A simple method for integrating a complex model into an ensemble data assimilation system using MPI

Environmental Modelling & Software 68 (2015) 122–128

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A simple method for integrating a complex model into an ensemble data assimilation system using MPI

P.A. Browne [a, *], S. Wilson [b, c]

[a] Department of Meteorology, University of Reading, RG6 6BB, UK
[b] NCAS-CMS, Department of Meteorology, University of Reading, RG6 6BB, UK
[c] UK Met Office Hadley Centre, Fitzroy Road, Exeter, EX1 3PB, UK

ARTICLE INFO

Article history:
Received 5 June 2014
Received in revised form
30 January 2015
Accepted 9 February 2015
Available online 7 March 2015

Keywords:
Data assimilation
Model coupling
MPI

ABSTRACT

This paper details a strategy for modifying the source code of a complex model so that the model may be used in a data assimilation context, and gives the standards for implementing a data assimilation code to use such a model. The strategy relies on keeping the model separate from any data assimilation code, and coupling the two through the use of Message Passing Interface (MPI) functionality. This strategy limits the changes necessary to the model and as such is rapid to program, at the expense of ultimate performance. The implementation technique is applied in different models with state dimension up to .2.7 × 10^8 The overheads added by using this implementation strategy in a coupled ocean-atmosphere climate model are shown to be an order of magnitude smaller than the addition of correlated stochastic random errors necessary for some nonlinear data assimilation techniques.

1. Introduction

When a model is developed to simulate a process in environmental science, the accurate solution of the underlying physical process will be foremost in the mind of the developers. Once the model performs sufficiently well, only then might the developers consider using such a model to produce a forecast (Verlaan, 1998). In order to make a forecast, the model needs to be initialised using observations of the system, so that the state of the model represents reality as best as possible. This process of incorporating observations into a model is known as data assimilation (DA).

The most effective method of DA for a new model may not be known *a priori*. For instance, whether a *variational* method or an *ensemble* method of DA should be used, and within these choices whether linear or non-linear DA scheme should be used (see for example Evensen, 2007). In this circumstance, an implementation technique that can rapidly prototype different DA methods would be particularly useful, before substantial time and effort is spent on efficient implementation of any one particular algorithm.

In the same manner, an academic researcher will want the flexibility to test novel DA techniques with state of the art models (Browne et al., 2014). Such ongoing changes to the DA system are used as proof-of-concept tests and as such do not seek maximum performance and efficiency. The critical factor for the academic may be the amount of coding time spent modifying the model in order to perform scientific tests.

In this paper we propose a simple implementation strategy which does not focus on maximum efficiency of the code. Instead the focus is on the speed of implementation. The aim is to satisfy the needs of model developers seeking to prototype different existing DA methods and also to give academics swift access to new models for predictive purposes. Changes to the model should be as small as possible, and specifically we do not want to change the flow or structure of the model.

The goal of data assimilation is to represent the probability density function (pdf) of the state of the model, given some observations (Evensen, 2007). Ensemble based data assimilation will try and represent such a pdf (or its moments such as the mean and covariance) with a collection of separate model instances. Each of these models instances need be propagated forward in time, and are inherently parallel tasks. To combine the information in the ensemble with observations (using methods such as the Ensemble Kalman Filter/Smoother (Evensen, 1994; Evensen and van Leeuwen, 2000)) or particle filters (van Leeuwen, 2009) only the state vector of each model at each timestep is required. Hence it is

* Corresponding author.
E-mail address: p.browne@reading.ac.uk (P.A. Browne).

http://dx.doi.org/10.1016/j.envsoft.2015.02.003
1364-8152/

possible to make generic data assimilation software based only on knowledge of the dimension of the model.

To run an ensemble data assimilation system which does not use an adjoint model (Errico, 1997), the only task the model must perform is to integrate the model state forward in time. Other scientific questions related to data assimilation, such as handling observations and constructing and storing covariance matrices are effectively independent of the model code, hence fall outside the scope of this paper where we consider technically connecting the model with a data assimilation code. As such, the required operations will be to interrupt the model after each timestep and get the model state to a DA code which will operate on it and return to the model a modified state. This was noted by Verlaan (1998, Section 5.6.1) but implemented by combining the model and DA code within one Fortran executable. Message Passing Interface (MPI) is ideally suited to this and avoids the need of the model to write to disk as we shall show subsequently.

