ESAIM: PROCEEDINGS, Vol. ?, 2013, 1-10
Editors: Will be set by the publisher

AN ASYNCHRONOUS WRITING METHOD FOR RESTART FILES IN THE GYSELA CODE IN PREVISION OF EXASCALE SYSTEMS *

O. Thomine[1], J. Bigot[2], V. Grandgirard[1], G. Latu[1], C. Passeron[1] and F. Rozar[2]

Abstract. The present work deals with an optimization procedure developed in the full-f global **GY**rokinetic **SE**mi-**LA**grangian code (GYSELA). Optimizing the writing of the restart files is necessary to reduce the computing impact of crashes. These files require a very large memory space, and particularly so for very large mesh sizes. The limited bandwidth of the data pipe between the computing nodes and the storage system induces a non-scalable part in the GYSELA code, which increases with the mesh size. Indeed the transfer time of RAM to data depends linearly on the files size. The necessity of non synchronized writing-in-file procedure is therefore crucial.

A new GYSELA module has been developed. This asynchronous procedure allows the frequent writing of the restart files, whilst preventing a severe slowing down due to the limited writing bandwidth. This method has been improved to generate a checksum control of the restart files, and automatically rerun the code in case of a crash for any cause.

Résumé. Ce travail concerne une procédure d'optimisation développé dans le code de calcul GYSELA (**GY**rokinetic **SE**mi-**LA**grangian). L'optimisation de l'écriture de fichiers de reprise est nécessaire afin de palier aux crashs lors du calcul. Ces fichiers de redémarrage nécessitent un très grand espace mémoire, et ce d'autant plus que le maillage utilisé est gros. La bande passante entre les noeuds de calcul et le système de stockage étant limité, cela induit une non-scalabilité des écritures, ce qui limite actuellement la fréquence d'écriture de ces fichiers. Cette non-scalabilité se révèle être d'autant plus importante que les fichiers sont gros. En effet, ce temps d'écriture (une fois le réseau de communication saturé) dépend linéairement de la taille des fichiers à écrire. La nécessité d'introduire une méthode d'écriture désynchronisée se révèle donc d'une grande importance.

Un nouveau module a été développé dans GYSELA. Cette procédure asynchrone permet une écriture très fréquente de ces fichiers de redémarrage, en limitant un ralentissement important dû à la limitation de la bande passante. Cette méthode a été améliorée en introduisant une somme de contrôle de ces fichiers de redémarrage, afin de contrôler leur intégrité si leur utilisation s'avère nécessaire, et permettant un redémarrage automatique de la simulation après un crash.

1. Introduction

Modeling turbulent transport is a major goal in order to predict confinement issues in a tokamak plasma. The gyrokinetic framework considers a computational domain in five dimensions which allows for kinetic issues to be investigated in a plasma. Gyrokinetic simulations lead to huge computational needs. Up to now, the

* *This work was partially supported by the G8-Exascale action NuFUSE contract. GYSELA computations were granted access to the HPC resources of Mésocentre d'Aix-Marseille Université, TGCC (Saclay/France) and IFERC (Rokkasho/Japan).*

[1] CEA Cadarache, FR-13108 Saint-Paul-les-Durance

[2] Maison de la simulation, CEA Saclay, FR-91191 Gif sur Yvette

gyrokinetic code GYSELA has performed large simulations using from a few hundred up to tens of thousands of cores.

Our gyrokinetic model considers as main unknown a distribution function $\bar{f}$ that represents the density of ions at a given phase space position. This function depends on time and on 5 other dimensions. First, 3 dimensions in space $\mathbf{x}_G = (r, \theta, \varphi)$ with r and θ the polar coordinates in the poloidal cross-section of the torus, while φ refers to the toroidal angle. Second, velocity space has two dimensions: $v_\parallel$ being the velocity along the magnetic field lines and μ the magnetic moment corresponding to the action variable associated with the gyrophase. Let us consider the gyro-center coordinate system $(\mathbf{x}_G, v_\parallel, \mu)$, then the non-linear time evolution of the 5D guiding-center distribution function $\bar{f}_t(r, \theta, \varphi, v_\parallel, \mu)$ is governed by the so-called gyrokinetic equation which reads [GBB+06, Hah88] in its conservative form:

