PerfAndPubTools – Tools for Software Performance Analysis and Publishing of Results

Nuno Fachada1, Vitor V. Lopes2, Rui C. Martins3 and Agostinho C. Rosa4

1 Institute for Systems and Robotics, LARSyS, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
nfachada@laseeb.org
2 Universidad de las Fuerzas Armadas-ESPE, Sangolquí, Ecuador
3 Life and Health Sciences Research Institute, School of Health Sciences, University of Minho, Braga, Portugal
4 Corresponding author: Nuno Fachada

PerfAndPubTools consists of a set of MATLAB/Octave functions for the post-processing and analysis of software performance benchmark data and producing associated publication quality materials. More specifically, the functions bundled with PerfAndPubTools allow to:

1. Batch process files containing benchmarking data of computer programs, one file per run.
2. Determine the mean and standard deviation of benchmarking experiments with several runs.
3. Organize the benchmark statistics by program implementation and program setup.
4. Output scalability and speedup data, optionally generating associated figures.
5. Create publication ready benchmark comparison tables in LATEX.

These tools were originally developed to assess the performance of serial and parallel implementations of the PPHPC simulation model [3], as well as for producing some of the associated publication quality materials. However, the tools can be used with any computational benchmark experiment.

Implementation and architecture

Performance analysis in PerfAndPubTools takes place at two levels: implementation and setup. The implementation level is meant to be associated with specific software implementations for performing a given task, for example a particular sorting algorithm or a simulation model realized in a certain programming language. Within the context of each implementation, the software can be executed under different setups. These can be different computational sizes (e.g., vector lengths in a sorting algorithm) or distinct execution parameters (e.g., number of threads used).

PerfAndPubTools is implemented in a layered architecture using a procedural programming approach, as shown in Figure 1. From lowest to highest-level of functionality, the functions represented in this Figure have the following roles:

Figure 1: PerfAndPubTools architecture. Blocks in typewriter font represent functions. Dashed blocks represent directly replaceable functions.
The performance of four sorting algorithms, Bubble sort, Selection sort, Merge sort and Quick sort [10], are used to compare multiple implementations across different setups. The algorithms, specified in C [9], are compared using PerfAndPubTools to compare multiple implementations across different setups.

Implementation specs must have the same number of setups, and corresponding setups should have the same sname. Additionally, plotting with perfstats requires that the computational size, csize, is defined and has the same value for corresponding setups in different implementations.

Algorithm scalability: The perfstats function determines mean times and standard deviations of individual setups for each implementation. If the various setups correspond to different computational work sizes, perfstats can optionally plot a scalability graph. The following instruction performs this task for the experimental setup under discussion:

```matlab
[m, s] = perfstats(3, 'Bubble', bs, ...
    'Selection', ss, 'Merge', ms, ...
    'Quick', qs);
```

The contents of the returned variables, m and s, are as follows:

| m        | 36.0040  | 144.8210 | 325.1730 | 577.8600 |
|----------|---------|----------|----------|----------|
|          | 9.5270  | 38.0500  | 88.5130  | 153.6560 |
|          | 0.0200  | 0.0410   | 0.0600   | 0.0850   |
|          | 0.0100  | 0.0200   | 0.0300   | 0.0510   |
| s        | 0.8873  | 2.9223   | 6.1874   | 6.3846   |
|          | 0.0690  | 0.2829   | 3.6976   | 3.0600   |
|          | 0.0000  | 0.0032   | 0.0000   | 0.0127   |
|          | 0.0000  | 0.0000   | 0.0000   | 0.0032   |

The m variable represents mean times (in seconds), while s holds the respective standard deviations. Rows are associated with implementations (i.e., sorting algorithms), while columns represent setups (i.e., vector sizes). The first parameter of perfstats specifies whether to generate a scalability plot. More specifically, the value 3 orders the function to generate a semi-logarithmic plot,
as show in Figure 2. Negative values indicate that the figure should also display error bars representing the standard deviation in the measured computational sizes. No plot will be generated if zero is passed as the first argument.

Obtaining the speedup: The speedup function determines speedups against one or more reference implementations, across a number of setups. Its usage is similar to that of perfstats, requiring the identification of the implementation specs to compare:

```matlab
[s_avg, s_max, s_min] = speedup(-2, 1, ...
    'Bubble', bs, 'Selection', ss, ...
    'Merge', ms, 'Quick', qs);
```

The first parameter concerns the optional bar plot the function is able to generate. An absolute value of 2 states that a bar plot with a logarithmic scale should be generated, as shown in Figure 3. Since this value is negative, error bars representing the maximum and minimum speedups are drawn on top of the average speedup bars. The second parameter defines the reference implementation(s) to which the speedups are to be determined against. Passing 1 identifies the first implementation, Bubble sort, as the reference. The speedup function returns cell arrays containing the average, maximum and minimum speedup matrices for each reference implementation. In this case, one reference was defined, and thus only the first item in the returned cells is available:

```matlab
s_avg{1} = 1.0e+04 *
    0.0001    0.0001    0.0001    0.0001
    0.0004    0.0004    0.0004    0.0004
    0.1800    0.3532    0.5420    0.6798
    0.3600    0.7241    1.0839    1.1331
```

In a similar fashion to the mean and standard deviation matrices returned by perfstats, rows of speedup matrices are associated with implementations (i.e., sorting algorithms), while columns represent setups (i.e., vector sizes). Note that, in this case, the first row represents the average speedup of Bubble sort against itself, and, as such, the values are all ones.

