

Nbody6++

1. Scenario Overview

Give an introduction to the scenario, making the reader familiar with goals, main purpose and functional overview.

1.1 Background and Purpose

NBODY6++ is a member of a family of high accuracy direct N-body integrators used for simulations of dense star clusters, galactic nuclei, and problems of planet formation. It is a special version of NBODY6 optimised for massively parallel computers. NBODY1 to NBODY6 are provided free of charge for single workstations by

Sverre Aarseth, Cambridge (UK), while NBODY6++ is provided free of charge by our group at ZAH. The codes are used by a loosely knit international user group, who is provided with updates and bug fixes by Sverre Aarseth (NBODY1-6) and Rainer Spurzem (NBODY6++). The code is a Fortran code which has grown from its beginnings over more than 30 years. It is definitely not modular, it is not easy to use without support. However, in exchange for that it provides unique capabilities, where competing codes (there are only very few) still struggle to catch up. The unique features include a carefully balanced network of algorithms to treat close few-body encounters and stable and metastable few-body subsystems in a self-regulated way, keeping accuracy and efficiency, covering a very large dynamical range. The key are the regularisation methods, first applied by Kustaanheimo and Stiefel in the 60s, and much improved by Aarseth, Mikkola, Mardling and others.

Some of the most important applications are simulations of rich open and globular clusters with a large number of binaries and galactic nuclei with single and binary black holes. The dynamic range required for the simulation ranges from 10^{**8} years (relaxation time) over 10^{**6} years (typical orbital time of one star in the cluster) to several days (periods of most compact binaries). These data demand a high-order integration scheme, with regularisations, and an Ahmad-Cohen neighbour scheme. Only ten years ago the maximum number of particles we could simulate with the supercomputers was 10^{**4} , and only due to the use of special purpose GRAPE accelerator boards we can now with our recent parallel GRAPE clusters tackle the one million body limit.

Typical goals in our use of NBODY6++ are
(these are examples, the codes are physically very general,
just gravitating N-body problem, and there are many other goals):

- + for globular clusters: predict reliable theoretical rates of binaries involving compact degenerate objects - this is important to understand the role of globular clusters in galaxy formation (compare with X-ray binaries in the field for example)
 - and to predict the role of globular clusters as sources of gravitational waves (merging neutron stars, chirping binaries emitting in very low wavelengths).
- + for galactic nuclei: the dynamical interaction of single and binary black holes with stars and with other black holes coming in from inspiral of star clusters or major mergers is studied as a function of their host galaxy; the models aim to predict the rates of massive black hole mergers or ejections in the context of cosmological galaxy formation.

1.2 More information

<http://www.nbodylab.org>
<http://www.sverre.com>
<http://www.astrogrape.org>
<http://www.ari.uni-heidelberg.de/grace>
<http://www.ari.uni-heidelberg.de/mitarbeiter/spurzem/index.html>
<ftp://ftp/pub/staff/spurzem/nb6mpi/README>

Suggested Readings:

S.J. Aarseth, Gravitational N-body Simulations - Tools and Algorithms
Cambridge Univ. Press, 2003.

Spurzem R.: 1999, 'Direct N-body Simulations', Journ. Comp. Appl. Math. (JCAM)
109, pp. 407-432

E. Khalisi, C. Omarov, R. Spurzem, M. Giersz, D.N.C. Lin. Collisional
Dynamics of Black Holes, Star Clusters and Galactic Nuclei. In:
E. Krause W. Jaeger, M. Resch (eds.) High Performance Computing in
Science and Engineering '03, Springer Verlag, pp. 71--87, 2003.

----- 2. Current Scenario description -----

Describe the current scenario including as much details as possible.

Note, there is no special section to describe workflows/pipelines or
details about a phased execution of a program. If your scenario is a
workflow/pipeline OR your application is executed in several phases,
describe EACH step/part covering the sections 2.1 - 2.8. In addition
you must describe how these steps/parts are interrelated to each other
(in Section 2.8).

2.1 Environment

2.1.1 Hardware

Describe the hardware resources that are currently used.

- Processing

NBODY6++ simulations mostly run as parallel MPI runs on
supercomputers (IBM Jump in Juelich, Blue Gene/L, CRAY XD1)
and on Beowulf PC clusters such as the cluster provided within
the D-Grid from ZAH. Typical useful processor numbers are 32 to 512,
depending on hardware and problem size.

- Storage

local disk only, application is not very disk or data expensive
(exception: special socket used optionally for online remote
visualization, and also for remote control, the latter being
an experimental feature).

