parasweep: a template-based utility for generating, dispatching, and post-processing of parameter sweeps

Article

Published Version

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

To link to this article DOI: http://dx.doi.org/10.1016/j.softx.2020.100631

Publisher: Elsevier

www.reading.ac.uk/centaur

CentAUR

Original software publication

parasweep: A template-based utility for generating, dispatching, and post-processing of parameter sweeps

Eviatar Bach

Department of Atmospheric and Oceanic Science and the Institute for Physical Science and Technology, University of Maryland, College Park, MD, United States of America

ARTICLE INFO

ABSTRACT

We introduce parasweep, a free and open-source utility for facilitating parallel parameter sweeps with computational models. Instead of requiring parameters to be passed by command-line, which can be error-prone and time-consuming, parasweep leverages the model's existing configuration files using a template system, requiring minimal code changes. parasweep supports a variety different sweep types, generating parameter sets accordingly and dispatching a parallel job for each set, with support for local execution as well as common high-performance computing (HPC) job schedulers. Post-processing is facilitated by providing a mapping between the parameter sets and the simulations. We demonstrate the usage of parasweep with an example.

Code metadata

Current code version	2021.01
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_356
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	xarray [1] version 0.9+, NumPy [2], and SciPy [3]. To use the optional DRMAA functionality, DRMAA Python, DRMAA, and a DRMAA-compatible job scheduler and its DRMAA interface are required. To use the optional advanced template language, Mako is required.
Link to developer documentation/manual	http://www.parasweep.io/en/latest/
Support email for questions	eviatarbach@protonmail.com

Software metadata

Current software version	2021.01
Permanent link to this version	https://github.com/eviatarbach/parasweep/releases/tag/2021.01
Computing platforms/Operating systems	Operating systems with a Python interpreter (Linux, Microsoft Windows, and macOS, for example)

1. Motivation and significance

Parameter sweeps, whereby computational models are run repeatedly with different sets of parameters, are widely used in a plethora of scientific fields [4,5]. They can be done for a variety of reasons, such as testing sensitivity of a model to its parameters [6], exploring the qualitative changes in the behavior of a model as parameters are varied (for example, bifurcations) [7], or to find values of parameters that optimize some criterion [5]. The latter use is often employed for hyper-parameter optimization in machine learning [8]. Parameter sweeps are a classic example of an "embarrassingly parallel" problem, in that the set of simulations can easily be run in parallel because each simulation does

E-mail address: eviatarbach@protonmail.com.

not have to exchange information with the other simulations. However, most model software does not have built-in parameter sweep functionality that allows for generating parameter sets and running each instance in parallel.

We present parasweep, a free and open-source utility for easily carrying out parallel parameter sweeps for any computational model, with support for individual multi-core computers, clusters, and grids. It is written in Python, an interpreted, cross-platform language widely used for scientific applications; however, parasweep can work with models in any language. It makes use of configuration file templates in order to easily dispatch simulations with different parameter sets. The process of executing a parameter sweep and the full set of features of parasweep is discussed in Section 2.

Previous papers have focused on how to efficiently allocate resources for large parameter sweeps on various infrastructures [9–11]. parasweep does not incorporate any special scheduling strategies, but supports a number of cluster and grid schedulers through the Distributed Resource Management Application API (DRMAA), a standardized interface for communicating with job schedulers. Several tools have also been developed specifically for parameter sweep applications. One such tool, Nimrod [10], is only available for grid systems. ILab [12,13] used a similar concept of input file templates for parameter sweeping. Besides not being publicly available, this tool was less general than parasweep in the types of sweeps and the schedulers supported. The more recent preconfig [14] is a tool for generating configuration files, but does not handle dispatching or post-processing. Tools such as GNU parallel [15] and Slurm or PBS job arrays, while not designed solely for parameter sweeps, are sometimes used to facilitate them by automating the process of running the simulations with the different parameter values in parallel. However, these tools require the parameters to be passed through command-line arguments, which necessitates parsing within the model software. Moreover, those relying on job arrays only work with their respective schedulers. None of the tools in the latter group supports different types of parameter sweeps, keeps records of the parameters used, or facilitates post-processing. Thus parasweep, unlike previous tools, provides a complete cross-platform solution for generating, dispatching in parallel, and post-processing parameter sweeps, relying on a simple template-based system.