There are a number of existing frameworks for performing data assimilation with a new model, with varying techniques for linking with the model. Tools such as SESAM (Brankart et al., 2002) and the Ocean Assimilation Kit (OAK) (Barth and Vandenbulcke, 2013) are available to perform ensemble analysis on disk files. In these cases the dynamical model and data assimilation are not coupled at all. Instead a program such as a shell script has to control each individual code. In this case the model must run only a single timestep and write to disk at each step, resulting in poor performance (Nerger and Hiller, 2013). The data assimilation code requires the model state at every model timestep in order to be able to use methods such as the EnKS (Evensen and van Leeuwen, 2000) or other nonlinear sequential methods (van Leeuwen, 2009).

A powerful data assimilation tool is the Data Assimilation Research Testbed (DART) (Anderson et al., 2009). If a model is already in the form where a single timestep is callable as a subroutine, then DART will wrap around the model source code. If the model is not available as a subroutine then DART has the flexibility to work with specifically formatted netcdf files from disk. Another similar data assimilation tools is OpenDA (OpenDA, 2013) which requires either wrapping the model in Java or directly reading and writing disk files. Either case needs the model to produce a large amount of meta-data in XML files which allow the full flexibility of OpenDA to be utilised.

The Parallel Data Assimilation Framework (PDAF) (Nerger and Hiller, 2013) is a system in which a model's source code is modified in order to perform data assimilation. PDAF has two different modes; the fast *online* mode or the slower *offline* mode. In offline mode, disk files are used to transfer data between the model and the DA system, hence it suffers relatively poor performance. In online mode, the source code is modified to insert calls to the DA procedures. One executable file is created and the parallelism of the model is inherited by the DA algorithms (see Section 6).

The remainder of this paper is structured as follows. In Section 2 we give a brief overview of MPI. In Section 3 we discuss the concept of the mechanism by which the code performs the necessary set-up stages. In Section 4 the constraints on the design of a data assimilation system in order to use such a communication system are given. Section 5 explains the how communication between model and data assimilation code is performed. Section 6 details the advantages and disadvantages of adopting the proposed technique. Section 7 lists details of how this system has already been implemented in large-scale models and gives performance details for these. Finally we draw some conclusions on the implications of adopting this implementation for data assimilation. For complete clarity, we include Appendix A to give the technical details of the construction of the MPI communicators introduced in Section 3, and Appendix B shows the proposed implementation technique in the code a short model.

2. Overview of MPI

Message Passing Interface (MPI) is a standardised system for utilising distributed computing (Gropp et al., 1996; Gabriel et al., 2004). It has become the *de facto* standard (Desouza et al., 2005) for High Performance Computing (HPC) systems, and works on the basis of sending packets of information between *processes* running (possibly different) program executables. When launched, each process is assigned a unique integer known as the *rank*. By convention, we will refer to the process whose rank has the value 0 as the *master process*.

MPI is commonly thought of in Single Program Multiple Data (SPMD) form (Pieterse and Black, 1999), where the same program is executed on multiple processes. In this paradigm, the program computes which section of the code to execute based on a variable which it is given at run-time. This variable is typically the global rank of the process.

In contrast to SPMD, the Multiple Instruction Multiple Data (MIMD) form allows different programs to be executed so that they are able to communicate with each other. Here, multiple executables are launched using one `mpiexec` command. Hence one program can be written for one task, and a different program can be written to perform a completely separate task. All processes which are able to pass information between each other are said to share a *communicator*. When two processes share a communicator a simple instruction can be given to send or receive data between the processes. When more than two processes are involved in transfer of information, MPI finds an efficient strategy for exchanging the information. Such strategies are architecture dependent and are optimised without input from the user.

