Preprint typeset using LaTeX style emulateapj v. 11/10/09

CODES AS INSTRUMENTS: COMMUNITY APPLICATION AND SIMULATION SOFTWARE FOR THE HARDWARE ARCHITECTURES OF THE NEXT DECADE

Falk Herwig (UVic) and L. Jonathan Dursi (CITA/SciNet)

ABSTRACT

Modern astronomical research requires increasingly sophisticated computing facilities and software tools. Computational tools have become the fundamental tools to turn observational raw data into scientific insight. Complex multi-physics simulation codes have developed into tools for numerical experiments that provide scientific insight beyond classical theory. Canadian researchers need an environment for developement and maintenance of these critical tools. In particular, the drastically enhanced complexity of deeply heterogeneous hardware architectures poses a real challenge to using present and future HPC facilties. Without a national program in astrophysical simulation science and astronomy application code developement we are becoming vulnerable with respect to our ability to maximise the scientific return from existing and planned investments into atronomy. In addition, there are significant industrial/commercial HQP needs that a simulation and application code program could start to address, if it is properly aligned with academic training opportunities. We outline the framework and requirements for such a framework for developing Canadian astronomical application and simulation codes — and code builders. In the US decadal plan process, voices are calling for similar emphasis on developing infrastructure and incentives for open community codes (Weiner *et al.* 2009). We propose funding several small interdisciplinary teams of postdocs, graduate students, and staff, housed in departments at Universities that have or are about to make a commitment in a relevant area (e.g. applied math, computational physics, modeling science). These teams can, while training astronomical and computational HQP, focus on building tools that have been deemed to be high priorities by the astronomical and astrophysical communities in order to make the best scientific use of our new computational faciliites.

Subject headings:

1. CODES AS INSTRUMENTS

The Canadian computational landscape has changed remarkably since the last LRP, and in most ways for the better; Canadian astronomers now have access to much larger data and computational power, and there is some degree of staff supporting these new large facilities (for an overview, see Dursi et al 2010).

This white paper is singling out a final missing link and bottle neck that prevents Canadian astronomy and astrophysics to fully take advantage of the potential of the new computer age. This missing link is the underdeveloped and underfunded state of our simulation and application software developement efforts. The role of these activities has internationally long been recognized, e.g. in the NSF report *Computation As a Tool for Discovery in Physics*[1] that was the result of an expert workshop almost 10 years ago:

> Simply put, we must move from a mode where we view computational science as an applied branch of theory to a mode where its true resource needs as a distinct research mode are recognized. Concretely, this means providing support for building the infrastructure (software) of computational science at levels commensurate with their true costs, just as we support construction and operation of experimental facilities.

We like to employ this metaphor of simulation and application codes as instruments here as well[2]. The scientific success of new telescopes depends vitally on the development of increasingly sophisticated and complex instruments that turn photons collected with large dishes into meaningful astronomical data. Likewise, high-performance and super-computers (e.g. at Compute Canada sites) provide cycles (read photons) that need increasingly sophisticated and complex application and simulation codes to produce meaningful astronomical and astrophysical data. Just as the implementation of AO, multi-object and interferometric technologies are pushing the limits if instrument building, the accelerated introduction of increasingly complex hardware architectures pushes code developers to the next edge.

The last decade has seen much needed investments into the hardware platforms for scientific computing. There has not been, however, any kind of concomitant increase in funding for astronomical code development. To employ once more our metaphor, astronomy is not funding the instrument builders (read application and simulation code teams) which will develop and maintain the tools required to make the best possible scientific use of the new facilities available to us. The Compute Canada consortia have support staff (technical analysts) that can be thought of as support astronomers at observational facilities; they (including LJD) are available to help an astronomer make use of the facilities given the instruments that are available; but they have neither the time,

[1] `http://www.nsf.gov/pubs/2002/nsf02176`

[2] Of course, as all metaphors, this one has it's clear limits, and we apologize to the true instrument builders for simplifying their terrific and most important work for this illustration purpose.

resources, nor, in general, the expertise to create — or just significantly participate in creating the — new instruments from scratch for the astronomer to use.

