PPPL- 4279

PPPL- 4279

Optimizing Extender Code for NCSX Analyses

M. Richman, S. Ethier, and N. Pomphrey

January 2008

Prepared for the U.S. Department of Energy under Contract DE-AC02-76CH03073.

Optimizing Extender Code for NCSX Analyses

M. Richman[1], S. Ethier[2], and N. Pomphrey[2]

[1] The College of New Jersey, Ewing NJ 08628, USA
[2] Princeton Plasma Physics Laboratory, Princeton University, Princeton NJ 08543, USA

Abstract

`Extender`[1] is a parallel C++ code for calculating the magnetic field in the vacuum region of a stellarator. The code was optimized for speed and augmented with tools to maintain a specialized NetCDF database. Two parallel algorithms were examined. An even-block work-distribution scheme was comparable in performance to a master-slave scheme. Large speedup factors were achieved by representing the plasma surface with a spline rather than Fourier series. The accuracy of this representation and the resulting calculations relied on the density of the spline mesh. The Fortran 90 module `db_access` was written to make it easy to store `Extender` output in a manageable database. New or updated data can be added to existing databases. A generalized PBS job script handles the generation of a database from scratch.

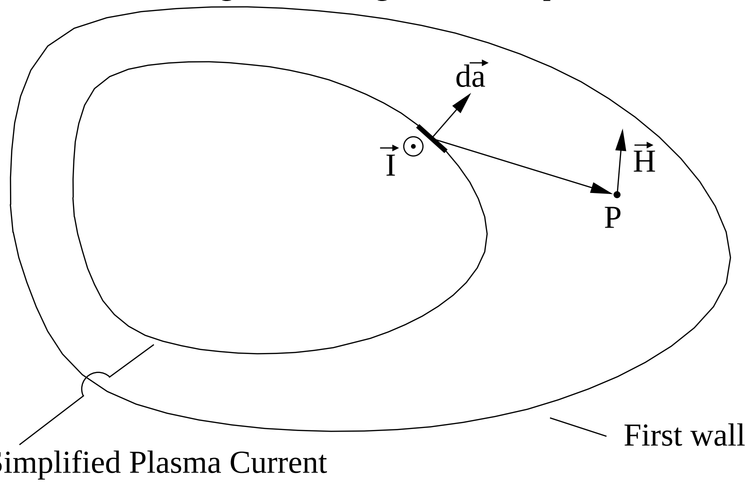
1 Background

1.1 Extender

`Extender`[1] is a parallel C++ code, originally written by Michael Drevlak of IPP Garching, for calculating the magnetic field in the vacuum region of a stellarator device. It operates in two phases; current simplification and field calculation. The plasma equilibrium is generated by `Vmec`[2] and stored in a file (`wout.ext` passed to `Extender` at runtime. The field contribution due to the plasma could be calculated by integrating over the volumetric currents within the plasma, but the computation would be too demanding. Instead, the internal plasma currents are represented by a single sheet current lying on the outer surface of the plasma (Figure 1). Because the sheet current is relatively smooth, it can be represented accurately by a spline. The sheet current is evaluated at points on a mesh on the plasma surface, and splines are generated.

After the sheet current splines are prepared, the field due to the plasma currents at a field point outside the plasma can be evaluated by adding up the field contributions from each sheet current element on the plasma surface. The integral is adaptive. For points closer to the surface, $d\vec{\mathbf{a}}$ must be smaller in order to achieve numerical accuracy. For points more distant from the surface, less points are needed.

Figure 1: Schematic diagram of integral with simplified surface current



The calculations of $\vec{\mathbf{H}}$ at each field point are independent of one another. For this reason, `Extender` is a prime candidate for parallelization.

1.2 Parallel Algorithms

It is assumed that there are N_w independent calculations, or tasks , to be completed, and there are N_p processes running on N_p processors available to do the computation. Two work distribution algorithms were considered:

- *master-slave algorithm*: One process, p_0, is devoted to distributing the work. The others, $p_1 \ldots p_N$, request work from and return results to p_0.
- *block-distribution algorithm*: The set of calculations are divided into blocks of size $\sim N_w/N_p$. p_0 may be responsible for collecting results for the final output, but the processes need not communicate during computation.