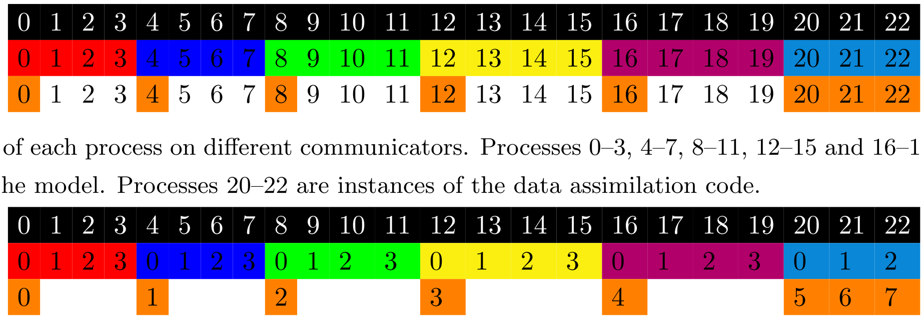
In this paper we will describe a strategy for setting up appropriate communicators so that models and DA codes can be run in MIMD form. Once the communicators are initialised then data transfer can be achieved with basic MPI send and receive commands.

3. MPI communicator initialisation for model processes

In this section we shall describe the initialisation step for the method that we propose that must be applied to the model code. The goal is to compute the appropriate information to allow communication to and from a data assimilation code. A more detailed technical discussion of how this can be achieved is given in Appendix A.

As the ensemble of models and a number of data assimilation processes will be launched in MIMD mode, they will all share the `MPI_COMM_WORLD` communicator. This default numbering and ordering of the processes must be appropriately divided up to allow for the various types of communication: the MPI concept of *splitting* communicators will allow us to do precisely this. Splitting communicators is based on another abstract concept of *colours*: once split those processes with the same colour are grouped together in a new communicator. (See Nerger et al., (2005) for an example of the use of different MPI communicators in a data assimilation system.)

Fig. 1 shows the different communicators which we will seek to create. From the point of view of the model code, this is a three stage process. Firstly we separate the processes from the models and those of the DA processes. We choose to do this by setting the colour of all model processes to 0 (DA processes will have colour 1) and using MPI_split to generate a temporary communicator `tmp_mdls_comm` from `MPI_COMM_WORLD`. It is mandatory that the



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

(a) The rank of each process on different communicators. Processes 0–3, 4–7, 8–11, 12–15 and 16–19 are different instances of the model. Processes 20–22 are instances of the data assimilation code.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2
0				1				2				3				4				5	6	7

(b) The rank of each process within each communicator

Fig. 1. Schematic to show the distribution of processes amongst the different communicators. Black indicates the global communicator. Red, blue, green, yellow and magenta represent 5 separate models each running on 4 processes. The cyan communicator is a communicator dedicated for the data assimilation code and the orange communicator links the master process of each of the models with all the data assimilation processes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

DA processes also call this split at this point, as discussed in Section 4.

Next we must split this temporary communicator into the different ensemble members, in order to account for any existing MPI parallelism within the model. The colour for this splitting is given by the integer value `[models_id/mdl_num_proc]`, where `models_id` is the rank of the process on the `tmp_mdls_comm` communicator and `mdl_num_proc` is the number of MPI processes launched for each ensemble member. Hence the different instances of the model will have colour = 0, 1, 2…. Splitting the temporary communicator based on these colours produces the new communicator `mdl_mpi_comm`. In Fig. 1 these different `mdl_mpi_comm` are shown in the second row, and given the colours red, blue, green, yellow and magenta. This communicator should replace the original communicator used for all communication between the processes of the model.

The next task is to set up another communicator, to couple the master process of each model and all the DA processes. A final split of `MPI_COMM_WORLD` is used to create this, in which the model process will set its colour to 9999 if and only if its rank on `mdl_mpi_comm` = 0. This value is somewhat arbitrarily chosen for our implementation, and will be the value that all the DA processes use for this splitting. For simplicity of implementation we want to ensure that the DA processes have the highest rank on this communicator. This can be achieved by setting the *rank_key* used in the MPI_split of the DA processes higher than that used by the model processes. To ensure this is possible, the *rank_key* used by the model processes in this splitting must be less than the total number of model processes launched. Thus this split creates the orange communicator in Fig. 1 which we denote `cpl_mpi_comm`.

