Parallelisation of MACOPA, a multi-physics asynchronous solver

Ronan Guivarch[1], Guillaume Joslin[1], Ronan Perrussel[2], Daniel Ruiz[1], Jean Tshimanga[1], and Thomas Unfer[2]

[1] University of Toulouse, INP(ENSEEIHT)-IRIT, France
[2] University of Toulouse, INP(ENSEEIHT)-LAPLACE, France

Abstract. Macopa is a partial differential equations solver based on a particular local time stepping technique dedicated to multi-physics and multi-scale problems. Here, some parallelisation strategies – multi-threading, domain decomposition, and hybrid OpenMP/MPI– are introduced for this solver. Their efficiency is evaluated on a few examples.

1 Context

Numerical simulation has become a central tool for the modeling of many physical systems (combustion, atmospheric plasmas, etc). Multi-scale phenomena make the integration of these models difficult in terms of accuracy and computation time. Time-stepping integration techniques used for modeling such problems generally fall into two categories: explicit and implicit schemes. In the explicit schemes, all unknown variables are computed at the current time level from quantities already available. Time step is then limited by the most restrictive stability condition over the whole computation domain. In the implicit method, the time step is no longer limited by a stability condition. However the scheme is generally not suitable for strongly coupled problems. To solve such problems, a number of local time stepping approaches have been developed. These methods are restricted by a local stability condition rather than the traditional global stability condition.

Macopa is a Partial Differential Equations (PDE) solver based on an asynchronous time stepping technique proposed in [1]. The asynchronous time stepping is an explicit local time stepping technique which is consistent in time for solving a system of conservative PDE. Two data description modes are considered, cell-centered schemes and cell-vertex schemes. It has been successfully applied to fluid mechanics, combustion, micro-wave propagation, plasma discharge modeling. Recent developments have extended the paradigm to higher order accuracy when it is used in combination with a Discontinuous Galerkin method [2].

The capability of handling a large number of different time steps is obtained assuming that the time steps themselves can be discretized using an elementary virtual sub-time step. Under this hypothesis a Discrete Time Scheduler (DTS) has been introduced in [1]. The architecture of the DTS relies on two concepts:

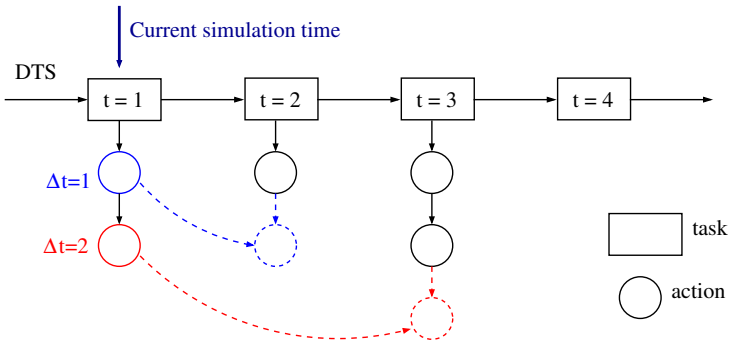


Fig. 1. Example of a sequential DTS

the "action" which is an elementary computation to be done, typically "refreshing" the values of the local variables in a cell of the mesh, and the "task" which is a list of actions to be treated at a given time tag. This list can eventually be empty when no actions are required at this given time tag. The DTS itself is a circular table of tasks with a particular task holding the current time. As the simulation moves in time, a new time tag in the future is assigned to the previous task (see Fig. 1). The horizon of the DTS is the size of the sub-time step multiplied by the number of tasks in the table. Actions, which have time steps larger than the horizon of the DTS, are managed with delays. The DTS insures a planning of n actions with a complexity of $\mathcal{O}(n)$. When it is required, the virtual sub-time step is rescaled dynamically during the simulation.

2 Parallelisation

Parallelisation of the asynchronous time stepping was realized using either multi-threading or mesh partitioning with possible hybridization of both approaches.

