Introduction to the Analysis with Paraver

Material

- Copy../tools/Paraver_intro.tar to your directory and extract its content. What you have:
  - A couple of source files a and makefile to built an OpenMP application.
  - trace_gen.sh: Script to launch the instrumentation run. Reference just in case you want to obtain a new trace.
  - An already generated trace: test_trace_static_hwc_8.prv and test_trace_static_hwc_8.pcf for the 8 threads run of the redBlackSOR code. Other trace for a 4 processors run.

Navigation and Basic concepts

Let as go though some basic navigation and analysis functionalities.

- Launch paraver (./launch_paraver.sh)

- Load trace: Tracefiles->Load trace: Select file trace_static_hwc_8.prv
- An initial timeline view will show up. Do some initial navigation:
  - Properties: Right Button (anywhere inside the window)->**Show Properties**: displays a table indicating the meaning for each color in this specific window.
  - Zoom: Right button (anywhere inside the window) ->**zoom**, then select left limit with the Left Button and the right limit with the Middle Button.
  - Undo Zoom and Redo Zoom commands are available on the Right Button menu.
  - The **Zoom XY** option will let you select a subset of threads. This is useful when analyzing runs with many processes and you are willing to concentrate on a few of them. After selecting the option you have to identify a rectangular area (clicking on top left and bottom right corners).
  - Color: Locate the color button in lower menu bar of the display window. If the bar does not appear click on the small button at the lower left corner of the window. Switch color button and redraw. You will see a piecewise constant function of time for each thread. This exposes THE KEY CONCEPT in Paraver. The tool generates based on the trace, a piecewise constant function of time that may then be displayed as such or translated through a color encoding mechanism. Coloring is nice for presentation purposes, to convey global qualitative information to the analyst, but the important thing is that Paraver works with functions of time. We will come back to this later.
  - Color translation table: in the "Global Controller" window (typically appears in the lower rightmost corner) click on the "v" symbol. On the "Visualizer Module" window that appears click on the colors button. You will see the value to color translation table. You may change the RGB for any value of interest (don't forget to click on the "apply" button. You may have to redraw after the apply)
  - Categorical or continuous color encoding. If the function of time to be displayed has only a limited categorical set of possible values, a translation table as the one you have seen is adequate to help you differentiate between them. This is the case for the "state as is" view you are looking at, but also for other view displaying the identifier of the user function in execution at a given point in time, MPI calls,…. For views displaying a continuous valued metric such as number of instructions executed in an interval, the IPC, the bandwidth achieved by MPI calls, etc…. a gradient translation mechanism is more adequate. You can switch to gradient color translation mode by Right Button→Color type→Gradient Color or Not Null Gradient Color.

- To measure time between any two points in the trace
  - Right Button→Timing, then select two points in the window with the Left Button. The time between the two selected points of the trace is displayed in a small window. You can measure time between two display windows.
- Click: Left button click on any point in the window. It will list in textual form the actual value of the function of time displayed by that widow. It will also list the events in the trace around the point you clicked. The listing can be in symbolic or numeric form (change "Text Mode" button in lower panel to switch between forms. You can activate or deactivate symbolic listing as well as listing of event or communications with the button in the left lower part of the window.

- Configuration files: Paraver has an extremely flexible mechanism to generate the functions of time to be displayed out of the records in the tracefile. Extreme flexibility usually comes with some complexity in setting it up. This set up on how to generate a view from the trace records can be stored in a configuration file, providing a useful mechanism for novice users but also to enable the easy reuse of metrics that may be considered of interest by an expert analyst. At any point in an analysis you can right Button->Save As on a timeline or table window and save its setup into a configuration file (give the file name including the .cfg extension) you may later reload.

- Some configurations are provided with the distribution to easily display some interesting views ($PARAVER_HOME/etc/cfgs). In this way, a user can navigate and analyze a fairly large set of views/metrics without having to know about the internals of Paraver and how it generates the views. Use Configuration->Load Windows to select some of these views:

  - **General/views/user_functions.cfg**: creates a display window with the timeline of which user function is being executed at each point in time. In an OpenMP application, it is typical that the only thread executing user functions is the main thread.
    - Flags: Button in lower menu bar of the window. Switch button and redraw to see the flags indicating the events (entry and exit of the routine) from which the function identifiers timeline is computed.
      - Zoom till you and identify a few user function invocations.
  - Load configuration file **useful_instructions.cfg** from the current directory (selector on the top right corner of "Load Windows" set to SELECT DIRECTORY). This configuration file shows a timeline with the number of instructions executed in each interval of useful computation. The function is valued to 0 in the regions where processes are idle or in the OpenMP run time in order to let us focus on the actual useful computation parts.
    - The function of time is represented as a Not Null Gradient Color. Use "show properties" to see the actual scale.
    - Display as a function of time: click in the Color button towards the lower central part of the window and then the Redraw button.
    - Fit Y scale: The timeline has a limited space to represent a function of time whose dynamic range may be large. A linear scale is used in the Y direction. If the scale is such that the whole dynamic range does not fit in the allocated space, then the function display is truncated. To automatically fit the scale to the whole dynamic range of the function click Right Button->Scale->**Fit Y scale**. To manually control the scale use the fields in the Visualizer Module window that appears when clicking in Global Controller->V.
    - Synchronize windows: Go back to the "user functions" view. Zoom till you identify an area with a few user function invocations. On this window click Right Button->**Copy**, then on the "instructions" window click Right Button->**Paste X scale**. Both windows now represent different views of the same part of the trace. If you put one above the other there is a one to

one correspondence between points in vertical. To take full advantage of this feature it is recommended not to change the size of windows. Other paste choices let you have the same window size and representing the same set of processes and

- Load configuration file OpenMP/**views/parallel_functions.cfg**. The colors in the timeline identify the outlined routine being executed by each thread (use Show Properties to check). Zooming to see a couple of invocations to the red parallel function (click on the "Flag" button and Redraw to see the flags indicating the entry/exit of a parallel function and use it for reference in the zooming process). Copy the scale to the "state as is" window and you should identify the structure of two worksharing do loops inside the parallel region, matching the source code strcuture.