There are advantages and disadvantages to each algorithm. The correct choice depends on the nature of the calculations and the parallel computing network being used. In the master-slave scenario, one disadvantage is that p_0 does no useful computation, which effectively reduces N_p by one. Furthermore, at least some inter-process communication is required for the master process to direct the others. Communication overhead can be especially significant on a workstation cluster, where communication is done via high-latency ethernet interconnect and TCP/IP protocol. These penalties are offset by the ability of this algorithm to handle variable amounts of computational work between processors, leading to an improved load balance for p_1 thru p_N.

The block-distribution algorithm avoids some of the disadvantages of the master-slave algorithm. Each process can independently determine the block of calculations it should work on based on its id. If $N_w = Q \cdot N_p + R$, then p_0 thru p_{R-1} take blocks each of size $Q + 1$, and p_R thru p_N take blocks of size Q. Therefore, every process does useful computation, and no inter-process communication overhead is introduced during computation. However, the efficiency of the block-distribution algorithm depends on the blocks taking equal time to complete. If the blocks are uneven, then some processes finish before others; this is *load imbalance*. If there is no way to ensure that the inter-block differences are negligible, a master-slave algorithm can be more efficient.

2 Speed Optimization

`Extender` was optimized for speed by two approaches. The first was to ensure that the code was parallelized efficiently. The second was to represent the plasma surface as splines rather than the original Fourier series. The latter change, which was suggested by a `gprof` profile of the code, had a significant effect on performance.

2.1 Efficient Parallelization

`Extender` originally used a master-slave parallel algorithm. The implementation worked as follows: Each process computes the surface current splines separately. Then, p_0 has a list of field points, and p_1 thru p_N stand in a queue waiting for assignments from p_0. When the process at the head of the queue is assigned a field point, it leaves the queue, computes the magnetic field at that point, and returns to the tail of the queue where it waits for its next assignment. p_0 is in a loop, where it assigns one field point at a time to the current head of the queue during each iteration until the set of points is exhausted. After all the calculations are done, p_0 loops through the slave processes, collecting their results.

Under this master-slave implementation, there are several inter-process communications for each field point. It was hypothesized that this lead to some inefficiency that could be countered by using a block-distribution algorithm. In the new implementation, each process had a copy of the list of points. Each determines independently the block of points it should work on. When every processes is finished, p_0 gathers the results from all the processes including itself.

Timing the computation phase under each implementation shows that the two algorithms yielded similar performance. The two algorithms experienced different setbacks of similar total cost to performance.

As discussed above, the master-slave algorithm results in some load imbalance because p_0 does not participate in the main calculation. While this was the most significant loss, `Extender`'s implementation also suffered from excessive inter-process communication. An efficient master-slave algorithm could instead assign large chunks of work at the beginning the computation phase, and then assign the rest of the work in blocks of decreasing size until all work was assigned and completed. The existing implementation assigns each point one at a time. Not only does this mean that thousands of excess `MPI` communcation calls are made, but it is more likely for a slave process to be stuck waiting in the queue while p_0 assigns work to those processes closer to the beginning of the queue.

The block-distribution implementation suffers from a different kind of load imbalance. Because calculation of the magnetic field at a point uses an adaptive integral, each field point takes a different amount of time to complete. This variation means that each block takes a different amount of time to finish. As a result, some processes are finished sooner than others. These processes remain idle while others are busy. Ideally, some work should be moved from the busy processes to the idle ones so that every process is being used. In the case of `Extender`, the load imbalance the master slave algorithm incurred by communication overhead and not using p_0 is comparable to that incurred under the block-distribution algorithm by not using all processes efficiently at all times.

In summary, for the main calculation, using a block-distribution algorithm did not have a significant effect on execution time. Nonetheless, the analysis was productive. We needed to verify that we were using an efficient algorithm, rather than simply accept the existing one. However, because the block-distribution method was simpler to implement, it was straightforward to apply it to additional parts of the program. Setup time was thereby decreased by a factor of $10\times$. This speedup in the setup phase was, however, offset by the changes discussed in the next section.