Throughout the paper, we refer to the program on which we run a parameter sweep as the *model*, a particular assignment of values to each of the parameters as a *parameter set*, a single run of the model with a particular parameter set as a *simulation*, and the collection of all the simulations as the *sweep*.

2. Software description

2.1. Software architecture

parasweep is written in Python, a cross-platform, general-purpose language widely used for scientific applications. Although Python is an interpreted language and generally slower than compiled languages such as C or C++, this is not likely to be a bottleneck since the time for generating parameter sets and filling out a template is insignificant compared to the simulation time for the vast majority of applications. An object-oriented structure makes the sweep type, dispatching, template engine, and generation of simulation identifiers entirely modular, allowing parasweep to be easily extensible. All features are documented and tested with a test suite.

The idea of parasweep is to leverage the existing configuration files of the given model. These files have a single value for each

parameter, but parasweep allows parameter values to be swept over with little effort. This is done by providing parasweep with a *configuration file template*, which is identical to the configuration file, except with placeholders where the parameters to be swept over will be inserted. The user specifies the parameter sweep, which produces sets of parameters to be given to the model. Using the template, parasweep generates a configuration file for each parameter set, and assigns this set of parameter values a unique identifier (the simulation ID). (This is explained in more detail below.) The only modification that needs to be made to the model is to receive the simulation ID as a command-line argument, read the generated configuration file corresponding to that ID, and write the output to a file also corresponding to that ID. This approach thus requires no major changes to the configuration system of the model, no parsing of parameter values through the command-line (which can be time-consuming and must be modified for every parameter added), and is easy to set up in whatever language the model is written. The simulation ID provides a way to associate the parameter set with the output for every simulation in the sweep.

The basic sequence for running parameter sweeps with parasweep is:

1. Generate the sets of parameter values.
2. For each set of parameter values:

    (a) Assign a simulation ID.
    (b) Using the configuration template, fill in the parameter values into a configuration file with the simulation ID in the name.
    (c) Dispatch a simulation with the simulation ID as a command-line argument.
    (d) In the model program, open the configuration file with the given simulation ID, read the parameters, and run the simulation. Output to a file corresponding to the same ID.

3. Return a mapping between the sets of parameter values and the simulation IDs.

The sweep type, assignment of simulation IDs, template engine, dispatching, and mapping type are all configurable and several options are provided for each within parasweep. We discuss the options for sweep types, dispatching, and mapping below. As mentioned above, parasweep's modular structure makes it easy to extend.

2.2. Software functionalities

The implemented sweeps are Cartesian product sweeps, filtered Cartesian product sweeps, set sweeps, and random sweeps. In Cartesian product sweeps (sometimes known as grid sweeps), all the possible combinations of the given parameter values are run. Filtered Cartesian product sweeps allow the user to specify in addition a filtering function of the parameters, and only those parameter sets that meet the condition of the filter are run. This can be used, for example, to run a parameter sweep of a model that takes parameters x and y, but with the condition that $x > y$. Set sweeps run only the parameter sets specified by the user. Random sweeps sample each variable as an independent probability distribution, with a wide variety of distributions from which to select.

Simulations can be dispatched by spawning processes locally, a useful option for multi-core computers. Alternatively, a large number of job schedulers typically found on high-performance computing (HPC) systems, both cluster and grid, are supported

using the Distributed Resource Management Application API (DR-MAA) [16] if it is installed on the system. This includes Slurm and PBS/Torque among a number of others.