- Network

network bandwidth and compute speed must be in a balance. For a

given physical problem size N the optimal processor number becomes larger for larger network bandwidth. Presently we require Myrinet or Infiniband (approx. 10 Gbit/s) node-to-node for the largest applications. With smaller bandwidth smaller processor numbers must be used. Primarily bandwidth dependent, latency is dominant for smaller N (about 10^{**4}), which some users need. Low Latency (few microsec of Myrinet or Infiniband) is needed. We know that with present bandwidth and latency data a distributed simulation between different sites makes no sense. But this will change in the near future and e.g. between supercomputing centres, within the D-Grid? The traffic volume say for a few 10^{**5} particles is several 100 MBytes/sec. in and out synchronously in a ring-like (systolic) communication pattern.

- Describe special hardware or other hardware resources that are relevant for the scenario.

Optionally special purpose cards are used (GRAPE) which accelerate gravitational force computations by a factor of about 50-100 compared to a present Pentium 4 Xeon. In the near future these will be complemented or ultimately substituted by reconfigurable FPGA boards, which are more flexible, and become as fast as GRAPE. GRAPE is so fast, that at a given network bandwidth, the number of processors one can use drops dramatically. A job on 32 processors with GRAPE competes with another one on say 512 nodes without GRAPE (as e.g. in the supercomputing centres).

2.1.2 Software

- Describe used software such as operating system, software libraries, the use of GRAPE requires an interface library, with subroutines called from the NBODY code. The same holds for the reconfigurable boards.
- What programming language is used and what compiler/linker version is required?

The NBODY code is programmed in Fortran77 with MPI-calls. The GRAPE interface library is coded in C. Compilers such as g77, gcc, pgf77, pgf90, ifort (Intel) have all been successfully used.

- How is the program deployed?
 - . source code in a tar.gz form. Users keep their own variants.
 - An authorised version of NBODY6++ is on the cited ftp-location at ARI and updated from time to time. We do not have a cvs system.
 - Bug fixes and reports are communicated between updates via an email communication list.
- How is the program compiled?
 - . make ; the Makefile is manually adapted by us for different target architectures. There is no configure script or similar thing (yet?)
- State the program license and any commercial 3rd party licenses.

Apart from compiler licenses, no other licenses are involved in building and running NBODY6++ executables.

2.2 User Interaction

Describe the user interaction necessary for starting the program and additional interaction with a running program.

2.2.1 Initiation

- Describe how the program is started and any steps needed before the actual initiation.

Our resources are partly managed by batch systems (eg. PBS) or directly accessible by users via manual start of a script executing an mpirun.

Users describe their NBODY6++ job in a batch or interactive script

- executable name (supplied by the user)
- parameter file, data input files (supplied by the user)
- number of processors
- move output files of job from their generic names to unique names (the latter supplied by user)
- requested runtime (if batch)

and then send it off to the queuing system or start it interactively.

- compilation (cf. Section 2.1.2),
if a certain system is been configured in the Makefile manually (this is typically done by us at ARI-ZAH) just make should suffice with the proper target name programmed into the Makefile.

retrieve/copy/generate data/files (cf. Section 2.3.2),
parameters (cf. Section 2.3.1)

three types of parameter / input files may be needed:

- (i) parameter file with control parameters
 - (ii) data file with N-body configuration from other applications in a standard format to start with.
 - (iii) data file with N-body configuration from earlier NBODY6++ run in specific binary format.
- (ii) and (iii) are alternatives, they exclude each other usually

- Where is the program executed?

typically via a batch system on a PC cluster or supercomputer. Optionally (by a target in the Makefile) also single CPU runs can be compiled and started from the same source. Compiler directives are used to exclude the compilation of MPI calls in such a case. This code has not yet been run on a grid, but similar codes have been tested on a grid from our colleagues in Amsterdam.

- How is the program initiated?
. command line, web portal, ...
Usually command line, although we also use Unicore to submit it at NIC in Juelich.

2.2.2 Monitoring/Steering/Visualization during the run-time of the program

Definitions:

Monitoring - Information retrieval regarding the state of the program. For example, "Program is running" or more application specific information such as "Current simulation iteration is 42".

Monitoring - the Unix stdout listing contains many data for monitoring. There are typical lines occurring in predefined intervals, which one may get by a Unix grep command easily, such as energy checks, checks of time step distributions, error conditions (if they occur), step numbers and wall clock times, and many more.

