SoftwareX 22 (2023) 101329

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

TDF: A compact file format plugin for FEniCS

Anthony Dowling, Lin Jiang, Ming-Cheng Cheng, Yu Liu *

Department of Electrical & Computer Engineering, Clarkson University, Potsdam, NY, 13699, USA

ARTICLE INFO

Article history:
Received 18 November 2022
Received in revised form 11 January 2023
Accepted 27 January 2023

Keywords:
Hierarchical Data Format 5 (HDF5)
Type-Length-Value (TLV)
Finite Element Method (FEM)
FEniCS

ABSTRACT

The Finite Element Method (FEM) is an important numerical method to solve Partial Differential Equations (PDEs). The widely used open source FEM platform — FEniCS, supports Hierarchical Data Format 5 (HDF5), which is very effective at storing and organizing large amounts of simulation data. However, HDF5 files can become prohibitively large for certain engineering applications, such as the thermal simulation of CPUs. Thus, this paper introduces a Type-Length-Value data format (TDF) plugin for compact storage of the FEM simulation solutions. Our Type-Length-Value (TLV) encoding implementation can be readily expanded to store more generic mathematical data generated by engineering and scientific applications. The evaluation results indicate that the TDF format could lead to a ≈95% space savings in our illustrated example. Meanwhile, the read/write speed has been improved compared to HDF5 used by FEniCS.

Code metadata

Code metadata description	Metadata
Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00380
Permanent link to reproducible capsule	none
Legal Code License	GPL
Code versioning system used	git
Software code languages, tools, and services used	C++, MPI
Compilation requirements, operating environments & dependencies	GCC, G++, CMake, Make, Dolfin, Linux
If available Link to developer documentation/manual	https://github.com/dowlinah/tdf#readme
Support email for questions	yuliu@clarkson.edu

1. Motivation and significance

Partial Differential Equations (PDEs) are important to model nearly all fields of science and engineering, and the Finite Element Method (FEM) is a widely used numerical method to solve PDEs. To implement the FEM, the domain of interest is discretized into small elements through mesh generation techniques. Usually, numerical accuracy in the FEM benefits from a higher resolution mesh, but this greatly increases the computation time and storage needs.

Hierarchical Data Format (HDF) is a set of file formats (e.g., HDF4, HDF5) designed to store and organize large amounts of data [1]. The open source platform — FEniCS [2], is widely used in the FEM simulation to solve engineering problems, and it utilizes the HDF5 as the principle file format. However, HDF5 files can become prohibitively large, which creates a challenge to collecting the FEM data obtained from FEniCS simulation. In this paper, we study the storage size required by our illustrated example — the thermal simulation of CPUs. In this example, we performed the FEM simulation to predict the thermal behaviors on all functional units of an AMD Athlon CPU, and a dataset of 20,000 time step solutions can easily achieve a storage size of 1 terabyte with a coarse mesh of 129 × 129 × 14. For the finer mesh of 512 × 512 × 14, the storage size requested for the same number of time steps is around 12 terabytes, which is difficult for most research projects. Note that only ≈87 ms of the thermal behavior is simulated, even with 20,000 time steps, and longer simulation time is required to match real-world scenarios. Therefore, it is necessary to propose a new compact file format plugin for FEniCS to accommodate such scenarios.

During our examination of the HDF5 format, it was observed that storing an FEM solution could take more than 200 ms when

* Corresponding author.
E-mail address: yuliu@clarkson.edu (Y. Liu).

https://doi.org/10.1016/j.softx.2023.101329

Table 1
Type field bits.

Bits	Description
0	Vector/Matrix indicator
1–3	Data type indicator
4–7	TDF field number

Table 2
Data type indicator values (Note that 110 & 111 are unused).

Value	000	001	010	011	100	101
Type	`unsigned`	`int`	`float`	`double`	`char`	`long unsigned`

a higher resolution mesh was used. This leads to a large time overhead when a long simulation is performed. In the example of 20,000 time steps, 200 ms to store a solution accounts for more than 60 min of the total run time. Therefore reducing the time to store solutions is necessary to reduce the time requirements for FEM simulation.