2.2 Optimizing Representation of Plasma Surface

Examining the parallel algorithm in `Extender` did not present opportunities for large speedups. To identify additional optimization possibilities, the code was profiled using `gprof`. The process was carried out as follows:

1. Identify a function that represents a significant portion of the execution time.

2. Discern the purpose and method of implementation of the function.
3. Try to optimize the function itself.
4. Try to optimize routines that call the function in order to limit the number of function calls.
5. Test any new methods for timing and precision.

In the `Extender` profile, one function stood out: `v3 Surface_f_c::operator() (double const&, double const&, long const&) const`. This function prototype indicates the overloaded operator `()` for the class `Surface_f_c`. An example call to the function might be "`my_surface_f_c (0.1, 0.1, 32)`". This function represented 58% of the execution time. Therefore, it was an excellent candidate for optimization.

Examination of the function code and comments revealed the purpose of the function. The surface integral is carried out by iterating over two-dimensional coordinates (u,v) relative to the plasma surface. However, the absolute coordinates (x,y,z) are required for calculating the field due to each differential sheet current. The function suggested by profiling performs the coordinate transformation.

The position of the plasma surface is stored using Fourier series. `Surface_f_c::operator()` first computes the cylindrical coordinates (r,ϕ,z) of a point on the plasma surface, and then it converts to Cartesian coordinates. For the first conversion, the expressions

$$r = \sum_{i=0}^{N} c_{r_i} \cos[2\pi(m_i u + n_i v)]$$
$$z = \sum_{i=0}^{N} c_{z_i} \sin[2\pi(m_i u + n_i v)]$$

are used, where (u,v) are the poloidal and toroidal angles over a single period of the stellarator, and c_{r_i}, c_{z_i}, m_i and n_i are Fourier coefficients. Then, using $\phi = v/n_{\mathrm{fp}}$, where n_{fp} is the number of field periods (3 for NCSX), the Cartesian coordinates are

$$x = r\cos\phi$$
$$y = r\sin\phi$$
$$z = z.$$

Fourier series provide an efficient way to store plasma surfaces. However, they are slow to use during the main calculation. There are too many calls to the system's `sin()`

and `cos()` routines. There is no way to achieve a significant speedup by optimizing this function alone. To minimize execution time, an alternative representation of the plasma surface is needed.

The plasma surface can be represented using splines. Plasma surfaces are generally very smooth. Therefore, provided that a sufficiently dense mesh is used to generate splines, there should be no significant loss in precision.

It was not difficult to implement these "surface splines" in `Extender`. In fact, they are generated in much the same manner as the "sheet current splines" in the original code. Essentially, an array of values for points in a toroidal grid is passed to a routine that creates a new spline. Three splines are created this way: one for x, y and z. These splines may thereafter be used in lieu of the Fourier series to obtain the Cartesian coordinates of a point on the surface.

2.2.1 Results

Error analysis. After introducing the surface splines, the first task was to verify that no significant errors were introduced by simplifying the representation of the plasma surface. The original code was used as the given from which the new code might deviate. In the error measurements, the most stringent criterion was used: for a given set of calculations done by both the original and new codes, we sought to minimize the *greatest single deviation* between the results of the modified and original code.

First the plasma surface representation itself was examined. The results are summarized in Figure 2. It was found that for a toroidal grid of $N = 200$ points in each dimension, the greatest error in the position of the surface was only $\sim 1 \times 10^{-7}$ m. Most of the points were represented even more accurately. This suggests that the surface can be represented very accurately with splines.

After verifying that the surface representation itself was very accurate, the final results were examined. The final output, a set of magnetic field $\vec{\mathbf{H}}$ values at the specified locations, of the new code was compared to that of the original code. The results are summarized in Figure 3. It was found that for a toroidal grid of $N = 200$ points in each dimension, the greatest error in the $|\vec{\mathbf{H}}|$ was only $\sim 5 \times 10^{-7}$. Again, the values were even more accurate at most field points. This indicates that representing the surface with splines does not have a detrimental effect on `Extender`'s output.

