Introduction to Paraver and Dimemas methodology (MPI analysis)

This tutorial is structured as a bunch of rules that can be verified during the analysis process. The results of looking at one rule diagnosis may open new rules to look at, like in a search tree. The goal of this tutorial is not to describe the full search tree but to focus on the first steps and show how to decide which branches are interesting to explore. As a general characteristic of the analysis methodology with Paraver and Dimemas, the approach is based on looking at the temporal and spacial distribution of the performance data to understand the application behavior, detect its different phases and identify the behavioral structure (that may be different from the procedural structure). This detailed analysis allows to extract a lot of information from the performance data collected during the run.

WARNING::: The first section is based on a tracefile without samples. If the tracefile was obtained with samples the configuration files related to correlate hardware counters with code regions have to be selected from cfgs/sampling.

The first question to answer when analyzing a parallel code is "how efficient does it run?". The efficiency of a parallel program can be defined based on two aspects: the parallelization efficiency and the efficiency obtained in the execution of the serial regions. These two metrics would be the first checks on the proposed methodology.

- To **measure the parallel efficiency** load the configuration file `cfgs/mpi/mpi_stats.cfg` This configuration pops up a table with %time that every thread spends in every MPI call. Look at the global statistics at the bottom of the outside mpi column. Entry *Average* represents the application parallel efficiency, entry *Avg/Max* represents the global load balance and entry *Maximum* represents the communication efficiency. If any of those values are lower than 85% is recommended to look at the corresponding metric in detail. Open the control window to identify the phases and iterations of the code.
- To **measure the computation time distribution** load the configuration file `cfgs/general/2dh_usefulduration.cfg` This configuration pops up a histogram of the duration for the computation regions. The computation regions are delimited by the exit from an MPI call and the entry to the next call. If the histogram does not show vertical lines, it indicates the computation time may be not balanced. Open the control window to look at the time distribution and visually correlate both views.
- To **measure the computational load (instructions) distribution** load the configuration file `cfgs/papi/2dh_useful_instructions.cfg` This configuration pops up a histogram of the instructions for the computation regions. The computation regions are delimited by the exit from an MPI call

and the entry to the next call. If the histogram doesn't show vertical lines, it indicates the distribution of the instructions may be not balanced. Open the control window to look at the time distribution and correlate both views.

- To **measure the serial regions performance** look at the IPC timeline loaded with `cfgs/general/2dh_usefulduration.cfg`. What it's a reasonable IPC would depend on the machine used to run the application, but typically values lower than 1 identify poor performance sections. You can correlate the IPC with the computation time modifying the `Statistic` of the useful duration histogram to use `correlate with metric` and verify that the selected `Metric` is `Instructions per cycle`. Now the cell color corresponds to the IPC showing the correlation between duration (position) and IPC (color). Zooming into an unbalanced region of the histogram would allow you to verify if the unbalance is related to a different IPC. Change the `Metric` to `Instructions` to correlate the duration with the instructions.

With these 4 views we've been able to get a first assessment of the parallel efficiency of the code, to identify if either balance or communication are limiting the performance, to analyze the structure and distribution of the computation with respect to duration and instructions and to measure the performance of the sequential regions (IPC) correlating its impact on the region duration. Before a deeper analysis of the facts we detected as bad performing, it's a good idea to look a bit more at the behavioral structure.

- To further **identify the computational structure** apply clustering to a representative region of your code and load the configuration file `cfgs/clustering/cluster_id.cfg`. This configuration pops up a timeline that colors the different regions detected. Load the gnuplot generated by the clustering module to correlate the regions with their distribution on the scatter plot. Both comparison between clusters and within a cluster are interesting. Use the Paraver timeline to check if more than one cluster are executed at the same time by different processes. Correlate the potential unbalance detected on the previous views (duration, instructions and IPC) with the cluster(s) distribution. The metrics can also be correlated within the Paraver 2d table using for instance the configuration file `cfgs/clustering/2d_usefulduration_vs_cluster.cfg`.

Next steps would be driven by the results of the basic previous analysis. Select the metrics you consider interesting to look at.