Type-Length-Value(TLV) encoding has very successful implementations in the data encoding of network protocols (e.g., Transport Layer Security Protocol [3]) and video formats (e.g., QuickTime File Format [4]). We realized that it can be readily customized for versatile uses in the data storage of a variety of applications. For this work, this encoding is utilized with the goal of being able to store FEM solution data effectively, and our implementation for FEniCS can be readily expanded to store more generic mathematical data generated by engineering and scientific simulations. Our evaluation indicates that the proposed TLV data format (TDF) could lead to a ≈95% space savings per time step in our illustrated example of thermal simulation compared to FEniCS' HDF5 interface, while the read performance is improved by 4.9% on average. The write performance is improved by 70.1% on average and 98.9% on average when not storing the metadata.

2. Software description

2.1. TDF design

Besides the enormous storage size needed, the HDF5 implementation in FEniCS is restrictive to the user, and does not allow fine-grained control over the file operation, especially when parallel operation is considered. Thus, we need a new format allows full control over key operations with sufficient simplicity, allowing the user to make use of the data efficiently. Also, it can be easily parsed and searched by generalized parsing functions. Thus, the TLV encoding is an ideal choice for such purposes.

TLV is comprised of three components: the type of the stored data, the length of the stored data, and the actual stored data (i.e., a "length" amount of data representing the value for the "type"). The type field of TLV encodes the type of data stored in the TLV field. Table 1 shows the bit fields of the type header. The first bit encodes whether the TLV field is a vector or a matrix, the second through fourth bits encode the data type, and the last 4 bits are kept to encode an application-chosen number for the TLV field. The last 4 bits are optional, but kept for potential future use, and to maintain the byte-alignment of the format. Table 2 shows the different values that the data type indicator can take. Note that a single file (chunk) of TLV encoded data is allowed to include multiple TLV fields, demarcated by the TLV headers, but in this plugin, each file is comprised of only a single TLV field. Moreover, matrices and vectors are basic data structures used in numerical simulations, such as building linear equations. For these goals, there is a bit reserved to indicate if the data stored is a matrix or a vector. In FEniCS, FEM solutions are stored in vectors. Thus, support for matrix storage has not yet been implemented, but can readily be added at a later stage to expand the versatility of the format. For matrix data, the length field would need to include the number of rows and the number of columns of the stored matrix.

The length field of the TLV header is encoded as a 64-bit integer. This allows for the data length to become extremely large if needed. In the case that the field is storing a matrix, this would be encoded as a pair of 64-bit integers. This integer value stores the number of elements in the value field. The actual size of each element may vary greatly between different data types, for instance a `char` value may only take one byte, while a `double` may take 8 instead. Encoding the length as the number of values helps to keep the logic of the format consistent between different data types and avoids the overhead of accounting for potential variations in data size between types.

2.2. TDF implementation for storing FEM solutions in FEniCS

DOLFIN is the C++/Python computational high-performance backend library of FEniCS, and it provides data structures and algorithms for finite element meshes, automated finite element assembly, and numerical linear algebra [2]. Through study of the DOLFIN library implementation, it was found that there are a total of four vectors of data that are needed to properly load a FEniCS Function object into a mesh for use. Three of these vectors consist of metadata used for ordering the actual solution values, while only the last vector contains the solution data. Thus, the metadata takes up a large majority of the space, especially for the finer mesh. However, such data only needs to be saved once for the entire simulation run. The method described to use HDF5 files for storing FEniCS functions implies that these vectors are stored for every time step's FEM solution. This leads to an unnecessary and massive increase in the disk usage.

DOLFIN `Function` objects store the data values at each mesh node and their indices for ordering and access. In the DOLFIN API, there is a function used by the HDF5 utility that uses these vectors to order the data into a `Function` object for use. This function is used by our implementation for ordering points into the `Function` objects. This is allowed because the logic for ordering does not change as long as the metadata vectors are available along with the solution vector. This also retains simplicity in our implementation, as the logic for ordering the data into the mesh does not need to be re-implemented.