- To get a profile of the execution:
  - Load configuration file **OpenMP/analysis/omp_profile.cfg**. A table pops up with one row per thread and one column per parallel_function as seen before. Each entry in the table tells the percentage of the total time the corresponding thread has been inside the specific parallel function.
  - To see a different statistic change the **Statistic selector** in the upper part of the 2D window. Interesting options at this time may be:
    - **%time**: to get the accumulated percentage of time each process has spent in each parallel function.
    - **#burst**: To count the number of invocations to each parallel function.
    - **Average burst time**: to compute the average duration of the parallel function.
  - All the above statistics are computed based on a single timeline window, which we call "**control window** " and which can be popped up by clicking on the icon just right of the selector of control window. The values of the control window determine to which column is a given statistic accumulated/accounted.
  - The statistic inside a cell can actually be performed on a different window, that we call "**data window**". For example, if you select the "**average value**" statistic and you select as data window the useful instruction window we loaded before, the entry will report (you may need to click the repeat button) the average number of instructions executed by each computation burst within the specific parallel function. If you change the statistic to "**average burst time !=0** " you will get the average duration of those computation bursts. You may check whether there is a direct relation between instructions and time.
  - To apply the analysis to a subset of the trace, zoom on any of the timelines to the time region you are interested on. Right button→ copy on this window and Right button→ paste scale on the table. The analysis will be repeated just for the selected time interval.

- To get a histogram of continuous valued metrics:
  - Load configuration file **2dh_useful_IPC.cfg**. A table pops up whit one row per thread and one column per bin of IPC values. In this case, the columns go from IPC 0 to 1.5 in bins of 0.1. The statistic in each cell is the number of instances such thread has had a specific IPC. Moving the cursor on top of an entry will display textual information at the lower left corner of the window. The coloring scheme is a gradient from light green for a low valued statistic to a dark blue for a high value. If the value is exactly 0, the cell stays grey.
  - As before, it is possible to change the statistic so that the actual value represented is total time, average duration of the bursts or average value of other metric (ie. IPC). This mechanism is very useful to analyze correlations between metrics.
  - Click on the magnifying glass icon on the top right corner and you will se the actual numeric information in a tabular form.

Further details and methodological hints

- In a loaded system, time sharing of the processors may result in significant perturbations of the traces as some thread of an application may be preempted and the delay it suffers will propagate to all other threads through the synchronizations. A good practice is to identify parts in the trace where the applications threads where preempted. This can be done with configuration file **cycles_per_us.cfg**. The ratio should match the processor frequency. Areas with a smaller ratio reflect that the process was preempted (the cycle counter does not increase while actual elapsed time always goes by).

- The tracing package stores the events in a buffer in memory. When the buffer fill and at the end of the run, the buffer is dump to disk. This may take a significant time compared to the fine grain activity typical in MPI/OpenMP programs. The result may be a significant perturbation of the trace at some points in time. Use ${OMPITRACE_HOME}/etc/cfgs/General/sanity_checks/flushing.cfg to identify those regions and discard them from your analysis.

- The default instrumentation only generates events on the entry and exit of user routines that have some OpenMP call inside. If you are interested in identifying entry and exit of other routines, you can use the -include <functions> option to the ompitrace command. <functions> is a file with the names of the functions (one per line) you want to instrument. This feature is only available on platforms where the instrumentation is injected into the code by means of Dyninst. This is the case for the Altix platforms.

- For further detail, in the case you may be interested in emitting your own events to the trace you can instrument your source code with calls to the API described in

  ${OMPITRACE_HOME}/include/ompitrace.h

  and link with

  OMPITRACE_LIB = -L$(OMPITRACE_HOME)/lib -lompitrace

  The most relevant call would be OMPItrace_eventandcounters which injects into the trace an event with the type and value that you specify plus the hardware counter events. There are a couple of encoding practices that we recommend

  - If you use this API to identify regions of code use a single type (integer starting at 1000) for all regions and a different value (integer starting at 1) to represent the specific region you are entering. We also recommend that you emit an event with the same type and value 0 when you exit a region. This encoding will support the later analysis with Paraver of nested regions.

  - If you want to emit to the trace the values of variables in your program use a different type for each variable and emit as value of the event the actual value of the variable. In this way you will be able to later display the evolution of the variable value with time.

  - For documentation on the tracing package and instrumentation API

    ```
    export MANPATH=/apps/CEPBATOOLS/OMPItrace/man:${MANPATH}
    man ompitrace
    man ompitrace_event
    ```

Question

- Obtain 4 traces for 4 and 8 processors with static and guided scheduling and compare them. Send a report to jesus.labarta@bsc.es