$$B^*_{\parallel s}\frac{\partial \bar{f}}{\partial t} + \boldsymbol{\nabla}\cdot\left(B^*_{\parallel s}\frac{\mathrm{d}\mathbf{x}_G}{\mathrm{d}t}\bar{f}\right) + \frac{\partial}{\partial v_\parallel}\left(B^*_{\parallel s}\frac{\mathrm{d}v_\parallel}{\mathrm{d}t}\bar{f}\right) = B^*_{\parallel s}\left(\mathcal{D}_r(\bar{f}) + \mathcal{K}(\bar{f}) + \mathcal{C}(\bar{f}) + \mathcal{S}\right) \tag{1}$$

where $\mathcal{D}_r(\bar{f})$ and $\mathcal{K}_r(\bar{f})$ are respectively a diffusion term and a Krook operator applied on a radial buffer region, $\mathcal{C}(\bar{f})$ corresponds to a collision operator (see [DPDG+11] for more details) and $\mathcal{S}$ refers to source terms (detailed in [SGA+10]). The scalar $B^*_{\parallel s}$ corresponds to the volume element in guiding-center velocity space. The expressions of the gyro-center coordinates evolution $\mathrm{d}\mathbf{x}_G/\,\mathrm{d}t$ and $\mathrm{d}v_\parallel/\,\mathrm{d}t$ are not necessary. The useful information for this paper is that they depend on the 3D electrostatic potential $\Phi(\mathbf{x}_G)$ and its derivatives. In this Vlasov equation, μ acts as a parameter because it is an adiabatic motion invariant. Let us denote by N_μ the number of μ values. We have N_μ independent equations (Eq. 1) to solve at each time step. In the code, electrons are assumed adiabatic, i.e electron inertia is ignored. The electrostatic potential is determined by solving the self-consistent coupled quasi-neutrality equation

$$-\frac{1}{n_0(r)}\nabla_\perp\cdot\left[\frac{n_0(r)}{B_0}\nabla_\perp\Phi(r,\theta,\varphi)\right] + \frac{1}{T_e(r)}\left[\Phi(r,\theta,\varphi) - \langle\Phi\rangle_{\mathrm{FS}}(r)\right] | = \tilde{\rho}(r,\theta,\varphi) \tag{2}$$

The r.h.s of Eq. (2) is defined as $\tilde{\rho} = \frac{2\pi}{n_0(r)}\int B(r,\theta)d\mu \int \mathrm{d}v_\parallel J_0(k_\perp\sqrt{2\mu})(\bar{f} - \bar{f}_{eq})$ with $\bar{f}_{eq}$ representing a local ion Maxwellian equilibrium. The perpendicular operator $\nabla_\perp$ is defined as $\nabla_\perp = (\partial_r, \frac{1}{r}\partial_\theta)$. $B(r,\theta)$ represents the magnetic field with B_0 being its value at the magnetic axis. The radial profiles $n_0(r)$ and $T_e(r)$ correspond respectively to the equilibrium density and the electron temperature. J_0 which denotes the Bessel function of first order reproduces the gyro-average operation in Fourier space, $k_\perp$ being the transverse component of the wave vector. $\langle\cdot\rangle_{\mathrm{FS}}$ denotes the flux surface average defined as $\langle\cdot\rangle_{\mathrm{FS}} = \int\cdot\mathcal{J}_{\mathrm{x}}\,\mathrm{d}\theta\,\mathrm{d}\varphi/\int\mathcal{J}_{\mathrm{x}}\,\mathrm{d}\theta\,\mathrm{d}\varphi$ with $\mathcal{J}_{\mathrm{x}}$ the jacobian in space of the system.

Therefore, GYSELA code is a global nonlinear electrostatic code which solves the gyro-kinetic equation in a five dimension phase space with a semi-Lagrangian method [GBB+06, GSG+08]. The Semi-Lagrangian time integration technique [SRBG99] couples the Lagrangian and Eulerian points of view. The main advantage offered by the semi-Lagrangian technique is that time steps are not restricted by the CFL condition. We combine this scheme with a second order in time Strang splitting method.

