Mechanic: a new numerical MPI framework for the dynamical astronomy

M. Słonina[1], K. Goździewski[2], C. Migaszewski[3]
Toruń Centre for Astronomy, Nicolaus Copernicus University, Gagarina 11, 87-100 Toruń, Poland,
1. slonina@astri.umk.pl, 2. k.gozdziewski@astri.umk.pl, 3. c.migaszewski@astri.umk.pl

Introduction

In the field of the Solar system and planetary dynamics, extensive computational experiments become useful and often necessary research tools. These include direct numerical integrations of complex equations of motion to study the long-term orbital evolution and stability (e.g., [1]), analysis of the fine structure of the phase space and particular types of solutions, like periodic orbits, in terms of dynamical maps [2], modeling various types of observations of extrasolar planetary systems (radial velocities, astrometry, eclipse and TOA timing, photometric transits) by quasi-global evolutionary algorithms (e.g., [3]), investigating qualitative features of basic dynamical models in the framework of the dynamical systems theory (e.g., [4]), spacecraft trajectory optimization (e.g., [5]), to mention just a few subjects of this rapidly developing branch of dynamical astronomy.
Usually, these experiments are equivalent to performing the same or very similar numerical operation or procedure on a large set of initial conditions (e.g., numerical integrations, dynamical maps) or intermediate data (e.g., when evaluating the goal function during model optimization process). In the language of computing, such calculations may be understood as standalone numerical tasks taking seconds, but also hours and days of single CPU-time. If processing of large sets of such tasks is required, one can split a given numerical experiment into smaller parts, and distribute them over a computing pool (usually, CPU cluster or a network of workstations). This leads however to task management issues, which may be handled efficiently only by dedicated software tools.
There are different such task management systems available. Likely, the best example is the well known Condor package [6]. Within this type of numerical framework, the user–supplied, stand-alone software performing computations are distributed over a computing pool. However, the input and output of each software instance must be then handled by the host node, that requires an efficient – possibly unified task preparation scheme – check-pointing and keeping intermediate results, as well as data storage and post–processing. Focusing on particular applications in the field of the dynamical astronomy, we address these issues by developing a new management code, called `Mechanic` that is – unlike Condor and similar software systems – built on the basis of the Message Passing Interface (MPI) [7], highly-standardized and portable message-passing system widely used on a variety of parallel computers and CPU-clusters. We shortly present here some key features of that software (Section 1.), and we describe its basic usage for investigating simple dynamical system (Section 2.). Moreover, `Mechanic` may become a new helper tool in a wide range of applications, particularly focusing on processing large data sets. Indeed, it has been already applied to study the global dynamics of the ν Octantis planetary system (see our second paper in this volume). More details will be described elsewhere [8].

1. Overview of the `Mechanic` framework

`Mechanic` distributes tasks within the well-known task farm (TF) communication pattern, which introduces the master-worker relationship between nodes (CPUs) in a computing pool (Fig. 1). The architecture of our software mimics the Unix system architecture in terms of the core-module scheme. The core of the `Mechanic` handles the MPI communication, basic setup, task pool configuration and assignment (Fig. 2). A particular numerical problem, implemented in the user-supplied module, communicates with the core through provided Application Programming Interface (API), which makes it possible to manage the computations basically in any detail. It applies to any C-interoperable programming languages (such as C++, Fortran 2003+, Nvidia CUDA or OpenCL frameworks) and allows to reuse existing serial

software without a need of extensive modifications. The `Mechanic` core may be installed system-wide and used under common queue control systems. By design, it should be possible to run it under control of any UNIX-like system (Linux and Mac OS are actively maintained) and it works uniformly in single- and multi-CPU environments. The framework may be used in two basic modes: the task farm (as the default) and a master-alone mode (computations are performed only on the master node).
One of the key features of `Mechanic` is dedicated and unified data storage model built on the top of HDF5 standard [9]. By default, no data management is performed on worker nodes. Instead, the user-specified results are sent by these nodes, received by the master node and stored in *one* global datafile. This helps to reduce the overall data processing time and to avoid check-pointing issues. Yet the flexible HDF5 storage standard makes it possible to use the computation results independently from the host software, thanks to a number of helper applications.
By default, the task assignment has the structure of a two-dimensional grid (Fig. 3). The master node creates a pool of computational tasks. Each worker node receives a task from the pool, does the computations, returns the result to the master node, and takes the next task to run, if available. Coordinates of a specific task are used to define the initial state of single-run (Fig. 4), for instance, to compute a dynamical map of a planetary system, we create a mesh grid of initial conditions, which are usually constrained through observations. Each initial condition is mapped then to standalone numerical tasks processed by worker nodes. The grid scheme is governed through the API (Fig. 4).