What remains is to identify with which DA process the master process of each model should transfer data. To do this, we first detect the total number of DA processes and models which are running and we call them `nda` and `nens` respectively. The rank of each instance of the model is detected on `cpl_mpi_comm` and we denote this `particle_id`. For each `particle_id` we must choose a particular DA process on `cpl_mpi_comm` with which it should communicate. The rank of this DA process we shall call `cpl_root` and we choose to define it such that

$$\texttt{cpl\_root} := \left\lfloor \frac{\text{nda} \times \text{particle\_id}}{\text{nens}} \right\rfloor + \text{nens}.$$

In doing so the work is distributed almost uniformly across all the DA processes. The benefits of this choice of `cpl_root` are greatest for the DA implementation (see Section 4).

4. Data assimilation code design

In order to make use of a model with the communication capabilities as given in the previous sections, a data assimilation code must be written to link with such a generic model. As for the model code there are two stages: defining the communicators and then evolving the model. In this section we describe the requirements of a data assimilation code to use the system we have introduced.

The data assimilation code that the models will be linked to will have to set appropriately the communicators which we have discussed earlier. Firstly, an inter-DA process communicator is created at the same time that the temporary models communicator, `tmp_mdls_comm`, is made. This is done by an MPI_split of `MPI_COMM_WORLD` where the DA processes set their colour to 1. The resulting communicator we denote `da_mpi_comm` (and is shown in cyan in Fig. 1). To create `cpl_mpi_comm` the DA processes must again perform a split of `MPI_COMM_WORLD` at the corresponding time that the models create each `mdl_mpi_comm`. The couple colour must be set to 9999, the value used by the master process of the models. To ensure that the DA processes are given the highest ranks, the rank_key used in this split must be at least the number of ensemble members; the size of `MPI_COMM_WORLD` satisfies this condition. Once this split is complete, the number of ensemble members can be computed by differencing the sizes of the communicators `cpl_mpi_comm` and `da_mpi_comm`.

Each DA process must then compute the rank of the ensemble members which it will communicate with. For the DA process with rank `da_rank` on `cpl_mpi_comm`, the ensemble members which it communicates with lie in the interval

$$\left[\left\lceil\frac{(\text{da\_rank} - \text{nens})\text{nens}}{\text{nda}}\right\rceil, \left\lceil\frac{(\text{da\_rank} - \text{nens} + 1)\text{nens}}{\text{nda}}\right\rceil - 1\right]$$

This distribution of ensemble members to DA processes is a direct result of the choice of `cpl_root` chosen in Section 3. This ensures a relatively even number of ensemble members per data assimilation process for load balancing purposes. The main advantage and reason that this distribution strategy has been adopted is that contiguous ensemble members are associated with each DA process, thus allowing easy use of MPI methods such as MPI_gatherv and MPI_scatterv for transferring information about ensemble members between the DA processes.

In order for the data assimilation code to evolve ensemble member *j* forward one timestep, it is simply a matter for the appropriate DA process to send model *j* the appropriate initial conditions and receive back the evolved state. Hence DA process

$\left\lfloor \frac{\mathrm{nda} \times j}{\mathrm{nens}} \right\rfloor + \mathrm{nens}$ must send the state vector to process rank j on `cpl_mpi_comm` using an MPI_send command and receive the updated state vector from the same process. The MPI receive here can be non-blocking to allow the DA process to perform other calculations while the model is evolving the state vector forward in time, provided the data assimilation method in question permits such extra computations.

To start the DA code, the ensemble of initial states have to be received from the connected models using an MPI_recv command. To allow the models to end correctly, the final step of the DA code should be to send the state vector of each connected ensemble member back to the corresponding models. This has the effect of ensuring that all the send/receive pairs match up. Algorithm 1 shows pseudocode of a data assimilation code showing where the appropriate communication occurs.

