Open Compute Stack (OpenCS) Framework

Dragan D. Nikolić
http://daetools.sourceforge.io
DAE Tools Project, Belgrade, Serbia

Introduction

Large scale systems of non-linear (partial-)differential and algebraic equations are found in many engineering problems. One of the methods to solve this type of problems is by using the equation-based approach. According to this approach, all equations and variables which constitute the model representing the process are generated and gathered together. Then, equations are solved simultaneously using a suitable mathematical algorithm. Equations are given in an implicit form as functions of state variables and their derivatives, degrees of freedom (the system variables that may vary independently), and parameters.

In general, a typical equation-based model consists of a coupled set of *partial-differential equations* and *auxiliary* algebraic and ordinary differential equations. An individual partial-differential equation is often treated as a distinct group of equations – a *kernel equation*. *Kernel equations* represent a homogeneous group of identical mathematical expressions operating on different variables and can be efficiently evaluated on both general purpose and streaming processors/accelerators (and heterogeneous systems). Typical examples represent mass, heat and momentum balance equations and various scalar/vector transport equations. The *auxiliary equations* represent, for instance, boundary conditions, phase equilibrium, connectivity between units and various process performance indicators. The *auxiliary equations* are evaluated on general purpose processors.

In general, simulation programs for this class of problems are developed using:

- General-purpose programming languages such as C/C++ or Fortran, one of available suites for scientific applications such as SUNDIALS[Sundials], Trilinos[Trilinos] and PETSc[PETSc] and high-performance computing SDKs such as Intel oneAPI[oneAPI], AMD ROCm[ROCm], NVIDIA HPC[NVidia-hpc] and ARM HPC[ARM-hpc].
- Libraries for finite element (FE) analysis and computational fluid dynamics (CFD) such as deal.II[dealII], libMesh[libMesh], FEniCS[FEniCS], Firedrake[Firedrake], FreeFem++[FreeFem], and OpenFOAM[OpenFoam].
- Computer Aided Engineering (CAE) software for finite element analysis and computational fluid dynamics such as HyperWorks[HyperWorks], STAR-CCM+ and STAR-CD[Siemens-MDE], COMSOL Multiphysics[COMSOL], ANSYS Fluent/CFX[ANSYS] and Abaqus[Abaqus].

The equation-based problems are found in the process, chemical, petrochemical, pharmaceutical and other engineering areas, natural sciences and financial systems. They are employed for tasks such as: *simulation*, *optimisation*, *parameter estimation*, *sensitivity analysis*, *model predictive* and *optimal control*, and *supply chain optimisation*. Their numerical solution requires the following computationally intensive tasks:

1. *Numerical integration in time* (requires evaluation of model equations, residuals for DAE and right hand side for ODE systems).
2. *Linear algebra operations* (BLAS L1 vector-vector and a subset of the BLAS L2 matrix-vector operations).
3. *Solution of systems of linear equations* (both direct and iterative linear solvers require computation of analytical derivatives).
4. *Integration of sensitivity equations* (local gradient-based sensitivity analysis methods require evaluation of sensitivity residuals).

Parallelisation of these tasks leads to a faster numerical solution.

Simulation programs can be parallelised at different levels and, for maximum performance, all available resources should be exploited:

- *Level 1*: inter-node parallelism in a distributed memory system.
- *Level 2*: intra-node task parallelism utilising multiple cores and streaming processors / accelerators.
- *Level 3*: data parallelism utilising multiple vector units at a single work item level.

The task parallelism comes from a large number of cores in the processor/accelerator while the data parallelism comes from support for vector instructions, which make every compute unit a single-instruction multiple-data (SIMD) processor.

Different applications utilise different levels of parallelism.

- *Simulation* of large scale models requires a numerical integration of a single model on a distributed memory system. They can take advantage of all three levels of parallelism.
- *Sensitivity analysis* requires a huge number of simulations performed simultaneously and independently from each other in parallel. In general, the task and data parallelisms are used. The distributed memory systems can be employed as well since there is no inter-node dependency.
- *Optimisation* requires a huge number of simulations performed typically sequentially. In general, they are performed on shared memory systems utilising task and data parallelisms. However, many modern optimisation solvers provide parallel algorithms. In addition, some solvers support execution on the distributed memory systems.
- *Parameter estimation* is a special class of optimisation problems with the fixed problem definition and the objective function (i.e. least squares). Similarly to the optimisation problems, it can utilise all three levels of parallelism.
- *Model Predictive Control* / *Optimal Control* are optimisation problems that require very fast simulations (near real-time capabilities). The models are typically small scale and significantly simplified. Simulations are often performed on embedded systems in a

sequential fashion. Depending on the hardware, the task and data parallelisms might or might not be supported.