Generating tables: PerfAndPubTools can generate plain text or publication quality tables summarizing the performed computational benchmarks. The process is divided in two steps using the times_table and times_table_f functions, respectively. The former determines and returns a matrix containing partial or complete information to generate a table, while the latter effectively generates tables. This division is useful because times_table_f can accept more than one matrix returned by times_table, allowing the generation of more complex tables.
The `times_table` function, like `perfstats` and `speedup`, requires the identification of the implementation specs to compare, as shown in the following command:

```matlab
tdata = times_table(1, ...  
    'Bubble', bs, 'Selection', ss, ...  
    'Merge', ms, 'Quick', qs);
```

The first argument designates the references implementation or implementations, in a similar fashion to the second parameter of `speedup`. The return value, `tdata`, can be passed to `times_table_f` in order to generate a table:

```matlab
times_table_f(0, 'vs Bubble', tdata)
```

The first argument, 0, instructs the function to generate a plain text table, as shown in Figure 4. Setting this value to 1 would generate a LATEX table, as shown in Figure 5. Note that LATEX tables require the `siunitx`, `multirow` and `booktabs` packages.

**Complete example**: The complete example is available in the user manual bundled with the software. It contains the necessary steps required to reproduce these results, also showing how the return values of `perfstats` and `speedup` can be used to generate custom publication quality plots. The user manual also details an additional example concerning the performance of serial and parallel implementations of the PPHPC simulation model [3], namely different ways of contextualizing the concept of computational size, and the generation of more complex tables.

**Quality control**
The available functions are covered by unit tests in order to ensure their correct behavior. The MOxUnit framework [11] is required for running the unit tests. Additionally, all the examples available in the user manual (bundled with the software) have been tested in both MATLAB and Octave.

**Availability**

**Operating system**
Any system capable of running MATLAB R2013a or GNU Octave 3.8.1, or higher.

**Programming language**
MATLAB R2013a or GNU Octave 3.8.1, or higher.

**Dependencies**
There are no additional dependencies for the package tools. However, unit tests depend on the MOxUnit unit test framework for MATLAB and GNU Octave.
List of contributors
The software was created by Nuno Fachada.

Software location
Archive
Name: PerfAndPubTools
Persistent identifier: https://zenodo.org/record/50190
Licence: MIT License

(3) Reuse potential
These utilities can be used for analyzing any computational experiment. As described in ‘Implementation and architecture’, other benchmark data formats can be specified by implementing a custom function to replace get_gtime and setting its handle in the gather_times function. Results from perfstats and speedup functions can be used to generate other types of figures. The same is true for times_table, the results of which can be integrated in table layouts other than the one provided by times_table_f.

Competing Interests
The authors declare that they have no competing interests.

Acknowledgements
This software is enhanced by the matlab2tikz script and by the siunitx, multirow and booktabs LATEX packages.
Note
1 e.g., files containing the output of GNU time.

References
1. The MathWorks 2013 Inc. Natick, Massachusetts, USA MATLAB and Statistics Toolbox Release 2013a.
2. Eaton, J W, Bateman, D, Hauberg, S and Wehbring, R 2015 GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations, CreateSpace Independent Publishing Platform, fourth edition, (March).
3. Fachada, N, Lopes, V V, Martins, R C and Rosa, A C 2016 Parallelization strategies for spatial agent-based models. International Journal of Parallel Programming, (January) pp. 1–33.
4. Keppel, D, MacKenzie, D, Juul, A H and Pinard, F 1990 GNU time, available: https://www.gnu.org/software/time/.
5. Wright, J 2016 siunitx: A comprehensive (SI) units package, (January) available: https://www.ctan.org/pkg/siunitx.
6. van Oostrum, P, Bache, Ø and Leichter, J 2010 The multi-row, bigstrut and bigdelim packages, (February) available: https://www.ctan.org/pkg/multirow.
7. Fear, S 2005 Publication quality tables in LATEX, (April) available: https://www.ctan.org/pkg/booktabs.
8. Scholmer, N 2008 matlab2tikz, available: http://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz-matlab2tikz.
9. Fachada, N 2016 Self-contained ANSI C program for benchmarking sorting algorithms, available: https://github.com/fakenmc/sorttest_c.
10. Sedgewick, R 1997 Algorithms in C, Parts 1–4: Fundamentals, Data Structures, Sorting, Searching, Addison Wesley, (September).
11. Oosterhof, N N 2015 MOxUnit – An xUnit framework for Matlab and GNU Octave, available: http://www.mathworks.com/matlabcentral/fileexchange/54417-moxunit.