It should be noted that HDF5 files have full support for use with the Message Passing Interface (MPI) [5]. They are able to store `Function` objects and load them with a differing number of MPI processes than were used to create the original file. This functionality is highly desirable to ensure the usability and performance of this format, and thus our TDF implementation fully supports this feature.

2.3. TDF plugin APIs

From a plugin perspective, the use of this file format for storing and loading FEM solution data should not add unacceptable overhead to the user compared with HDF5 provided by the DOLFIN library. Listing 1 below shows a snippet of C++ code that can be used to load a `Function` from a HDF5 file, then write that same `Function` to a different HDF5 file.

Listing 2 illustrates the interface of our TDF plugin with DOLFIN, and Listing 3 shows how an equivalent operation can be achieved using the TDF format. The primary read and write operations are extremely similar to the HDF5 interface. The main change being that while constructing the file object, instead of

Listing 1: HDF5 File Input and Output

```
1 Function u;
2 HDF5File in_file=HDF5File(mesh->mpi_comm(),"file.h5","r");
3 input_file.read(u);
4 HDF5File out_file=HDF5File(mesh->mpi_comm(),"outfile.h5","w");
5 output_file.write(u);
```

Listing 3: TDF File Input and Output

```
1 Function u;
2 TDFFile in_fh=TDFFile(mesh->mpi_comm(),"file.tdf","meta");
3 in_fh.cache_metadata = true; //enables caching of metadata
4 in_fh.read(u);
5
6 TDFFile out_fh=TDFFile(mesh->mpi_comm(),"out.tdf","out_meta");
7 out_fh.save_metadata = true; //only needed for first write
8 out_fh.write(u);
```

specifying if the object is in read or write mode, the metadata file prefix must be given. For simplicity of use, no explicit mode needs to be specified.

Listing 2: Function TDFFile Interface

```
1 TDFFile::TDFFile(
2         MPI_Comm comm,
3         const std::string& name,
4         const std::string& mname = "" )
5
6 comm: The MPI communicator to be used.
7 name: The filename for the solution vector
8 mname: The prefix for the meta data vectors.
9        Note suffixes will be added automatically.
```

The primary difference between the usage of the file formats is on lines 3 and 6 between Listing 1 & 3. Line 3 enables the caching of metadata vectors that have been previously loaded. Once the first file is read, these vectors are stored in the object for later use, allowing the file to be loaded to be changed, and a new Function to be loaded without reloading the metadata vectors. Line 6 enables the storage of the metadata vectors. Leaving the `save_metadata` option set to false will cause the object to only store the Function values. This feature allows for great improvements to the speed of storing FEM data.

3. Illustrative example

3.1. FEM based thermal simulation on CPUs

With the rapid miniaturization of integrated circuits (ICs) in the last several decades, high-performance microprocessors are becoming more thermally constrained due to the increasing power density [6,7]. High temperature could seriously degrade the microprocessor performance and reliability [8]. Thermal management and thermal-aware exploration are the effective ways to decrease the possibility of CPU failure and improve processor performance, but accurate prediction of the thermal distribution in the processor is needed [9,10]. The FEM is one of the most accurate approaches for thermal simulation of semiconductor chips. In this case, the heat transfer equation (1) is solved by the FEM to predict the temperature of the CPU, and it has been implemented on the FEniCS platform.

$$\rho C \frac{\partial T(\vec{r},t)}{\partial t} = \nabla \cdot k\nabla T + P_d(\vec{r},t) \tag{1}$$

Table 3
One time-step file sizes.

Mesh	$129 \times 129 \times 14$	$256 \times 256 \times 14$	$512 \times 512 \times 14$
TDF	1.8 MB	7.1 MB	29 MB
HDF5	41.4 MB	163.1 MB	652 MB

where k, ρ and C are the thermal conductivity, density and specific heat, respectively. $P_d(\vec{r},t)$ is the interior power density.