Steering - Remote alteration of the programs state. For example, program stop or application specific information like "at iteration 42, set parameter x = 78".

at present there is only one such possibility, this is the possibility to create a file named STOP in the run directory, which will cause a smooth FORTRAN Stop. We are working in experimental version where there is a special socket interface, through which parameters can be varied at runtime (not in standard version yet).

Visualization - Remote access of the application data needed for visualization of, for example, a simulation.

Again, in an experimental version where there is a special socket we write out data for visualization (xnbody, in cooperation with ZAM Juelich). The present standard version only works with large data files which are post-processed for visualization.

- What type of data is produced by the program during run-time used for monitoring/steering/visualization?
 - . log-files, stdout/err, intermediate results, ...
 - . Unix stdout, which is used for monitoring (unit 6, see above)
 - . several files for post-processing and visualization (formatted and unformatted)
 - . binary files containing common block dumps for restarts.

- What methods/tools exists for accessing data produced by the program during run-time?
 - . which protocols are used (SSH, GSISSH, SOAP, HTTP, arbitrary, ...?)
 - . IP sockets, web portal, shell, program exported API, ...
 - . shall these interfaces be used for automated monitoring/steering?

an IP socket is in experimental stage, and yes it should be used automated monitoring/steering/visualization.

- Does your application support any standard for monitoring/steering?

no, user specific, work still under progress

- Describe any security measures related to program access for monitoring/steering/visualization.

experimentally we have used the communication via an ssh-tunnel.

- Who can access the running program OR run-time produced monitoring data?

. dynamic groups (defined by the user/owner),

- From where can run-time produced monitoring data be accessed?

. specific IP/netmask (experimental)

- How is the program termination detected?

Presently by mail or manual online monitoring. The stdout file contains a diagnostic line to detect program termination and its conditions.

- How much monitoring data and how often is monitoring data transferred during a program run (min/max/avg)?

- . all monitoring data are transferred, but they are typically small, even for large runs (order one MB).
- . all output data files are usually not automatically transferred, they stay remotely, the user manually acquires those needed for visualisation post-processing.

- Does your program generate metadata and stores this externally (e.g. in a catalog)?

no, not applicable.

- Who accesses this metadata? From where? Does your program access metadata generated by other programs?

not applicable

- How many executions/jobs must be monitored/steered in parallel? By how many users?

2.3 Input

2.3.1 Parameters

Describe the program parameters in detail.

There are three classes of program parameters. All of them are provided as real or integer numbers. The following description is exemplary not exhaustive.

- (i) job control data: e.g. maximum physical or wall clock time for this job
maximum number of steps
parameters which determine the frequency of

diagnostic

output on stdout
parameters which determine the number and kind of
different output data files other than stdout
parameters defining whether we continue a run or
create a new initial model

- (ii) algorithmic parameters: tolerance parameters for time step adjustments
parameters determining the balance of different
interconnected parts of the algorithm, e.g.
the neighbour number for the neighbour scheme,
parameters determining when to use

regularization

of close encounters

- (iii) astrophysical parameters: how many particles? what initial model?
selection of physical situation (is there
an external tidal field? is stellar

evolution

taken into account?)

2.3.2 Input data

- How is the input data prepared?
 - (i) copy/move from a resource,
 - (ii) pre-execution data generation,
- Where is the input data stored? Describe all central and distributed locations.
 - . local file system
- Are file-names known in advance (before the program is started)?
 - . they are known and defined by the code, but usually we give afterwards unique names to the files to identify their origin (e.g. date and NQS batch job reqid)
- Are data locations (directory, server, ...) known in advance?
 - yes
- Describe the different ways data is accessed.
 - local disk
- Non-file based data access (XML, database, ...) should include
 - does not apply
- How much data is accessed at each run?
 - . number of files/data sets: min/avg/max 1/1.5/2
 - . total-size: min/avg/max, few kB/50 MB/100 MB
- Is it possible that a data set/file is accessed multiple times over a short period of time?
 - No all data files are accessed sequentially by a single thread (the root thread). Since computation is so overwhelmingly large compared to I/O this is not a problem.
- How many users are using the same data simultaneously?
 - Are these users geographically distributed?
 - Data are only used by one job at a time.
- Elaborate on the use of metadata related to input data.
 - right now we are not able to define a metadata structure
 - we are discussing this for the future.

2.3.3 Additional Notes

Describe any additional information regarding the input data which has not yet been covered.

2.4 Output

This covers what data products are generated (INTERMEDIATE and FINAL results), where they are generated and how they are handled after the program finished (transferring data or removing it, ...).

