A Multiphysics and Multiscale Software Environment for Modeling Astrophysical Systems

Simon Portegies Zwart[1], Steve McMillan[2], Breanndán Ó Nualláin[1], Douglas Heggie[3], James Lombardi[4], Piet Hut[5], Sambaran Banerjee[6], Houria Belkus[7], Tassos Fragos[8], John Fregeau[8], Michiko Fuji[9], Evghenii Gaburov[1], Evert Glebbeek[10], Derek Groen[1], Stefan Harfst[1], Rob Izzard[10], Mario Jurić[5], Stephen Justham[11], Peter Teuben[12], Joris van Bever[13], Ofer Yaron[14], and Marcel Zemp[15]

[1] University of Amsterdam, Amsterdam, The Netherlands
`spz@science.uva.nl`
[2] Drexel University, Philadelphia, PA, USA
[3] University of Edinburgh, Edinburgh, UK
[4] Allegheny College, Meadville, PA, USA
[5] Institute for Advanced Study, Princeton, USA
[6] Tata Institute of Fundamental Research, India
[7] Vrije Universiteit Brussel, Brussel, Belgium
[8] Northwestern University, Evanston IL, USA
[9] University of Tokyo, Tokyo, Japan
[10] Utrecht University, Utrecht, the Netherlands
[11] University of Oxford, Oxford, UK
[12] University of Maryland, College Park, MD, USA
[13] Saint Mary's University, Halifax, Canada
[14] Tel Aviv University, Tel Aviv, Israel
[15] University of California Santa Cruz, Santa Cruz, CA, USA

Abstract. We present MUSE, a software framework for tying together existing computational tools for different astrophysical domains into a single multiphysics, multiscale workload. MUSE facilitates the coupling of existing codes written in different languages by providing inter-language tools and by specifying an interface between each module and the framework that represents a balance between generality and computational efficiency. This approach allows scientists to use combinations of codes to solve highly-coupled problems without the need to write new codes for other domains or significantly alter their existing codes. MUSE currently incorporates the domains of stellar dynamics, stellar evolution and stellar hydrodynamics for a generalized stellar systems workload. MUSE has now reached a "Noah's Ark" milestone, with two available numerical solvers for each domain. MUSE can treat small stellar associations, galaxies and everything in between, including planetary systems, dense stellar clusters and galactic nuclei. Here we demonstrate an examples calculated with MUSE: the merger of two galaxies. In addition we demonstrate the working of MUSE on a distributed computer. The current MUSE code base is publicly available as open source at http://muse.li.

Keywords: Stellar Dynamics And Evolution; Radiative Transfer; Grid Computing; High-Performance Computing; Multi-Scale Computing.

1 Introduction

The Universe is a multi-physics environment in which, from an astrophysical point of view, Newton's gravitational force law, radiative processes, nuclear reactions and hydrodynamics mutually interact. Astrophysical problems are generally multi-scale, with, in the extreme, spatial and temporal scales ranging from 10^4 meters and 10^{-3} seconds on the small end to 10^{20}m and 10^{17}s on the large end. The combined multi-physics, multi-scale environment presents a tremendous theoretical challenge for modern science. While observational astronomy fills important gaps in our knowledge by harvesting ever wider spectral coverage with continuously increasing resolution and sensitivity, our theoretical understanding lags behind these exciting developments in instrumentation.

Computational astrophysics is situated between observations and theory. The calculations generally cover a wider range of physical phenomena, whereas purely theoretical studies are often tailored to a relatively limited range of spectral coverage. On the other hand, extensive calculations can support observational astronomy by mimicking observations and support the interpretation by enabling wide parameter space studies. They can elucidate complex consequences of physical theories. But extensive computer simulations in order to deepen our knowledge of the physics require large programming efforts and a good fundamental understanding of the underlying physics.