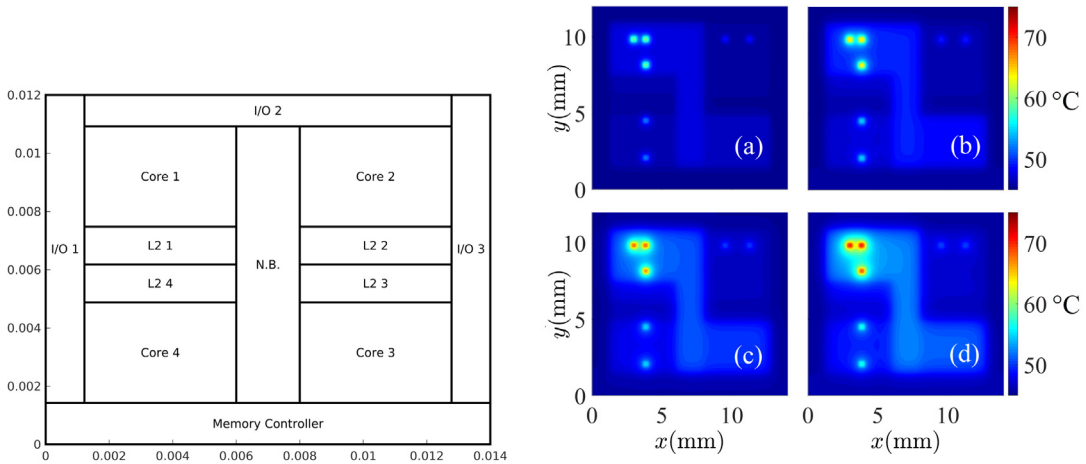
A quad-core CPU, AMD Athlon II X4 610e [11], is selected in this example to demonstrate the thermal simulation model in the FEM. As shown in Fig. 1(a) for the floorplan, this quad-core processor includes the following units: four 512 KB L2 caches, a Northbridge in the center and I/O and DDR3 placed around the periphery. The simulation domain of this processor covers a volume of 14 mm $\times$ 12 mm $\times$ 650 μm in the x, y and z directions, respectively. All the surfaces of the chip, except for the bottom surface, are assumed adiabatic. Heat dissipation from the bottom of each unit to the ambient is modeled by a convective boundary condition. The dynamic power in each unit applied in the DNS is calculated using McPAT [12] based on statistics information for each time step as generated by gem5 [13]. Using these power traces, DNS is performed to calculate the temperature of the CPU at each time step. Some snapshots of the dynamic temperature maps are shown in Fig. 1(b), where a mesh as fine as 512×512 on the xy plane is needed to capture many high-temperature small-diameter hot spots.

However, FEM requires a large number degrees of freedom (DoF), and therefore require extensive computational time and storage resources to provide detailed temperature distributions. It could be calculated that the storage size required for 20,000 time steps is around 12 terabyte if the mesh of $512 \times 512 \times 14$ is used.

3.2. Performance evaluation

This evaluation was done on a Dell Precision 7810 with two Intel Xeon E5-2683v4 CPUs, 32 GB DDR3 RAM, and a 256 GB Solid-State Drive. The OS used is Ubuntu 18.04.6 LTS with Linux kernel 5.4.0. Table 3 shows the sizes when the data is saved to disk as the TDF and HDF5 format respectively. The comparison shows our TDF plugin offers a $\approx$95% space savings in total by eliminating redundant mesh data in HDF5 files.