2.1 Multi-threading

Multi-threading of the algorithm is possible as long as two issues are tackled: thread-safety has to be insured when accessing the data within the mesh, but it also has to be handled when performing the scheduling of the actions.

For thread-safe schedulling of the actions, the strategy that has been followed is to duplicate the DTS: creating one circular table of tasks for each thread. At the beginning of the tasks, action lists are balanced from one local DTS to the others if needed (see Fig. 2). Then each thread processes its actions and replaces them in the future tasks within its local DTS.

The mesh data issues arises because for conservation equations, say for instance in the finite volume approach, data from the cells on both side of a face are needed to compute a numerical flux through the face. Then this flux is used

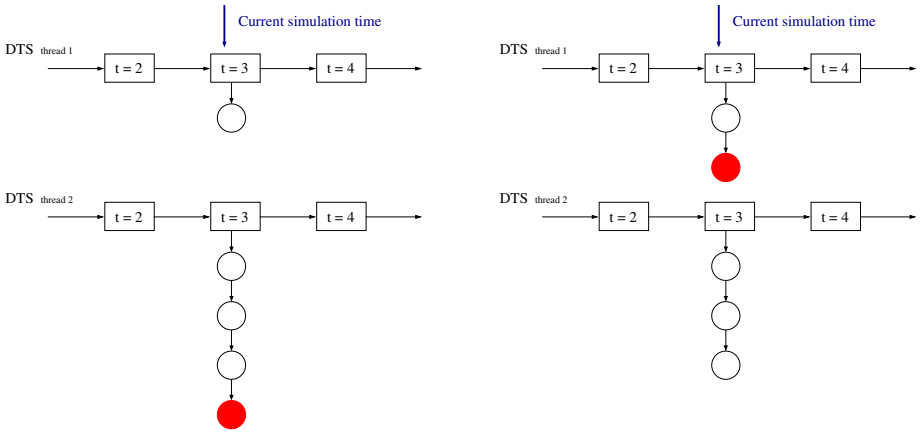


Fig. 2. Example of a multi-threaded DTS: before starting the treatment at the current simulation time (left), the Master balances the actions between the threads (right).

in the flux balance computation needed to obtain the time derivatives of the unknown in both neibouring cells. So if two threads try to refresh two adjacent cells at the same time, a thread-safety issue occurs. The workaround consists in splitting the algorithm in two phases. During the first phase, threads treat their actions. They are allowed to access cell data such as states and time derivatives for reading but not for writing. In this phase, only intermediate variables such as fluxes or source terms are written, and for each cell in which the states and time derivative have to evolve an integration action is then created. This action is inserted into the task of the thread creating it (so that those updates can be done safely in parallel). The integration action consists in synchronizing the states at present simulation time and computing the new time derivatives. For the integration to be thread safe, we must insure that a single cell appears only once in the integration lists of all threads. A synchronization step is performed to remove every duplicated cell from the union of these lists of integration actions. Then the size of the lists are balanced again and each thread performs its integration actions.

2.2 Mesh partitioning

Mesh partitioning has been done using the SCOTCH mesh partitioner [3]. The mesh cells have been weighted with the refreshment frequency (inverse of the local time step) and the faces are weighted with the average refreshment frequency of both cells. Process communications are done using MPI. Each process has its own DTS. For two adjacent cells on both side of the MPI boundary that have different time steps, the refreshment time tags do not necessarily match. The approach that has been developed is to send fluxes/partial residuals with their

time of validity. The time of validity is the next simulation time tag at which the MPI boundary cell must be updated again.

So from the sender point of view, a message shall be sent each time an intermediate variable (flux or partial residual) is recomputed. The receiver has to receive as many messages as needed until the message that contains a validity time that is beyond its own simulation time. So that the current values can be incorporated into the current computations.

For instance, in Fig.3, the blue processor 2 has to wait the message `(flux_1_to_2@t4, t6)` in order to perform its action at time t_5.