Performance data. Using the new code, some things were immediately clear, qualitatively: `Extender` ran faster; the setup phase took longer because of the time needed to set up the surface splines; and the computation phase ran faster because of the use of the surface splines. To quantify these results, the original and new codes were modified to output

Figure 2: Greatest residual in surface position vs. spline generation mesh density

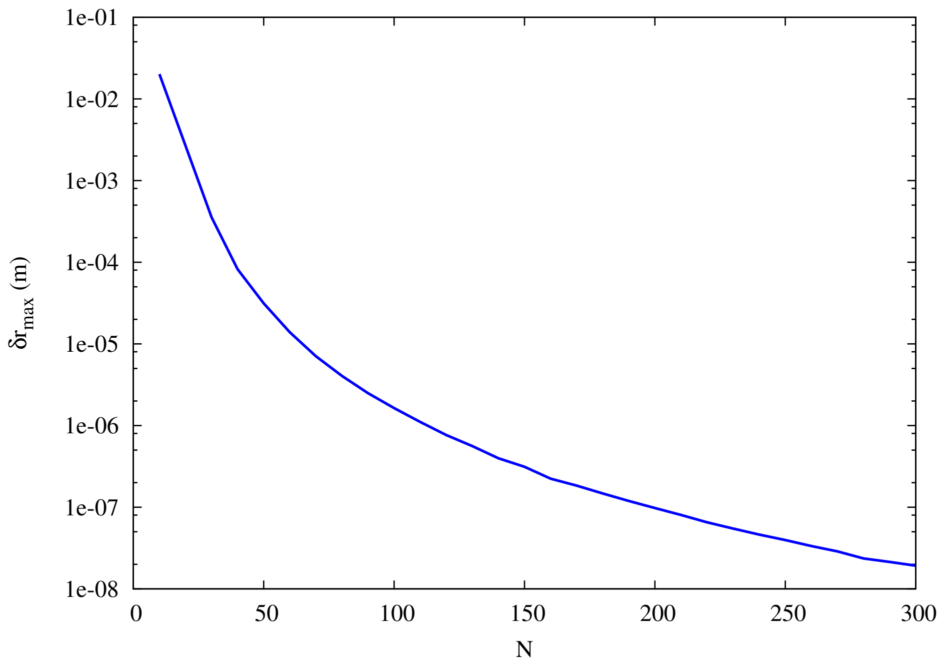
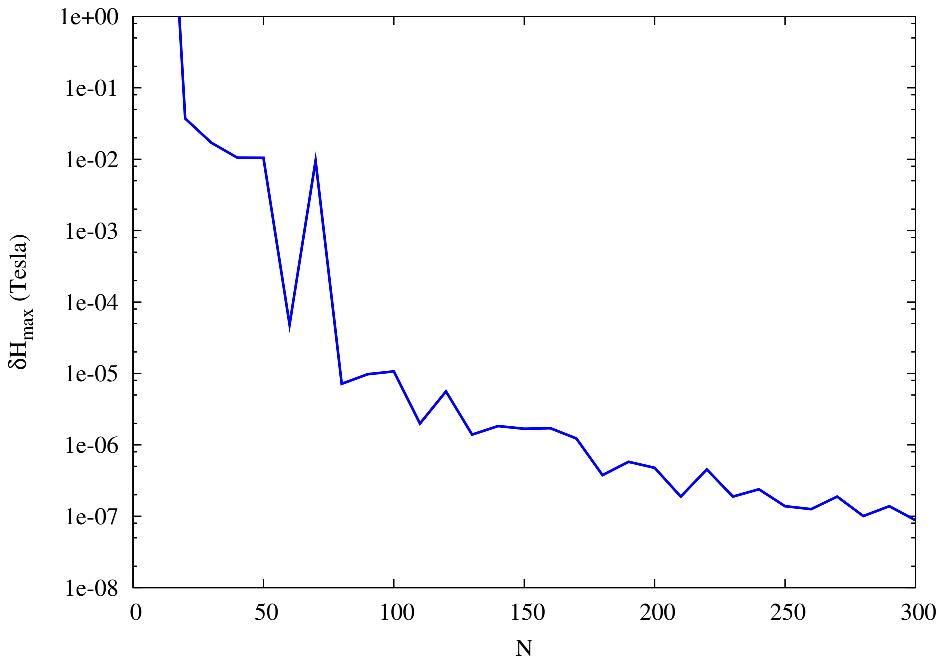


Figure 3: Greatest residual in magnetic field vs. spline generation mesh density



timing information including the durations of both the setup phase and the computation phase, and the total execution time.