- **If the global load balance is poor** the previous views already give you hints on the reason of the unbalance (instructions, IPC or a combination of both). If the tracefile was obtained with callers, load the configuration file

cfgs/general/callers.cfg. This configuration pops up two timelines with the caller function and the caller line of the MPI call previous to the region start. If the values are showing as unknown, try a deeper level on the call stack increasing in 1 the `Caller Level` tag of the window. If none of the levels are available the tracefile may have not been properly merged or the binary was not compiled with -g.

- **If the performance of some of the main computing regions is low** one alternative is to look at the hardware counters available on the tracefile that can be analyzed using the configuration files included on the Paraver distribution (`$PARAVER_HOME/cfgs/counters_PAPI`). For a detailed analysis a new tracefile can be generate with Extrae activating the sampling. In combination with the clustering, the folding module would obtain very detailed evolution of the performance hardware metrics. The folded data can be loaded in Paraver or directly with gnuplot. This analysis would provide the internal performance of the computing regions for the different hardware counters available on the tracefile.

All the previous steps have been done using Paraver analyzer and the modules implementing performance analytics (clustering, folding). The second part of this introductory analysis methodology uses the Dimemas simulator to evaluate different scenarios. The Dimemas predictions should have been taken as trends of the application behavior under different conditions.

- To **analyze the communication efficiency** we can use Dimemas with the configuration of an ideal network (no latency, infinite bandwidth). With this simulation we can separate the communication time that was caused by the data transfer from the communication time caused by the synchronization. To do so, load the simulated trace and apply the mpi profile configuration file (cfgs/mpi/mpi_stats.cfg). The *Maximum* global statistic at the bottom of the table corresponds to the synchronization efficiency. The transfer efficiency can be computed dividing the communication efficiency by the synchronization efficiency. If the synchronization efficiency is close to the communication efficiency the limitation is not on the application but on the network capacity. Otherwise, the high time in MPI is not due to the need to transfer data between processes but to the chain of dependencies and serializations imposed by the code.
- To **analyze the sensitivity to Latency and/or Bandwidth**, if we have found that transfer is limiting our performance, we can use Dimemas to check if the code is sensible to bandwidth, latency or both. The first step would be to check that the nominal simulation provide similar results than the instrumented run. Small differences are acceptable and do not try to match each MPI call separately but that the global communication time both in collectives and point to point are similar enough. Then maintaining the

nominal configuration, modify only latency or bandwidth either with an improvement or a deterioration. The extreme scenario of a nominal configuration with unlimited bandwidth would tell us if the application is limited by latency and we should consider grouping some communication or reducing them. Load both simulations with Paraver and compare the time in the different communication phases.

- Even if **the transfer isn't a limiting factor** may be interesting to look at the nominal and ideal network simulations speeding up the processor. This simulation would give some hints on the application scalability. Making 10 times faster the processor may be considered an optimistic prediction of multiplying by 10 the number of tasks as we don't increase neither the number nor the size of the communications. Load both traces with Paraver and compare the time in the different communication phases.
- To **evaluate the benefits of accelerating only some of the computing regions** that can be achieved with OpenMP or porting some kernels to accelerators, we can use the clustered tracefile that allows to select which regions are affected by the improvement. Try speeding up the main clusters and not the small computations within the communication phases that typically wouldn't be accelerated. This scenario allows you to measure the Amdahl's Law effect on your code. Load both traces with Paraver and compare the time distribution.
- To **evaluate the benefits of balancing some of the computing regions** that you detected unbalanced during the analysis initial steps, we can use the clustered tracefile to select the region and fix its duration to the average burst time that can be computed with Paraver using the configuration file `cfgs/clustering/2dp_clusters.cfg` and looking at the *Average* row in the global statistics at the bottom of the table. Dimemas configuration requires to express the time in seconds. Load both traces with Paraver and compare the effect on the communication phase after the selected region.

Some of the analysis steps results may raise new questions. If balancing a region doesn't have the desired impact on the communication phase after it, it may indicate that the problem of the communication phase isn't the initial unbalance but the serializations within the communication phase. The proposed methodology has some fixed initial steps for the analysis covered in this tutorial. After these steps the exploration would depend on the previous results and some examples have been provided. Keep your investigator attitude and enjoy the analysis!