The code is written in Fortran 90 and parallelized by using an hybrid MPI/OpenMP paradigm. Large data structures are used in Vlasov and quasi-neutrality solver: the 5D data $\bar{f}$, and the electric potential Φ which is a 3D data structure along with some derivatives of the gyro-average of Φ. Let N_r, N_θ, N_φ, $N_{v_{||}}$, N_μ be respectively the number of points in each dimension r, θ, φ, $v_{||}$, μ, therefore the size of 5D and 3D data are $(N_r N_\theta N_\varphi N_{v_{||}} N_\mu)$ and $(N_r N_\theta N_\varphi)$.

With the continuous increase of the number of computers nodes, the crash probability of one or several nodes will increase in the future. The origins of these crashes have been studied during the last few years at the Los Alamos Laboratory reliability data sets (http://institutes.lanl.gov/data/fdata/). These crashes induce minor or major problems depending on the seriousness of the crash. Minor seriousness errors are frequently produced, approximatively every 8 hours on the 100 000-nodes Blue Gene computer [GCG+07]. A severe crash on the

other hand (inducing a code interruption), is produced every 7 to 10 days on the same computer [GCG$^+$07]. In Exascale prevision, the mean computing time between two successive severe crashes will be reduced from 3 to 26 minutes [SG07, VS09]. If the crash probability is too high, the numerical codes need to be resilient enough to recover the simulation after each crash. A compromise is needed between the time spent for restart file saving and the time spent in computing. If the restart files are saved too often, the time spent in restart file writing will impact the computation time, because of the interaction of the writing procedures and computing threads. Another major problem is the bandwidth between the storage array and the processing units. The size of the data to save increases with the mesh resolution. For large to very large data files, the bandwidth will decelerate the computation. Indeed, each node will wait for the end of the writing process before continuing the computation.

An asynchronous method for writing restart files has been implemented to tackle this problem. A checksum of the data array has been saved in the restart files to be able to detect if a crash happened during the saving and corrupted data. We will see that the asynchronous method induces computing and memory overheads. We will estimate this overhead and define the strategy to adopt in function of the file size to be written. A scientific computing dedicated study has been performed by Sancho [SPJ04] which studies the feasibility of incremental check-pointing. Due to the data size needed for this method, we cannot use it for GYSELA. The incremental method needs a reference-field, which is in our case has a size of some TB. Another kind of check-pointing method consists of mirror checkpointing. This checkpointing algorithm consists in replicating the data contained on a node to a neighbor node. The major problem of this algorithm is the assumption that only one node can crash [CD96]. If two nodes (one containing the original data and the other containing the saved data) crash simultaneously, the failure cannot be recovered.

Several libraries have been developed to improve this writing feature. *DejaVu* [RHV07] is one of the most well known asynchronous libraries. However, we do not use this library for the following two reasons which *DejaVu* does not allow. Alternating between two set of distinct datas allows us to be sure that at least one set of restart files is able to be used to restart the simulation. An other reason is to use the HDF5 library capabilities, like chunk decomposition of domain for very large fields ($> 4GB$). A specific routine, who alternates writing between two distinct set of restart files, and with HDF5 file support, has been developed specifically for GYSELA.

2. Crash frequency on new architecture

Increasing the number of nodes used for a parallel simulation induces an increase of the crash probability. The crash probability of one node is given by a Weibull law, with a mean lifetime for a node of $\lambda = 4$ years, and an exponential value $k = 0.78$ [SG06]. This cumulated probability is:

$$P(t) = 1 - exp(-(t/\lambda)^k) \ . \tag{3}$$

In the case of N nodes, this probability reads:

$$P_N(t) = 1 - (exp(-(t/\lambda)^k))^N \ . \tag{4}$$

The mean computing time between 2 crashes with a success probability of $s \in [0, 1[$ is given in equation (5) and is shown on figure 1 for a probability of success of 10%.

$$\bar{\tau}_s(N) = \lambda \left(ln \left(\frac{1}{1-s} \right) \frac{1}{N} \right)^{1/k} \ . \tag{5}$$

For a large number of nodes $N = 10^5$, this mean time $\bar{\tau}_s$ is of the order of the minute. For this reason, an upgrade of GYSELA that allows fault tolerance is very important.