The total speedup depends strongly on the number of field points and the number of processors; a larger number of points per processor results in a larger speedup factor. Two types of problems are likely to be handled with `Extender`: finding the magnetic field at points on one surface in space that surrounds the plasma, and finding the magnetic field in a mesh in the region within the first wall of the vacuum vessel. Sets of points corresponding to these two applications were used to benchmark `Extender`. Runs were done with 4 and 16 processors. The timing results are summarized in Tables 1–4.

Performance analysis. Tables 1–4 verify that the new code improves performance to a greater extent with more field points per processor. Each section of the code was affected differently:

Setup In each run, the spline-generation parameters were the same. Because the surface splines are generated with relatively dense grids, they take the longest to set up.

Table 1: Timing summary, few (1138) field points, 4 processors

Segment of run	t_{original} (s)	t_{new} (s)	Speedup
Setup	19.2	32.0	.6×
Computation	93.1	7.47	12.5×
Collection	0.18	0.02	9.×
Total	112.	39.5	2.9×

Table 2: Timing summary, many (35938) field points, 4 processors

Segment of run	t_{original} (s)	t_{new} (s)	Speedup
Setup	19.4	32.7	.59×
Computation	3980	310.	12.8×
Collection	2.	0.46	4.3×
Total	4010.	343.	11.7×

In the new code, some parts of setup were parallelized. Still, with the additional splines, setup took about 1.5× as long in each run.

Computation The most significant change in the computation phase was the use of the surface splines. In the four runs, there was an average speedup of 11×. With lower N_p and higher N_w, computation dominates time of execution, so the total speedup approaches 11×.

Collection A C++ `class` with a corresponding `MPI::Datatype` was created to simplify inter-process communcation of data. Also, small communication calls in loops were replaced by bulk communcation calls. As a result, the sharing of the total output became nearly instantaneous in every run.

For a dense mesh of field points, the new code approaches the computation speedup factor of 11×. For a small number of points, especially when more processors are available, the speedup dissappears. Such small runs take a relatively short time ($\sim 1/2$ min) to complete. However, such runs also may be executed thousands of times for different plasma equilibria. This would warrant optimizing the spline generation routines in the setup phase.

A look at the spline constructor in `Extender` reveals that it uses a Cholesky factorization to determine the cubic coefficients. Using an original or existing parallel routine to

Table 3: Timing summary, few (1138) field points, 16 processors

Segment of run	t_{original} (s)	t_{new} (s)	Speedup
Setup	19.3	31.1	$.62\times$
Computation	18.7	2.08	$9.\times$
Collection	0.85	0.01	$> 20\times$
Total	38.9	33.1	$1.2\times$

Table 4: Timing summary, many (35938) field points 16 processors

Segment of run	t_{original} (s)	t_{new} (s)	Speedup
Setup	19.4	31.8	$.61\times$
Computation	796.	77.7	$10.\times$
Collection	10.8	0.5	$22.\times$
Total	826.	110.	$7.5\times$

complete the factorization would decrease the setup time, leading to a significant speedup factor for runs with small N_p. For present-time applications, `Extender` runs fast enough. However, if `Extender` is to be used in a more real-time situation for analysing NCSX diagnostics, the spline routines should be optimized.

3 Automation

3.1 Fortran 90 module

In the immediate future, one important application already alluded to is to run `Extender` on thousands of possible plasma equilibria to generate a database that can be used for subsequent analyses. The output needs to be stored in an organized, structured and compact way. NetCDF, a platform-independent data format [3], provides mechanisms to do all of this. It is a highly general I/O library suitable for problems of any shape and size. Therefore a Fortran 90 module, `db_access`, was created to facilitate the use of NetCDF in conjunction with `Extender`.

