To measure performances (CPU) under Linux

Summary:

There exist tools making it possible to trace times CPU used (*profiling*).

On Linux, the tool is used `gprof`. The use of this tool forces to compile all the sources with the option "`- pg`". The overcost of the instrumentation is negligible. The result of *profiling* is a textual file which should be interpreted.

To simplify interpretation, one proposes at the end of the document a tool to trace a graph starting from the textual file produced by "profiled" calculation.

Contents

1 Gprof

1.1 Construction of a version with gprof

To function, gprof requires that the sources were compiled with the option `- pg`.

To preserve construction by default of Code_Aster (without the options of gprof), one uses one alias `waf_prof` (it is enough to create a link symbolic system for that).

Then, it is enough to include the configuration `gprof` after the configuration of the machine to add the options necessary.

On the machines Gauges and the official waiters, the option `--use-config` is optional in normal weather (automatically given when the environment `env_unstable.sh` is charged). Here, we are obliged to name it explicitly. Thus replace XXX by `calibre7` or `aster5` or...

```
    Cd $HOME/dev/codeaster/src
    ln - S waf_variant waf_prof
            ./waf_prof   configures   --use-config=XXX,   gprof   --prefix=.
/installation/professor
    ./waf_prof installation
```

Notice

Mode optimized is a priori preferable to measure "the true" performances of the code. On the other hand, mode "debug" is necessary if one wants to know the most consuming lines (use then `./waf_prof intall_debug`).
One has unfortunately observed an unexplainable problem in mode "debug" : the result of `profiling` indicated links of incoming call routines which did not exist! One can however hope that this anomaly entirely does not invalidate the rest of measurement.

1.2 Overcost of the instrumentation

As example, on the test `ssnv506c`, one has got the following total results:

* in mode `nodebug` without instrumentation	138s
* in mode `nodebug` with instrumentation	139s
* in mode `debug` without instrumentation	218s
* in mode `debug` with instrumentation	228s

It is noted that the instrumentation has a negligible cost CPU.

1.3 Execution DE the achievable one instrumented with waf

To start a test with `waf`, one can add one line this type in the file `.export` to recover the file of `gprof` and to copy it, for example, in `/tmp` :

```
    F name /tmp/gmon.out R 0
```

It is then enough to launch: `./waf_prof test - N ssnv506c`

See the following paragraph for the use of this file `gmon.out`.

1.4 Execution DE the achievable one instrumented with hasstk

One can also to carry out the study which one wants “to profile” with the achievable one instrumented in astk.
For that, the version should be declared instrumented in the list of the versions usable locally.
In the file $HOME/.astkrc/prefs, to add the line:

```
towards: DEV_PROF: $HOME/dev/codeaster/installation/professor
```

In astk, it is then enough to select the named version DEV_PROF.

The execution of the study produces a file (called gmon.out) in the temporary repertoire of execution.

To preserve the invaluable file gmon.out, one adds a field in the profile astk of type "name" and whose value is "… /gmon.out". A file of name then will be recovered gmon.out in the shown way.

1.5 Analysis of the results

The produced file, without overcost, by the achievable one instrumented is not however not usable directly. It is necessary to carry out gprof to make it readable (note that it is necessary to indicate the name of achievable instrumented) :

```
gprof $HOME/dev/codeaster/installation/professeur/bin/aster gmon.out >
listing
```

The analysis thus consists in stripping the file listing product.

Caution

IL is not necessary to be discouraged. For an execution of 30s, one already saw, the order gprof to consume more than 5 minutes of CPU. The time of gprof does not depend too much (a priori) on the shaped execution time.

The interpretation of the file obtained (listing) is described below. An excellent document describing all the process of profiling is that written by Jay Fenlason and Richard Stallman: "Gnu gprof The GNU to profile". One easily finds it on the Web.

Notice

Even if one recompile all the sources of Aster, the “depth” of the analysis of the performances stop with the libraries which one uses with the edition of the links and who were not compiled with "- pg" . It is for example the case of the routines blas. The time spent in these libraries cannot be attached to the routines DE Code_Aster which calls them. This defect can be important, for example, if one wants to measure the performances of the solveurs MUMPS or MULT_FRONT because most of spent time is in routines blas.