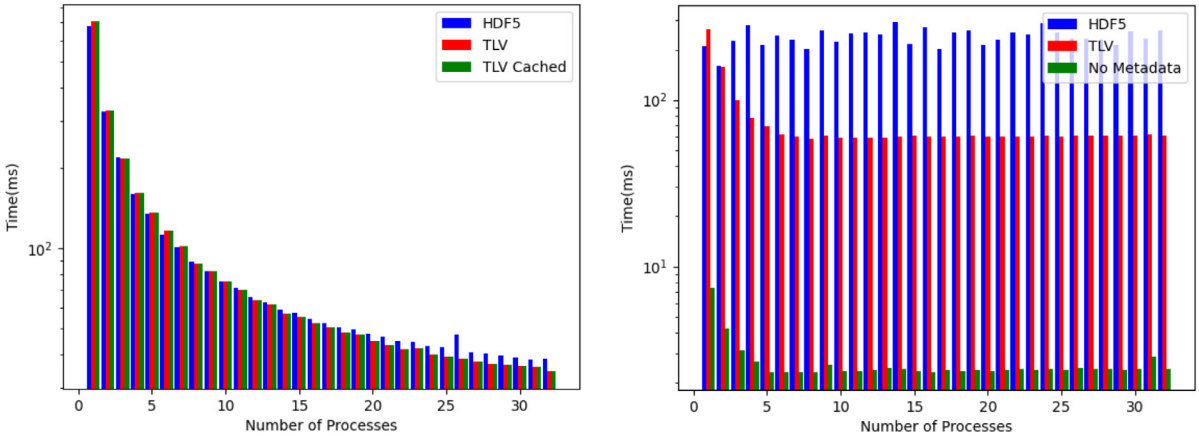
The timing comparison of the file types is the time to store and load a `Function` object, which is shown in Fig. 2. The latency to read a `Function` in the TDF format is very similar



(a) AMD Athlon II X4 610e floorplan

(b) AMD CPU heat maps at certain time steps as calculated by FEM simulation with varying power distributions

Fig. 1. Thermal simulation of the AMD Athlon CPU.



(a) Time to read FEM function

(b) Time to write FEM function

Fig. 2. I/O time comparison between HDF5 and TDF format.

to HDF5. This is likely due to overhead from ordering the data points into each `Function` object. However, the write time is greatly improved with the new format. Further improvement is achieved when only the `Function` values are stored without the metadata. This optimization allows for eliminating a great deal of time needed when performing the FEM simulation. The time to load a `Function` is currently dominated by the ordering of the values into the mesh, but on a machine restricted by the disk, such as a Hard Disk Drive, or when a larger mesh is used, the TDF format with cached metadata will likely outperform HDF5, as shown by the slight improvement over HDF5 when more MPI processes are used. The read time is CPU-bound, as the values have to be ordered into the mesh, so the use of more processes leads to a speedup of that operation. Write time, however, is disk-bound, meaning more processors can speed that operation up, but once the limit of the disk is reached, no further speedup will occur. Overall, the read/write performance has been improved by 4.9% and 70.1% respectively in the default configuration; enabling caching optimizations improves performance by 4.9% and 98.9% respectively. For this evaluation, the default HDF5 configuration of FEniCS was used.

TDF caching can further improve the reading performance shown in Fig. 2. The idea behind the TDF caching is that since the metadata vectors do not change, they can be kept in memory for use when loading other solution vectors. This is very useful when loading multiple time steps of data. Once the metadata is loaded, it just does not get loaded again, it is kept in vector objects in the TDF file object, then for subsequent time steps the same metadata vectors are used to order the solution into the mesh. This makes it so the metadata vectors are only loaded once, no matter how many time steps of data are loaded.

4. Impact

FEniCS is a very popular open-source FEM platform to solve PDEs, which enables users to model their problems with efficient finite element codes. HDF5 is a standard data format for FEniCS, but HDF5 files can become prohibitively large for FEM simulations that require high resolution results. The illustrated example in this paper has shown the enormous storage requested if the HDF5 format is used. Our TDF plugin aims to store the FEM data obtained from FEniCS simulation in a more effective manner, which

does not harm the I/O performance. With the assistance of this novel data format, the storage issue of large-scale FEM simulation can be mitigated for certain scenarios. Our TDF plugin for FEniCS can also be readily expanded to store more generic mathematical data generated by engineering and scientific simulations due to the flexibility of the TLV encoding scheme. Also, we provide the source code and building instructions of the TDF plugin implemented in FEniCS to the research community using numerical simulation through a public release on Github [14], which enables researchers to adapt our design and implementation for use in other open source packages.