For post-processing, parasweep keeps track of the simulation IDs assigned to each parameter set. For a Cartesian sweep, this mapping can be naturally represented as an *n*-dimensional array, where *n* is the number of parameters in the sweep. The mapping for Cartesian sweeps is thus a labeled array provided by xarray, a powerful library for handling multidimensional labeled data [1]. This array can be saved to disk as a netCDF file for future reference. For the other types of sweeps, since a multidimensional array is not a parsimonious representation, the mapping is a dictionary (hash map) between the simulation IDs and the parameter sets used. This can be saved to disk as a JSON file.

3. Illustrative example

We present the following example of the usage of parasweep. More examples, showing all the major features of parasweep, are available in the documentation.

3.1. The model

Our model in this case is a Fortran program `lorenz`, which simulates the Lorenz '63 model of convection [17] and outputs its largest Lyapunov exponent. The Lorenz model takes three parameters, β, σ, and ρ, and it is known that it is chaotic (exhibits sensitive dependence on initial conditions) for some values of these parameters and not for others. We wish to know for which parameter sets it is chaotic, and we can determine this by checking whether the largest Lyapunov exponent of the system is positive. The definition and algorithm[1] for computing the largest Lyapunov exponent is not important for our purposes. The full code for this example is provided in the parasweep code repository, but in this section we discuss only the necessary changes to be able to conduct parameter sweeps with it.

The model reads a configuration file `params.nml` which contains the values of β, σ, and ρ; we now modify it to instead use the file `params_{sim_id}.nml`, where the simulation ID `sim_id` is provided as a command-line argument.

It suffices to change

```
namelist /params/ beta, sigma, rho

open(1, file="params.nml")
read(1, nml=params)
```

to

```
namelist /params/ beta, sigma, rho
character(30) :: sim_id

call get_command_argument(1, sim_id)
open(1, file="params_" // trim(sim_id) // ".nml")
read(1, nml=params)
```

We also modify the model to output to the filename `results_{sim_id}.txt` instead of `results.txt`. We change

```
open(2, file="results.txt", action="write")
write(2, *) lyap
```

to

```
open(2, file="results_" // trim(sim_id) // ".txt",
action="write") write(2, *) lyap
```

3.2. The configuration template

Suppose the `options.txt` looked like the following:

```
&params
beta = 2.67,
sigma = 10,
rho = 28
/
```

Here β, σ, and ρ are hard-coded. To make the parameters able to be swept over, we simply need to indicate where they must go and give them an identifier surrounded by curly braces:

```
&params
beta = {beta},
sigma = {sigma},
rho = {rho}
/
```

This is the template, into which the parameter values will be substituted for every simulation in the sweep. We save it as `template.txt`. Note that this is the format of the configuration file for this particular model, and a different template has to be created for every model in order to run a parameter sweep on it.

3.3. The command

We can now run a parameter sweep. Suppose we want to try 3 evenly spaced values of β between 2 and 4, 10 values of σ between 2 and 20, and 10 values of ρ between 2 and 30. Then the sweep can be run as follows:

```
import numpy
import xarray

from parasweep import run_sweep, CartesianSweep

sweep_params = {'beta': numpy.linspace(2, 4, 3),
                'sigma': numpy.linspace(2, 20, 10),
                'rho': numpy.linspace(2, 30, 10)}

sweep = CartesianSweep(sweep_params)

mapping = run_sweep(command='./lorenz {sim_id}',
                configs=['params_{sim_id}.nml'],
                templates=['template.txt'],
                sweep=sweep)
```

This means the following:

- `command`: specifies the command to run a simulation with the model. Note that `{sim_id}` indicates where the simulation ID for each simulation in the sweep is to be substituted in the command; `sim_id` is a special keyword that must be used in both the `command` and the `configs` arguments.

---

[1] The algorithm tracks two points close to each other on the attractor and rescales the vector that connects them [18].

