THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 14, Number 4/2013, pp. 385–392

TRAFFIC REPLAY COMPRESSION (TRC): A HIGHLY EFFICIENT METHOD FOR HANDLING PARALLEL NUMERICAL SIMULATION DATA

Alin ANTON[1], Vladimir CREȚU[1], Albert RUPRECHT[2], Sebastian MUNTEAN[3]

[1] "Politehnica" University of Timisoara, Faculty of Automatic Control and Computing,
2nd Vasile Pârvan., 300223, Timisoara, Romania
[2] Universität Stuttgart, Institut für Strömungsmechanik und Hydraulische Strömungsmaschinen,
10 Pfaffenwaldring, 70550 Stuttgart, Germany
[3] Romanian Academy – Timisoara Branch, Center for Advanced Research in Engineering Science,
24th Mihai Viteazu, 300223, Timisoara, Romania
E-mail: alin.anton@cs.upt.ro

Numerical simulations produce large amounts of data. A key challenge is how to cope with the resulting bottlenecks. This paper introduces a highly efficient data handling method using parallel inter-processor traffic, called TRC. It is used to reconstruct the internal fields inside one or more regions of interest (ROIs) defined by the user. The method is implemented on top of OpenFOAM®, an opensource computational fluid dynamics (CFD) platform [1]. A well known CFD benchmark is used to demonstrate its efficiency in terms of the space savings obtained. The benchmark test-case is validated using experimental data from the literature. Several compression algorithms are applied to assess the efficiency of the method within a user-defined ROI, including ZIP and RAR [2, 3]. It is shown that in terms of space savings TRC has an efficiency of an order of magnitude higher, and that the efficiency improves as the size of the uncompressed data gets bigger. The space savings obtained with classic algorithms remain constant for more than 60 Gb of floating point data. The obtained results bring new perspectives to attention regarding data handling techniques for parallel numerical simulations.

Key words: numerical simulation, data handling, inter-processor communication, computational fluid dynamics, OpenFOAM.

1. INTRODUCTION

Modern day engineering cannot be conceived without the extensive use of computers, and sometimes of computer simulations [4]. One of the hardest challenges of parallel high performance computing is how to cope with the "big data" phenomenon [5]. In particular, the results originating from parallel numerical simulations are often large volumes of floating point data which are tedious to move around and download. The present results are a continuation of the developments in [27].

Fig. 1 shows the generic organization of a HPC center [6]. The nodes with processing elements (PEs) do not own a significant amount of storage space – they rely instead on the scratch space area provided by a networked file system. Data on the scratch space has a limited quota and may get deleted automatically. Because of that, storage is delegated into specialized data warehouses, outside the HPC center. The volume of generated floating point data is so large, that even the high-speed internal networks connecting the PE nodes require compression at MPI level [7]. The external, slower connections are bound to several bottlenecks that appear during the data transfers. The most prominently perceived one happens at user side when the client downloads the results, as a consequence of Internet bandwidth limitations. Another bottleneck resides on the link between the gateway and the HPC facility, and finally the gateway itself is a bottleneck between the end-user and the HPC center.

All methods that can compress or reduce the size of the result data have an impact on all three levels of bottlenecks. However since the most prominent problem is the end-user being unable to download the results in a feasible manner, the subject of this paper is how to solve this specific problem.

After completing a scientific simulation, the user is faced with the option to either post-process the data in situ according to a very tight schedule, or to migrate it outside the HPC facility. Many attempts have been made to deal with these issues. For unsteady simulations, the first thing that comes to mind (and does not require additional preparations) is to subsample the results by skipping some of the time steps but this is not always possible – for instance when a form of principal component analysis called proper orthogonal decomposition (POD) is required [17].

