SOFTWARE SUPPORT FOR SOLVING THE MECHANICAL PROBLEM OF BEAM BUCKLING

SOFTVERSKA PODRŠKA ZA REŠAVANJE MEHANIČKOG PROBLEMA IZVIJANJA STUBOVA

P. Marić[1], Ž. Živanov[1], M. Hajduković[1], D. D. Milašinović[2]

[1] University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia,
{petarmaric, zzarko, hajduk}@uns.ac.rs
[2] University of Novi Sad, Faculty of Civil Engineering, Subotica, Serbia,
ddmil@gf.uns.ac.rs

Abstract (English): *For the purpose of solving problems in mechanics, such as beam buckling, it is necessary to use and develop complex software solutions. This paper presents the evolution of software architecture used for multi-dimensional parametric modeling of beam buckling, based on the finite strip method. The re-engineering of the software architecture for model variation has been described with the goal of supporting the variation across four mutually independent dimensions, while discussing the constraints of the starting solution and the advantages of the target solution. The presented software solutions and their source code have been made publicly available under the BSD Open Source license.*

Apstrakt (Serbian): *Za potrebe rešavanja problema u mehanici, kao što je problem izvijanja stubova, neophodna je upotreba i razvoj složenih softverskih rešenja. U ovom radu je predstavljena evolucija arhitekture softvera za multidimenzionalnu varijaciju parametara prilikom modeliranja problema izvijanja stubova primenom metoda konačnih traka. Opisan je reinženjering arhitekture softvera za varijaciju modela tako da se podrži varijacija u četiri međusobno nezavisne dimenzije, uz diskusiju ograničenja polaznog rešenja i prednosti ciljnog rešenja. Predstavljeno softversko rešenje i njegov izvorni kod su javno dostupni pod BSD Open Source licencom.*

Keywords: *FSM, HDF5, Python, Open Source*

1. INTORDUCTION

This paper discuses software development cycle during research of behavior of mechanical structures for which both geometry and material properties can be considered as constants along a main direction, straight or curved, while only the loading distribution may vary. Analyses of such structures is based on the semi-analytical harmonic coupled finite strip method (HCFSM) for geometric non-linear analysis of reinforced concrete plate structures under multiple loading conditions [1, 2]. The HCFSM augments conventional finite strip method (FSM) by taking into account the important influence of the interaction between the buckling modes. Coupling of all series terms in the HCFSM formulation increases the computation complexity, which depends on the number of used harmonics. Application of HCFSM requires development of complex software that combines different parallelization models to speed up the calculation time and requires appropriate visualization techniques to be able to apprehend numerous numeric results [1, 2]. Development of such software is in great deal event driven, as results of the fact that some research steps give new ideas of future research direction and change previous plans. As consequence the structure of such complex software becomes unmanageable and requires re-engineering. That was the case for the developed MKTB software project used for multi-dimensional parametric modeling of beam buckling. We describe the need and the approach of the MKTB software redesign process, and describe architecture of the resulting FSM eigenvalue software solution.

2. The MKTB project

The MKTB project has been created 8 years ago, and has been used since for multi-dimensional parametric modeling of buckling and free vibration in prismatic structures, performed by solving the eigenvalue problem in HCFSM [2]. It takes the semi-analytical finite strip model data file (geometry, materials, loading) as its input and then performs its computations as a parameter sweep over 3 separate dimensions:

- D1: performs the buckling and free vibration analysis
- D2: iterates over all strip lengths, in the range specified by the input
- D3: iterates over all modes, in the range specified by the input

The total number of iterations is then D1 x D2 x D3, and they can be computed independently of each other, making this an "embarrassingly-parallel" problem.

In general, the finite strip model of structure requires fewer geometry elements than their equivalent finite element model [1, 2]. Upon analyzing the MKTB project RAM and CPU requirements it was found that the MKTB project can fit most problems on a single computer, so the task-oriented local-level parallelism in a form of multithreading/multiprocessing were found to be sufficient.

However, during development it was found that this 3-D iteration split is problematic because the resulting iterations were too fine-grained (due to their sub-second execution time) and much of the execution time was spent on synchronizing tasks and their results. Because of this we decided to make the iterations coarser by collapsing some of dimensions together with the goal of increasing the individual iteration time and decreasing the proportion of time spent on task synchronization.

The multiprocessing computation MKTB plugin supports the following levels of optimization:

- "[O0] do not optimize"
- "[O1] pre-distribute immutable data", which distributes iteration immutable data to all workers only once, when initializing the process pool
- "[O2] coarse grained parallelization", which uses coarse-grained parallelization by running all analysis and mode iterations for a specific strip length in a sequential loop for each multiprocessing process -- effectively collapsing the D1 and D3 dimensions together
- "[O3] O1 + O2", which combines [O1] and [O2] approaches.

The MKTB project can also be instructed to reorganize and group individual iterations into task "chunks", to further manage the inter-process communication overhead. It can either skip chunking, use the multiprocessing process pool default setting, or try and determine the optimal number of chunks through a heuristic - based on the number of available CPU cores.