- configs: sets the name of the configuration file that will be created for each simulation in the sweep, where {sim_id} indicates where the simulation ID is to be substituted in the filename.
- templates: specifies the location of the configuration file template.
- sweep: specifies the sweep type. In this case, we select a Cartesian product sweep and provide the parameter values for each parameter we would like to sweep over. Since there are 3 possible values of β, 10 possible values of σ, and 10 possible values of ρ, 300 simulations will be run.
- wait: indicates whether to wait for the completion of all simulations before returning. Since the post-processing in the following section requires the results of all the simulations, we set this to True.

Descriptions of all the arguments to run_sweep is available in the documentation.

3.4. Post-processing

We now want to extract the results of the simulations and plot them. We use the mapping object returned after calling the run_sweep function. It is an xarray DataArray object, a labeled N-dimensional array. The coordinates are the sweep parameters and the "data" is the simulation IDs. This makes it easy for programs to retrieve the simulation output by the parameter values rather than having to specify the simulation IDs manually. The example below, executed after the code in Section 3.3, selects the first β (in this case, $\beta = 2$) and plots the largest Lyapunov exponent as a function of ρ and σ.

```python
def get_output(sim_id):
    filename = f'results_{sim_id}.txt'
    return numpy.loadtxt(filename)


lyap = xarray.apply_ufunc(get_output, mapping, vectorize=True)
lyap = lyap.rename('Largest Lyapunov exponent')

lyap.isel(beta=0).plot()
```

This will produce Fig. 1. The chaotic regime of the parameter space can then be easily read off as those parameter sets which result in a positive largest Lyapunov exponent (the red regions of the plot). This is just one example of the types of post-processing that can be done.

4. Impact

parasweep considerably simplifies the process of running parallel parameter sweeps, with applications to many scientific fields. As of January 2021, parasweep has had over 8500 downloads from PyPi (the official Python package repository) alone, not counting downloads from GitHub, which are not tracked. The author is aware of parasweep being used for running parameter sweeps of a coupled atmosphere–ocean model, a mathematical model of epithelial cells, electronic circuit simulations, and an ensemble forecasting method for dynamical systems [19].

5. Conclusions

We present parasweep, a Python utility for generating, dispatching, and post-processing of parameter sweeps. parasweep allows for easy generation of parameter sweeps with existing models by using a template-based system. We discuss its potential to be useful in a wide variety of scientific applications, and present an illustrative example.
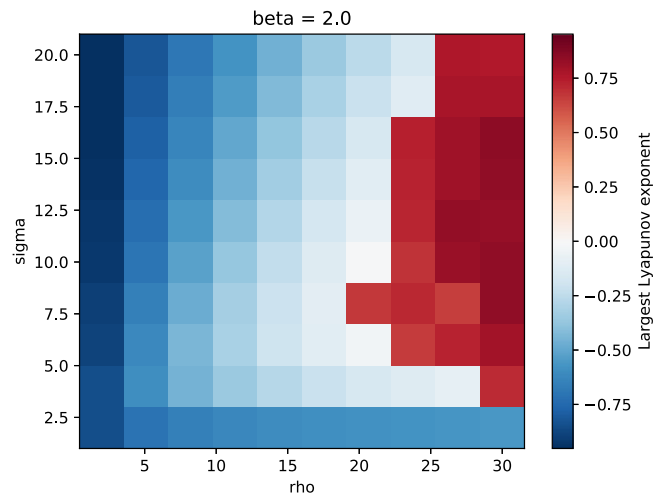

Fig. 1. The largest Lyapunov exponent of the Lorenz model as a function of ρ and σ, with fixed $\beta = 2$.