Until now GYSELA used to write restart files only at the end of each run. If a crash appeared during the simulation, the whole run had to be re-computed. A first solution is to write many restart files while GYSELA

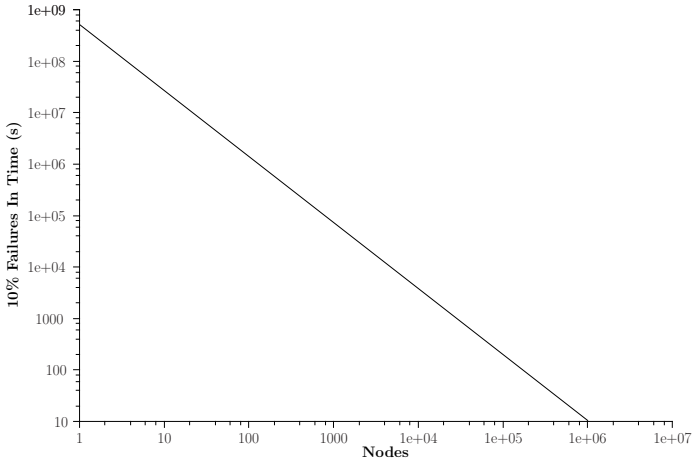

FIGURE 1. Evolution of mean computing time with a probability of success of 10% in function of the number of nodes.

is running. With this procedure, when a crash occurs, the time that is lost is of the order of the time between two restart files savings. The checksum integrated in each restart file allows to make sure of their integrity. In this case, a new run with these restart files is launched, as shown on Fig. 2. To prevent issues during writing (file system failure for example), the initial restart file is not erased to preserve at least one non-corrupted file. In the case of a crash during restart file writing, the files will be corrupted. The next run cannot use these files as initialization data. So the last run has to be completely re-run, as shown on Fig. 3.

3. The asynchronous method used in the GYSELA code

3.1. **Synchronous and asynchronous methods**

Until now, the restart files were written synchronously and used the HDF5 library due to its accessibility to the data stored. The GYSELA code waits for the end of this writing procedure before it resumes computing. This waiting time is lost CPU time. For very large files to write, the delay due to the data transfers from local memory to file system is a big penalty.

The method used to desynchronize this writing procedure consists in launching a new thread dedicated to the data writing. During this file writing, the computing can continue. The major problem due to this method is the concurrent access to the memory containing the fields to write, between the computing subroutines and the writing thread activity. To prevent these concurrent accesses, a copy of the fields to be written is necessary. This copy is the overhead of this method: CPU time to copy the original fields to temporary fields, and memory needed to contain these fields. This overhead has been estimated and the results are shown in section 4. We show that the overhead of field copy before writing is of the order of 0.3% of the total simulation time. The major problem of this field copy is the memory needed for this operation. Most of the memory needed for each run is taken by the 5D field. We need two times more memory to keep a copy of this field.

3.2. **Optimizing the number of restart file savings**

One can deduce the optimal number of restart files during each simulation to minimize the average time which is lost due to asynchronous method. Let us suppose that the crash probability during an entire simulation is $\mathcal{P}$ (depending on the mean node lifetime, the number of nodes and the simulation time). Let T_e be the time

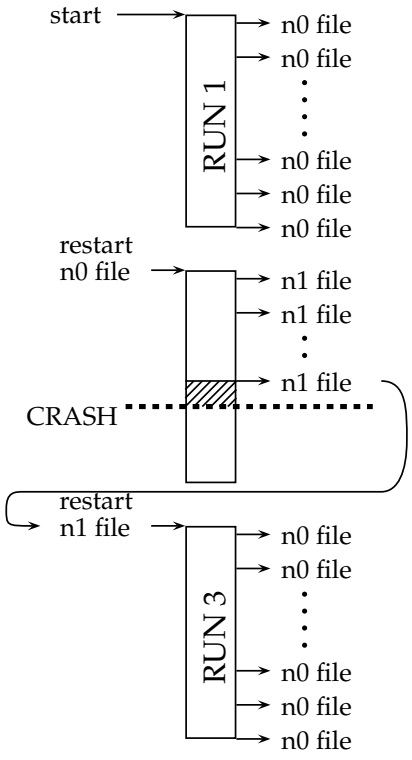

FIGURE 2. Crash during computation, restart files integrity checked.