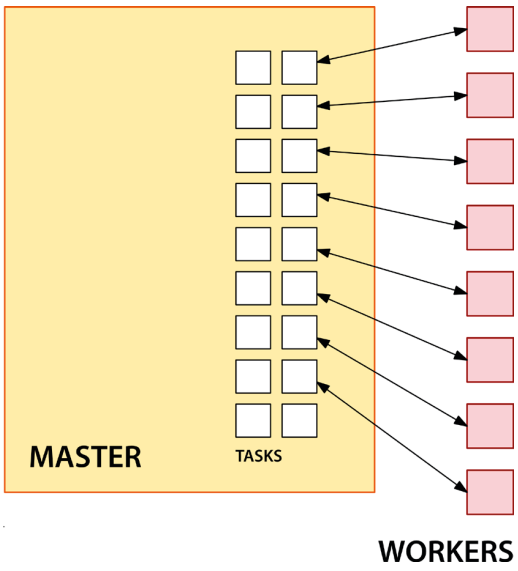

Figure 1: The task farm model. The master node creates and manages a task pool. Each worker node takes a task from the pool, returns the result to the master and takes the next task, if available. Computations in the pool are finished, when all tasks are processed.

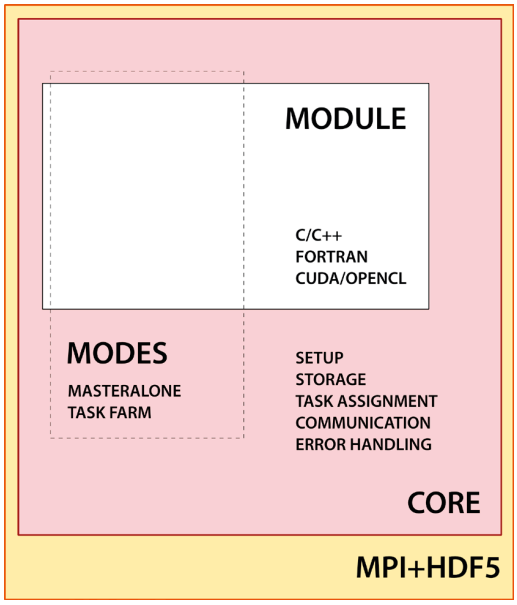

Figure 2: Design of `Mechanic`. An external, user-supplied software communicates with the *core* through provided API (*module*). A *module* is a C-interoperable code compiled in the form of shared library.

The `Mechanic` has been already extensively tested in different computing environments, such as large CPU-clusters, workstations or laptops. The code remains in early stages of its development, however, it provides basic core-functionality, and is suitable to host many computational problems of a dynamical astronomy. In principle, the MPI and HDF5 layers are hidden (or rather transparent) to user, so that the basic knowledge on parallel programming is not necessary. Yet advanced topics, such as additional communication between worker nodes or implementation of different storage layout need parallel computing experience. For details of using and developing the `Mechanic` framework and its modules, please refer to the user-guide available through the project website [10]. Sample numerical modules are available as separate projects at the alternate public website [11].

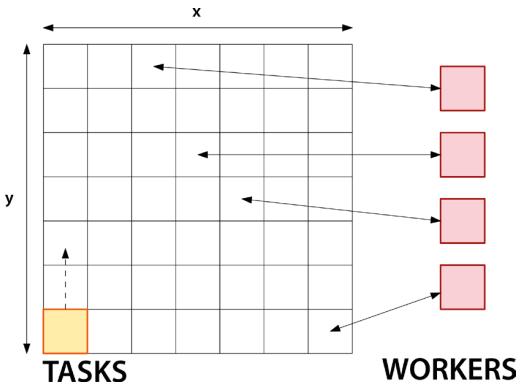

Figure 3: Default task assignment has a two-dimensional grid structure , which suits computation of the dynamical map. Each pixel is a standalone numerical task, handled by the worker node.

```
module_task_process(task) {

  xstep = (task->xmax - task->xmin)/
          task->xres;
  ystep = (task->ymax - task->ymin)/
          task->yres;

  x = task->xmin + task->x*xstep;
  y = task->ymin + task->y*ystep;

  task->result = your_task_code(x,y);
}
```

Figure 4: A sample task preparation (C-pseudocode). Since all information on the task is available to the worker node, it may be used to define the initial state for the run.