`db_access` is a small module that allows user code to access one database during execution. Before accessing the database, there is one initialization function: `db_access_init()`. This ensures that the database file exists and reads the number of points per equilibrium and the number of equilibria. Then, to read the three components of the magnetic field, for each point and each equilibrium into a three dimensional allocatable array, `db_access_read_raw()` is used. For example,

```
real, allocatable :: H(:,:,:)
...
call db_access_init ("my_data.nc")
call db_access_read_raw (H)
```

Now the data can be analyzed. In one example program, `db_gen_Hn`, which is included with the module, an array of normal vectors is read in from a file and the values of $\vec{\mathbf{H}} \cdot \hat{\mathbf{n}}$ are computed. This results in an array of indices (field point, equilibrium). A `real` array of this form can be added to the database with one call:

```
call db_access_add_var ("Hn", ncreal, data_real=Hn)
```

or an `integer` array of the same dimensions could be added like this:

```
call db_access_add_var ("Int_Array", ncint, data_int=my_int_array)
```

A database can be extended to include additional information. The function calls and their purposes are listed in Table 5. There are additional calls, but they are used by the included programs `db_add_pointfile` and `db_append`. They should not be required by new programs, but if necessary, they can be understood from the source code of the module and the include utilities. The exact semantics of any of the listed calls can also be understood from the source code of the module.

3.2 Generalized PBS job scripts

Because `Extender` is a parallel code, it must be run on a parallel computer system. At PPPL, it is run on a cluster. Parallel jobs are submitted to a cluster using `PBS` [4], a *Portable Batch System* widely used on parallel computers. Two generalized job scripts were written to simplify the use of `Extender` with `PBS`. These are `batch-job-template` (see Appendix A) and `job-template`, and they are used for generating a database with many equilibria or running `Extender` one time, respectivly. There are many advantages to using these scripts.

- All files, directories, and `Extender` parameters that must be tracked are consistently named with shell variables. File names are prefixed with `FILE_`; directory names are prefixed with `DIR_`; and `Extender` parameters are prefixed with `PARAM_`. This makes it easy to use the same script under different conditions.

Table 5: Read / Write calls for data in `Extender` output database

Subroutine	Summaray
— Reading Data —	
`db_access_read_raw`	Read an array of indices (coordinate, field point, equilibrium). One value for each of (r,ϕ,z) for each field point, for each equilibrium.
`db_access_read_var`	Read an array of indices (field point, equilibrium). One value for each field point, for each equilibrium.
`db_access_read_profile`	Read an array of indices (surface, equilibrium). One value for each plasma surface, for each equilibrium.
`db_access_read_attribute`	Read an array of indices (equilibrium). One value for each equlibirum
`db_access_read_var_pointwise`	Add an array of indices (field point). One value for each field point.
— Writing Data —	
`db_access_add_var`	Add an array of indices (field point, equilibrium). One value for each field point, for each equilibrium.
`db_access_add_var_profile`	Add an array of indices (surface, equilibrium). One value for each plasma surface, for each equilibrium.
`db_access_add_var_attribute`	Add an array of indices (equilibrium). One value for each equlibirum
`db_access_add_var_pointwise`	Add an array of indices (field point). One value for each field point.

- A directory structure is set up to store input files and output files separately. For `batch-job-template`, `Extender` outputs are the final database files are also stored separately.

- The scripts are organized so that they are easily extendible. For example, one extension has already been written for `batch-job-template` that adjusts the coils for each equilibrium during the main loop.

- `batch-job-template` allows the creation of a database from scratch or the extension of an existing database. There is are simple mechanisms for specifying the equilibria that should be used and for avoiding running `Extender` when a given equilibrium is already in the database. The script initializes the database at the beginning with `db-add-pointfile`, adds each run with `db-append`, and computes $\vec{\mathbf{H}} \cdot \hat{\mathbf{n}}$ for the whole database at the end with `db-gen-Hn`.

The scripts are commented so that it only requires a basic knowledge of shell scripting to understand and modify them. A more detailed description of the important variables in `batch-job-template` is given in the `README` file in the same directory. Because the batch job script is mostly a generalization of the one-time job script, the information in the `README` file applies to the one-time job script as well.

4 Conclusion

`Vmec` generates stellarator plasma equilibria and stores them using Fourier series. `Extender` has been modified so that, prior to computing the magnetic field at each specified field point, it recomputes the plasma surface in terms of splines in Cartesian coordinates, (x,y,z). This optimization can be used without compromising the accuracy of the obtained results. When the number of field points per processor is large, a speedup of $12\times$ or more may be achieved. For a small number of field points per processor, the additional time spent setting up the splines decreases the speedup factor.