Where modern instruments are generally built by tens or hundreds of people, the development of theoretical models and software environments are generally one-person endeavors. Theory lends itself excellently for this relatively individualistic approach, but scientific computing is in a less favorable position. Developing a simulation environment suitable for multi-physics scientific research is not a simple task. In contrast to purely theoretical studies computer models often require a much broader scope which non-linear couplings between various physical domains. As long as the physical scope remains relatively limited the software only needs to address the problem of solving sets of differential equations in a single physical domain and with a limited range in size scales and time scales. Such software can be built by a single scientific programmer or a numerically well educated astronomer. Regretfully, these packages are often "single-written single-use", and thus single purpose: reuse of scientific software within astronomy is still rarely done.

Problems which encompass multiple time or size scales are sometimes coded by small teams of astronomers. There are several examples of successful projects, such as FLASH [1], GADGET [2] and starlab [3], in which a team of several scientists collaborates in writing a large scale simulation environment. The resulting software of these projects has a broad user base and is applied several times for a variety of problems. These packages, however, address one very specific task, and their use is limited to the type of physics that is addressed and the solver that is used. In addition, it requires considerable expertise to use these packages.

In this paper we describe a software framework that targets multi-scale, multi-physics problems in a hierarchical and somewhat consistent implementation. The development of this framework is based on the philosophy of "open knowledge" [1].

2 The Concept of MUSE

The development of MUSE was initiated during the MODEST-6a [2] workshop in Lund (Sweden [12]), but the first lines of code were written during MODEST-6d/e in Amsterdam (The Netherlands). During the two workshops MODEST-7f in Amsterdam and MODEST-7a in Split (Croatia) the concept of Noah's Arc was initiated and realized (see Sect. 2.1).

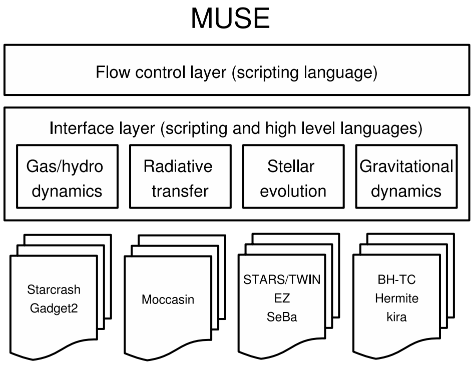

Fig. 1. Basic structure design of the framework (MUSE). The top layer (flow control) is connected to the middle (interface layer) which controls the command structure for the individual applications. These parts and the underlying interfaces are written in Python, whereas the applications can we written in any language. In this example only a selection of numerical techniques is shown for each of the applications, such as smoothed particle hydrodynamics (such as `starcrash` [4] and `gadget` [2]) to solve the gas dynamics, Metropolis-Hastings Monte Carlo for addressing the radiative transfer and ionization/thermal/chemical balance (such as `Moccasin` [5]), Henyey (`STARS`[6], `EZ`[7]) or parameterized code (like in `SeBa` [8]) for stellar evolution and direct integration for the stellar dynamics (Barnes-Hut tree code [9], `Hermit0` or the `kira` integrator in `starlab`[3]).

The development of a multi-physics simulation environment can be approached from a monolithic or from a modular point of view. In the monolithic approach

[1] See for example `http://www.artcompsci.org/ok/`.

[2] MODEST stands for MOdeling DEnse STellar Systems, and the term was coined during the first MODEST meeting in New York (US) in 2001. The web page for this coalition is `http://www.manybody.org/modest`. See also, [10,11]

a single numerical solver is subsequently expanded to include more physics. Basic design choices for the initial numerical solver are petrified in the initial architecture. Nevertheless, such codes are sometimes successfully redesigned to include two or possibly even three solvers for a different physical phenomenon (see FLASH where hydrodynamics has been combined with magnetic fields). Rather than forming a self consistent framework, the different physical domains in these environments are made to co-exist. This approach is prone to errors and the resulting large simulation packages are often hampered by bugs, redundancy in source code, chunks of dead code and a lack of homogeneity. The assumptions needed to make these codes compile and operate without fatal errors often hampers the science. In addition, the underlying assumptions are rarely documented and the resulting science is at best hard to interpret. We address these issues by the development of a modular numerical environment, in which independently developed specialized numerical solvers are coupled at a meta level, resulting in a framework as depicted in Fig. 1.

