Introduction to the Use of Dimemas

Tutorial material

This tutorial contains an example and guidelines to get started on the use of Dimemas. In the directory where it is located you can find:

- A Paraver trace `Iberia-128-CA.chop1.1it.shifted.prv` for a 128 processor run of the WRF application. The traces was obtained on MareNostrum with Extrae, using the `LD_PRELOAD` mechanism to intercept entries and exits to MPI. The trace contains events on entry and exit to the MPI calls and hardware counter (cache misses, instructions and cycles) events at these points.
- Its corresponding *.pcf* file with the symbolic information for the numerically encoded records in the trace.
- A directory `./cfgs` with some paraver configuration files that will be used during the analysis of the traces obtained in this session.
- Four initial Dimemas configuration files named `MN.128.{1,2,4,8}ppn.cfg`
- A directory `./full_session.1ppn` with all the derived Dimemas configuration files and its simulation outputs for the 1-processor-per-node configuration. We recommended to initially ignore its content and follow the tutorial to get acquainted with Dimemas and DimemasGUI.

Objective

We provide a Paraver trace file describing an actual run (one iteration) of the WRF code in MareNostrum2 supercomputer. Our objective is to perform several Dimemas simulations to identify the sensitivity of the application performance to interconnect parameters.

A first step would be to model with Dimemas the actual MareNostrum2 configuration and check whether the prediction matches the actual behavior. Then we can proceed to study the potential benefit of hypothetical changes in the architecture.

Generating a Dimemas trace and performing a simulation

The first step is to convert the Paraver trace into a Dimemas file using `prv2dim`. To do so, execute the following command in a terminal window:

```
> $(DIMEMAS_HOME)/bin/prv2dim
Iberia-128-CA.chop1.1it.shifted.prv
Iberia-128-CA.chop1.1it.shifted.dim
```

The Dimemas configuration file `MN.128.1ppn.cfg` included in the tutorial directory describes an architecture model idealized from MareNostrum2 interconnection network (8 us latency, 250 MB/s but no contention in the network) with only one process per node. Run the following the command in a terminal window:

```
> $(DIMEMAS_HOME)/bin/Dimemas -S 32K -p
```

```
prediction.1ppn.prv MN.128.1ppn.cfg
```

In a few seconds you should get the `prediction.1ppn.prv` Paraver trace trace which is a reconstruction of what would have been the behavior on the machine modeled by the `MN.128.1ppn.cfg` file.

We can now load the original trace (`Iberia-128-CA.chop1.1it.shifted.prv`) in Paraver and the `cfgs/mpi_call.cfg`. We can do the same thing with the predicted trace (`prediction.1ppn.prv`). Also load `cfgs/mpi_call.cfg`) with this second trace. In order to ensure that both windows are at the same timescale, we should copy it from the original to the synthetic trace (right click in the original trace timeline window, select "*Copy*", and then move to the timeline of the synthetic trace and do "*Paste Size*" and "*Paste Time*"). We can see that the prediction of Dimemas is slightly optimistic, but quite close to the reality.

Changing the machine model

Let us assume that we are interested in finding out what would be the impact of a slower network (i.e. only 10 MB/s). We should use the Dimemas GUI to tune the *cfg* file. First load the GUI executing following command in your terminal:

```
> $(DIMEMAS_HOME)/bin/DimemasGUI
```

Once the it has load, follow the next sequence of action:

1. "*Configuration*" → "*Load configuration*" and select the file `MN.128.1ppn.cfg`
2. "*Configuration*" → "*Target configuration*". Click on the "*Config*" button by the "*Environment information*" to change the "*Network Bandwidth*" to 10.0 MB/s (please introduce the trailing dot and zero characters). Then click "*Save*".
3. "*Configuration*" → "*Save configuration*" to a file (e.g. `MN.128.1ppn.10MBps.cfg`)

Perform the Dimemas simulation specifying a name for the output Paraver file and the just saved Dimemas configuration file:

```
> $(DIMEMAS_HOME)/bin/Dimemas -S 32K -p
prediction.1ppn.10MBs.prv MN.128.1ppn.10MBps.cfg
```

Now, load the new Paraver trace, load the `cfg/mpi_call.cfg` on it and copy its timescale to the other two traces we had already loaded. We can see that although the reduction in bandwidth was very significant (divided by 25), the actual impact on performance was not huge (just 15%)

We may thus wonder what would be the impact of reducing bandwidth further. Let us say to just 5 MB/s. You can repeat this process generating a new configuration file with this modification in the interconnection network bandwidth. Now the impact starts to be larger. It is also apparent that the impact is not the same in all phases of the time-line. you can load the `cfgs/p2p_size.cfg` configuration and it will be apparent that the communication phases that are more sensitive to the bandwidth are those with larger messages as one would expect. In general, the actual impact of a reduction in bandwidth will depend on the computational granularity of the application, the message sizes, but also on the level of load balance and the actual use of asynchronous communications within the application or its tolerance to shifts in process pipelining.