2.4.1 Output data

- Where is the output data stored? Describe all centralized or distributed

locations.

- . local file system

- How is the output data structured?
 - . a collection of single files
 - . data formats: plain text, and some are binary
- Describe what happens when the program finishes? How are the results used?
 - . the stdout output listing goes immediately to the user for check
 - . a couple of text and binary files are stored and later retrieved by the user for post processing and visualization. They can be deleted at the computing site and are not needed for restart.
 - . two binary files are complete dumps for restarting the run. They can remain local, unless the job is to be continued at another location.
- Describe the different ways data is created/changed.
 - . write by fortran write statement
- Non-file based data access (XML, database, ...) should include description of not applicable
- How much data is written by the program at each run?
 - . number of files/data sets: min/avg/max 2/5/10
 - . total-size: min/avg/max, 1 MB/100 MB/several GB
- Describe the parameters which influence the amount of data and number of files/data sets generated.

see above input parameters. Some parameters just decide whether a certain file will be created at all or not. Other parameters determine how often information will be written, i.e. they determine the size of the files.

- Elaborate on the use of metadata related to output data.

again, we do not have a good idea about metadata definition for this usecase. Typically information about the run is stored in the stdout log file, and other output file names keep the creation date and some identification number.

2.4.2 Additional Notes

Describe any additional information regarding the output data which has not yet been covered.

The program is able to work in two computing modes, one is massively parallel (one application running on many processors), the other is job farming (many physically identical runs but with different random number initialisations). In the job farming case all output files will be produced as many times as there are instances of the job (processors) - and the file names will reflect the originating processor id.

2.5 Information resources

Give a summary of each information resource that is accessed by the program. Include information about data input/output, locations,

access methods (XQuery, SQL, ...), security related restrictions, search of metadata (exact key search i.e. "ABC", range queries i.e. "AB*", ...)

The program first reads the input parameters from a small text file (local). Then it reads input data from another text file (local) or in case of a restart from the binary file (local) produced in the previous run. Then it writes text data and binary data in regular intervals to local files.

In case of our experimental version which includes runtime job monitoring and steering the job very frequently writes data to an IP socket, which has to go out through an ssh-tunnel to the user.

2.6 Data Stream Management

Definitions:

Data Stream - intermediate results can be processed by the subsequent processing module before the current module has processed the last element of the input.

- Can single operations be performed on any compute node or do they need special hardware or software?

Yes and No, both is right: the code runs on standard parallel systems as well as on GRAPE systems (to be honest right now we have not one single source code system to do this, but two realizations of the code, one for GRAPE one for standard parallel machines). We work on this...

Some of the physical input parameters (number of particles, integration time needed) determine whether the use of the special purpose hardware GRAPE is

- (i) inefficient (for small problem size)
- (ii) possible but not strictly required (intermediate problem size)
- (iii) necessary, otherwise the job is impossible

Similarly an optimal processor number can be defined from the parameters of the request and the hardware data (CPU speed, communication bandwidth, is GRAPE there or not).

- Are data exchanged between distributed parts of the application? does this happen at the beginning, during run-time or at the end?

It is of utmost importance to the efficiency of the application that all nodes can communicate with their neighbours with maximum bandwidth through most of the time. The best overall performance is achieved if this communication time is balanced against the computing time. Our communication topology is a ring, but alternatively also a tree is implemented. On low latency systems both are equivalent.

- Are operations compute intensive?

Yes, on a single CPU. For parallel runs a balance between communication and computation has to be sought. The present version of the code uses blocking communication. In the future also non-blocking communication may be used.

2.7 Resource Security and Access Restriction

We are happy with user based access to all material.

2.8 Additional Information

Give additional information not covered by the sections above.

Typically individual jobs are of the same size and have similar requirements. Each job has wall clock times of order a day (supercomputer centres) or few days (our own cluster). Linear sequences of such jobs, consisting of order ten restarts make up a typical physically complete result.

3. Future Scenario and AstroGrid-D Usage

- we expect from the grid application:

+ a more user friendly interface to start jobs without having to worry about selecting the hardware (GRAPE or not?) or the optimal processor number on different systems. (by the way: the code has unique capabilities, but suffers from a very difficult and antique style of user interface. we hope that the grid application also gives us a momentum to improve that on the user side). There is a very particular aspect that this code package has been developed continuously in a simple way of coding by very few people over about 30 years. To preserve the knowhow and bring it to a more modern level of application we need the momentum of a project such as the grid application.