The way this plays out in practice is that the technical analysts can help astronomical researchers take existing simulation or data-analysis codes and port them to the new platforms, including doing significant work on parallelizing the code to make use of new, large computers or suggesting more efficient algorithms for already working programs. However, in general they will not have any domain experience in the astronomical or astrophysical problem at hand; while they can make it run faster and/or in parallel, they will have no knowledge as to whether the code works well in the first place, will continue working for larger problems, provides accurate answers, or is even considering the correct questions.

2. WRITING PROFESSIONAL SCIENTIFIC CODE REQUIRES PROFESSIONALS

So, what is so different about simulation and application code building compared to regular research activities of NSERC discovery grant funded researchers? To build scalable, efficient simulation or data analysis code for science requires expertise in the domain science and the mathematical techniques used therein; expertise in the basics of numerical methods (approximation with finite precision, domains of stability, convergence and consistency, validation and verification), knowledge of the current state of the art for numerical methods for all the mathematical operations to be used in the code (this is non-trivial; even something as straightforward as choosing pseudo-random numbers, bread-and-butter for any Monte Carlo method, become non-trivial when looking at large numbers of samples, and there is a large literature on the subject); knowledge of computer architectures, necessary for efficient computation (cache architecture and performance; vectorization/SSE operations); techniques for efficient parallel computation (shared- and distributed-memory, and increasingly heterogeneous and hybrid architectures such as GPGPU, IBM Cell), and basics of software engineering (unit testing; version control; software architecture; refactoring). This represents a volume and complexity of expertise which requires a team effort and sustained support to build-up. In particular, some of these capabilities by themselves clearly fall out of the reward model that is typically employed in astronomy (i.e. an essential but more technical contribution to a large and complex code project will get less than its in the peer-reviewed scientific paper reward model, e.g. technical aspects are often defered to an appendix).

Of course, to write 'one-off' proto-type code of relatively simle tasks does not require this level of knowledge; but we argue that having many scientists cobble the same sort of things together independently, and with varying degrees of quality and often making less efficient use of the latest parallel architecture, is simply not the best use of resources of the Canadian astronomical community. From the experience of looking at such code as technical analysts at the Compute Canada consortia, it is not unusual to find easy speed-ups of code by factors of several due to improving cache use, or to find obvious bugs that by pure luck have (hopefully) not affected numerical results to date. When such bugs do occur, of course, they can be disastrous (*eg*, Brunini 2006; Petsko 2007; Hall and Salipante 2007).

3. IMPORTANCE OF COMMUNITY CODES TO ASTROPHYSICS MODELING

But when quality software is developed and publicly released, and subsequently earns the trust of the community, it can start a chain reaction which greatly contributes to simulation and theory research in its subfield. The MESA stellar evolution code which is presently pushed forward by a small team around a privately funded master programmer (Bill Paxton) is an example (Paxton etal, in prep).[3] Over time such community codes become used by a growing community of scientists. Any serious bugs get shaken out, or, more colloquially, "Given enough eyes, all bugs are shallow", Raymond (1999). These codes are then increasing the efficiency of groups in solving science problems, thereby addressing some of the often stated needs of observers to have better models and simulations at their disposal to interprete observations. The community code becomes a platform from which to build extensions by adding new physics or methods, and these extensions in turn get fed back to the rest of the community.