1.6 To strip the results of the profiling

By default, the file is heavy. It is possible to limit the posting of information while playing with the options of gprof. “Times systems” are indicated in the form of many instructions used.
One will detail a little, while starting with the end of the file:

```
Index by function name
 [401] PyArg_Parse            [591] cftabl_   [1000] proc_at_0x1213acb50
 [212] PyArg_ParseTuple       [84] cftyli_    [660]  proc_at_0x1213ad470
[1137] PyArg_ParseTupleAnd    [310] cgmacy_   [453]  proc_at_0x1213ad560
[1605] PyBuffer_FromObject    [79] charme_    [680]  proc_at_0x1213aeac0
[1256] PyCFunction_Fini       [476] chlici_   [1221] proc_at_0x1213aedc0
 [531] PyCFunction_New        [190] chloet_   [217]  proc_at_0x1213b18e0
[1549] PyCObject_AsVoidPtr    [226] chmano_   [629] proc_at_0x1213b1e00Y
```

Each function called during the execution is located by a number between hook.
Just with the top:

```
granularity: instructions; units: inst' S; total: 201924201580.70 inst' S

 <A>  <B>            <C>           <D>  <E>         <F>         <G>
49.6  100384307222  100384307222  161  623505013   623596299 tldlr8_ [16]
31.0   63144941823  62760634601   506  124032874   124101882 rldlr8_ [17]
```

This table summarizes the most frequent calls.

- COLUMN <A> : percentage amongst instructions carried out by this function by report on the whole of the execution.
- COLUMN <B> : many instructions cumulated by this function and those which precede.
- COLUMN <C> : many instructions for this function.
- COLUMN <D> : many calls have this function
- COLUMN <E> : relationship between the column <B> and the column <D> (many instructions means by call of the function)
- COLUMN <F> : median number of instructions per call of the function and of its descendants.
- COLUMN <G> : name of the function and its reference number (between hooks).

In this example, the function `tldlr8` took 49.4% of the total of calculation while being called 161 times.

Lastly, at the beginning of the file, we have the tree of complete call. It will be sorted by order of call (one starts with the hand and one goes down) or by a function (see the options of `gprof`).

Let us take the example of `tldlr8` :

```
<A>     <B>  <C>              <D>          <E>          <F>

             100263313681.76  14679301.29  161/161      tldlgg_ [15]
[16]    49.7 100263313681.76  14679301.29  161          tldlr8_ [16]
             3129121.03       6207534.02   4485/30537   __upcUpcall [352]
             35974.59         2749927.50   522/195235   jelibe_ [65]
             192341.36        1770419.18   1005/775659  jeveuo_ [56]
             47302.73         140745.02    161/202579   jedema_ [102]
             18938.92         126525.05    322/63148    jeexin_ [196]
             27722.26         85430.33     94/49118     jeecra_ [154]
             17033.41         67779.29     94/13206     jecreo_ [257]
             45068.75         84.88        1044/1075446 jexnum_ [163]
             13618.68         2023.63      161/202581   jemarq_ [205]
             1710.66          0.00         161/3481     infniv_ [853]
```

One locates the instruction of the tree of call by the number between hooks on the left. Here, the number [16] indicates the function `tldlr8_` (as indicated at the end of the file for example). It is the function-reference (the node of the tree). The lines with the top are the appealing ones of this function (they are the function-parents), those in lower part are the functions called (they are the function-children). Each function has two principal digits: the number of instructions carried out in itself (“final” instruction of `FORTRAN`) and the number of instructions carried out in the function-children.

```
Function-parent
Function-parent
…
Function-reference
Function-child
Function-child
…
```

For the function-reference:

- COLUMN <A> : number of location of the function-reference.
- COLUMN <B> : figure 49.7 is the percentage amongst instructions carried out by this function-reference compared to the total of the execution (idem table precedent)
- COLUMN <C> : many instructions for the function-reference itself.
- COLUMN <D> : many instructions for the function-children of the function-reference.
- COLUMN <E> : many times or the function was called
- COLUMN <F> : name of the function-reference

For the function-parents and the function-children:

- COLUMN <A> : vacuum
- COLUMN <B> : vacuum
- COLUMN <C> : many instructions for the function itself.
- COLUMN <D> : many instructions for the descendants of the function
- COLUMN <E> : give two digits a/b whose direction varies according to the type of function (relative or child compared to the function reference):
  - For the function-parents (above the function reference) a/b: <a> is the number of times where the function-reference was called by this function-parent compared to the <b> full number of calls of the function-reference.
  - For the function-children (below the function reference) a/b: <a> is the number of times where the function-child was called by the function-reference compared to the <b> full number of calls of the function-child.
- COLUMN <F> : name of the function

Note:

If the number of instructions for the descendants of a function is worth zero, it is that the function considered does not call any other of it. One is "with the end" of the tree, it has there only basic calls FORTRAN in the function (it is the case of `infniv` for example).
For a given function-reference, if one makes the sum of the <a> in the column <E> of the functions parents, one obtains the total number of calls of the function reference.
For a given function-reference, if one makes the sum of the columns <C> and <D> of his function-children, one obtains the figure of the column <D> of the function-reference.