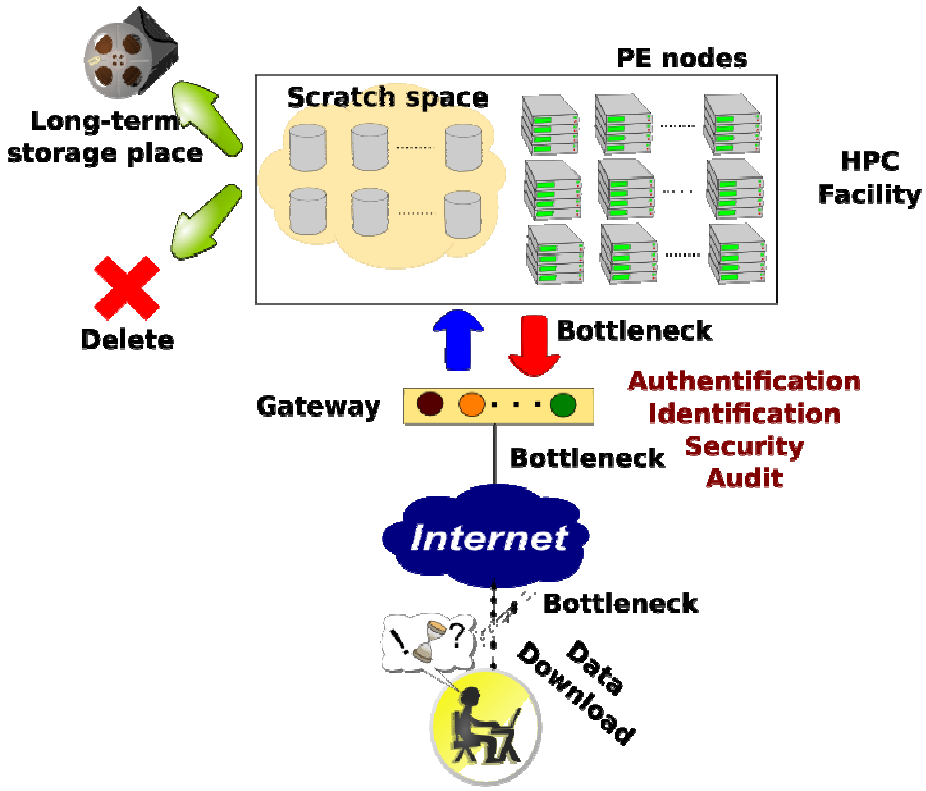

Fig. 1 – Bottlenecks when downloading big data.

A very specific family of lossless compressors has been designed for scientific floating point data sets [8–10]. They use bit predictors and dictionaries and deliver very high throughput. Whenever lossy compression is acceptable, the least significant bits can be discarded. A different method for obtaining lossy compression is to decompose the stream of numbers using wavelets [11, 12]. The less important components are ignored, and data can be progressively transmitted to a remote network. Another way to put this is by negotiating the regions of the mesh where it can be coarsened up to an acceptable degree. This is called adaptive coarsening (AC) [13].

Peer to peer techniques are used to move data outside the HPC facility [6] and to create distributed storage spaces for scientific data sets [14]. And finally feature extraction and tracking improve the visualization process with the help of artificial intelligence and statistical methods. The essential features of the phenomenon are compacted and transmitted without having the user to download the global fields [15].

There are, however, cases when the accuracy of the simulation data and the time granularity of the fields must be preserved. The paper addresses this problem by presenting a data reduction method based on the interception of parallel inter-processor traffic.

The next section describes our solution for CFD. The experimental setup used for demonstration is explained in Section 3, followed by results in Section 4. This section also contains a discussion on our findings and compares them with state of the art compression algorithms. It also explains the validation process for our test case. The conclusions are drawn in the last section.

2. METHOD

TRC takes advantage of local mesh refinement techniques [16]. In frequent cases of numerical simulation it is a common practice to simulate a larger analysis domain than what the user is really looking for. Consequently, a ROI can be defined in both space and time in order to capture the interesting data, but the exact boundary conditions for the ROI are not known – which is why the solution inside it cannot be obtained prior to computing the global simulation in the first place. TRC reconstructs the internal fields inside the ROI based on the exchange of communication between the ROI and the remaining sub-domain.