- What is the impact of the latency?

You may start from the original `MN.128.1ppn.cfg` description. Come back to the Dimemas GUI, load the configuration file asdo the following steps:

1. *“Configuration*" → "*Target configuration”*. Push the "*Config*" button by the "*Node information*" label.
2. Change the "*Startup*" value under the "*Inter-node communication parameters*" section and set it for example to 0.0001, to model a 100 us latency.
3. After changing it click on the "*Do all the same*" button to apply the new latency to all nodes (you could actually specify different latencies for different nodes, the node number's showed up at the top of this window).
4. "*Save*" the specified latencies (startups). In Dimemas terms, latency is not end to end but actually represents the local overhead an MPI implementation has. It is assumed to use the CPU and after it, data transfer can start.
5. Create a new Dimemas configuration file (again "*Configuration*" → "*Save configuration*") and perform the simulation (executing the simulator in the terminal window).

If you load the resulting trace in Paraver, you can see that the performance impact of such bad latency is negligible for this trace.

- Would I benefit from multi-rail adapters?

Start from the configuration of 5 MB/s. Go back to the Dimemas GUI. In the "*Node information*" window change the "*Input links*" and "*Output links*" under the "*Inter-node communications parameters*" section to 2. Save the configuration file and simulate.

You can see a certain potential gain by this configuration. One might also study increasing the input links but not the output links. Although not a foreseeable feature of future architectures, this structure can give insight on the structure of the application. For example if one region of an application shows an improvement in this situation, it means that there is significant end point contention. One would probably suggest to the application developer to restructure the schedule of communications so that not all processes send to the same processes at the same time, a frequently disregarded problem in many codes.

- What is the impact of contention?

Starting again from the original machine description you can wonder whether contention caused by bad routing can hurt the performance. One way of modeling this at a very abstract level is to change the "*Number of buses*" parameter in the "*Environment information*" window. The number you put in this field is the maximum number of possible concurrent transfers (except that a 0 means no limit on the number of concurrent communications). A value of 1 would mean the network topology would be a bus, with only one possible transfer at any time. You can vary this parameter and see how sensitive is the application to contention. In our case, the application still performs well with a very small number of concurrent transfers (i.e. 2 is actually enough).

- What would be the impact of a faster processor?

For the previous machine description change the "*Relative processor speed*" of the "*Node information*" window to 5.0. This will model a processor 5 times faster in the execution of the sequential computation burst between MPI calls. You will observe that now the application is more sensitive to contention. You will need to increase the number of buses to achieve a good efficiency.

An interesting simulation is to assume infinite bandwidth (you have to put a 0 in the "*Network bandwidth*" field on the "*Environment information*" window) and zero latency ("*Startup*" value under the "*Inter-node communication parameters*" on the "*Node information*" window). This will give a limit of the efficiency of the application due to load imbalance and dependence chains.

- What happens if we simulate 2 processors per node? What about 4 and 8?

At this point, try to repeat the whole analysis described in this section varying "*Number of processors*" per node, and the processes (or "tasks") assigned to each node. To do so, run the simulator using the Dimemas configuration files `MN.128.2ppn.cfg` (2 processes per node), `MN.128.4ppn.cfg` (4 processes per node), and `MN.128.8ppn.cfg` (8 processes per node). Compare the results with the simulations with less processors.

Visualization of the internals of the communication

Given that Dimemas has knowledge of the actual point in time where a data transfer takes place, it can emit such information to the generated Paraver file. The following Paraver configuration files shows how this information can be visualized or which measurements can be made. Let us assume we do a simulation with 10x processor speed, only 20 concurrent communication.

- Configuration file `/cfgs/used_network_bw.cfg` >displays the aggregated instantaneous bandwidth through the network.