+ to provide the functionality of extremely performant special purpose hardware (GRAPE and new reconfigurable FPGA hardware) to a wider circle of users without having to worry about technical details.

+ to be in a better position to use our code in European and international grid networks.

3.0 General goals

make better use of distributed resources (e.g. GRAPE hardware),
make it more available and visible for more users.
disseminate and preserve specific knowhow in the
present codes for the future use.

prepare the software environment for using the grid for one
job on more than one site (although present bandwidths
do not make this profitable now, we expect things to
develop fast, and we want to be in a good position for
groundbreaking large applications across sites once the
hardware allows this with reasonable bandwidth).

3.2 Environment

- Are there any constraints due to your participation in other projects or

international collaborations?

we are expecting to start in the DEISA project this year
we want to provide compatibility with other European and international sites if possible (Cambridge, Amsterdam, Marseille, and also sites in the US and Japan).

3.3 User Interaction

- Which parts should be automated?
 - . resource selection,
 - . data transfer before initiation and after termination,
- Which user interface are you planning to use?
 - . WWW portal, ...
- Are you planning to use any standard for application monitoring/steering?
 - . Do you want to use such standards in collaboration with the DGI or the other communities OR will you develop your own methods?

I hope we can comply with DGI standards. Right now we use our own method developed together with ZAM Juelich.

- Aspects of a Portal / WWW based interface:
 - . Which portal features are mandatory/optional (e.g. credential management, job management, job monitoring/steering, data transfer, ...)?

yes all of them.

- . How are user managed? Where is information about users defined / stored?

local machines, user authentication

- . Which authentication/authorisation methods are needed ?

normal unix user authentication

- . Do you want to access specific data services (web services, databases, etc.) via a portal?

no

- . Are there any existing programs, on which the user interface should be based OR which should be replaced by the portal?

At NIC Juelich we use the Unicore web portal. So this would make the transition easier. But we do not insist on a specific portal.

- . Should there be a central AstroGrid portal OR do you want to set up a portal server for each scenario/application ?

I do not mind a central portal.

- . Does the scenario require any special interfaces OR is it sufficient to use generic interfaces ?

Well I don't know exactly now.

- Aspects of a generic Grid Application Programming API (GAT)
 - . Which GAT functionality would you like to make use of (eg. job submission, file handling, resource brokering, etc.) ?

yes all of them.

- . What programming languages must be supported ? Which platforms ?
fortran, c, PC clusters, IBM or CRAY parallel machines, GRAPE hardware.
- . Which Grid Middleware should be supported (Globus, Unicore, gLite, etc.) ?

As I said, now we use Unicore at some sites, so we might be happy to continue that, but we may be flexible if there are good reasons. We will also explore Globus, where there is no experience yet.

- . For specific GAT functionality, which protocols/packages/tools should be supported ?
eg. for job management: clusters with PBS, SGE, Condor

right now we use PBS.

3.4 Input

- Do you handle input data manually or do you need an automated management of data?

manually, except the binary output input files for job chains.

3.5 Output

- Do you handle output data manually or do you need an automated management of data?

manually, except the binary output input files for job chains.

3.6 Additional Information

- How long (avg) does the scenario execute (minutes, hours, days)? Do you aim at a specific speedup?

days at minimum. No speedup - because we already use parallel execution.

- How often will the scenario be executed?

I can't say this - it depends on the number of projects and users and the distribution of them to resources. Or do you mean for one particular physical run - there as said above about ten jobs in a chain is a typical values, each job taking days on a multi-CPU system. (32 up to 256 processors).

- Which restrictions of the current approach (as described in section 2) do you want to overcome?

better portability and user friendliness of the run used on different hardware, better use of resources, preparation for large parallel jobs across more than one site.

4. Bigger Picture for the far future

4.1 Organization of Multiple Runs

Maintain a list of all simulations, to repeat simulations with a different binary, with different input data, to check if a program was already executed with a certain set of parameters/input data,

Yes, at this stage we would start to require and define more metadata. Right now all is done manually by the users and other problems of the implementation appear more important to us. But in the far future collections of runs could be linked (parameter studies, statistically independent runs for job farming).

4.2 Handling relationships between data products

For example, store metadata on how a data product was generated (from which input data, by which program, with which parameters) and how it can be used by others.

4.3 Constructing More Complex Runs

Provide more varieties of parallelization algorithms (blocking and non-blocking communication, systolic or tree structure) and algorithms for different type of hardware (GRAPE, standard, new FPGA hardware) in one environment.

The code is also constantly being developed to include more physical features. So the grid application should be able to grow with this.