- *Supply chain optimisation* is a class of optimisation problems for the strategic level decisions, such as the number of plants, the modes of transport, or the relocation of warehouses. In general, all three levels of parallelism can be employed.

Software description

The *Open Compute Stack* (`OpenCS`) framework is a common platform for modelling of problems described by large-scale systems of differential and algebraic equations (ODE or DAE), parallel evaluation of model equations on diverse types of computing devices (including heterogeneous setups), parallel simulation on shared and distributed memory systems, and model exchange.

The framework provides a platform-independent binary interface for model description with the data structures to describe, store in computer memory and evaluate large scale systems of equations. The same model specification can be used on different high performance computing systems and architectures. Model equations are specified in a symbolic form using the `OpenCS` API, transformed into the bytecode instructions or compute kernels using the operator overloading technique and stored as an array of binary data (a *Compute Stack*) for direct evaluation by simulators on all platforms/operating systems (including heterogeneous systems). Models can contain a coupled set of kernel equations and grouped auxiliary algebraic and differential equations. The framework automatically generates C++ and OpenCL source code for kernels. The compute kernels can be generated for automatic vectorisation or explicitly vectorised by the framework. The framework support user-defined functions by injecting function calls into the kernel expressions. The source code for C++ shared library kernels is automatically compiled and loaded by the framework and each group or kernel can be assigned to a different computing device (processor or accelerator).

The OpenCS framework supports different architectural designs of modern supercomputers:

1. Large number of processors having many cores and hardware threads and wide SIMD registers (i.e. Fugaku machine installed at RIKEN in Japan).
2. Large number of processors equipped with GPU accelerators (most supercomputers in USA/Europe).

For the first design option the framework provides vectorised compute kernels. Apart from 512 bit AVX-512, the framework offers variable width vector kernels (i.e. for vector extensions such as SVE 2). Regarding the second design option, the OpenCS framework supports heterogeneous CPU/GPU systems where compute kernels can be executed using the OpenMP API and the OpenCL framework or one of performance portable programming models such as Kokkos and SYCL.

The `OpenCS` framework is based on several algorithms and methodologies for parallelisation of equation-based simulation programs:

1. Parallelisation on heterogeneous computing systems[opencs-shared-memory].
2. Parallelisation on distributed memory systems[opencs-mpi].

3. Parallelisation using compute kernels code generation techniques[opencs-kernels].

It consists of the following components:

- Model specification data structures for a description of general systems of differential and algebraic equations (ODE or DAE).
- Method to describe, store in computer memory and evaluate model equations on diverse types of computing devices (the *Compute Stack* approach).
- Algorithm for partitioning of general systems of equations in the presence of multiple load balancing constraints.
- Algorithm for inter-process data exchange.
- An Application Programming Interface for model specification, parallel evaluation of model equations and model exchange.
- An Application Programming Interface for specification of kernel equations.
- Method for generation of the source code for kernels in multiple languages targeting different APIs/frameworks.
- Method for evaluation of coupled kernels and auxiliary equations on multiple computing devices/architectures.
- Method for automatic and explicit vectorisation of compute kernels.
- Cross-platform simulation software for parallel numerical solution of general ODE/DAE systems of equations on shared and distributed memory systems.

Main capabilities

The OpenCS models can be developed in C++ and Python or exported from simulators using the provided API. The architecture and the main components of the framework are illustrated in Fig. 1. The OpenCS framework provides the following libraries:

- *cs_machine.h* (header-only *Compute Stack Machine* implementation in C99).
- *libOpenCS_Evaluators* (sequential, OpenMP and OpenCL *Compute Stack Evaluator* implementations).
- *libOpenCS_Models* (*Compute Stack Model*, *Compute Stack Differential Equations Model*, *Compute Stack Model Builder* and kernel generator implementations).
- *libOpenCS_Simulators* (ODE and DAE simulators implementations).

and a standalone simulator (*csSimulator*, for both ODE and DAE problems).