2. An example of `Mechanic` application: the Arnold web

To illustrate a simple but non-trivial application of `Mechanic`, we consider the dynamical system investigated by Froeschlé et al. [4]:

$$\mathscr{H}_\varepsilon(I_1,I_2,I_3,\phi_1,\phi_2,\phi_3) = \frac{1}{2}I_1^2 + \frac{1}{2}I_2^2 + I_3 + \frac{\varepsilon}{\cos\phi_1+\cos\phi_2+\cos\phi_3+4},$$

where actions $I_1,I_2,I_3 \in R$ and angles $\phi_1,\phi_2,\phi_3 \in S$ are canonically conjugated phase-space variables, and ε is the perturbation parameter. For $\varepsilon = 0$ the dynamics are integrable. It is well known, that many models of the classical mechanics, dynamical astronomy, theoretical physics and dynamical systems theory have such a general form formulated by Poincaré as the very basic problem of the mechanics. According to the Kolmogorov–Arnold–Moser theorem (KAM, see e.g. [12]), if the perturbation is small enough, certain non-degeneracy conditions are fulfilled, and if the unperturbed motion is sufficiently non-resonant, solutions of the perturbed system are close to the unperturbed one, and lie on invariant tori. On the same torus, all motions are quasi-periodic with the same frequencies. However, the KAM theorem does not say much on the dynamics close to the unperturbed tori with the resonant frequencies. In the neighbourhood of such a set, called the Arnold web, the dynamics may be strongly chaotic and complex. To study such solutions in detail, basically only numerical methods can be applied.

The structure of the Arnold web can be illustrated at two-dimensional plane of actions, (I_1,I_2) that each point corresponds to fundamental frequencies of the unperturbed torus [4]. Resonances in this dynamical system are represented by straight lines, and due to infinite Fourier expansion of the perturbation, resonances of any order are possible. To distinguish between regular and chaotic motions in the neighborhood of the resonances, we apply numerical fast indicator, called the Mean Exponential Growth factor of Nearby Orbits (MEGNO) invented by Cincotta & Simó [13]. MEGNO converges to ~ 2 for the regular motions, decreases to 0 for periodic orbits, and grows linearly with time for chaotic orbits. This behaviour can be colour-coded at high-resolution dynamical maps, constructed at the frequencies plane (Fig. 5) with the help of `Mechanic`.

To solve the equations of motion and variational equations of the test Hamiltonian dynamical system, we applied symplectic integrator *SABA*3 by Laskar & Robutel [14] and the tangent map [15]. Figure 5 illustrates the Arnold web calculated with very fine resolution of 2048×2048 pixels for two values of ε, and relatively long integration times (10^4–10^6). Subsequent panels are for $\varepsilon = 0.01$ and $\varepsilon = 0.04$. The

