Controlling a Solver Execution: the *runsolver* Tool

Olivier ROUSSEL

CRIL - CNRS UMR 8188

roussel@cril.univ-artois.fr
http://www.cril.univ-artois.fr/∼roussel/runsolver/

Outline

- The goal
- The main problem
- First attempts (2005)
- A better solution (2006 – today)
- Impact on the solver

The goal

When one is experimenting with a solver, it is useful to:

- limit the resources consumed by the solver
  - time (obviously!)
  - memory (to be detailed)
  - cores allocated to the solver
  - size of the output (solvers can be very, very verbose!)
  - ...
- collect some information on what happened during the run
- know at what time a solver printed a line
- interrupt the solver in a nice way, so that it's still able to provide useful information (e.g. an approximate answer)
- ...
- and all this should come for free!

runsolver is designed to fulfill these requirements, *except the last one*

Different times

- WC: wall clock time = real time that elapses between the start and the end of a computing task.
- CPU: CPU time = time during which instructions of the program are executed by a processing unit.

Some remarks:

- On a host with 1 processing unit and no interrupts, CPU time=WC time
- On a host with 1 processing unit and a time sharing system, WC time $\geq$ CPU time
- On a host with n processing units, and for a perfect parallel program, CPU time=$n\times$ WC time
- WC time= user's perception of the program efficiency
- CPU time= actual computational effort

Different memories

RSS (Resident Size) amount of RAM occupied by the program

VSIZE (Virtual Size) amount of memory (RAM or swap space) occupied by the program

Some remarks:

- RSS is under the operating system control and can change arbitrarily during the solver execution. Not a candidate for enforcing a limit.
- VSIZE is under the program control (sum of program/library code + static data + dynamic memory allocations)
- VSIZE is the parameter to limit to prevent a solver from swapping

Swapping to disk

Until we have solvers which are able to handle swap space in a clever way, it's a good idea to prevent the solver from swapping:

- magnetic disks are approximately 6 order of magnitude slower than main memory: the solver performances would be dominated by the disk performances
- too frequent swapping might kill the hardware
- As an example, due to a configuration error, one solver was actually allowed to swap in the competition. The host became unresponsive (no way to login) and kept swapping for **27 hours**. Neither *runsolver*, nor torque (the batch system) were able to kill the job (were not even executed)!

This policy should be revised once we have swap space on SSD devices.

Straightforward approach

```
(limit cputime 1200 ;
   limit vmemoryuse 1G ;
     time solver instance.cnf)
```

- Easy approach for enforcing limits (except on WC time)
- Doesn’t satisfy all our requirements (collecting information about the running solver for example)
- May print that the solver used 1 second CPU time and a total of 1200 seconds WC time!!
- May allow a solver with multiple processes to use much more than 1200 s CPU time!!

The source of the problem

- The CPU time of a process only includes the “resources used by those of its children that *have terminated and have been waited for*” (man 2 times).
- *Consequence 1:* if a parent process doesn’t call wait(2), the resources used by the child will be forgotten.
- *Consequence 2:* ulimit/limit(1) cannot enforce reliable limits for multi-process solvers because the resources used by the child are only reported when it terminates (too late to enforce a limit!).

First attempt (2005)

The idea:

- intercept memory allocation requests, in order to be able to gracefully terminate the solver when it requests too much memory
- intercept process creation calls to maintain a list of the solver processes

Additional requirements:

- must also work for static binaries
- must not require any privilege

Solution:

- run the solver in trace mode to intercept system calls

Works, but severely degrades the solver performances!

Current solution (2006–today)

Idea:

- periodically scan the list of processes to identify new children of the solver (once per second)
- periodically scan the list of the solver processes to update their CPU usage (cheaper, ten times per second)

Advantages/Disavantages:

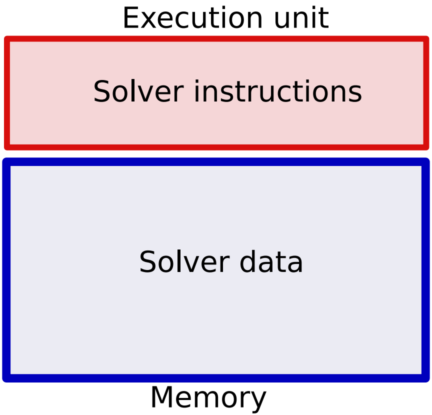
- works well
- low (but non nul) impact on the solver performances
- used in the PB/SAT competitions since 2006 as well as other competitions (ASP, MISC,...).
- cannot terminate the solver gracefully if it allocates too much memory in one call

Additional features

- timestamp each line printed by the solver (very useful)
- periodically save a list of the solver processes with the corresponding data from /proc (very useful for post analysis)
- limit the size of the solver output
- allocate a subset of the available cores to the solver
- and a few other options

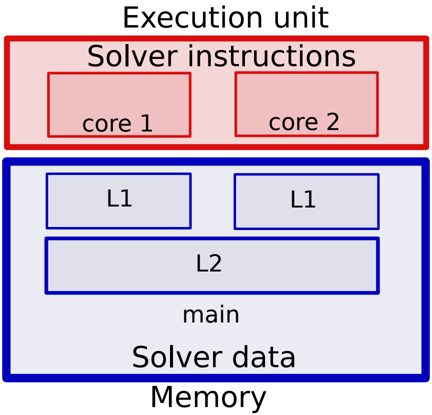
Impact on the solver (1)

In a perfect world (~30 years ago)



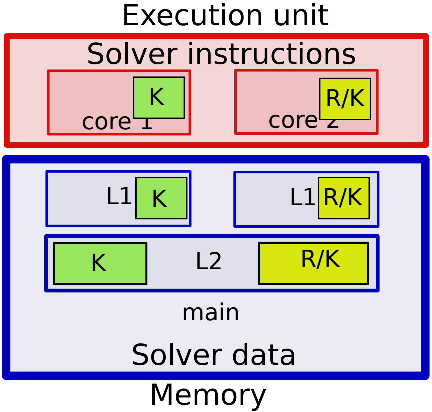
Impact on the solver (2)

In a perfect modern world



Impact on the solver (3)

In a real world, under the control of runsolver



runsolver, the kernel and the other processes running on the host are **stealing CPU power and cache memory to the solver!**

Impact on the solver (4)

- Any software tool will have an impact on the solver!
- *runsolver* attaches itself to the last core to limit its impact
- The competitions are a nice test-bed for runsolver:
  - the CPU time used by *runsolver* is low ($\sim$30 seconds CPU time for a run of 5000 s, less than 1 %)
  - for sequential solvers, generally CPU time is equal to WC time (almost)
  - for parallel solvers, evaluating the impact of *runsolver* is difficult because of the non-determinism of the solver and the sequential parts of the solvers (CPU/WC $<$ number of CPU). Some parallel solvers in the competition achieved a ratio CPU/WC of 7.98 on a host with 8 cores, so the impact of *runsolver* is probably around 1%.

Conclusion

- *runsolver* offers a number of interesting features to control a solver
- It benefits from the experience gathered during various competitions
- *runsolver* is not perfect, but is just a pragmatic answer to the problem
- There is necessarily an interaction between the solver and *runsolver* (measuring modifies the experiment!) but the perturbation is limited (depends on the hardware and the solver).
- The balance benefits/disadvantages is positive (IMHO)
- Available under a GPL license at http://www.cril.univ-artois.fr/~roussel/runsolver
- The latest version used during this year competitions will be available soon.