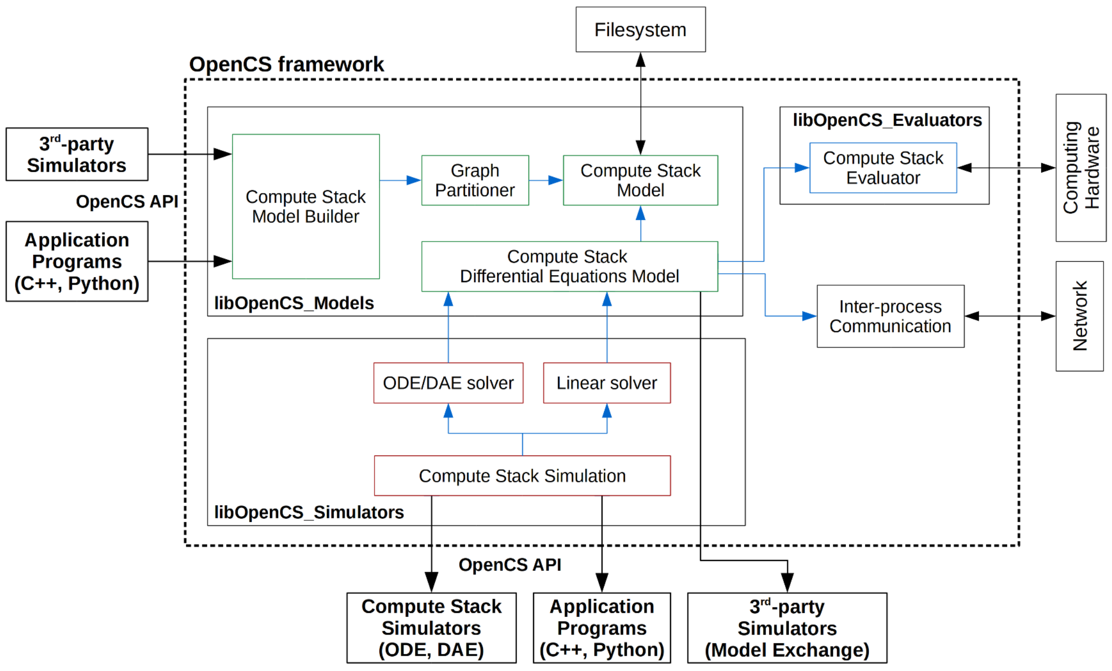

Figure 1. The structure and the main components of the OpenCS framework.

In the OpenCS approach, the model specification contains only the low-level information directly required by solvers and, in general, it can be generated from any modelling software. The OpenCS model specification, represented by a *Compute Stack Model*, provides a common interface to ODE/DAE solvers and can be generated using the OpenCS API in two ways (Fig. 2):