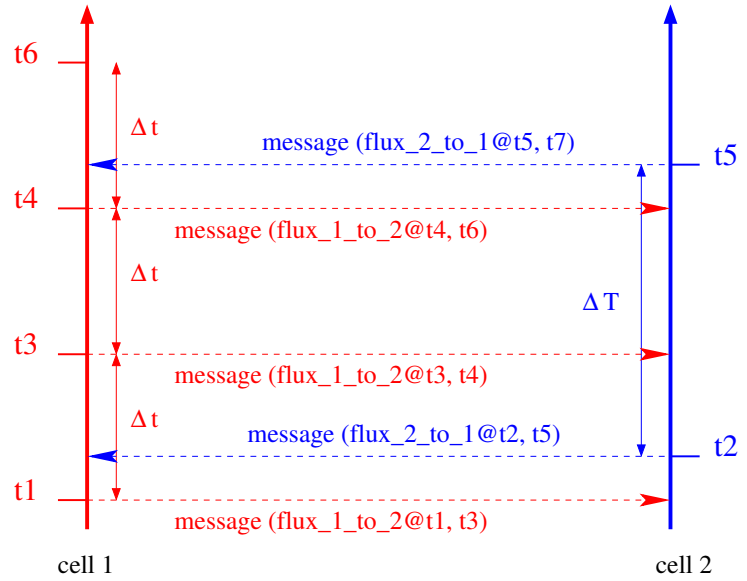


Fig. 3. Example of communication between two processors: each message contains, for two adjacent cells, some physical fields and the time of validity of those fields.

A single MPI message could be sent/received for every single cell at the MPI border, using for instance different message tags. But this approach faces MPI latency because of too many very small messages. The workaround that has been implemented is to manually create larger messages with all single messages that shall be sent/received at the current simulation time of a MPI process. All MPI calls are non blocking, so if an action needs a MPI message which is not available yet, it is pushed back at the end of the list in the task. By doing this, other actions, which do not need MPI messages (such as interior cells), can be performed in the meantime.

Furthermore MPI cell actions are considered more urgent than inner cell actions, so when scheduling the action, MPI cell actions are inserted on top of the task whereas inner cell tasks are inserted at the bottom of the task.

Note: with this paradigm, from the simulation time point of view, every process is also "asynchronous".

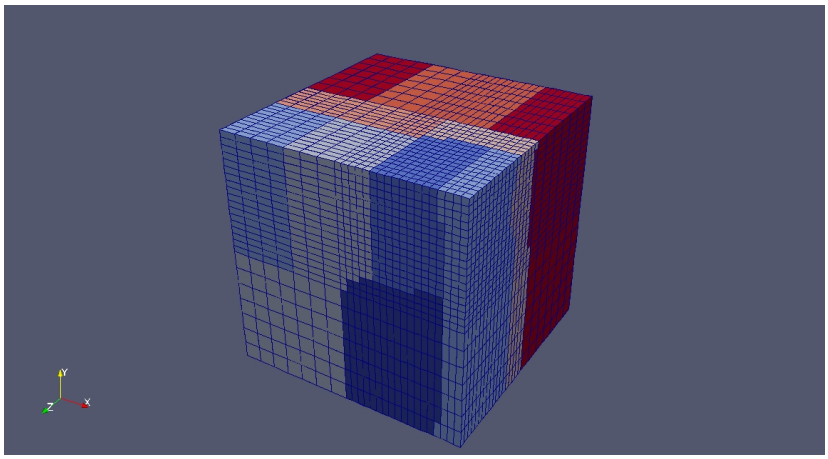


Fig. 4. Example of partition with SCOTCH on a 3D case

2.3 Hybrid OpenMP/MPI parallelisation

For the hybrid mode, MPI is used in multiple thread mode so that every thread in every process can communicate via MPI to any other thread in a locally connected process. When solving PDE using the cell-vertex data description, two threads refreshing two adjacent cells could try to read the same boundary point MPI message. To insure thread-safety in this case, the OpenMP lock concept is used. A thread shall lock a boundary point prior to try to access to its MPI message.