Consider the domain of analysis in Fig. 2 which is split into sub-domains S_1 and S_2 via domain decomposition. S_2 belongs to a space-time window covering the time interval from t_1 to t_2. Let Ω_1 be the mesh of S_1, and Ω_2 the mesh of S_2. The common boundary between S_1 and S_2 is defined by $\partial\Omega_1 = \partial\Omega_2$ and consists of cell faces with edges in 3D, or cell edges in 2D.

For simplicity in Fig. 2 it is considered that the parallel simulation involves two processor elements communicating via message passing. "Processor 0" (PE_0) and "Processor 1" (PE_1) are two independent, equally performing machines. Starting with simulation time t_1 and up to t_2 the exchange of MPI traffic from PE_1 towards PE_0 is intercepted in situ at OSI layer 7, and dumped into a storage file called "the traffic archive" which contains floating point numbers.

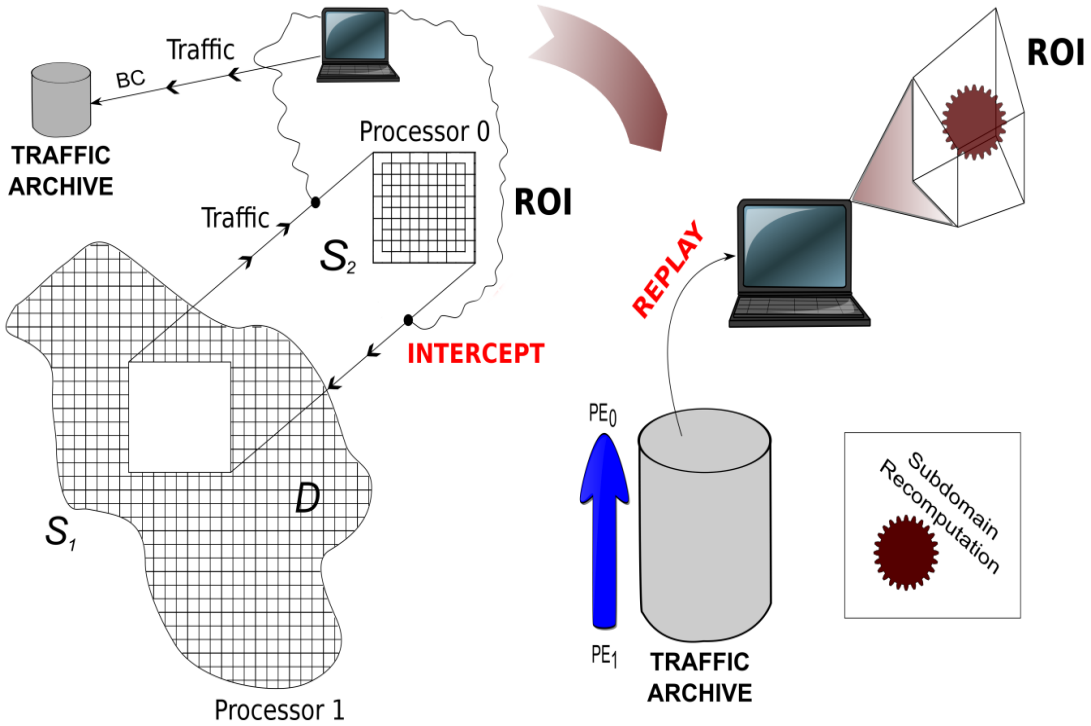

Fig. 2 – Traffic interception and later replay during the reconstruction of the fields.

The same numerical solver and configuration is used at client side to recover the information inside the ROI, by replaying the boundary data from the traffic file.

If TRC is used in conjunction with POD [17], an initial estimation for the efficiency of the method can be made. POD requires all the intermediary time steps for the ROI to be stored on disk without any sub-sampling. Given a cubical ROI with n cells in each direction one needs to store F bits of data on disk as given by equation (1) where S is the number of scalars in a cell, V the number of vectors, T the number of tensors and P is 32 or 64 depending on the precision used for storing the floating point values.

$$F = n \times (S + 3V + 9T) P. \tag{1}$$

In equation (2), C is an estimation of the number of bits communicated between t_1 and t_2, and K_{avg} is the average number of numerical iterations during each time step. S', V' and T' are the number of scalars, vectors and tensors transferred during the communication. Given that $C < F$, TRC compresses the data inside the ROI by C/F

$$C = K_{avg} \times 6n^2 \times \left(S' + 3V' + 9T'\right)P. \tag{2}$$

The method is best described by the following steps:

1. the global simulation is run on a coarse mesh until it starts producing interesting features at t_1
2. a ROI is defined and refined inside the global domain
   a. the simulation is continued for a few time steps to observe K_{avg};
   b. if $C/F \geq 1$ jump to step 2;
3. the overall domain is partitioned between the ROI and the remaining mesh;
4. the simulation from t_1 to t_2 is continued, and the inter-processor traffic is intercepted;
5. the internal fields inside the ROI are reconstructed by replaying the intercepted traffic;

TRC is implemented on top of OpenFOAM®, an opensource CFD platform designed to enable users to develop their own numerical solvers, tools and experiments [1, 18]. Industry grade applications have been built on top of this software and are deployed in large HPC centers.

```
if (myProcNo() == 0) {
      s = ("mydump" + ss.str());
      fileName directory = "dumpdir";
      mkDir(directory);

      OFstream mystream(directory/s);
      (*this).writeEntry("", mystream);
      mystream.setEof();
}
```

Fig. 3 – Dumping of floating point values into the traffic archive.

In Fig. 3 the *fvPatchFields* class is modified in order to intercept the floating point numbers communicated during the parallel simulation run. These numbers are dumped into a directory called the *traffic archive.* The next section describes the test case which has been used for validation.

3. TEST CASE SETUP

A well known test-case is selected in order to demonstrate the method. The European Research Community on Flow, Turbulence and Combustion (ERCOFTAC) has designed a number of benchmarks which are well known and widely used for validating CFD software. The square cylinder benchmark shown in Fig. 4 is one of them. It is an unsteady simulation of a turbulent flow where experimental data is available [19]. The test is designed to monitor the flow around a rectangular cylinder and in particular to observe the features of the unsteady flow behind it.

The side of the square cylinder is $D = 0.04$ m with a wideness $W = 0.392$ m. It is located in a domain of the same wideness, with the length of $L = 1.36$ m and height $H = 0.56$ m. Point (0; 0; 0) of the coordinate system is in the center of the square cylinder on the surface of the viewpoint. The overall bounding box of the analysis domain, in meters, is given by (–0.4 –0.28 0) (0.96 0.28 0.392).

The continuity and momentum equations are written in differential form in equations (3) and (4), where $\mathbf{v}$ is flow velocity, ρ is fluid density, p is pressure, $\mathbf{T}$ is the stress tensor and $\mathbf{f}$ are the body forces.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \mathbf{v}\right) = 0, \tag{3}$$

$$\frac{\partial \left(\rho \mathbf{v}\right)}{\partial t} + \nabla \cdot \left(\rho \mathbf{v}\mathbf{v}\right) = \rho \mathbf{f} + \nabla \cdot \mathbf{T}. \tag{4}$$