1. Direct implementation in C++ or Python application programs.

2. Export of existing models from third-party simulators.

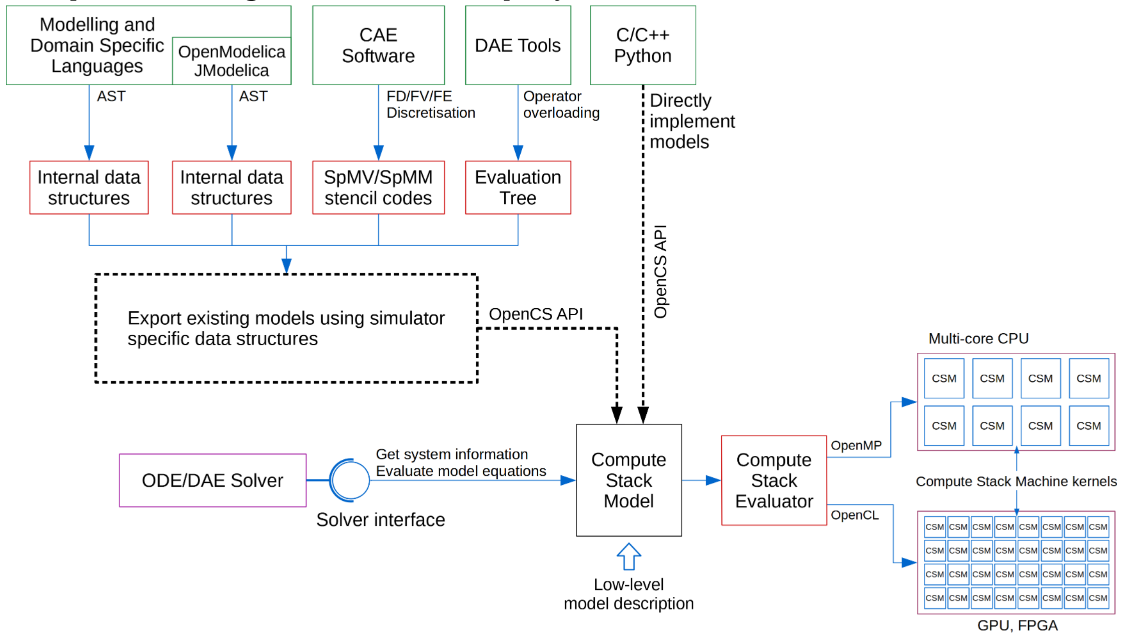

Figure 2. The OpenCS modelling approach. The low-level model specification is created using the OpenCS API and stored in a Compute Stack Model data structure which provides a generic interface to ODE/DAE solvers. Model equations, transformed into the bytecode instructions and stored as an array of binary data, are evaluated using the Compute Stack Machine kernels managed by a Compute Stack Evaluator.

The model specification data structures are stored as files in a binary format and used as inputs for parallel simulations on all platforms. This way, they provide a simple binary interface for model exchange. Model equations are stored as an array of binary data (bytecode instructions). A limited set of bytecode instructions is utilised (only memory access to supplied data arrays and unary and binary mathematical operations). Individual equations (*Compute Stacks*) are evaluated by a stack machine (*Compute Stack Machine*) using the Last In First Out (LIFO) queues or executed as C/C++ or OpenCL compute kernels. Systems of equations are evaluated in parallel using a *Compute Stack Evaluator* interface which manages the *Compute Stack Machine* kernels. Two APIs/frameworks are used for parallelism:

- The Open Multi-Processing (OpenMP) API for parallelisation on general purpose processors.
- The Open Computing Language (OpenCL) framework for parallelisation on streaming processors and accelerators (GPU, FPGA, heterogeneous CPU+GPU and CPU+FPGA systems).

Switching to a different computing device for evaluation of model equations is straightforward and controlled by an input parameter.

A generic simulation software is provided by the framework to utilise the low-level information stored in *Compute Stack Models*[opencs-mpi]. Simulations can be executed sequentially on a single processor or in parallel on message passing multiprocessors (Fig. 3). Simulation inputs are specified in a generic fashion as files in a (platform independent) binary format. The input files are generated using the `OpenCS` API, one set per processing element (PE)) and contain the stored model specification data structures and solver options. Simulation results are available in HDF5 or Comma Separated Value (.csv) formats.

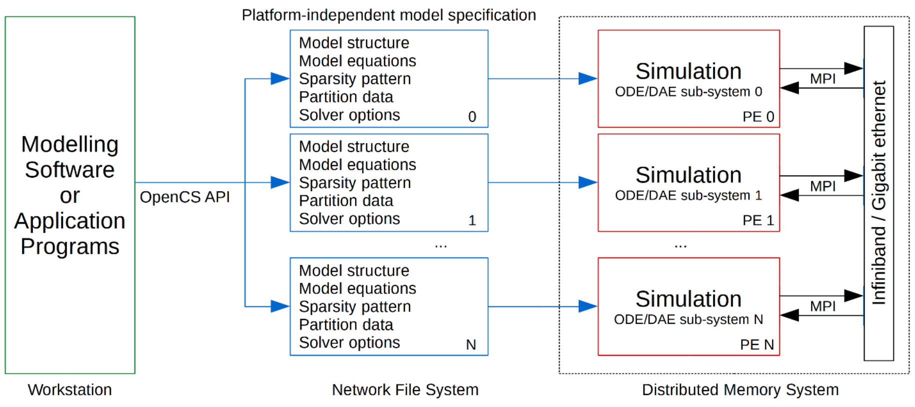

Figure 3. Parallel simulation using the `OpenCS` framework.

On shared memory systems simulations are executed on a single processing element utilising the available computing hardware such as multi-core CPUs and GPUs (illustrated in Fig. 4). On distributed memory systems simulations are executed on a number of processing elements where every processing element integrates one part (sub-system) of the overall ODE/DAE system in time and performs an inter-process communication to exchange the data between processing elements. Hence, the software for numerical solution on shared memory systems is used as the main building block for distributed memory systems (given in Fig. 5).