3 Performance Results

We present in this section some performance results obtained on the supercomputer EOS of CALMIP.

Its system is a *Bullx DLC B710 Blades, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR* system. EOS got 122,440 cores and its performances are 255,078 Gflop/s as RMAX and 274,176 Gflop/s as RPEAK and is 399 in the last Top500 list.

3.1 OpenMP results

Our test case is a 2D CH4/air premixed laminar flame (see Fig. 5 for a snapshot of the temperature field). The mesh consists of unstructured triangles for a total of $73,850$ nodes.

Fig. 5. Temperature field

In Fig. 6 it is shown the speed-up obtained on 1 node where we vary the number of threads from 1 to 20 (number of cores by node). We notice that the

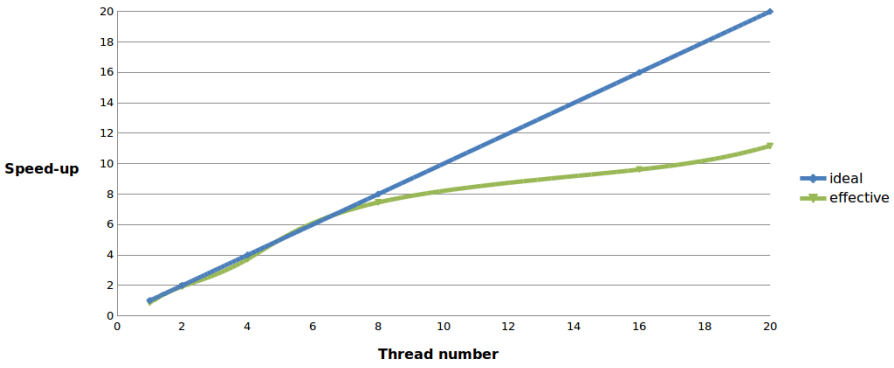


Fig. 6. OpenMP Strong Scaling on EOS

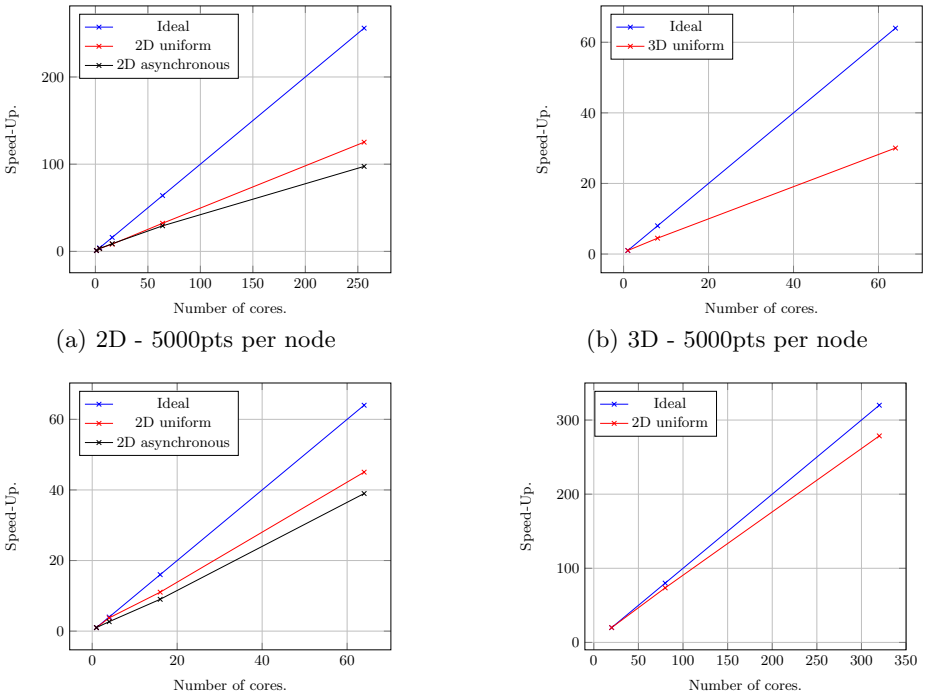
results are perfectly scalable until 8 threads where the number of actions by task is probably the limiting factor.