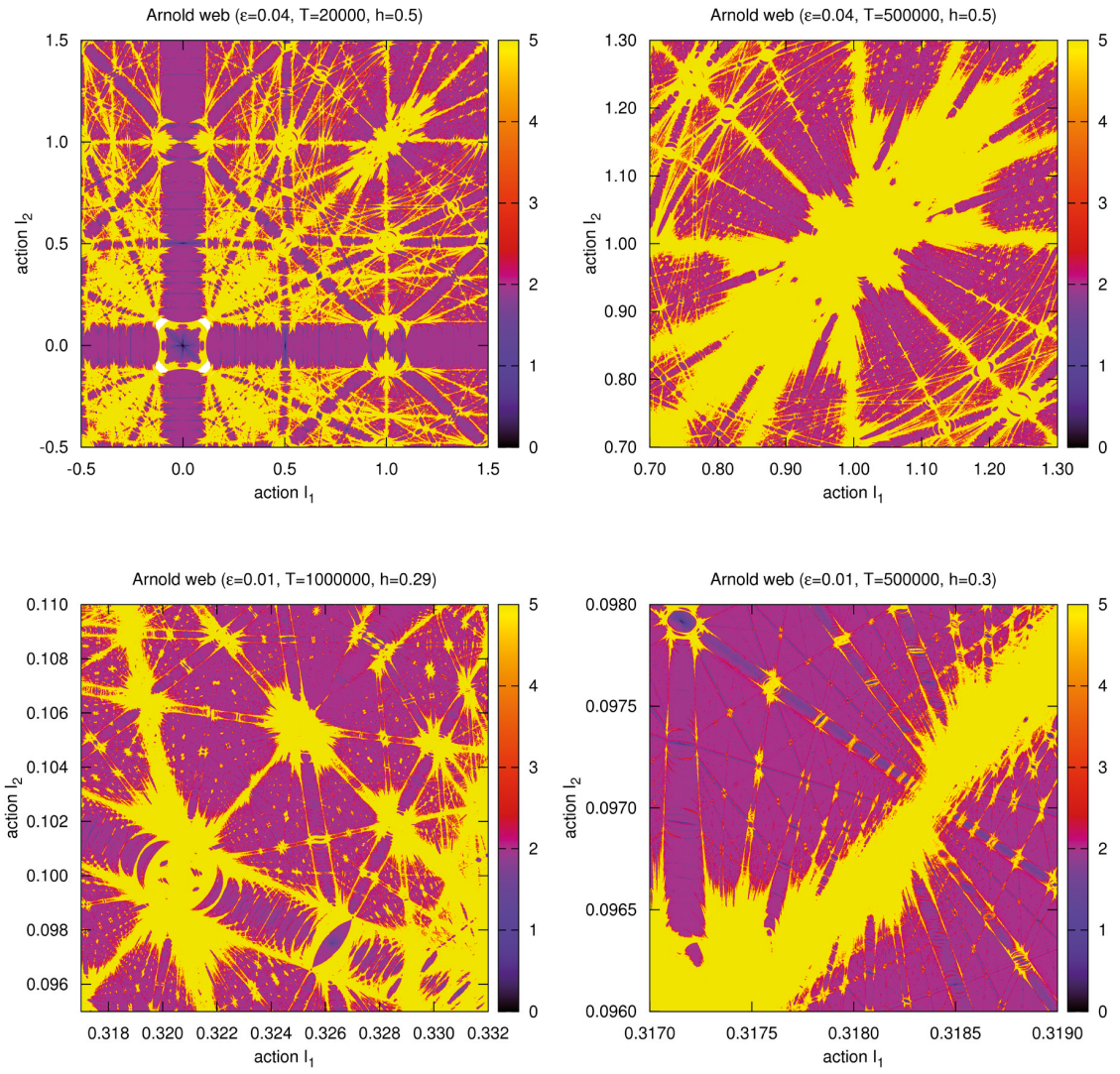

Figure 5: The Arnold web for different values of perturbation parameter ε. Subsequent panels are for parts and close-ups of the web for $\varepsilon = 0.01$ $\varepsilon = 0.04$. The fast indicator MEGNO has been computed at each point of these maps with the help of *SABA3* symplectic integration scheme, based on the so called tangent map, to distinguish between regular (violet regions) and chaotic (yellow regions) motion. Resonances are represented and visible as so called straight lines. The resolution is 2048 $\times$2048 pixels (see the text for details). Step-size h and the total integration time T are labeled.

close-ups reveal stunning details of the structure of the phase-space. Calculations have been run on CPU clusters up to 2048 cores, installed at the Poznań Supercomputer Centre (`reef` and `chimera` clusters), and took up to a few hours, depending on the integration time and step-size. The source code of the numerical module that computes the Arnold web, as well as technical details are available at the project website [11]. An application to modeling of the radial velocity observations, and a study of the global dynamics of the ν Oct planetary system are described in this volume [8].

Acknowledgements

This project is supported by the Polish Ministry of Science and Higher Education through grant *N/N203/402739*. Computations have been conducted within the POWIEW project of the European Regional Development Fund in Innovative Economy Programme *POIG.02.03.00-00-018/08*.

References

[1] Laskar J. & Gastineau M. 2009. *Nature*, 459.

[2] Cincotta P. M., Giordano C. M., and Simó C. 2003. *Phys. D*, 182:151–178.

[3] Goździewski K., Migaszewski C., and A. Musieliński 2008. *IAU Symposium 249*, 447–460.

[4] Froeschlé C., Guzzo M. & Lega E. 2000. Science, 289.

[5] Conway B. A., Chilan C. M. & Wall B.J. 2007. *Celest. Mech. and Dyn. Astr.*, 97.

[6] The Condor Research Project, `http://www.cs.wisc.edu/condor`

[7] Gropp W. et al. 1998. *MPI: The Complete Reference*, The MIT Press

[8] Słonina M., Goździewski K., Rozenkiewicz A., Migaszewski C. 2012. *this volume*

[9] The HDF Group, *The HDF5 Standard*, `http://www.hdfgroup.org`

[10] The Mechanic User Guide, `http://github.com/mslonina/mechanic`

[11] The Mechanic Modules, http://github.com/mslonina/MechanicModules

[12] Arnold V.I., Weinstein A., Vogtmann K. 2001. *Mathematical Methods of Classical mechanics*, Springer.

[13] Cincotta P. & Simó C. 2000. *A&A*, 147.

[14] Laskar J. & Robutel P. 2001. *Celest. Mech.*, 80

[15] Mikkola S. & Innanen K. 1999. *Celest. Mech. and Dyn. Astr.*, 74.