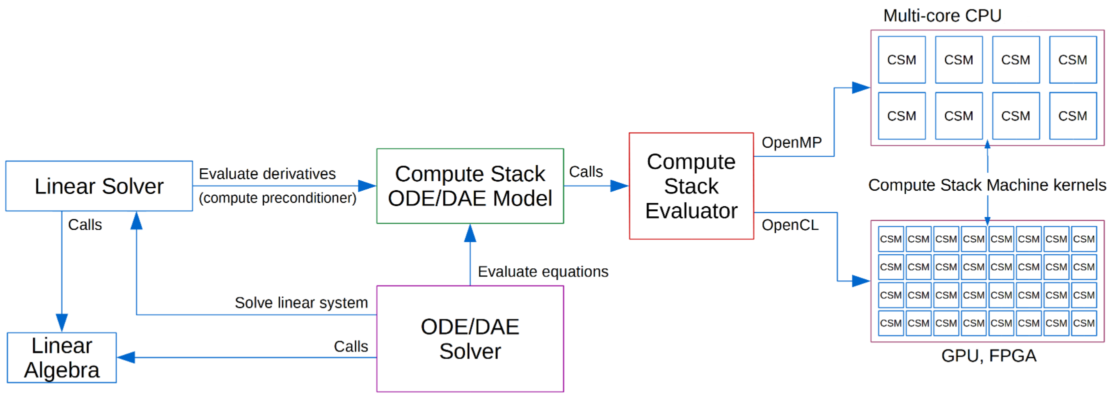

Figure 4. `OpenCS` simulation on shared memory systems.

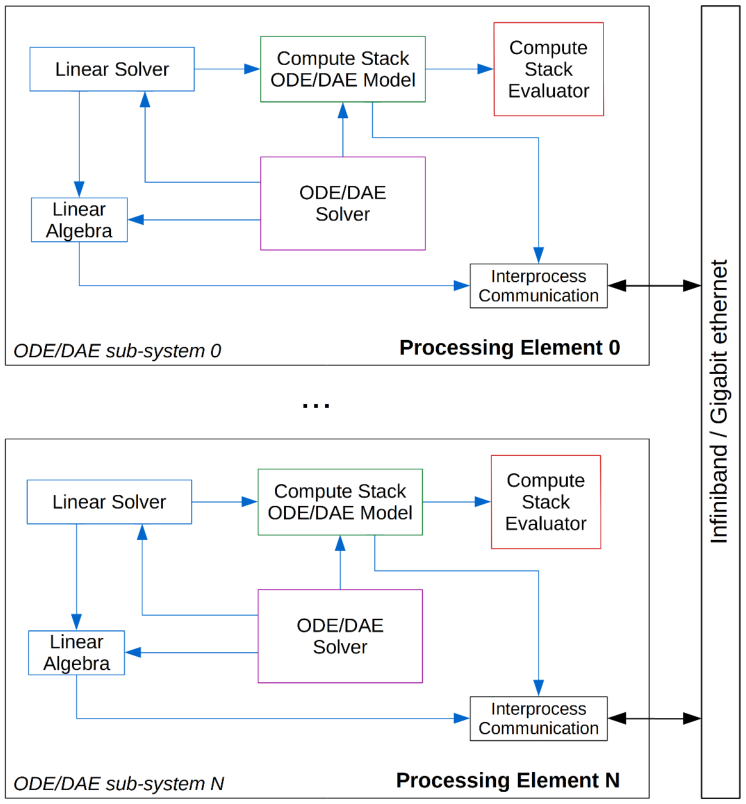

Figure 5. `OpenCS` simulation on distributed memory systems.

The framework offers the numerous benefits. A single simulation software is used for numerical solution of systems of differential and algebraic equations on all platforms. Model equations, either stored as the bytecode instructions or generated as compute kernels, can be evaluated on virtually all computing devices with no additional processing. Each group of equations or kernel can be assigned to a different computing device (controlled by an input parameter). The low-level model specification data structures, stored as files in binary format, are used as an input for parallel simulations on all platforms and provide a simple platform-independent binary interface for model exchange. The partitioning algorithm can accurately balance the computation and memory loads in all important phases of the numerical solution.

The quality of the software is a very important aspect and the formal code verification techniques (such as the Method of Exact Solutions and the Method of Manufactured Solutions) and the most rigorous acceptance criteria (such as the order-of-accuracy) are applied to test almost all aspects of the software.

The typical use-cases of the `OpenCS` framework include:

- Development of custom large-scale models in C++ and Python.