It may be worthwhile to optimize `Extender` even further for applications involving a small number of field points. This would require optimization of the spline routines. One possibility is to use a parallelized, rather than serial, Cholesky decomposition function during the spline setup phase. Another, perhaps more involved, possibility is to replace completely the spline routines included with `Extender` with parallel ones from another package.

Tools were developed to simplify the use of `Extender`. Template `PBS` job scripts were written in a very general way so that they can be customized and executed quickly for a particular application. Fortran 90 programs were written to provide an interface for creating and maintaining NetCDF databases of `Extender` outputs. The Fortran 90 module `db_access` was written to allow more programs to extract information from and add analyses to these databases.

Acknowledgment

This work was funded by the National Undergraduate Fellowship program through the US Department of Energy. We thank Michael Drevlak for making his Extender code available to us.

References

[1] D. Monticello M. Drevlak and A. Reiman. PIES free boundary stellarator equilibria with imporoved initial conditions. *Nucl. Fusion*, **45**:731–740 (2005).

[2] S. P. Hirshman and J. C. Whitson. Steepest-descent moment method for three-dimensional magnetohydrodynamic equilibria. *Phys. Fluids*, **26**:3553–3568 (1983).

[3] NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. `http://www.unidata.ucar.edu/software/netcdf/`

[4] Portable Batch System `http://www.openpbs.org/`

A PBS batch job template

Below is the file `batch-job-template`. For more information on NCSX, see `http://ncsx.pppl.gov`.

```
#! /bin/bash

## PBS---------------------------------------------------------------------
#PBS -N extender_batch
#PBS -l nodes=4:ppn=2
#PBS -V
#PBS -m ae
#PBS -q kestrel
#PBS -j oe
#PBS -o job-out
rm -f job-out


## DEFINITIONS ------------------------------------------------------------

## Options
OPT_REDO_IF_DONE="n" # "y" to redo if done,
# "n" to preserve existing

## DIRS
## Specify here the locations for input, output, and database-editing
## programs.
DIR_START="/p/ncsxeqdb/ExtenderStuff/mrichman/extender-batch"
DIR_EQUILIBRIA="/p/pies/Pomphrey/Ncsx_fields_from_database_on_VV/Wout_files"
DIR_INPUT="${DIR_START}/extender-input"
DIR_OUTPUT_RAW="${DIR_START}/db/raw"
DIR_OUTPUT_RECORDS="${DIR_START}/db/records"
DIR_DB_ACCESS="/p/ncsxeqdb/ExtenderStuff/mrichman/src/db-access"
rm -f ${DIR_START}/job-out


## Equilibrium -- specify a list of valid equlibria to be found in
## DIR_EQUILIBRIA.
FILE_VALID_EQUILIBRIA="${DIR_INPUT}/valid-equilibria"
# FILE_VALID_EQUILIBRIA="${DIR_INPUT}/redo-equilibria"
```

```
## Extender
## Specify Extender executable location and input file paths.
## Paths may be in terms of DIR_* variables.
## Files may be in terms of DIR_* and SUFFIX_* variables.
## PARAM_* variables are passed directly as whole parameters to
## Extender.
EXTENDER="/p/ncsxeqdb/ExtenderStuff/mrichman/src/PC6/EXTENDER_QS"
SUFFIX_RPHIZ="rphiz_points_vv"
SUFFIX_COILS="coils.c08r00x"
FILE_RPHIZ="${DIR_INPUT}/${SUFFIX_RPHIZ}"