Algorithm 1 Pseudocode of DA algorithm

```
Set up communicators
Compute connected ensemble members
FOR each connected ensemble member:
    receive state vector of ensemble member using MPI_recv
DO timestep loop:
    perform DA specific computations
    when ensemble needs to be evolved by the model:
      FOR each connected ensemble member:
          send state vector of ensemble member to model using MPI_send
      FOR each connected ensemble member:
          receive state vector of ensemble member from model using MPI_recv/MPI_irecv
    perform DA specific computations using the now evolved ensemble
END timestep loop
FOR each connected ensemble member
    send state vector of ensemble member to model using MPI_send to allow the model to finish
```

5. Communicating the state vector by the model code

In this section we describe how we propose to communicate the model state between model and data assimilation codes. The entire model state has to be passed back and forth from the model to the DA code in two places. Firstly, this is done once the model is initialised. Secondly, this must happen after the model has completed an iteration of its timestepping scheme.

In order to send the model state to the DA code, all of the model's prognostic variables are first gathered to the model master process so that the DA code will interact with a single complete state vector. This is a straightforward MPI procedure which we do not show in Algorithm 3 of the Appendix as the state vector is already stored in a single vector on a single process, however care must be taken in a model which employs domain decomposition in order to not duplicate any variables which are stored in multiple locations. As every model will store its variables in a different manner, locating them within the model may present its own complications. As these are the very variables that we wish to influence through data assimilation, finding them is an unavoidable process in setting up any data assimilation system. However, as the model must collect the state variables when it performs its output, it may be possible to reuse these sections of output code for this task.

Once all the prognostic variables are on the master model process, they are sent with a single MPI send command to the DA process with rank `cpl_root` on the `cpl_mpi_comm` communicator. The source of this rank is given in Section 3. We make the convention that all of the state variables are communicated as `MPI_DOUBLE_PRECISION` variables with MPI tag set to 1.

Immediately following the MPI send commands, MPI receive instructions should be issued, to receive the state vector from the same DA process as used in the MPI send. The receive tag should be `MPI_ANY_TAG` to allow extensions to be made for variational-type DA methods. Once this receive is completed, the model state should be scattered throughout the model to ensure that the updated state is in the appropriate variables.

The initial MPI send and receive commands are located after the model is initialised so that the DA code has the possibility of creating an initial perturbed ensemble. For instance, if each model has an identical initial state then DA code could add some stochastic variation initially.

The remainder of the MPI send and receive commands must occur after the model has completed a timestep. Hence the model has integrated forward in time the model state which was provided to it by the DA code. This integrated state is returned to the DA code so that its algorithm can continue. In the case of, say, a leapfrog integration scheme, this should be done for all parts of the model's timestepping scheme.

It should be noted that all of the MPI send and receive commands can be included in an if statement within the code or included as a compile time option. This will allow the user the functionality of running the model with or without data assimilation. Hence this functionality could then be left within the trunk of the model code, giving scope to progress the development of the model independently of the DA system being used.

6. Pros and cons of this strategy

Using MPI communicators in a data assimilation framework is a powerful tool, and indeed forms the basis of PDAF (Nerger, 2004, Section 8.3.2). The difference with our technique is that separate model and DA executables will be run in MIMD form with a single `mpiexec` command, as opposed to the SPMD setup of PDAF. PDAF may be used with a model written in a language that can call Fortran, and requires that the model executable be rebuilt each time the data assimilation codes are updated. The implementation technique described in this paper will be applicable for any model written in a language which has MPI bindings. These are generally directly callable from C, C++ and Fortran. Other programming languages such as C#, Perl, Python, R, Ruby, Java, CL and Matlab are able to interface with the MPI libraries and are thus compatible with this technique, though care must be taken to ensure that both model and DA code use compatible MPI libraries.

The major advantage of adopting the implementation technique described in this paper is the speed of implementation. In the authors' experience it has been finding the appropriate position in the model to insert the appropriate lines of code which has taken the most time. Once this has been found in the code, then the code modifications are similar for each different model.

