>fully</topology>
    <config>tcp1</config>
  </rail>
</rails>
```

The final step is to defined the different options that the user can choose with the option `--net` of `mpcrun`.

Define network modes

```
<cli_options>
  <cli_option>
    <name>ib</name>
    <rails>
      <rail>rail_ib1</rail>
    </rails>
```

```
  </cli_option>
  <cli_option>
    <name>tcp</name>
    <rails>
      <rail>rail_tcp1</rail>
    </rails>
  </cli_option>
  <cli_option>
    <name>tcpoib</name>
    <rails>
      <rail>rail_ib1</rail>
      <rail>rail_ib2</rail>
    </rails>
  </cli_option>
</cli_options>
```

The configuration system allow the user to define multirail by selecting many rails for a network mode. Using the previous code, the options for the option `--net` are `ib`, `ib` and `tcpoib`.

6 mpc_print_config

`mpc_print_config` is a small executable that print the configuration in different ways:

- XML mode:

```
<profile>
  <name>default</name>
  <modules>
    <launcher>
      <nb_node>4</nb_node>
      <nb_processor>4</nb_processor>
      <share_node>false</share_node>
    </launcher>
    <allocator>
      <numa>true</numa>
      <debug>false</debug>
    </allocator>
  </modules>
<profile>
```

- text mode:

```
config:
    modules:
        launcher:
            nb_node: 4
            nb_processor: 4
            share_node: false

        allocator:
            numa: true
            debug: false
```

To get more information, execute `mpc_print_config --help`.