FILE_COILS="${DIR_INPUT}/${SUFFIX_COILS}"
FILE_EXTENDER_OUTPUT="${DIR_OUTPUT_RAW}/${SUFFIX_RPHIZ}.output"
PARAM_POINTS="-points ${FILE_RPHIZ}"
PARAM_NU="-NU 80"
PARAM_NV="-NV 80"
PARAM_NPS="-NPS 200"

## Database
## Specify database-handling executables, control files, database
## file, and normal vector file.
DB_ADD_POINTFILE="${DIR_DB_ACCESS}/db-add-pointfile"
DB_APPEND="${DIR_DB_ACCESS}/db-append"
DB_DUMP_NAMES="${DIR_DB_ACCESS}/db-dump-names"
DB_GEN_HN="${DIR_DB_ACCESS}/db-gen-Hn"
CONTROL_DB_ADD_POINTFILE="${DIR_OUTPUT_RAW}/control-db-add-pointfile"
CONTROL_DB_APPEND="${DIR_OUTPUT_RAW}/control-db-append"
CONTROL_DB_GEN_HN="${DIR_OUTPUT_RAW}/control-db-gen-Hn"
FILE_DB="${DIR_OUTPUT_RECORDS}/Extender-test.nc"
FILE_RPHIZ_NORM="${DIR_INPUT}/rphiz_norm_vv"

## Node count -- number of nodes we are using on the cluster
NODE_COUNT=‘cat ${PBS_NODEFILE} | wc -l‘

## Beyond this point, little or no customization should be necessary
## in most cases.

## INITIALIZE DATABASE ---------------------------------------------------

## Make control file
rm -f ${CONTROL_DB_ADD_POINTFILE}
cat > ${CONTROL_DB_ADD_POINTFILE} <<EOF
```

```
${FILE_DB}
${FILE_RPHIZ}
(3E20.10)
‘cat ${FILE_RPHIZ} | wc -l | awk ’{print $1}’‘
49
EOF

## Set up database (all database stuff controlled from ${DIR_OUTPUT_RAW}
cd ${DIR_OUTPUT_RAW}
${DB_ADD_POINTFILE}


## BEGIN LOOP -------------------------------------------------------------
for i in ‘cat ${FILE_VALID_EQUILIBRIA}‘
do

## loop-dependent definitions
FILE_EQUILIBRIUM="${DIR_EQUILIBRIA}/$i"
FILE_TMP_LIST="${DIR_OUTPUT_RAW}/tmp_finished_list"
STDERR="${DIR_OUTPUT_RAW}/job-err-$i"
rm -f ${STDERR}
cd ${PBS_O_WORKDIR}

## check if already done
rm -f ${FILE_TMP_LIST}
${DB_DUMP_NAMES} ${FILE_DB} > ${FILE_TMP_LIST}
FOUND="n"
for j in ‘cat ${FILE_TMP_LIST}‘
do
if test "$i" == "$j"
then
FOUND="y"
fi
done

## if it’s already done, don’t do it again
if test "${OPT_REDO_IF_DONE}" == "y"; then FOUND="n"; fi
if test "${FOUND}" == "n"
then

ERRORS="n"
```

```
for (( j = 1; j < 5; j++ ))
do
## run Extender!
mpirun -np ${NODE_COUNT} ${EXTENDER} -v2000 ${FILE_EQUILIBRIUM} \
${PARAM_POINTS} -s suffix \
${PARAM_NU} ${PARAM_NV} ${PARAM_NPS} \
-outdir ${DIR_OUTPUT_RAW} >& ${STDERR}

## check for pbs errors
if test "‘grep ’errno = ’ ${STDERR} | wc -l | awk ’{print $1}’‘" == "0"
then
j=6
else
if test "$j" == "5"; then ERRORS="y"; fi
echo "Handling error on try $j for equilibrium $i"
sleep 10s
fi
done

## (could add more error checks here?)

## if there are no errors, add to database
if test "${ERRORS}" == "n"
then
## make control file
rm -f ${CONTROL_DB_APPEND}
cat > ${CONTROL_DB_APPEND} <<EOF
${FILE_EXTENDER_OUTPUT}
${FILE_DB}
‘cat ${FILE_EXTENDER_OUTPUT} | wc -l | awk ’{print $1}’‘
$i
1
EOF

## append to database, kill extender raw output
cd ${DIR_OUTPUT_RAW}
${DB_APPEND}
rm ${FILE_EXTENDER_OUTPUT}

fi ## ‘end if’ for whether this equilibrium was done yet
fi ## ‘end if’ for whether there were errors

done ## end of main loop


## Finished ---------------------------------------------------------------

## generate Hn for database
rm -f ${CONTROL_DB_GEN_HN}
cat > ${CONTROL_DB_GEN_HN} <<EOF
${FILE_DB}
${FILE_RPHIZ_NORM}
EOF
cd ${DIR_OUTPUT_RAW}
${DB_GEN_HN}

exit 0
```