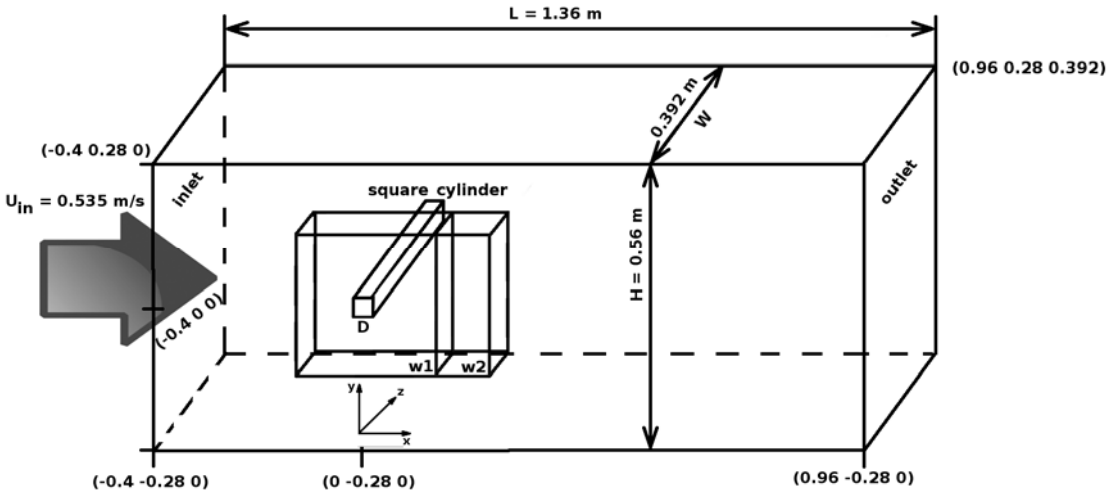

Fig. 4 – Computational domain and boundary conditions (BCs).

OpenFOAM® version 1.6 is used to perform three-dimensional turbulent unsteady numerical simulations. Large Eddy Simulations (LES) [20] are solved using a sub grid scale (SGS) model [21]. The pressure-velocity coupling algorithm is called PIMPLE, which is a mixture between the Pressure Implicit with Splitting of Operators (PISO) and Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) algorithms. The corresponding OpenFOAM® solver is pimpleFOAM [22].

Computational domain and boundary conditions (BCs) are shown in Fig. 4. Water enters the domain at the inlet with U_{in} = 0.535 m/s and 2% turbulence, and goes around the square cylinder forming vortices behind it. The Reynolds number computed with the side of the square cross section of the cylinder is Re = 21 400. Constant pressure is imposed at zero value on the outlet. The boundaries of the computational domain are modelled using slip conditions. The surfaces of the square cylinder are considered to be walls. Two sub-domain windows called w_1 and w_2 in Fig. 4 serve for two independent ROI cases. They are described by bounding boxes (–0.15 –0.15 0.05) (0.15 0.15 0.25) and (–0.15 –0.15 0.05) (0.25 0.15 0.25). These windows are used independently to test the reconstruction of the internal fields. The vortices that form behind the square cylinder are captured using w_1 and w_2 from t_1 = 6 s to t_2 = 7 s. The compression is measured by the compression ratio obtained when the traffic archive is used as a replacement for the internal fields of the ROIs covering the time period between t_1 and t_2.

4. RESULTS, VALIDATION AND DISCUSSION

The simulation starts with a global mesh of 200k cells. The mesh in w_1 and w_2 is refined on the order of 16 times the original internal resolution starting at t_1 = 6 s. This method is used to accelerate the convergence of the global solution. Experimental data from [19, 23] and a reference simulation having 3M cells are used to validate the computation. The drag coefficient C_d is a characteristic of the forces induced by the flow over the cylinder. It is compared against numerical data from the 3M cells benchmark, and against experimental data in [19, 23] where the averaged value for drag is C_d = 2.1 in Fig. 5. It can be seen that all of our numerical results are in agreement with experimental data.

The amplitudes for the drag signal are oscillating around the value of 2.1, in agreement with the experimental data. This means that the physics of the unsteady phenomenon in our simulations is captured correctly. Without mesh refinement, the average value for the drag coefficient on the 200k mesh stays below 2. The first 6 seconds are used in order to reach the unsteady phenomenon. Unsteady numerical simulations produce large amounts of result data, and it is common practice for the industry to compress these results. The proposed method is tested on the two ROIs w_1 and w_2 between t_1 = 6 s and t_2 = 7 s by capturing the inter-processor traffic between w_1 and w_2, at one hand, and the rest of the computational domain, on the other.