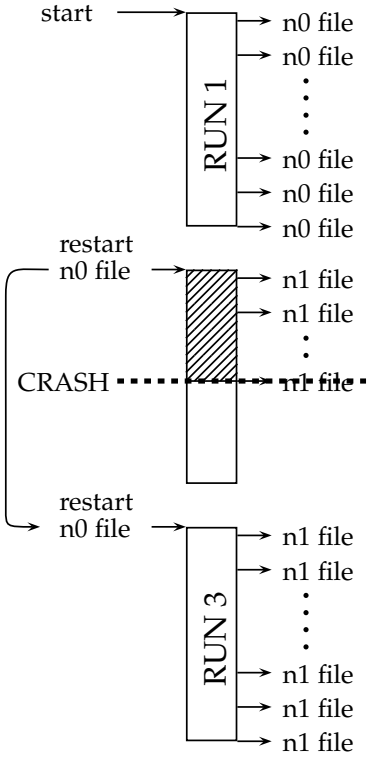

FIGURE 3. Crash during writing, restart files integrity unchecked.

needed to write one restart file, N_e is the parameter to optimize, namely the number of restart writing files in one run, and α a coefficient ($\in [0,1]$) which determines the CPU load needed for the writing procedure (in particular to compress and copy the fields to the temporary memory). We can consider that this parameter is equal to unity for synchronous writing. In one run, the time between two saving procedures is $\Delta T_e = T_r/N_e$, with the duration of one run being: $T_r = T_c + \alpha T_e N_e$. We can deduce the crash probability in a run, during the files writing with the asynchronous method as: $\mathcal{P}_W^{as} = \mathcal{P} N_e T_e / T_r$, and out of the restart files writing as: $\mathcal{P}_{NW}^{as} = \mathcal{P}(1 - N_e T_e / T_r)$. The mean time lost in this case is $T_r/2$ (Fig. 3). In the other case, i.e. when the crash does not occur during the restart file writing, the lost time is on average: $\Delta T_e/2$ (Fig. 2). We can thus deduce the mean simulation time $\bar{\tau}_s$ in function of N_e:

$$\bar{\tau} = \underbrace{N_t T_c}_{\substack{\text{Simulation} \\ \text{CPU time}}} + \underbrace{\mathcal{P}_W^{as} \frac{T_r}{2}}_{\substack{\text{lost time} \\ \text{due to crash} \\ \text{during write}}} + \underbrace{\mathcal{P}_{NW}^{as} \frac{T_r}{N_e}}_{\substack{\text{lost time due} \\ \text{to crash in-between} \\ \text{writings}}} + \underbrace{N_e N_t \alpha T_e}_{\substack{\text{CPU} \\ \text{overhead}}} , \tag{6}$$

with N_t the number of runs during the simulation. We can easily show that the lowest value of $\bar{\tau}$ is obtained for the following value of N_e :

$$N_e = \sqrt{\frac{\mathcal{P} T_c}{T_e (2\alpha N_t + \mathcal{P})}} . \tag{7}$$

Name	Mesh size	Number of nodes	Number of processors	Total file size
SIM_1	2.1 G	16	256	18.0 GB
SIM_2	4.3 G	32	512	36.0 GB
SIM_3	8.5 G	64	1024	72.0 GB
SIM_4	17.0 G	128	2048	143.9 GB
SIM_5	36.1 G	256	4096	287.8 GB
SIM_6	68.2 G	512	8192	575.7 GB
SIM_7	136.4 G	1024	16384	1.1 TB
SIM_8	272.7 G	2048	32768	2.3 TB

TABLE 1. Simulations parameters used for GYSELA weak scaling.