- Parallel evaluation of model equations (i.e. in simulators with no support for parallel evaluation or using the computing devices which are currently not utilised).
- Parallel simulation on shared and distributed memory systems.
- Model-exchange.
- Use as a simulation engine for Modelling or Domain Specific Languages.
- Benchmarks between different simulators, ODE/DAE solvers, computing devices and high performance computing systems (since a common model-specification is used on all platforms, `OpenCS` models can be used to benchmark memory and computation performance of individual computing devices or high performance computing systems; for example, benchmarks between heterogeneous CPU/GPU and CPU/FPGA systems could be performed without re-implementation of the model for a completely different architecture).

References

[Abaqus]: Dassault Systemes SE. (2018). Abaqus. http://www.simulia.com

[ANSYS]: Ansys, Inc. (2022). ANSYS Fluent. http://www.ansys.com

[ARM-hpc]: The ARM, Ltd. (2022). ARM HPC. https://developer.arm.com

[COMSOL]: COMSOL, Inc. (2022). COMSOL Multiphysics. http://www.comsol.com

[dealII]: Bangerth, W., Hartmann, R., & Kanschat, G. (2007). deal.II – a General Purpose Object Oriented Finite Element Library. ACM Trans. Math. Softw., 33(4), 24/1–24/27.

[FEniCS]: Alnaes, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., & Wells, G. N. (2015). The FEniCS project version 1.5. Archive of Numerical Software, 3. https://doi.org/10.11588/ans.2015.100.20553

[Firedrake]: Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., Mcrae, A. T. T., Bercea, G.-T., Markall, G. R., & Kelly, P. H. J. (2016). Firedrake: Automating the Finite Element Method by Composing Abstractions. ACM Trans. Math. Softw., 43(3). https://doi.org/10.1145/2998441

[FreeFem]: Hecht, F. (2012). New development in FreeFem++. Journal of Numerical Mathematics, 20(3-4), 251–266. https://doi.org/10.1515/jnum-2012-0013

[HyperWorks]: Altair Engineering, Inc. (2022). HyperWorks. https://altairhyperworks.com

[libMesh]: Kirk, B. S., Peterson, J. W., Stogner, R. H., & Carey, G. F. (2006). libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. Engineering with Computers, 22(3–4), 237–254.

[Mills]: R. T. Mills, M. F. Adams, S. Balay, J. Brown, A. Dener, M. Knepley, S. E. Kruger, H. Morgan, T. Munson, K. Rupp, B. F. Smith, S. Zampini, H. Zhang, and J. Zhang, Toward performance-portable PETSc for GPU-based exascale systems, Parallel Computing, 108 (2021), https://doi.org/10.1016/j.parco.2021.102831.

[NVIDIA-hpc]: The NVIDIA, Inc. (2022). NVIDIA HPC. https://developer.nvidia.com

[oneAPI]: The Intel, Inc. (2022). oneAPI. https://oneapi.io

[opencs-shared-memory] Nikolić, D. D. (2018). Parallelisation of equation-based simulation programs on heterogeneous computing systems. PeerJ Computer Science, 4, e160. https://doi.org/10.7717/peerj-cs.160

[opencs-mpi]: Nikolić, D. D. (2023a). Parallelisation of equation-based simulation programs on distributed memory systems. Zenodo. https://doi.org/10.5281/zenodo.8037490

[opencs-kernels]: Nikolić, D. D. (2023b). Parallelisation of equation-based simulation programs using kernel code generation techniques. Zenodo. https://doi.org/10.5281/zenodo.8037508

[OpenFOAM]: The OpenFOAM Foundation. (2022). OpenFOAM. http://www.openfoam.org/

[PETSc]: Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., & Zhang, H. (2015). PETSc Users Manual (ANL-95/11 - Revision 3.6). Argonne National Laboratory. http://www.mcs.anl.gov/petsc

[ROCm]: The AMD, Inc. (2022). ROCm. https://www.amd.com/en/developer/rocm-hub.html

[Siemens-MDE]: Siemens. (2022). Multidisciplinary Design Exploration. https://mdx.plm.automation.siemens.com

[Sundials]: Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., Woodward, C. S. (2005). SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. ACM Trans. Math. Softw., 31(3), 363–396. https://doi.org/10.1145/1089014.1089020

[Trilinos]: Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., & Stanley, K. S. (2005). An overview of the Trilinos project. ACM Trans. Math. Softw., 31(3), 397–423. https://doi.org/10.1145/1089014.1089021