The efficiency of TRC is compared against state of the art algorithms and tools using Table 1, by compressing the internal fields of the ROI from t_1 = 6 s to t_2 = 7 s at all time steps Δt selected in our investigation. The compression effect is measured by compression ratio (*CR*) defined by equation (5) while

the saved space metric (S) is given by following equation $S = 1 - CR$. As a result, the algorithm efficiency is directly proportional with the saved space metric (S) and inversely proportional with the compression ratio (CR).

$$CR = \frac{C(\text{incomig traffic})}{F(\text{data inside the ROI})} \times 100. \tag{5}$$

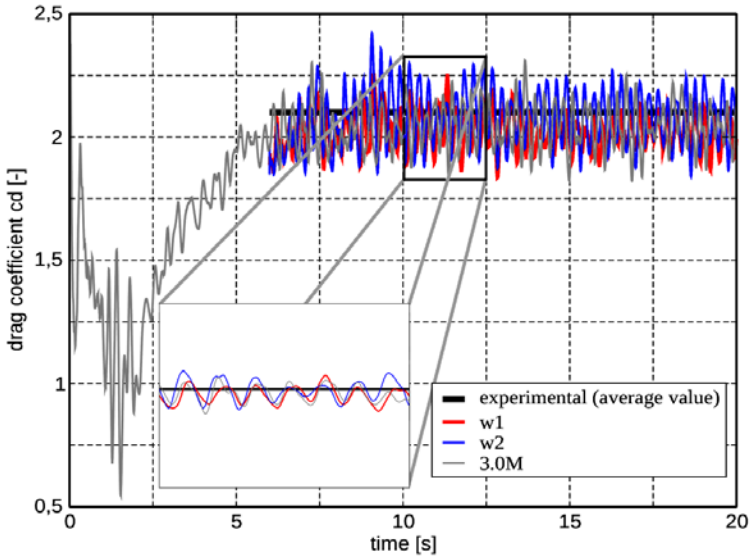

Fig. 5 – Validation of the numerical results against experimental data.

ZIP and GZIP are tools based on the Lemper-Ziv algorithm [2]. The main difference between them is that ZIP compresses collections of files independently of each other, even when there are many files in the same archive. GZIP on the other hand relies on external programs for archiving, like the popular unix tape archiver. This particularity enables it to produce better results than ZIP by exploiting the existing redundancy spread across multiple files. Because a unix ".tar" file was used for compression in both cases, ZIP and GZIP produce indistinguishably similar results in our tests. The other tools support their own version of file archives, but the difference in compression ratio is negligible. BZIP2 is based on the popular Partial Prediction Matching algorithm [24]. It is slower but usually yields better results than Lemper-Ziv. The 7Zip application uses Lemper-Ziv with Markov chains extension, and comes with a large number of compression schemes [25]. And finally RAR is a proprietary algorithm developed by Eugene Roshal - it combines Partial Prediction Matching [3] with a variation of the Lemper-Ziv algorithm published by Storer and Szymanski [26].

Table 1

Comparison TRC against different algorithms

	Δt (s)	Size (Gb)	(%)	**TRC**	BZIP2	GZIP	ZIP	7ZIP	RAR
w_1	1e-3	243	Compression ratio	18.7	97.11	94.65	94.65	88.47	90.12
			Saved spaced	**81.3**	2.89	5.35	5.35	11.53	9.88
w_2	1e-3	307	Compression ratio	16.6	97.06	94.78	94.78	88.92	90.22
			Saved spaced	**83.4**	2.94	5.22	5.22	11.18	9.18
	5e-3	62	Compression ratio	24.5	96.77	95.16	95.16	88.70	88.70
			Saved spaced	**75.5**	3.23	4.84	4.84	11.30	11.30
	1e-2	31	Compression ratio	37.0	96.77	96.77	96.77	90.32	90.32
			Saved spaced	**63.0**	3.23	3.23	3.23	9.68	9.68

Fig. 6 plots the data in Table 1 by connecting the saved space metric *S* with the size of the uncompressed data. It can be observed that the proposed method leads to a more efficient handling of the storage space. This happens because TRC has been designed with very specific scenarios in mind regarding how the result data is used.

The efficiency of TRC increases as the size of the uncompressed data grows. This trend gives TRC a clear advantage when large scale simulations are deployed. On the other hand, the efficiency of the generic algorithms shows a constant evolution regarding the space savings obtained. Their performance stales at 60 Gb of data. This happens because they have not been designed with very large volumes of data in mind, and consequently are not appropriate for parallel numerical simulations.