Although designed for parameter sweeps, parasweep can be useful for any application that requires generation of configuration files, dispatching tasks in parallel, and post-processing. The sweep type, assignment of simulation IDs, template engine, dispatching, and mapping type are all modular within parasweep, making it easily extensible beyond its current capabilities.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

References

[1] Hoyer S, Hamman J. Xarray: N-D labeled arrays and datasets in Python. J Open Res Software 2017;5(1). http://dx.doi.org/10.5334/jors.148.
[2] van der Walt S, Colbert SC, Varoquaux G. The NumPy array: A structure for efficient numerical computation. Comput Sci Eng 2011;13(2):22–30. http://dx.doi.org/10.1109/MCSE.2011.37.
[3] Jones E, Oliphant T, Peterson P, et al. SciPy: Open source scientific tools for Python. 2001.
[4] Wilkinson B. Grid computing: techniques and applications. CRC Press; 2009.
[5] Sudholt W, Baldridge KK, Abramson D, Enticott C, Garic S, Kondric C, et al. Application of grid computing to parameter sweeps and optimizations in molecular modeling. Future Gener Comput Syst 2005;21(1):27–35. http://dx.doi.org/10.1016/j.future.2004.09.010.
[6] Edwards NR, Marsh R. Uncertainties due to transport-parameter sensitivity in an efficient 3-D ocean-climate model. Clim Dynam 2005;24(4):415–33. http://dx.doi.org/10.1007/s00382-004-0508-8.
[7] Marsh R, Yool A, Lenton TM, Gulamali MY, Edwards NR, Shepherd JG, et al. Bistability of the thermohaline circulation identified through comprehensive 2-parameter sweeps of an efficient climate model. Clim Dynam 2004;23(7):761–77. http://dx.doi.org/10.1007/s00382-004-0474-1.
[8] Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge, Massachusetts: The MIT Press; 2016.
[9] Casanova H, Obertelli G, Berman F, Wolski R. The AppLeS parameter sweep template: User-level middleware for the grid. In: Proceedings of the 2000 ACM/IEEE conference on supercomputing. Washington, DC, USA: IEEE Computer Society; 2000.

[10] Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global grid?. In: Proceedings 14th international parallel and distributed processing symposium. 2000, p. 520–8. http://dx.doi.org/10.1109/IPDPS.2000.846030.

[11] Wilson LA, Fonner JM. Launcher: A shell-based framework for rapid development of parallel parametric studies. In: Proceedings of the 2014 annual conference on extreme science and engineering discovery environment. New York, NY, USA: ACM; 2014, p. 40:1–8. http://dx.doi.org/10.1145/2616498.2616534.

[12] Yarrow M, McCann KM, Biswas R, Van der Wijngaart RF. An advanced user interface approach for complex parameter study process specification on the information power grid. In: Buyya R, Baker M, editors. Grid computing. Lecture notes in computer science, Springer Berlin Heidelberg; 2000, p. 146–57.

[13] DeVivo A, Yarrow M, McCann KM. A comparison of parameter study creation and job submission tools. NASA advanced supercomputing technical report NAS-01-002, NASA Ames Research Center: Computer Sciences Corporation; 2001, p. 6.

[14] Nedelec F. preconfig: A versatile configuration file generator for varying parameters. J Open Res Software 2017;5(1):9. http://dx.doi.org/10.5334/jors.156.

[15] Tange O. GNU parallel: The command-line power tool. USENIX Mag 2011;36(1):42–7.

[16] Troger P, Rajic H, Haas A, Domagalski P. Standardization of an API for distributed resource management systems. In: Seventh IEEE international symposium on cluster computing and the grid. 2007, p. 619–26. http://dx.doi.org/10.1109/CCGRID.2007.109.

[17] Lorenz EN. Deterministic nonperiodic flow. J Atmos Sci 1963;20(2):130–41. http://dx.doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[18] Sprott JC. Chaos and time-series analysis. Oxford, UK: Oxford University Press; 2003.

[19] Bach E, Krishnamurthy V, Mote S, Sharma AS, Ghil M, Kalnay E. ?Ensemble Oscillation Correction (EnOC): Leveraging oscillatory modes to improve forecasts of chaotic systems, Under review.