5. Conclusion

The popular HDF5 format in the open source software FEniCS has unacceptable restrictions in terms of its usability, potential write speed, and required storage space for the FEM simulation to solve certain engineering and scientific applications. This is alleviated through the creation of an alternative binary file format, TDF, which uses a proposed TLV encoding to store data. Furthermore, through careful optimization of the use of the metadata needed to load the FEM solution data, the requirements for disk space and I/O operations can be greatly improved. With FEniCS, our evaluation of an FEM based thermal simulation case indicates that the proposed TDF format could lead to a ≈95% space savings in total, while maintaining or exceeding the I/O performance of HDF5. In addition, the API provided for use of the HDF5 file format in FEniCS is restrictive to the user and does not allow fine-grained control over the file operation. The new TDF format allows full control over key operations, allowing the user to make use of the data as needed.

Future work includes the completion of the expansion of the format to enable the storage of matrix data for use with other mathematical programs besides the FEM solutions. Also, operation with multiple TLV fields being stored in a single file may be investigated, which would enable the storage of all metadata vectors in a single file. Furthermore, creation of a plugin for popular open-source visualization tools would enable simplified viewing of solutions stored as TDF.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Yu Liu reports financial support was provided by National Science Foundation. Yu Liu reports a relationship with National Science Foundation that includes: funding grants.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by National Science Foundation under Grant No. ECCS-2003307 & OAC-2118079.

Data availability

No data was used for the research described in the article.

References

[1] The HDF group homepage. 2022, https://www.hdfgroup.org/ Last Visited: Nov. 5, 2022.
[2] FEniCS project. 2022, https://fenicsproject.org/, Last Visited: Nov. 5, 2022.
[3] RFC5246 - TLS. 2022, https://datatracker.ietf.org/doc/rfc5246/, Last Visited: Nov. 5, 2022.
[4] Apple Inc. Quicktime file format specification. 2001, https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html.
[5] MPI forum. 2022, https://www.mpi-forum.org/, Last Visited: Nov. 5, 2022.
[6] Jin W, Sadiqbatcha S, et al. Full-chip thermal map estimation for commercial multi-core CPUs with generative adversarial learning. In: Proc. ICCAD. 2020, p. 1–9.
[7] Guggari SI. Analysis of thermal performance metrics—Application to CPU cooling in HPC servers. IEEE Trans Compon Packag Manuf Technol 2021;11(2):222–32. http://dx.doi.org/10.1109/TCPMT.2020.3029940.
[8] Zhou X, Xu Y, et al. Thermal management for 3D processors via task scheduling. In: Proc. ICPP. 2008, p. 115–22.
[9] Jiang L, Liu Y, Cheng M-C. An effective and accurate data-driven approach for thermal simulation of CPUs. In: 2021 20th IEEE intersociety conference on thermal and thermomechanical phenomena in electronic systems (iTherm). 2021, p. 1008–14. http://dx.doi.org/10.1109/ITherm51669.2021.9503183.
[10] Jiang L, Liu Y, Cheng M-C. Fast accurate full-chip dynamic thermal simulation with fine resolution enabled by a learning method. IEEE Trans Comput-Aided Des ICs Syst 2022. http://dx.doi.org/10.1109/TCAD.2022.3229598.
[11] CPU-World. AMD athlon ii X4 610 CPU. 2022, https://www.cpu-world.com/CPUs/K10/AMD-Athlon%20II%20X4%20610e%20-%20AD610EHDK42GM%20(AD610EHDGMBOX).html, Last Visited: Nov. 5, 2022.
[12] Li S, Ahn JH, Strong RD, Brockman JB, Tullsen DM, Jouppi NP. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture. MICRO 42, New York, NY, USA: Association for Computing Machinery; 2009, p. 469–80.
[13] Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, et al. The gem5 simulator. ACM SIGARCH Comput Archit News 2011;39(2):1–7.
[14] FEniCS TDF plugin code repository on GitHub. 2023, https://github.com/dowlinah/tdf, Last Visited: Jan. 27, 2023.