To assess the performance of these different approaches, we benchmarked (Tab. 1) the available computation plugin configurations on a single 4 core CPU with a finite strip model composed of approx. 100,000 total iterations D1 x D2 x D3:

Table 1*:* MKTB computation plugins benchmark (4 CPU cores)

Computation plugin configuration	Time (s)	Speedup
multiprocessing (no chunking, [O3] O1 + O2)	258.2746	3.993
multiprocessing (default chunking, [O0] do not optimize)	260.4705	3.959
multiprocessing (no chunking, [O2] coarse grained parallelization)	260.5869	3.958
multiprocessing (default chunking, [O1] pre-distribute immutable data)	260.8271	3.954
multiprocessing (default chunking, [O2] coarse grained parallelization)	261.6354	3.942
multiprocessing (default chunking, [O3] O1 + O2)	263.3488	3.916
multiprocessing (no chunking, [O1] pre-distribute immutable data)	265.1048	3.890
multiprocessing (optimal chunking, [O0] do not optimize)	266.8663	3.864
multiprocessing (optimal chunking, [O3] O1 + O2)	267.4888	3.855
multiprocessing (optimal chunking, [O2] coarse grained parallelization)	267.8917	3.850
multiprocessing (optimal chunking, [O1] pre-distribute immutable data)	268.3978	3.842
multiprocessing (no chunking, [O0] do not optimize)	338.8451	3.044
Sequential	1031.3018	1.000

Best speedup over the sequential plugin is 3.993 times, or 99.825% of the maximum theoretical speedup achievable on a 4 core CPU. However, it's important to note that the total number of iterations may be substantially increased, i.e. by providing a model that iterates over strip lengths with a very small step – to precisely determine individual bifurcation points.

3. The fsm_eigenvalue project

Over the years the MKTB project has organically accumulated technical debt in such amount that it became very difficult to add new features or improve upon optimizations. In that time we have refactored several times, but the underlying technologies we base our work upon have changed substantially as well.
In 2017 we've decided it's faster and easier to re-architect our solution instead with the goal of paying off technical debt, adding flexibility, supporting the model variation across additional independent dimensions, achieving higher levels of modularization, better execution performance, and publishing the resulting project as Open Source Software.
The fsm_eigenvalue project [3] is a complete re-engineering of the MKTB project, and it provides a reference Open Source Software implementation for parametric modeling of buckling and free vibration in prismatic structures, performed by solving the eigenvalue problem in HCFSM.
The fsm_eigenvalue project is composed of the following loosely-coupled software modules:

- '.load', responsible for loading the parametric model data file
- '.compute', responsible for solving the HCFSM eigenvalue problem and modeling of buckling and free vibration
- '.compute.integral_db', responsible for downloading and working with the precomputed HCFSM integral tables [2, 4]
- '.compute.matrices', responsible for computing the local/global stiffness/stress/mass matrices
- '.compute.parameter_sweep', responsible for performing the multi-dimensional parameter sweep and distributing its workload onto multiple CPUs
- '.store', responsible for storing the computed results
- '.main', responsible for gluing the above modules together
- '.shell', responsible for providing the applications command-line interface

The fsm_eigenvalue project takes the semi-analytical finite strip model data file (geometry, materials, loading) as its input and then performs its computations as a parameter sweep over 4 separate dimensions:

- D1: performs the buckling and free vibration analysis
- D2: iterates over all strip lengths, in the range specified by the input
- D3: iterates over all strip thicknesses, in the range specified by the input
- D4: iterates over all modes, in the range specified by the input

The total number of iterations is then D1 x D2 x D3 x D4, and they can be computed independently of each other, making this an "embarrassingly-parallel" problem. The '.compute.parameter_sweep' module has been inspired by the MKTB multiprocessing computation plugin, so it also uses task-oriented local-level parallelisms in form of multithreading/multiprocessing to combat the increased computational complexity.
The 4-D iteration split was yet again found to be problematic because the resulting iterations were too fine-grained (due to their sub-second execution time) and much of the execution time was spent on synchronizing tasks and their results. This is why we made the iterations coarser this time as well, by collapsing some of dimensions together with the goal of increasing the individual iteration time and decreasing the proportion of time spent on task synchronization.
The '.compute.parameter_sweep' module automatically reorganizes and groups individual iterations into task "chunks", to further manage the inter-process communication overhead. It will try to determine the optimal number of chunks dynamically, through a heuristic based on the number of dimension-specific iterations within the parametric model data file.
During development it was found that even with all of the optimizations listed above the total number of iterations can still be very large – the fsm_eigenvalue_experiments [5] project provides data files that have over 1,000,000 iterations each.
Therefore, the '.compute.parameter_sweep' and '.store' modules have been reworked to use Python's generator/iterator protocols when computing iterations and collecting their results. In simplest terms, the '.compute.parameter_sweep' module computes the parametric model iterations and immediately "yields" their individual results through an iterator, while the '.store' module "listens" to this iterator. This effectively means that the fsm_eigenvalue project can write out its results just-in-time as they're computed, without waiting for the entire list of results to complete (or even actually forming the list for that matter). Thanks to the expressive power of the Python programming language this optimization has been implemented in less than 20 lines of code.
The HDF5 (Hierarchical Data Format 5) [6] format has been selected for storing the computed results, because of its innate ability to efficiently store and organize large amounts of numerical data. HDF5 is an extensible and open standard, comprised of platform independent technologies which are available under Open Source licenses. HDF5 format has been designed with the objective of creating data storages that are self-descriptive, flexible, and have extremely fast and efficient access patterns to the stored data.
For each combination of mode, strip length and thickness the fsm_eigenvalue project outputs a very large amount of numerical data: critical stress, natural frequency, their approximations via physical duality, approximation errors, minimal critical stress vector, minimal natural frequencies vector, etc. To approximate the natural frequency from stress via physical duality (and vice versa) a separate Open Source programming library has been created, named physical_dualism [7].