The resulting impact of community codes can be characterized by looking at their use. One of the authors keeps track of computational astrophysics, particularly simulation, papers on astro-ph[4]; in the four years from 1 Apr 2006 to 30 Mar 2010, there were approximately 3,435 of these. Using the ADS, one can count the citations of the code papers of some of the most-used community codes during that same period of time; these citations usually represent papers that use the code. Citation counts include Gadget2 (715; Springel 2005), Cloudy (483; Ferland *et al.* 1998), FLASH (235; Fryxell *et al.* 2000), PKDGRAV (98; Stadel 2001), Ramses (95; Teyssier 2002), Zeus (59; Hayes et al 2006), and Enzo (42; O'Shea *et al.*, 2004). There are some issues with comparing these numbers; the simulation paper count may be an undercount (but even so, it also includes non-simulation papers, such as new algorithms for data analysis); in addition, the citations of the code papers certainly overcounts, but probably only modestly, the number of papers using the codes. If we nonetheless take these numbers at face value, the implication is that the top three community codes are responsible for 40% of simulation papers during that period, rising to 50% if one includes the next three. Given the diversity of objects and physics that comprise the field, this is remarkable, and remains so even if a more careful accounting adjusts the percentages downwards somewhat.

That trusted community codes are so often used should not be surprising; once the code becomes accepted within the community as a good way to do certain types of simulations, it becomes a go-to approach for researchers; and in fact the very existence of a well-tested, freely available code to perform a certain task may encourage certain lines of research; whereas other possibly equally or more important research directions simply languish, lacking a comparably available tool. Experience also shows that once a certain seed investment has been made into a computational tool eventually community driven growth

[3] `http://mesa.sourcforge.net`
[4] `http://www.cita.utoronto.ca/~ljdursi/thisweekcomp/`

becomes self-perpetuating.

Therefore, building or even seeding a successful community code can actually shape research patterns; it provides a simulation tool that is not only less work for a researcher than stopping other projects to write their own, but the result is generally of higher quality. When it comes to research tools, "Better, for cheaper" should be something that as a matter of policy the astronomical community in Canada should encourage, and fund.

4. IMPORTANCE OF COMMUNITY CODES TO ASTRONOMICAL DATA ANALYSIS

For the same reasons, community codes play an important role on the observational side of astronomical research. As pointed out in a white paper for the US decadal survey (Weiner *et al.* 2009), several software packages or libraries that have been released to the community "have enabled easily as much science as yet another large telescope would have, at considerably lower cost". These can take the form of complete data reduction packages such as the venerable IRAF (`http://iraf.noao.edu/`), AIPS (`http://www.aips.nrao.edu/`) or MIRIAD (`http://www.atnf.csiro.au/computing/software/miriad/`); photometry and analysis packages such as DAOPHOT, by Peter Stetson at DAO (`http://www.star.bris.ac.uk/~mbt/daophot/`), SExtractor (`http://sextractor.sourceforge.net/`) and GALFIT (`http://users.obs.carnegiescience.edu/peng/work/galfit/galfit.html`) and various libraries used to build packages such as FITSIO, PGPLOT, the IDL Astronomy Library, and IDLUTILS. Many of these packages are used so routinely that it is almost impossible to track their usage in the literature.

5. INDUSTRUAL/COMMERCIAL IMPLICATIONS

Systematically building up and nurturing a Canadian simulation and application code capabilty in astronomy would likely have HPQ training implications beyond the immediate needs of the astronomy community. Some of our theory and simulations graduate students and postdocs end up using their computational skills in industry and commerce. Again, employing the instrument building metaphor, the skill and training of code building and maintenance has similar potential for technological implications for the private sector as has the development of high-tech skills and capabilities that are viewed as a desired and essential side effect of many instrument building funding decisions.

