

Automatic performance monitoring tools

Nicolas Desprès

LRDE seminar, November 16, 2005, revision 721

<http://www.lrde.epita.fr/>



Copying this document

Copyright © 2005 LRDE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being just “Copying this document”, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is provided in the file COPYING.DOC.

Introduction

We want to be able to collect many measures and to check their evolution with the less possible effort.

Table of Contents

Overview	4
Measurement Library	8
Data acquisition	23
Visualization/analysis interface	27

Overview

Context

Olena[3], Vaucanson[5], Transformers[4] get slower:

- Why?
- When?

Objectives

- Prevent next performance regressions.
- Find past regression reasons.

System overview

- Automatic build system[2]: configure, build, test run benchmark
- Measurement library:
 - Provide non intrusive measurement tool.
 - Ensure output format reliability.
- Data acquisition script: populate measurement into a repository.
- Visualization/Analysis application:
 - Store data.
 - Visualization/Analysis features.

Measurement Library

Old approach[1]: presentation

- Two macros:
 - `BEGIN_NAME_TYPE_SCALE`
 - `END_NAME_TYPE_SCALE (score)`
- One header:
 - Generated from the data storage manager.
 - Declares every `BEGIN_xxx/END_xxx` macros.

Old approach: disadvantage

- Macros are statically evaluated \Rightarrow can not dynamically iterate through a range of name/type/scale.
- Header consists in inlining the database content \Rightarrow no scalable.
- Not extensible by the end user.
- One benchmark = one type = one scale.

Old approach: advantage

Ensure benchmark writers do not misspell any previously registered benchmark name/type/scale.

⇒

Ensure database consistency.

New approach(libbranch-cxx): presentation

- Two methods:
 - `Bencher#start(...)`
 - `Bencher#stop(...)`
- One header:
 - Interface of the library.

New approach: disadvantage

- Users must link the library with their benchmark.
- Benchmark writers may misspell benchmark name/input name/output name.
⇒ the database may be inconsistent.

New approach: advantage

- Methods are dynamically evaluated.
- Extensible by means of sub-classing.
- One benchmark = several inputs = several outputs = several input values.

Example: determinize code

```
Ranch::Input::Input nb_states("nb_states");
Ranch::Inputs inputs;
inputs.add(nb_states);
Ranch::Output::Output nb_edges("nb_edges");
Ranch::Output::UTime utime;
Ranch::Outputs outputs;
outputs.add(utime).add(nb_edges);
Ranch::Bencher b("determinize", inputs, outputs);
for (int i = 9; i <= 15; i += 3) {
    automaton_t a = aut_2n(i);
    b.start(i);
    automaton_t ret = determinize(a);
    b.stop(utime(), (double)ret.edges().size());
}
```

Exemple: determinize output

```
- 'determinize':  
  type:  
    inputs:  
      - 'nb_states': ''  
    outputs:  
      - 'utime': 'sec'  
      - 'nb_edges': ''  
    comment: |  
  
  content:  
    -  
      inputs:  
        - 9 # nb_states()  
      benches:
```

```
outputs:
  - 0.05          # utime(sec)
  - 1536          # nb_edges()
-
inputs:
  - 12 # nb_states()
benches:
outputs:
  - 0.51          # utime(sec)
  - 12288         # nb_edges()
-
inputs:
  - 15 # nb_states()
benches:
outputs:
  - 5.87          # utime(sec)
  - 98304         # nb_edges()
```

Measurement reliability (1/3)

Requirements:

- The measurement library must not falsify program performance.
- Measurement tools provided by the library must be non intrusive with any programs.
- The library must accept measure result from other tools.

Measurement reliability (2/3)

Libbranch-cxx:

- Measurement tools are not started/stopped exactly before/after the measured function.
- Time measurement tool setup/teardown may invalid memory measurement.
- Memory measurement tool setup/teardown may invalid time measurement.

Measurement reliability (3/3)

But, we are doing regression/scalability benchmark:

- We do not care about absolute measure.
- overhead delta is:
 - constant,
 - negligible.

Measurement extension

- Inherit from the `Output` class.
- Implement the `start` method.
- Implement the `stop` method.

Output format extension

- Inherit from the `Dumper` class.
- Implement the `begin_bench/end_bench` methods.
- Implement the `begin_score/end_score` methods.

Data acquisition

The populate script

Old approach[1]: Reject and Register

- Reject non previously registered data.
- Interface to register benchmark.
- Registration done by hand.
- Generate language specific registered benchmark list.

New approach: Accept and clean

- Accept data if they syntactically fit with the database format.
- Manually clean unnormalized data to restore database consistency.
- Interface to clean incorrect benchmark information.

Comparison

- Both may deal with erroneous data.
- Benchmark writers can make mistakes with both language interface if they duplicate their code.
- It's cheaper to update the database than to rerun the benchmark.

Visualization/analysis interface

The ranch web application

Regression

Analysis:

- performance evolution over a range of revisions;
- fixed input values;
- fixed output.

⇒ curve based charts visualization.