Including the previous result to Eq. (6) gives the minimal simulation time $\bar{\tau}_{\text{MIN}}$ in function of N_t, T_c, T_e, $\mathcal{P}$ and α. For a unique simulation, i.e. for N_t, T_c, T_e, $\mathcal{P}$ fixed, we can show that $\bar{\tau}_{\text{MIN}}$ increases when α increases. With $\alpha = 1$, which corresponds to the synchronous method, $\bar{\tau}_{\text{MIN}}$ is maximum, as expected.

4. Profiling results

To compare the saved time between synchronous and asynchronous methods, weak-scaling simulations have been performed on the HELIOS machine at CSC, Rokkasho, Japan. This Bullx B510 computer contains 70560 thin cores, which constitutes a 1.52 PFlops computing power. The bandwidth of data transfer towards the file system is about 20 GB/s. In our case (weak scaling), each node contains the same number of points. Using this scaling method allows us to increase the writing time, while keeping the same mesh point per node ($N_R = 128$, $N_\theta = 128$, $N_\phi = 128$, $N_{v//} = 64$, $N_\mu = 1$, witch correspond to 134 M of points per node). The mesh size depending on the number of node is N_μ. The uncompressed restart file size is 1.12 GB. The simulation parameters are given in Table 1. Each MPI process is deployed on one node, the OpenMP parallelization allows each MPI process to use the 16 cores within the nodes. Three sets of simulations have been performed: one with the original synchronous method, one with the new asynchronous method, and one with no restart file writing.

Each simulation contains 10 iterations, and saves the restart files every 4 iterations, meaning twice considered per simulation (for the synchronous and asynchronous cases, the last save is synchronous). The total data to copy from node memory to storage bay scales from 18 GB for 16 MPI processes to 2.3 TB for 2098 MPI processes. The computation time for each simulation is shown on Fig. 4.

We can see in Fig. 4 that multiplying by 128 the number of nodes induces a total time increase of the order of 55% if no restart file writing is asked, 55% if the asynchronous method is used and 115% with the synchronous method. These encouraging results show that the asynchronous method scales as well that without restart files writing. The relative efficiency of *GYSELA* is shown in Fig. 5.

As we saw before, the asynchronous method induces an overhead due to the duplication of the data to write in addition to the thread dedicated to the writing. To estimate this overhead, the total time needed to copy the data has been estimated. These overheads are shown in Fig. 6. The time needed for the 5D-field copy is constant, which induce a perfect relative efficiency of this part. The time needed to launch the thread is constant too, and its impact is minor, compared to the copying procedure.

For the above cases, the overhead required for the synchronous writing, the asynchronous writing and the no writing cases are similar. However, when we increase the size of the data to write (to approximatively 1 TB), the bandwidth saturates. We can then see an important increase in the time needed to write the data. As the computing can continue in the asynchronous case while it cannot in the synchronous case, the total computing case strongly increases in the synchronous case, but not in the asynchronous case, as shown in Fig. 4.

However, as the bandwidth needed for the asynchronous writing is shared with the MPI communication, we can suppose that the bandwidth will not be optimal. We can see in Fig. 7 that we obtain a bandwidth of the

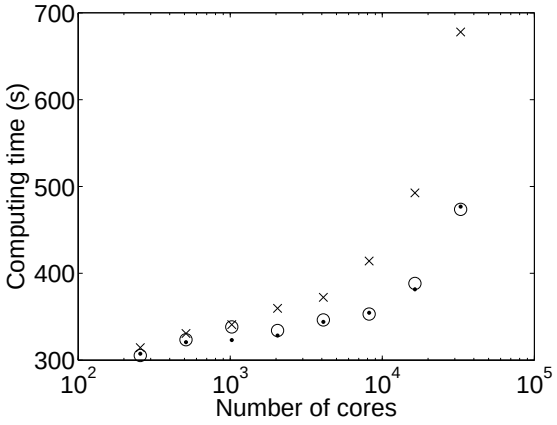

FIGURE 4. Total simulation time with synchronous (crosses), asynchronous (circles) and no writing (points) methods in function of number of cores.