From the exemplarary experience of one post-doc at UVic we know that scientists with experience in mathematical modeling and numerical simulations of multiphysics processes are a thought after human resource in research and development groups of companies that need to solve problems or use methods similar to those traditionally met in physical sciences, such as astrophysics. Examples of companies that this particular post-doc interviewed with include Maplesoft (a software company in Waterloo, ON), which has a contract with Toyota for mathematical modeling of cars. They needed a person who could solve a complex system of PDEs using their computer algebra software product Maple (similar to Mathematica). Automotive Fuel Cell Cooperation (AFCC) company in Vancouver wanted to hire a simulation scientist to model various physical and electrochemistry processes in hydrogen fuel cells that they are developing for cars in cooperation with Ballard, Daimler, and Ford. General Fusion Inc. in Vancouver was looking for a computational plasma physicist who would perform magneto-hydrodynamics simulations to model collision and interaction of two compact plasma configurations for a future thermonuclear reactor. Similar positions were recently advertised, e.g. with Atomic Energy of Canada Limited (AECL) and U.S. Steel. Another post-doc, who worked in the same simulation group at Lost Alamos as one of the authors was hired by a German subsidiary of Siemens to build a small research group in hydrodynamic simulations of industrial fans. These examples show that the same technological and industrial benefits that are commonly associated with instrumentation engeneering projects can also be associated with appropriately funded and organised simulation code developement and maintenance projects. In fact, we argue that an astrophysics program in this area could address the underserved human resource needs of some Canadian high-tech companies in the coming decade.

6. THE INCREASING DIFFICULTY OF DEVELOPING SCALABLE CODES FOR LARGE DATA/LARGE COMPUTING

Building a high-quality, correct, reliable piece of scientific software that others can use for their problems, even for running just on the desktop is difficult. Increasing a problem's complexity by 25% can increase the complexity of the software to solve it by a factor of two (Woodfield 1979). Building software to be reusable by others is three times more time consuming than to whip out something that will only be used once (Thomas, Delis & Basili 1997) – but if the software is to be used by more than three people, it is obviously worth it. Similarly, 40%–80% of software development cost is maintenance (Boehm 1973). But for a piece of software to be useful to the community, it must be general enough to be useful for many problems, reusable, and maintained.

Truly cutting edge scientific software that can take advantage of modern parallel computers, however, is still harder. "There is widespread agreement that trends in both hardware architecture and programming environments and languages have made life more difficult for scientific programmers" (Graham, Snir, & Patterson 2004).

The most obvious increase in complexity is in parallel computing. The primary library for programming in parallel on large clusters is with MPI (`http://www.mcs.anl.gov/research/projects/mpi/`), but this is an extremely difficult way to program complex parallel codes. Programmers who are new to parallel programming demonstrate a wide spread in development times (a factor of 10!) in developing parallel MPI code from scratch, even at a fixed level of quality; the spread between novice and experienced programmers is presumably even greater;(Almeh 2007). Even in a mature, parallelized code, the MPI sections require constant work; apparently roughly 25% of the maintenence of FLASH is done on the MPI parts (Hochstein, Shull, & Reid 2008).

For simple problems, OpenMP (`http://openmp.org/`), a shared memory approach which is viewed as much easier than MPI, only takes about 60% less effort than MPI, at the cost of being impossible to scale to

large clusters (Hochstein 2006).

But the true challenge for the coming decade for efficient simulation and application code developement will be the increasing heterogeneity of computing hardware. Even now a single compute node may already have 8-32 processing cores, suggesting different parallelization techinques within a node as vs. between nodes. In addition, accellerators such as GPUs within a node are becoming more common. The IBM Cell architecture is another example of hybrid compute nodes.[5] This shift to heterogeneous architectures makes programming even more difficult – for instance, the second biggest computer in the world has no fewer than three types of processors, using it to its fullest requires use of all three.

None of these hurdles are insurmountable, but it is becoming simply impossible for a scientist to be at the same time at the leading edge of all the aspects of simulations science; or to write high-quality, correct, reliable, software from scratch for the current crop of supercomputers while staying on top of the discipline area. To employ once more our instrument metaphor, we have to implement in the computing arena the type of collaboration and division of labor that is well established between observers and instrument builders in astronomy.

While in some other countries the era of every researcher using home-grown software is mostly gone (take for example the US with its long established system of national supercomputing centers that are at the same time nucleation points innovative code developement, complemented by the DOE SciDAC programs and previously the ASC program). Canada should follow those examples and enter an era of professionalization of the scientific software community, where scientists can rely increasingly on community codes.