1.7 Analysis of the example

In the example presented, the function tldlr8 is expensive since with it-only, it represents about half amongst instructions total of the execution. It is also seen that these are the own instructions which take time and not the call to his/her function-children (the relationship between the two reached 1000). Like only the function `tldlgg` call `tldlr8`, it is necessary to look at the tree of call for this function. It is seen whereas it is the algorithm of contact/friction (`fropgd`) who is more glouton (the 2/3 of the calls to `tldlgg` are made by the algorithm of contact).

1.8 Generation of a graph with *gprof2dot*

1.8.1 Presentation

To facilitate the interpretation of one *profiling*, one describes in this section a small utility Python (*gprof2dot*) who transforms the textual file produced by `gprof` in a graph simpler to read.

The produced graph is that of the routines traversed with carryforward of the columns <B> and <C> (in % of the full number of instructions), amongst calls of the routine. In addition the cells of the graph are colored (blue towards the red) to identify the critical paths very quickly.
One will find more information on the Web page of the developer of *gprof2dot* : http://code.google.com/p/jrfonseca/wiki/Gprof2Dot.

1.8.2 Use

Once the file of*output* product by `gprof` recovered, this one being called `listing`, one will carry out the following order in a terminal:

```
cat listing | gprof2dot.py | dowry - Tpng - O graphe.png
```

An example of the type D 'produced image is given in the following figure.

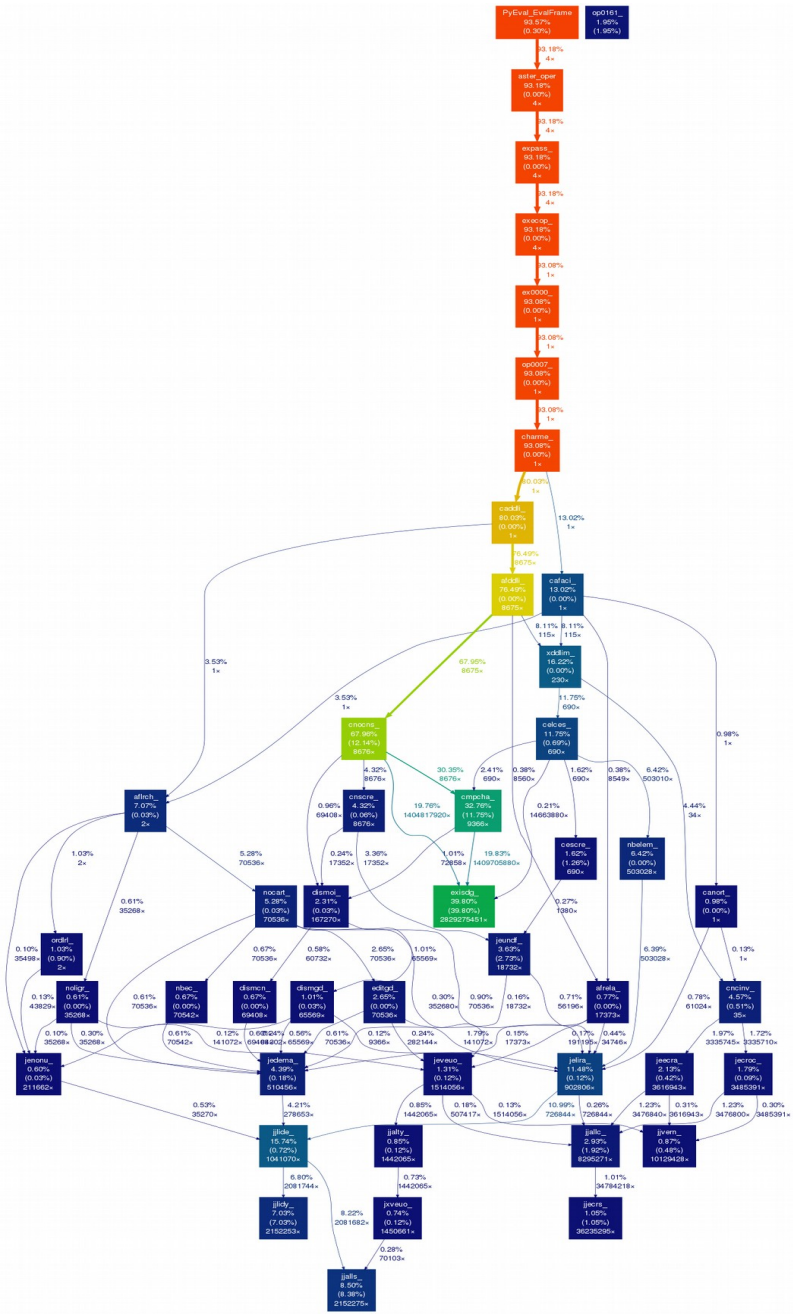

Figure 1 : graph of call generated with *gprof2dot*