The modular approach has many advantages. Existing codes which have been well tuned and tested in their own domains can be reused by wrapping these in a thin layer and interfacing them to a framework. The identification and specification of suitable interfaces for such codes allows the codes to be interchanged easily. An important element of the framework will be the provision of documentation and exemplars for the design of new modules, and their integration into the framework. A user can "mix and match" modules like building blocks to find the most suitable combination for his application. The resulting framework is also more easily maintainable, since the dependencies between modules is well separated from their functionality.

A particular advantage of a modular framework is that a motivated scholar can focus the attention on a narrower area, write a module for it and integrate it with a knowledge of only the bare essentials of the framework interfaces. For example it will take little extra work to adapt the results of a successful student project into a separate module, or a researcher working with his own code for one field of physics may wish to find out how his code interacts in a multiphysics environment. The shallower learning curve of the framework will lower the barrier for entry, will make it more accessible and ultimately leads to a more open and extensible system.

The only constraint that code must meet to be wrappen as a module is that it is written in a programming language with a Foreign Function Interface which can be linked to a contemporary Unix-like system. This includes many popular languages such as `C`, `C++` and Fortran as well as other high-level languages such as C#, Java or Haskell.

The flexibility of this framework allows a much broader range of applications to be prototyped and the bottom-up approach makes the code much easier to understand, expand and maintain. If a particular combination of modules is found to be particularly suited to an application, greater efficiency can be achieved, if desired, by hard coding the interfaces and factoring out the glue code, thus ramping up to a specialized monolithic code.

2.1 Noah's Ark

Instead of writing a new code from scratch, we envision a software framework in which a glue language is used to bind a wide collection of diverse applications. We call this environment MUSE, for MUltiphysics Software Environment.

The MUSE framework consist of a hierarchical component architecture that encapsulates dynamic shared libraries for simulating stellar evolution, stellar dynamics and treatments for colliding stars. Additional packages for file I/O, data analysis and plotting are included. Our objective is to eventually include gas dynamics and radiative transfer but at this point these are not yet incorporated.

We have so far included at least two working packages for each domain of stellar collisions (hydrodynamics), stellar evolution and stellar dynamics, in what we label the Noah's Ark milestone. The homogeneous interface which connects the kernel modules enables us to switch packages at runtime via a scheduler. In this paper we demonstrate modularity and interchangeability.

Stellar Collisions. The physical interaction between stars is incorporated by means of (semi)hydrodynamics solvers to the framework. At the moment two methodologies are incorporated, one is based on the `make-me-a-star` (MMAS) package [13][3] and the revised version `make-me-a-massive-star` (MMAMS) [14][4]; the other solution is based on `sticky spheres`. The former (MMAS and MMAMS) can be combined with full stellar evolution models, as they process the internal stellar structure in a similar fashion to the stellar evolution codes. The sticky sphere approximation only works with parameterized stellar evolution, as it does not require any knowledge of the internal stellar structure.

Stellar Dynamics. To simulate gravitational dynamics (e.g., between stars and/or planets), we incorporate packages to solve Newton's equations of motion by means of gravitational N-body solvers. Currently two N-body kernels are available: a direct force evaluation method and a tree code.

The direct N-body code is based on the 4th order Hermite predictor-corrector N-body integrator with block time steps [15]. If present, the code can benefit from special hardware like GRAPE [16] and modern GPUs [17,18]. This method provides the high accuracy needed for simulating dense stellar systems, but even with special computer hardware it lacks the performance to simulate systems with more than 10^6 particles. For simulating large N systems we have incorporated a Barnes-Hut [9] tree-code.

Stellar Evolution. Many applications require the structure and evolution of stars to be followed at various levels of detail. Examples are stellar masses and radii as a function of time (important for feedback on stellar dynamics), luminosities and photon energy distribution of the stellar spectrum (important for feedback on radiative transfer), mass loss rates, outflow velocities and yields of

[3] See `http://webpub.allegheny.edu/employee/j/jalombar/mmas/`
[4] See `http://modesta.science.uva.nl/`

various chemical elements (returned to the gas in the system and to be followed hydrodynamically) and even the detailed interior structure (to follow the outcome of a stellar merger or collision). Consequently the stellar evolution module should ideally incorporate both a very rapid but approximate code for applications where speed (i.e. huge numbers of stars) is paramount (like when using the Barnes-Hut tree code for addressing the stellar dynamics) and a fully detailed (but much slower) structure and evolution code where accuracy is most important (for example when studying relatively small but dense star clusters).