7. CASE STUDIES: WHAT IT TAKES TO DEVELOP A COMMUNITY CODE

To understand what is involved in developing a community code, we briefly discuss the development history of the top three simulation codes by citations, listed above. The three are very different types of code with quite different development histories, and distribution moels, but all demonstrate that to build a successful community code requires amounts of work measured in person-years, not -months; extensive documentation; continuous maintenance; test cases so that a user can ensure that it is working properly on their system; and that the code must have been written to address a real need to perform specific research.

7.1. *Gadget 2*

Gadget 2 is a SPH code which evolves ideal-gas hydrodynamics under the effect of self-gravity, as well as gravity-only dark matter dynamics. It includes two gravity solvers – one purely tree-based, and one that also involves a (possibly periodic) mesh. It consists of 12,500 source lines of code (eg, omitting comments and blank lines), and comes with 46 pages of documentation, the 30-page code paper, and data for most of the test-cases covered in the paper.

Gadget 2 was developed essentially solely by Volker Springel, at the time at the MPA in Garching, and was written for his own requirements in performing galactic and cosmological simulations. The coding, testing, and writing of documentation required several person-years of labour, by someone already well experienced in the area – the original Gadget-1 was produced during Springel's PhD work. The development was supported by the unusually independent and long-term nature of postdocs at CfA and MPA. As is often the case, the author published several results with Gadget 2 before widely releasing the code, allowing a balance between publicly releasing the code and still benefiting scientifically from the development (eg, not get 'scooped' by users of the code he wrote). Gadget is distributed with source code for reading results into various packages (IDL, sm, VTK) for analysis, and recently a 3rd party package has started to include support for visualizing results.

Gadget has a 'one-way' code distribution mechanism; the more or less complete Gadget-2 code was released when completed, and it has been updated only with very minor bug fixes since. A community of users has built up around it; many users of Gadget add their own subgrid models or other physics, sometimes making this more widely available to the community, but usually not – this is usually due to proprietary concerns, but also because of the time required to maintain and support publicly-released code. The author continues with his collaborators to develop a proprietary version with additional physics. There is a community mailing list where users post questions, which are often answered by other users.

7.2. *Cloudy*

Cloudy is a spectral synthesis / photoionization simulation code: the spectrum of material of a given density/composition/temperature profile is calculated. Cloudy has been actively developed since 1978; it started started in early versions of FORTRAN, and has since been rewritten in C++. It currently consists of 142,000 source lines of code, and 500 pages of documentation that extends into four volumes. In addition to the usual amount of work that goes into a scientific code, the Cloudy developers have also by necessity spent a great deal of time curating a database of atomic rates and line widths that are essential inputs to the code. Tending to and maintaining this database has required a great deal of work, and the developers provide it in a format which can be used by other codes fairly easily.

Estimating the amount of effort that has gone into a code over three decades is almost impossible, but simple software engineering models (Wheeler 2004) suggest that several dozens of person-years would be required, which seems immediately plausible given the length of time and the number of developers involved; there are typically three or so active developers at any given time. The NSF has funded cloudy development for 29 years continuously; this sort of stable funding for computational work is almost impossible to imagine in the current Canadian astrophysics funding model.

The Cloudy community is centred around a wiki and a web-forum where the developers and knowledgeable

[5] Recently, Los Alamos National Lab in collaboration with IBM have demonstrated how the future of parallel computing could look like when they combined 5 different architectures to work together in one hydrodynamic simulation code `http://www.youtube.com/watch?v=-GeB1HOt_U0`.

users answer user questions. It is distributed with its extensive documentation and several test suites, and tools for analyzing and visualizing results. Suggested updates are accepted from the community, and incorporated into the code distribution. New releases, involving significant rewrites are released infrequently (every several years) but versions, involving more minor changes, and 'hotfixes' — urgent patches — are released more often.