Scalability

Analysis:

- performance evolution over a range of value for one input;
- fixed revision;
- fixed output;
- fixed other inputs value.

⇒ curve based charts visualization.

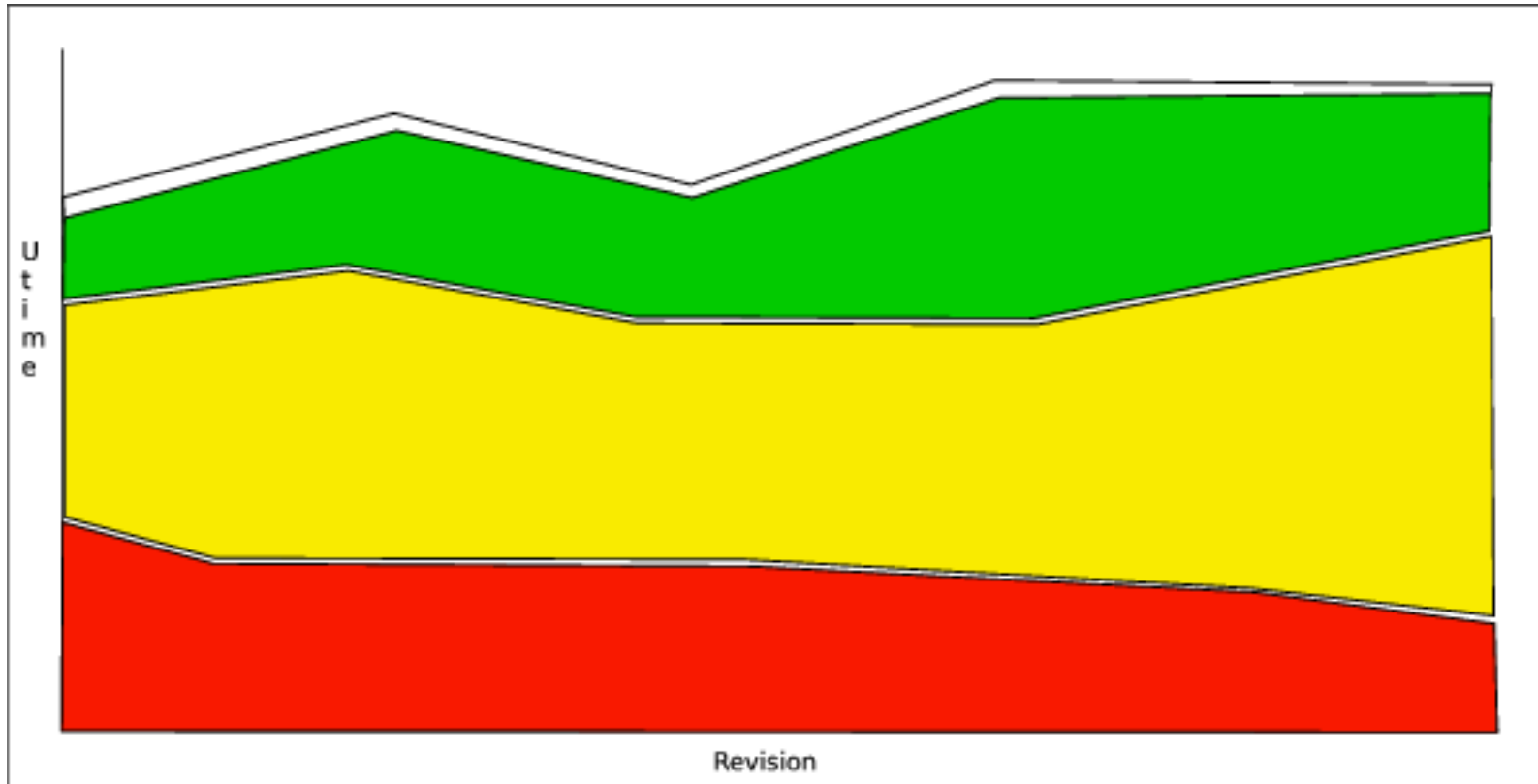
Program comparison

Analysis:

- Regression: other program revision fixed.
- Scalability: all programs input scale evolves.

⇒ curve overlapping based charts visualization.

Nested benchmark



Future work

- Measure memory usage using `mprof(1)` or `malloc_hook(3)`.
- Write documentation.
- Extract more data from past revision.
- Make prettier web interface.
- Finish to manage nested benchmark for Tiger support.

Conclusion

- Ranch has not yet been able to gather enough data in past revision to find key regression point, because it is very hard to recompile a program from any revision.
- If the AutoBuild, Ranch and the benchmarks are still maintained there will have no more silent regression in the future.

References

- [1] Nicolas Desprès. Automatic regression benchmark system. Technical report, LRDE, June 2005.
- [2] LRDE. Autobuild project home page, 2005.
- [3] LRDE. Olena project home page, 2005.
- [4] LRDE. Transformers project home page, 2005.
- [5] LRDE. Vaucanson project home page, 2005.

Questions and remarks