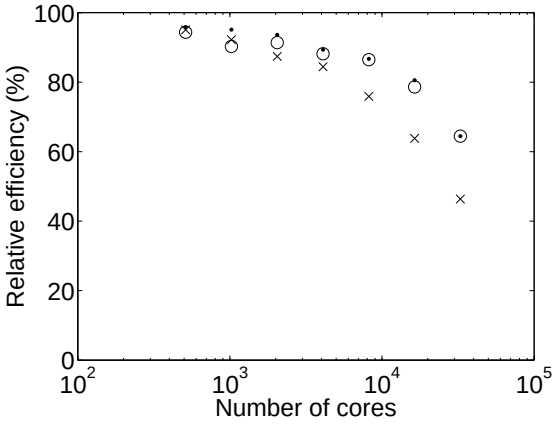

FIGURE 5. Relative efficiency obtained with synchronous (crosses), asynchronous (circles) and no writing (points) in function of number of cores.

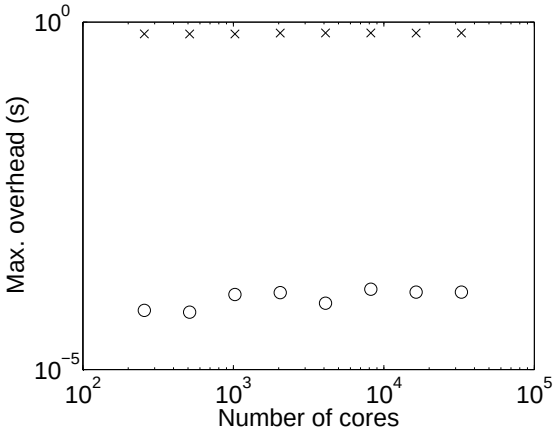

FIGURE 6. Overhead obtained with asynchronous method: 5D field copy (crosses) and thread launching (circles) in function of number of cores.

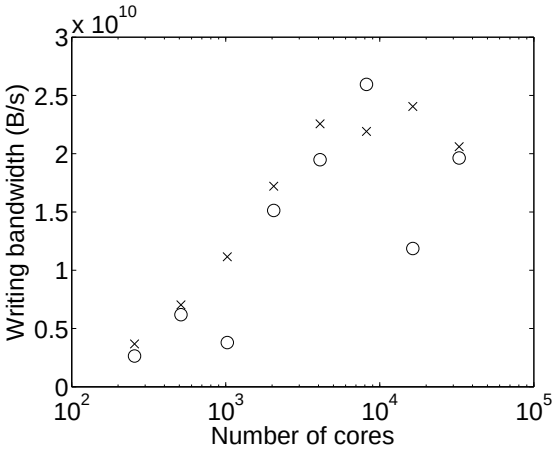

FIGURE 7. Bandwidth obtained with synchronous (crosses), asynchronous (circles) in function of number of cores.

same order of the asynchronous case in the synchronous case. This result denotes a low impact of file writing communication and MPI communications.

These comparisons have been made with a high frequency writing of the restart files. The optimum number of restart files to save during each simulation is detailed in section 3.2. In our case, this optimal number is of course near zero instead of 2, due to very low probability of a crash on this low number of nodes.

Writing method	Number of cores	Total file size	Max time (s)	Bandwidth
synchronous	256	$18.0GB$	5.23	3.7 GB/s
synchronous	512	$36.0GB$	5.48	7.0 GB/s
synchronous	1024	$72.0GB$	6.91	11.1 GB/s
synchronous	2048	$143.9GB$	8.96	17.2 GB/s
synchronous	4096	$287.8GB$	13.66	22.6 GB/s
synchronous	8192	$575.7GB$	28.14	21.9 GB/s
synchronous	16384	$1.1TB$	51.27	24.0 GB/s
synchronous	32768	$2.3TB$	119.73	20.6 GB/s
Max bandwidth			$24.0GB/s$	
asynchronous	256	$18.0GB$	7.30	2.6 GB/s
asynchronous	512	$36.0GB$	6.23	6.2 GB/s
asynchronous	1024	$72.0GB$	20.33	3.8 GB/s
asynchronous	2048	$143.9GB$	10.19	15.1 GB/s
asynchronous	4096	$287.8GB$	15.83	19.5 GB/s
asynchronous	8192	$575.7GB$	23.77	25.9 GB/s
asynchronous	16384	$1.1TB$	103.98	11.9 GB/s
asynchronous	32768	$2.3TB$	125.68	19.6 GB/s
Max bandwidth			$25.9GB/s$	

TABLE 2. Estimated bandwidth.