Currently two stellar evolution modules are incorporated. One is based on fits to precalculated stellar evolution tracks [19], the other solves the set of coupled partial differential equations of stellar structure and evolution [6]. The lower speed of the second method is inconvenient but the better physics allows for much more realistic treatment of unconventional stars, such as collision products.

2.2 Performance

Large scale simulations, in particular the multiscale and multiphysics simulations for which our framework is intended, require a large number of very different algorithms, many of which achieve their highest performance on a specific computer architecture. For example, the gravitational N-body simulations are best performed on a GRAPE enabled PC, the hydrodynamical simulations are accelerated using GPU hardware whereas the trivially parallel execution of a thousand single stars is best done on a Beowulf cluster computer.

The top level organization of where what should run is managed using a resource broker, which is grid enabled (see Sect. 2.4). The individual packages have to indicate on what hardware they operate optimal. Some of these modules are individually parallelized using the MPI library, whereas others (like stellar evolution) are handled via a master-slave approach by the top level manager.

Certain parts of the individual modules benefit enormously from dedicated computing. For example, the gravitational direct N-body calculations are sped up by special purpose GRAPE-6 [20,16] or GPU hardware to orders of magnitude faster than on workstations [17,18,21].

2.3 Units

A notorious pitfall in combining scientific software is the failure to perform correct conversion of physical units between modules. In a highly modular environment such as MUSE, this is a significant concern. One approach to the problem could have been to insist on a standard set of units for modules to be incorporated into MUSE but this is neither practical nor in keeping with the MUSE philosophy.

Instead we provide a Units module, in which is encoded information about the physical units used in all other modules, conversion factors between them and certain useful physical constants. When a module is added to MUSE, the programmer adds a declaration of the units which that module prefers. When several modules are imported into a MUSE experiment, the Units module then takes care of ensuring that all values passed to each module are in its preferred units.

Naturally the flexibility which this approach affords also introduces an overhead. But it is this flexibility which is MUSE's great advantage; it allows the experimenter to easily mix and match modules until the desired combination is found. When the desired combination is found, then the dependence on the Units module can be removed and conversion of physical units performed by explicit code. This leads to more efficient interfacing between modules, while the correctness of the manual conversion can be checked against that of the Units module.

2.4 MUSE on the Grid

Due to the wide range in computational characteristics of each of the modules, we plan on running MUSE on a computational grid with a number of specialized machines. Here we report on our preliminary grid interface which allows us to use remote machines to distribute individual MUSE modules on the grid. In this way we reduce the runtime by adopting computers which are best suited for each module, rather than continuing a calculation even though the selected machine may be less suitable for that particular part of the calculation. For example, we can select a large GRAPE cluster in Tokyo for a direct N-body calculation while the stellar evolution is calculated on a Beowulf cluster in Amsterdam.

The current preliminary interface uses the MUSE scheduler as the manager of grid jobs and replaces internal module calls with a job execution sequence. This is implemented with PyGlobus, an application programming interface to the Globus grid middleware written in Python. The execution sequence for each module consists of:

Write the state of a module, such as initial conditions, to file
- Transfer state file to the destination site.
- Construct a grid job definition using Globus resource specification language.
- Submit the job to the grid. The launched job subsequently:
  - read the state file,
  - execute the specified MUSE module,
  - write the new state of the module to a file,
  - and copy the state file back to the MUSE scheduler.
- Then read new state file and resume the simulation

The grid interface has been tested successfully using the distributed ASCI computer (DAS-3). We executed individual invocations of stellar dynamics, stellar evolutions and stellar collisions modules on remote machines.