3.2 Domain decomposition results

Our test case is the propagation of an acoustic wave. It is possible to simulate in a 2D or 3D simple domain.

For the 2D results, two meshes are used. The first mesh is uniform and made of quadrangles. In the second mesh, quadrangle size varies according to a polynomial law in both directions. Thus for the last one, the simulation faces a large amount of different local time steps.

The uniform mesh permits to validate the MPI strategy presented in section 2.2. The non uniform mesh should illustrate the benefit of the asynchronous behavior of Macopa in a strongly asynchronous case.



(a) 2D - 5000pts per node

(b) 3D - 5000pts per node

(c) 2D - 20,000pts per node

(d) 2D - 80,000pts per node/20 threads

Fig. 7. Speed-Up on the propagation of an acoustic wave

Fig. 7 shows that it is possible to implement a parallel version using MPI of the asynchronous algorithm with a reasonable speed-up. However our implementation is a first step and many optimizations are still possible (see last section for some perspectives).

The results of Fig. 7 (a) show that for the current parallel implementation of Macopa, $5,000$ points per core are too low. The MPI boundaries are large

with respect to the interior mesh. In this case, overlapping computation with communication is not effective. The results for the 3D case (Fig. 7(b)) shows the same trends. The performance is improved with more points per core (Fig. 7(c)).

The difference between the uniform and the asynchronous cases means there is still some improvements to propose for the mesh partitioning.

For the hybrid case (open-mp + mpi), we assume that the speed-up with one node and 20 threads is linear. We know that this assumption is far too optimistic (see section 3.1) but the point was to assess the MPI performance in the natural way of using the EOS super-computer: openMP on one node and MPI between the nodes. In this situation, with $80,000$ points per node, the scaling is interesting (Fig. 7(d)).

4 Future work

We noticed that it is difficult with SCOTCH to take into account the amount of work on each cells with asynchronous meshes in order to generate a balanced partitioning. We are looking for some other partitioners, for instance hypergraph partitioners [4,5] in order to better express the constraints and obtain a better partitioning.

Finally, a new version of the standard MPI is now complete. For instance, the MPI Remote Memory Access (RMA) interface has been re-examined and permits efficient one-sided programming model within MPI. We should investigate these new functionalities to determine if they could be useful in Macopa.

Acknowledgements

This research is granted by the project MACOPA (ANR-11-MONU-0019).
This work was performed using HPC resources from CALMIP (Grant 2015-[p1528]).
We also thank Alfredo Buttari for his support and advises.

References

1. T. Unfer, J.P. Boeuf, F. Rogier, F. Thivet, An asynchronous scheme with local time stepping for multi-scale transport problems: Application to gas discharges, J. Comp. Phys, 227, 2007, pp.898-918.
2. A. Toumi, G. Dufour, R. Perrusel, T. Unfer, Asynchronous numerical scheme for modeling hyperbolic systems, Comptes Rendus Mathematique, vol 353, N°9, pp.843-847.
3. F. Pellegrini, SCOTCH 5.1 User's Guide, Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.
4. Ü.V. Çatalyürek, C.Aykanat, PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/umit/software.htm, 1999.
5. M. Rietmann, D. Peter, O. Schenk, B. Uçar, M.J. Grote, Load-Balanced Local Time Stepping for Large-Scale Wave Propagation, IEEE CPS. 29th IEEE International Parallel & Distributed Processing Symposium, May 2015, Hyderabad, India, pp.925-935, <hal-01159687>.