The time used for both the synchronous and asynchronous writing procedures can be used to estimate the data transfer bandwidth. Tab. 2 summarizes the maximum time necessary to write one file (of about $175MB$). Using this time together with the total size of the restart files allows us to estimate the bandwidth. This estimation is of about $20GB/s$ for synchronous methods. In the asynchronous case, the MPI communications interacts with the data writing and decreases the mean bandwidth. These bandwidths are in accordance with the nominal HELIOS specifications.

5. Conclusion

In this work, we have shown that the asynchronous method used for the restart file saving allows to circumvent the worst problems encountered in the synchronous cases. All the crashes that occurred have been resumed successfully and automatically due to this method. One can deduce that this asynchronous method is thus able to limit the lost time due to crashes without requiring severe constraints on the code structure and its complexity.

In this work we show that the asynchronous writing method used has a very low overhead, for files size from some GB to more than 1 TB. This procedure has been adapted to easily restart the simulation in case of several crash. We have deduce too the most efficiency frequency of restart file writing in function of crash probability of one node.

References

[CD96] T.-C. Chiueh and P. Deng. Evaluation of chekpoint mechanisms for massivly parallel machines. *Proceedings of FTCS-26*, 1996.

[DPDG+11] G. Dif-Pradalier, P. H. Diamond, V. Grandgirard, Y. Sarazin, J. Abiteboul, X. Garbet, Ph. Ghendrih, G. Latu, A. Strugarek, S. Ku, and C. S. Chang. Neoclassical physics in full distribution function gyrokinetics. 18(6):062309, 2011.

[GBB+06] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrucker, J. Vaclavik, and L. Villard. A drift-kinetic semi-lagrangian 4d code for ion turbulence simulation. *Journal of Computational Physics*, 2(217):395–423, 2006.

[GCG+07] J.N. Glosli, K.J. Caspersen, J.A. Gunnels, D.F. Richards, Rudd R.E., and F.H. Streitz. Extending stability beyond cpu millenium: a micron-scale atomistic simulation of kelvin-helmoltz instability. *In proceedings of the 2007 ACM/IEEE conference on supercomputing*, 2007.

[GSG+08] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, P Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrucker, N. Besse, and P. Bertrand. Computing itg turbulence with a full-f semi-lagrangian code. *Communications in Nonlinear Science and Numerical Simulation*, 1(13):81 – 87, 2008.

[Hah88] T. S. Hahm. Nonlinear gyrokinetic equations for tokamak microturbulence. *Physics of Fluids*, 9(31):2670 – 2673, 1988.

[RHV07] J.F. Ruscio, M.A. Heffner, and S. Varadarajan. Dejavu: Transparent user-level checkpointing, migration, and recovery for distributed systems. *IPDPS2007 Proceedings*, 2007.

[SG06] B. Shroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. *Proceedings of ICDSN*, 2006.

[SG07] B. Shroeder and G. Gibson. Understanding failure in petascale computers. *Journal of Physics Conference Series: SciDAC*, 2007.

[SGA+10] Y. Sarazin, V. Grandgirard, J. Abiteboul, S. Allfrey, X. Garbet, Ph. Ghendrih, G. Latu, A. Strugarek, and G. Dif-Pradalier. Large scale dynamics in flux driven gyrokinetic turbulence. 50(5):054004, 2010.

[SPJ04] J.C. Sanlos, F. Petrini, and Fernandez J. On the feasibility of incremental checkpointing for scientific computing. *Proceedings of IPDPS'04*, 2004.

[SRBG99] E. Sonnendrucker, J. Roche, P. Bertrand, and A. Ghizzo. The semi-lagrangian method for the numerical resolution of the vlasov equation. *Journal of Computational Physics*, 2(149):201 – 220, 1999.

[VS09] E. Vivek Sarkar. Exascale software study: Software challenges in exascale systems. 2009.