3 MUSE Example: Two Black Holes in Merging Galaxies

Here we demonstrate the possibility of changing the integration method within a MUSE application during runtime. We deployed two integrators for simulating

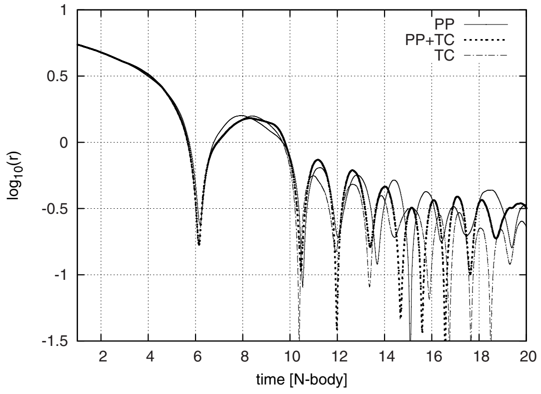

Fig. 2. Time evolution of the distance between two black holes which initially reside in the center of a galaxy with 2048 particles that is a hundred times more massive than the black hole. Initially the two "galaxies" were located far apart. The curves indicate calculations with the direct integrator (PP), a tree code (TC) and using the hybrid method in MUSE (PP+TC). The units along the axis are in dimensionless N-body units [22].

the merging of two galaxies, each with a central black hole. The final stages of such a merger with two black holes orbiting each other can only be integrated accurately using a direct method. Since this is computationally expensive, the early evolution of such a merger is generally ignored and these calculations are typically started some time during the merger process, for example when the two black holes form a hard bound pair inside the merged galaxy.

These, rather arbitrary, starting conditions for binary black hole mergers can be improved by integrating the initial merger between the two galaxies. We use the `BHTree` code to reduce the computational cost of simulating this merger process. When the tree code fails to produce accurate results the simulation is continued using the direct integration method. Overall this results in a considerable reduction of the runtime while still preserving an accurate integration of the close interactions.

In Fig. 2 we show the results of such a simulation. The initial conditions are two Plummer spheres with 1024 particles each, all with the same mass. Each "galaxy" receives a black hole with a mass of 1% of that of the galaxy. The two stellar systems are subsequently set on a collision orbit, but at fairly large distance from each other. The simulation is performed three times, once using `Hermite0`, once using `BHTree` and once using the hybrid method. In the latter case the equations of motion are integrated using `Hermite0` if the two black holes are within 0.3 N-body units [22][5], otherwise we use the tree code. In Fig. 2 we show the time evolution of the distance between the two black holes.

The integration with only the direct `Hermite0` integrator took about 4 days on a normal workstation and the tree code took about 2 hours. The hybrid code

[5] Or see `http://en.wikipedia.org/wiki/Natural_units#N-body_units`.

took almost 2 days. As expected, the relative error in the energy of the direct N-body simulation ($< 10^{-6}$) is orders of magnitude smaller than the error in the tree code ($\sim 1\%$). The energy error in the hybrid code is comparable to that of the tree code, which in part is caused by regularly changing methodology. Even though we were unable to further reduce the energy error in the hybrid code, it seems safe to assume that the close encounters of the two black holes are treated much more accurately in the hybrid approach compared to the pure tree-code simulation. This is supported by the very small $\mathrm{O}(10^{-6})$ error in the energy during the close interactions between the two black holes, which were computed using the direct integrator. If the system was integrated using the tree code, the energy errors were also characteristic for the adopted methodology. The latter then obviously dominates the total error, irrespective of the accurate integration of the close encounters.

Acknowledgments. We are gretefull to Atakan Gürkan, Junichiro Makino, Stephanie Rusli and Dejan Vinković for many discussions. This research was supported in part by the Netherlands Organization for Scientific Research (NWO grant No. 635.000.001 and 643.200.503), the Netherlands Advanced School for Astronomy (NOVA), the Leids Kerkhoven-Bosscha fonds (LKBF), the International Space Science Institute (ISSI) in Bern, the ASTRISIM program of the European Science Foundation, by NASA ATP grant NNG04GL50G and grant No. NNX07AH15G, by the National Science Foundation under Grant No. PHY99-07949 (S.L.W.McM.) and No. 0703545 (J.C.L.), the Special Coordination Fund for Promoting Science and Technology (GRAPE-DR project), the Japan Society for the Promotion of Science (JSPS) for Young Scientists, Ministry of Education, Culture, Sports, Science and Technology, Japan and DEISA. Part of the calculations were done on the LISA cluster and the DAS-3 wide area computer in the Netherlands. We are also grateful to SARA computing and networking services, Amsterdam for their support.