7.3. *FLASH*

FLASH is a multi-physics adaptive mesh hydrodynamics code, including self-gravity through multipole and multigrid solvers, hydrodynamics (several solvers), MHD (several solvers), several equations of state including for degenerate matter, optically thin heating/cooling, nuclear reactions. The FLASH code consists well over 400,000 source lines of code, and a 330 page manual. The user community is arranged around a web page and several mailing lists, and the FLASH centre gives tutorials which are posted online. A test suite is run on the code daily, identifying problems as soon as they are checked into the code base. The FLASH-centre internal version of the code contains test versions of solvers which are not publicly released; typically the internal users publish using the solvers before releasing them at large. An analysis/visualization package based on IDL is distributed with the code, and an open-source visualization/analysis package is available separately. The FLASH code is not as freely available as the other two codes on this list; its funding agency (the US DOE NNSA) requires users who want access to the code to sign a license agreement forbidding re-distribution and to fax the result back in. Users from countries the US considers 'unfriendly' cannot get access to the code.

FLASH was funded as part of a larger research project, and it is difficult to disentangle the costs and contributions of the code development as apart from the rest of the project. However, well over two dozen people have committed significant portions of code to the code base over the 10-year period of development, which in turn began with existing code bases from PROMETHEUS, PARAMESH, and microphysics packages previously written by original FLASH team members. The numerical methods for hydrodynamics (i.e. the piece-wise parabolic method, PPM, ¡ Colella & Woodward 1984) preceeds even 20 years back. Over most of the official ten year period of funding of the project, at least two or three people were employed full time to develop the code and associated tools; thus, not including the scientists who contributed physics solvers, this already constitutes 20-30 person years of supported effort.

FLASH is continuously updated with bug fixes, and new solvers; only FLASH team members can add code. Typically the developers have a chance to test, and then publish with, new solvers before they are incorporated to the main distribution; new versions of the FLASH code are released every year or so. Very occasionally new code is accepted from the community, but this is typically from close collaborators. Many users have added their own solvers for, eg, radiative transport; but as with Gadget, these are very seldom publicly released.

7.4. *Commonalities*

We see that for a community code to be successful, it needn't be especially large; the development team size can vary, as can the distribution method and the frequency (if at all) of updating the code. All that seems to be required is that: the code be high quality; moderately easy to use and understand, including having appropriate documentation; come with tests to make sure it is installed properly; and come with all the tools needed to perform the necessary workflow (eg, create initial conditions, visualize/analyze results). Most importantly, however, the code must address a real research need of a broad community, and be of high quality; and high quality means supporting person-years of effort. These are for future projects person-years involving different persons in a team with a combination of capabilities and expertiese as described above. Such teams can only work with the required stability if they are matched with a corresponding funding instrument.

8. A CODE PROGRAM FOR ASTRONOMY

Several ways are conceivable to start to remove the present limitations in Canadian simulation and application code developement and maintenance. Whatever the instrument, we recommend the following three-part mission: to (a) build up the simulation and data analysis capability that is needed to take full advantage of the increasing fidelity of HPC; and (b) enable large-scale application code development, maintenance and user support, (c) provide the Canadian astronomical community with the software it needs, directed through a broad advisory and consultation program.

The result of such a program would be highly qualified (and widely employable) personnel in the increasingly important areas of simulation science and large-scale analytics, and software that serves the needs of the Canadian astronomical community. A specific plan for executing such a program would be to fund 1-2 teams in year one each involving 5-7 interdisciplinary members, from undergraduate and graduate students to postdocs and small buy-out for a faculty team leader, preferably at an institution which already has a relevant accademic program; this would be a competitively allocated process. Over several years, this process would ramp up to 3-5 teams across Canada. These teams would generate codes with support for the duration of the program, including documentation and user support, to be deployed on Compute Canada systems. They would focus their work on things that are not readily available from other sources, just as instrument teams propose projects to be decided on through community consultation. There would be recompetition every five years with the intention to provide competitive continuity. The code teams and user groups could be loosely joined by some national virtual institute, and if successful this strategy may serve as a template to be employed by other disciplines. Besides the immediate benefits of software development and training, this would help lay out the pathway to academic, scalable peta-scale computing in the next decade.