When larger finite strip models are analyzed, such as the example data files provided by the fsm_eigenvalue project which have 51 modes, 7800 strip lengths (100-4000 mm with a 0.5 mm iteration step) and 140 strip thicknesses (2-9 mm with a 0.05 mm iteration step), computed results can amount to over 50 GB of compressed data per HDF5 file.
Such data can't be inspected manually, so we've built several Open Source projects that support automated visualization and modal analysis of the parametric model of buckling and free vibration in prismatic shell structures [4, 5, 7-19].

4. CONCLUSION

The lessons learned during development of MKTB software and FSM eigenvalue software solutions stress the need of modular design, where each module should have the simple and clear function. In such case redesign means recombination of existing modules with addition of several new ones.
In the end, several Open Source Software implementations have been produced: for approximating the natural frequency from stress via physical duality (and vice versa), parametric modeling of buckling and free vibration in prismatic structures, automated experiment management, their visualization and modal analysis.

Acknowledgements

The work presented in this paper is a part of the investigation conducted within the research project OI 174027 "Computational Mechanics in Structural Engineering", supported by the Ministry of Science and Technology, Republic of Serbia. This support is gratefully acknowledged.

REFERENCES

[1] Clough R. and Penzien J. (1995), "Dynamics of Structures". McGraw-Hill, USA.
[2] P. Marić, D.D. Milašinović, D. Goleš, Ž. Živanov, M. Hajduković (2017), "A Hybrid Software Solution for the Harmonic Coupled Finite Strip Method Characteristic Equations", in P. Iványi, B.H.V. Topping, G. Várady, (Editors), Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, Stirlingshire, UK, Paper 31, 2017. doi:10.4203/ccp.111.31
[3] P. Marić, D.D. Milašinović, „fsm_eigenvalue public code repository", https://bitbucket.org/petar/fsm_eigenvalue
[4] P. Marić, D.D. Milašinović, „beam_integrals public code repository", https://bitbucket.org/petar/beam_integrals
[5] P. Marić, „fsm_eigenvalue_experiments public code repository", https://bitbucket.org/petar/fsm_eigenvalue_experiments
[6] The HDF Group, "HDF5", https://www.hdfgroup.org/HDF5/
[7] P. Marić, D.D. Milašinović, „physical_dualism public code repository", https://bitbucket.org/petar/physical_dualism
[8] P. Marić, „dynamic_pytables_where_condition public code repository", https://bitbucket.org/petar/dynamic_pytables_where_condition
[9] P. Marić, „export_beam_integrals public code repository", https://bitbucket.org/petar/export_beam_integrals
[10] P. Marić, „friendly_name_mixin public code repository", https://bitbucket.org/petar/friendly_name_mixin
[11] P. Marić, „fsm_damage_analysis public code repository", https://bitbucket.org/petar/fsm_damage_analysis
[12] P. Marić, „fsm_effective_stress public code repository", https://bitbucket.org/petar/fsm_effective_stress
[13] P. Marić, „fsm_load_modal_composites public code repository", https://bitbucket.org/petar/fsm_load_modal_composites
[14] P. Marić, „fsm_modal_analysis public code repository", https://bitbucket.org/petar/fsm_modal_analysis
[15] P. Marić, „fsm_strip_length_analysis public code repository", https://bitbucket.org/petar/fsm_strip_length_analysis
[16] P. Marić, „fsm_strip_length_damage_analysis public code repository", https://bitbucket.org/petar/fsm_strip_length_damage_analysis
[17] P. Marić, „fsm_strip_thickness_analysis public code repository", https://bitbucket.org/petar/fsm_strip_thickness_analysis
[18] P. Marić, „fsm_strip_thickness_damage_analysis public code repository", https://bitbucket.org/petar/fsm_strip_thickness_damage_analysis
[19] P. Marić, „simple_plugins public code repository", https://bitbucket.org/petar/simple_plugins