References

1. Fryxell, B., et al.: FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. Apjs 131, 273–334 (2000)
2. Springel, V., Yoshida, N., White, S.D.M.: GADGET: a code for collisionless and gasdynamical cosmological simulations. New Astronomy 6, 79–117 (2001)
3. Portegies Zwart, S.F., McMillan, S.L.W., Hut, P., Makino, J.: Star cluster ecology - IV. Dissection of an open star cluster: photometry. MNRAS 321, 199–226 (2001)
4. Lombardi Jr., J.C., Warren, J.S., Rasio, F.A., Sills, A., Warren, A.R.: Stellar Collisions and the Interior Structure of Blue Stragglers. apj 568, 939–953 (2002)
5. Ercolano, B., Barlow, M.J., Storey, P.J.: The dusty mocassin: fully self-consistent 3d photoionization and dust radiative transfer models. MNRAS 362, 1038–1046 (2005)
6. Eggleton, P.P.: The evolution of low mass stars. MNRAS 151, 351 (1971)
7. Paxton, B.: EZ to Evolve ZAMS Stars: A Program Derived from Eggleton's Stellar Evolution Code. PASP 116, 699–701 (2004)

8. Portegies Zwart, S.F., Verbunt, F.: Population synthesis of high-mass binaries. A & A 309, 179–196 (1996)
9. Barnes, J., Hut, P.: A Hierarchical O(NlogN) Force-Calculation Algorithm. Nat 324, 446–449 (1986)
10. Hut, P., et al.: MODEST-1: Integrating stellar evolution and stellar dynamics. New Astronomy 8, 337–370 (2003)
11. Sills, A., et al.: MODEST-2: a summary. New Astronomy 8, 605–628 (2003)
12. Davies, M.B., et al.: The MODEST questions: Challenges and future directions in stellar cluster research. New Astronomy 12, 201–214 (2006)
13. Lombardi, J.C., Thrall, A.P., Deneva, J.S., Fleming, S.W., Grabowski, P.E.: Modelling collision products of triple-star mergers. MNRAS 345, 762–780 (2003)
14. Gaburov, E., Lombardi, J.C., Portegies Zwart, S.: Mixing in massive stellar mergers. MNRAS 9, L5–L9 (2008)
15. Makino, J., Aarseth, S.J.: On a hermite integrator with ahmad-cohen scheme for gravitational many-body problems. Publ. Astr. Soc. Japan 44, 141–151 (1992)
16. Makino, J.: Direct Simulation of Dense Stellar Systems with GRAPE-6. In: Deiters, S., Fuchs, B., Just, A., Spurzem, R., Wielen, R. (eds.) ASP Conf. Ser. 228: Dynamics of Star Clusters and the Milky Way, p. 87 (2001)
17. Portegies Zwart, S.F., Belleman, R.G., Geldof, P.M.: High-performance direct gravitational N-body simulations on graphics processing units. New Astronomy 12, 641–650 (2007)
18. Belleman, R.G., Bédorf, J., Portegies Zwart, S.F.: High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. New Astronomy 13, 103–112 (2008)
19. Eggleton, P.P., Fitchett, M.J., Tout, C.A.: The distribution of visual binaries with two bright components. 347, 998–1011 (1989)
20. Makino, J., Taiji, M.: Scientific simulations with special-purpose computers: The GRAPE systems. In: Makino, J., Taiji, M. (eds.) Scientific simulations with special-purpose computers: The GRAPE systems, John Wiley & Sons, Chichester, Toronto (1998)
21. Hamada, T., Fukushige, T., Makino, J.: PGPG: An Automatic Generator of Pipeline Design for Programmable GRAPE Systems. In: ArXiv Astrophysics e-prints (March 2007)
22. Heggie, D.C.: Binary evolution in stellar dynamics. MNRAS 173, 729–787 (1975)