9. CONCLUSIONS

In this white paper we describe the role of simulation and application code developement and maintenance as a criticial component in the national astronomy and astrophysics landscape. Presently this component is un-

derdeveloped. This situation poses a risk for the healthy progression of the field in the next decade. It will take the concerted effort of researchers, teachers and practitioners on the ground as well as changes to the funding environment in order to address this situation. Over the past decade an important step has been made with funding priorities for computational hardware. These developements have to be sustained and need to be complemented now by comparatively modest investments in a science oriented systematic software development program. The benefits of such a program would reach far beyond the immediate needs of the astronomy community.

REFERENCES

Weiner, B. *et al.* 2009. "Astronomical Software Wants To Be Free: A Manifesto", In Astro2010: The Astronomy and Astrophysics Decadal Survey, arxiv.org:0903.3971.

Dursi, L.J. *et al* 2010, *Computing, Data, and Networks LRP2010 Whitepaper*, `http://casca.ca/lrp2010/Docs/LRPReports/CDandN_WP.pdf`.

Colella, P. & Woodward, P. R. 1984, Journal of Computational Physics, 54, 174

Compute Canada Midterm Report, `https://computecanada.org/?pageId=887`

Brunini, A. 2006. "Retraction: Origin of the obliquities of the giant planets in mutual interactions in the early Solar System". Nature 443, 1013. doi:10.1038/nature05298.

Petsko, G., 2007. "And the second shall be first". Genome Biology, 8:103 doi:10.1186/gb-2007-8-2-103.

Hall BG, Salipante SJ, 2007. "Retraction: Measures of Clade Confidence Do Not Correlate with Accuracy of Phylogenetic Trees". PLoS Comput Biol 3(7): e158. doi:10.1371/journal.pcbi.0030158.

Raymond, Eric S., 1999. *The Cathedral & the Bazaar*. O'Reilly. ISBN 1-56592-724-9 .

Wheeler, D., 2004. "More than a Gigabuck: Estimating GNU/Linux's Size", `http://www.dwheeler.com/sloc`.

Springel, V., 2005. "The cosmological simulation code GADGET-2", MNRAS **364**, 1105-1134.

Ferland, G. J., *et al.*, 1998. "CLOUDY 90: Numerical Simulation of Plasmas and Their Spectra", PASP **110**, 761-778.

Fryxell, B. *et al.*, 2000, "FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes", ApJSS **131**:1, 273-334.

Stadel, J. G., 2001. "Cosmological N-body simulations and their analysis", Ph.D. thesis, University of Washington.

Teyssier, R., 2002. "Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES", A&A **385**, 337-364

Hayes, J. C., *et al.*, 2006. "Simulating Radiating and Magnetized Flows in Multiple Dimensions with ZEUS-MP", ApJSS, **165**:188:228

O'Shea, B. W., *et al.*, 2004. "Introducing Enzo, an AMR Cosmology Application", eprint arXiv:astro-ph/0403044

Woodfield, S.N. (1979). "An experiment on unit increase in problem complexity", IEE Transactions on Software Engineering, **5**:2, p.76–79.

Thomas, W.M., Delis, A. & Basili, V.R. (1997). "An analysis of errors in a reuse-oriented development environment", Journal of Systems and Software **38**:3 pp 211-224.

Boehm, B.W. "The high cost of software". In Proceedings of Symposium on High Cost of Software, Monterey, Calif., 1973, pp. 27–40.

Graham, S.; Snir, M. & Patterson, C.A. (2004),"Getting Up To Speed: The Future of Supercomputing", Technical report, [U.S.] National Research Council.

Almeh, R., 2007. "Investigating the effects of novice HPC programmer varations on code performance". M.Sc. thesis, University of Maryland, College Park.

Hochstein, L., 2006. "Development of an empiral approach to building domain-specific knowledge applied to high-end computing". Ph.D. thesis, University of Maryland, College Park.

Hochstein, L. and Shull, F. and Reid, L.B. (2008). "The role of MPI in development time: a case study", in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press, pp. 1–10.