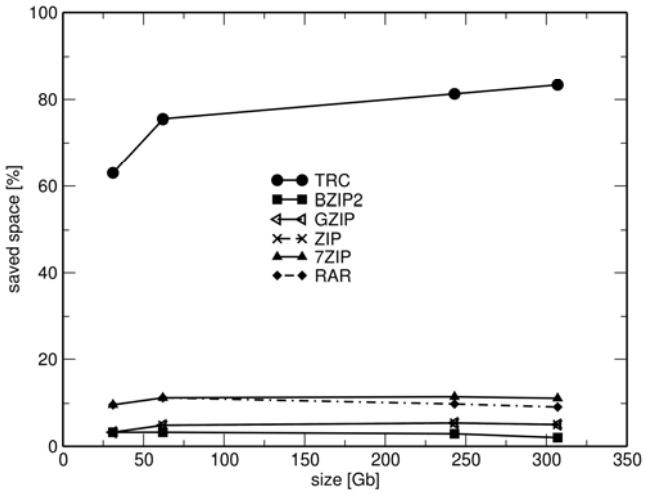

Fig. 6 – Space savings versus size of data.

5. CONCLUSION

This paper introduced a highly efficient method for data handling of parallel numerical simulation results. The topic is a key problem in modern high performance computing. TRC involves the interception of parallel inter-processor traffic into a storage file, and the later replay of this traffic in order to reconstruct the internal fields inside one or more ROIs defined by the user. The solution was applied on a well known CFD benchmark. Experimental data and a high resolution simulation were used for the validation of the numerical analysis. The implementation was developed on top of OpenFOAM®, an opensource CFD platform. Several compression algorithms were used on the same benchmark to assess the space savings efficiency of TRC (including ZIP [2] and RAR [3]). Our method called TRC delivers space savings within the range of 63–83.4%, an order of magnitude higher than the classic compression algorithms. This happens because TRC takes advantage of how the data was generated and what it is used for in the first place. The space savings obtained by the competitors stay below 10%. Another important finding is that the efficiency of TRC grows when the size of the uncompressed data gets bigger. This happens while the efficiency of the other algorithms stales to a constant limit. The obtained results bring new perspectives to attention regarding data handling techniques for parallel numerical simulations, and can be easily reproduced in other sciences.

ACKNOWLEDEMENTS

The authors thank Niklas Nordin who implemented the initial test-case in OpenFOAM® and made it publicly available. They also gratefully thank the bwGRiD project for the computational resources [28]. Dr. Sebastian Muntean was supported by the Romanian Academy program.

REFERENCES

1. Weller, H.G., Tabor, G., Jasak, H., Fureby, C., *A tensorial approach to computational continuum mechanics using object-oriented techniques*, Comput. Phys., **12**, *6*, pp. 620–631, 1998.
2. Ziv, J., Lempel, A., *A universal algorithm for sequential data compression*. IEEE Trans. Inf. Theor., **23**, *3*, pp. 337–343, 2006.
3. Shkarin, D., *PPMd – fast PPM compressor for textual data*; ftp://ftp.elf.stuba.sk/pub/pc/pack/ppmdh.rar 2001.
4. Benioff, R.M., Lazowska, D.E., *Computational Science: Ensuring Americas Competitiveness*, Report to the President of the United States, June 2005.