A disadvantage of this implementation technique is that it does impact on the performance of the model. As we are including extra communication we are adding an extra overhead in terms of wall-clock time. This means that the implementation technique will be theoretically less efficient than converting the model to a subroutine callable by a data assimilation system such as DART or OpenDA, however the task of implementing this is the impracticality that the implementation technique proposed in this paper avoids. It is, however, much more efficient to use MPI to transfer the data than writing to disk. A simple test, where a state vector of dimension 2.3×10^6 is written to disk by one process, read in by another, then reversed so that it is returned to the original process, took approximately 28 s. In comparison, on the same machine, the same task accomplished by MPI took approximately 0.03 s. These figures are more pronounced than those quoted by Nerger and Hiller (2013) as here we consider a model with approximately 2.5 times larger state dimension.

One other potential complication is that the proposed technique requires the communicator used in the model to be changed to the one defined in Section 3. This complication is only applicable if the model in question is already parallelised using MPI. In the models we have considered, this is a simple find and replace task. Using the MIMD approach may also make debugging and compilation more problematic, as errors in consistency between different executables may only be noticed at run-time.

The method we propose in this paper necessary launches more processes than those to run the models. These additional data assimilation processes can be performing useful tasks such as generating random terms while the models are running. This may save some wall-clock time over other implementation strategies, but would of course remain model dependent.

Notice that the set-up of the model is done entirely by the model itself. This allows the methods of operating and defining the model to remain unchanged when it is run within the DA system. Advantages of keeping the model interface consistent include improved productivity, shorter learning times, fewer user frustrations and reduced training costs (Nielsen, 1989).

The implementation technique proposed in this paper could be easily incorporated into an existing framework which uses subroutines. A dummy subroutine would have to be created to use MPI to interface with the model. The core of this subroutine would be an MPI_send followed by a corresponding receive, which would instruct the model to integrate the dynamical system forward one timestep. These links are planned to be developed in the near future and would give any model with the MPI capability proposed in this paper access to the wealth of tools available in, say, DART or PDAF.

The data assimilation methods which can make immediate use of a model with the MPI functionality are sequential methods. Most variants of the Ensemble Kalman Filter (localised EnKF, square root EnKF, morphing EnKF etc, see for example (Lei et al., 2010; Tippett et al., 2003)) could easily be implemented in this setting. Fully nonlinear ensemble data assimilation methods, such as particle filters (van Leeuwen, 2009) can also be implemented. Examples of such methods would include the SIR filter (Gordon et al., 1993), the auxiliary particle filter (Pitt and Shephard, 1999), the implicit particle filter (Chorin et al., 2010) or the equivalent weights particle filter (van Leeuwen, 2010). Parameter estimation and sensitivity analyses could be implemented using these methods by including the parameters as prognostic variables of the model in an augmented state approach.

The implementation technique proposed in this paper has the potential to be extended to allow for variational data assimilation methods. This would require that the adjoint of the model (Errico, 1997) was also available to provide the necessary gradient information. A similar MPI structure to that proposed in this paper for the forward model could be added to the adjoint model code. In doing so, the data assimilation code would be able to use a gradient based optimization method necessary for variational DA.

7. Large scale examples

Models such as Lorenz 1963 (Lorenz, 1963) shown in the appendix are of course not the scale of model for which the implementation proposed in this paper is designed. We are indeed interested in much larger models. The MPI coupling has been incorporated into a Barotropic Vorticity model with 65536 variables that has been used for research (Ades and van Leeuwen, 2014) and teaching of particle filters. It was quickly implemented into TELEMAC (Hervouet, 2000), an unstructured finite element model of the North Sea with 114288 state variables, for the assimilation of tidal gauge data (Thainuruk, 2013). It has also recently been incorporated into the land-surface models JULES (Best et al., 2011) and DALEC (Williams et al., 2005), as well as the coupled ocean biogeochemical model GOTM-ERSEM (Allen et al., 2004).