5. Kramer, W.T.C., Shoshani, A., Agarwal, D.A., Draney, B.R., Jin, G., Butler, G.F., Hules, J.A., *Deep scientific computing requires deep data*, IBM J. Res. Dev., **48**, *2*, pp. 209–232 (March 2004).
6. Monti, H.M., Butt, A.R., Vazhkudai, S.S., *Timely result-data off loading for improved HPC center scratch provisioning and serviceability*, IEEE Trans. Parallel Distrib. Syst., **22**, *8*, pp. 1307–1322, 2011.
7. Filgueira, R., Atkinson, M., Nuez, A., FERNNDEZ, J., *An adaptive, scalable, and portable technique for speeding up MPI-based applications*. In Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.), EuroPar2012 – Parallel Processing, Volume 7484 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 729–740.
8. Burtscher, M., Ratanaworabhan, P., *FPC: A high-speed compressor for double precision floating-point data*, IEEE Trans. Comput., **58**, *1*, pp. 18–31, 2009.
9. Burtscher, M., Ratanaworabhan, P., *gFPC: A self-tuning compression algorithm*, Proceedings of the 2010 Data Compression Conference. DCC '10, Washington, DC, USA, IEEE Computer Society, 2010, pp. 396–405.
10. O'Neil, M.A., Burtscher, M., *Floating-point data compression at 75 Gb/s on a GPU*, Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-4), New York, NY, USA, ACM, 2011, pp. 7:1–7.
11. Wilson, J., *Wavelet-based lossy compression of turbulence data*, Proceedings of the Conference on Data Compression (DCC '00), Washington, DC, USA, IEEE Computer Society, 2000, p. 578.
12. Wilson, J., *Wavelet-based lossy compression of barotropic turbulence simulation data*, Data Compression Conference, 2002, Proceedings. DCC 2002, p. 479.
13. Unat, D., Iii, T.H., Baden, S.B., *An adaptive sub-sampling method for in-memory compression of scientific data*, Proceedings of the 2009 Data Compression Conference, Washington, DC, USA, IEEE Computer Society, pp. 262–271, 2009.
14. Vazhkudai, S.S., Ma, X., Freeh, V.W., Strickland, J.W., Tammineedi, N., Simon, T., Scott, S.L., *Constructing collaborative desktop storage caches for large scientific datasets*, Trans. Storage, **2**, *3*, pp. 221-254, 2006.
15. Frits, H.P., Benjamin, V., Helwig, H., Robert, S.L., Helmut, D., *The state of the art in flow visualization: Feature extraction and tracking*, 2003.
16. Berger, M. J., Colella, P., *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys., **82**, *1*, pp. 64–84, 1989.
17. Rudolf, P., Jízdný, M., *Decomposition of the swirling flow fields*, Proceedings of the 4th IAHR International Meeting on Cavitation and Dynamic Problems in Hydraulic Machinery and Systems, Belgrade, Serbia, 2011, pp. 131–141.
18. Buntic-Ogor, I., Krappel, T., Kirschner, O., Ruprecht, A., *Numerical investigation of the swirling flow and the vortex control in a straight diffuser*, 5th OpenFOAM Workshop, Gothenburg, June 2010.
19. Lyn, D.A., Rodi, W., *The applying shear layer formed by flow separation from the forward corner of a square cylinder*, Journal of Fluid Mechanics, **267**, pp. 353–376, 1994.
20. Ruprecht, A., Helmrich, T., Buntic, I., *Very large eddy simulation for the prediction of unsteady vortex motion*, Proceedings of the 12th International Conference on Fluid Flow Technologies, Budapest, 2003.
21. De Villiers, E., *The Potential of Large Eddy Simulation for the Modeling of Wall Bounded Flows*, PhD Thesis, Imperial College of Science, Technology and Medicine, 2006.
22. Penttinen, O., *A pimplefoam tutorial for channel flow, with respect to different LES models*, Technical Report, Chalmers University of Technology, November 2011.
23. Lyn, D.A., Einav, S., Rodi, W., Park, J.H., *A laser-doppler velocimetry study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder*, Journal of Fluid Mechanics, **304**, pp. 285–319, 1995.
24. Cleary, J., Witten, I., *Data compression using adaptive coding and partial string matching*, IEEE Transactions on Communications, **32**, *4*, pp. 396–402, 1984.
25. Pavlov, I., *LZMA SDK (Software Development Kit)*; http://www.7- zip.org/sdk.html , 2007.
26. Storer, J.A., Szymanski, T.G., *Data compression via textual substitution*, J. ACM, **29**, *4*, pp. 928–951, 1982.
27. Anton, A., *Space-Time Window Reconstruction in Parallel High Performance Numeric Simulations. Application for CFD*, PhD Thesis, 2011.
28. *** bwGRiD: Member of the German D-Grid initiative, funded by the Ministry of Education and Research (Bundesministerium für Bildung und Forschung) and the Ministry for Science, Research and Arts Baden-Württemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg); http://www.bwgrid.de/; 2007–2010.

Received July 2, 2013