The implementation technique as described in this paper has been incorporated into HadCM3 (Gordon et al., 2000) (a coupled ocean-atmosphere global climate model). Using this, we have implemented the equivalent weights particle filter, and we show results here to illustrate simple performance measures. For a comprehensive overview of the equal weights particle filter see (van Leeuwen, 2010) and (Ades and van Leeuwen, 2012). Specific details of the assimilation method including the choice of the model error covariance matrix, will be thoroughly described in a forthcoming paper, along with the implications for this new strategy for initialising a climate model. For the implementation described here artificial sea surface temperature data were assimilated into the model and we present timings for one analysis cycle (24 h model simulation). The model has state dimension 2314430 and observation dimension 27370. For fully nonlinear data assimilation, this is a large system (Snyder et al., 2008). Note that this method updates the model state vector at every timestep, not just at observation time.

Table 1 shows wall-clock timings when the models and data assimilation system were run together. The ensemble size starts from 1 for completeness of the computational timings. The first column gives the number of models in the ensemble. The second column shows the total number of processing cores used; each model used 24 and was associated with its own DA MPI process which was distributed over another 12 cores in a hybrid MPI/OpenMP/multithreaded implementation. The third column shows the wall-clock time taken to run the model to simulate a single day when the MPI send and receives as described in this paper are *not* used. Hence it is constant as no communication happens except for the intrinsic model communication.

The forth column in Table 1 can be compared with the third column to see how much of an overhead using the MPI implementation described in this paper adds to the model run-time. The penultimate and final columns show the wall-clock time when

Table 1
Relative timings of HadCM3 with and without the MPI coupling over one analysis step, run on ARCHER, the UK's national HPC machine. It is a Cray XC30 with an Aries interconnect.

Number of ensemble members	Number of cores used	Model only time (s)	Model + MPI communication time (s)	Model + MPI + stochastic error (s)	Model + MPI + equivalent weights filter time (s)
1	36	4.6	5.5	25.6	39.5
3	108	4.6	5.5	26.1	39.3
7	252	4.6	5.5	26.4	39.7
15	540	4.6	5.5	26.4	37.9
31	1116	4.6	5.5	27.5	39.3
62	2232	4.6	5.5	26.5	39.7

running a stochastic ensemble and the specific data assimilation scheme respectively. These are an order of magnitude larger than the time to run the deterministic model itself, and even more of an overhead than the additional time that the MPI communication has added. Note how the timings for each different type of ensemble remains approximately constant as the number of ensemble members increases, showing excellent scaling properties for this implementation strategy. This is due to the fact that, as the ensemble size increases, we launch extra DA processes to do the computations associated with the extra ensemble members.

The computational cost of the equivalent weights particle filter, and indeed a stochastic ensemble, is dominated by the cost of computing correlated random errors. To do so requires a matrix-vector multiplication by $Q^{\frac{1}{2}}$ at each timestep of the model where, with N_x the size of the state vector of the model, $Q \in \mathbb{R}^{N_x \times N_x}$ is the model error covariance matrix. The addition of correlated random error should be included in any consistent DA method that considers the model stochastic and so is not a limitation of the equivalent weights particle filter. There are 72 timesteps per day in the model, so the stochastic version of HadCM3 requires 72 of these matrix-vector multiplications. The equivalent weights particle filter required an extra 28 matrix-vector multiplications as part of its relaxation scheme. For this application, $Q^{\frac{1}{2}}$ exists only as a sparse matrix, and not in operator form, requiring 16 GB of disk space to store only its upper part. To maximise the efficiency of these matrix vector multiplications they have been implemented in parallel using the highly efficient sparse BLAS implementation LIBRSB (Martone et al., 2010).

Note that all these performance numbers are highly model dependent. However, the more computationally expensive the model, the less overhead using the proposed MPI coupling will add. For instance, this coupling has been included within the UK MetOffice's operational forecast model (UM vn8.2) configured as an N512L70 global model. This has 40 km horizontal resolution and 70 vertical layers. The size of the state vector for this system is $275537920 \approx 2.7 \times 10^8$.

For a single instance of this model, a single model timestep *without* MPI communication to a DA code took 28.4 s. Running the code *with* the MPI communication to a DA code resulted in a wall-clock time for a single model timestep of 30.5 s. The communication overhead is smaller with 7.4% for the larger model compared to 20% with HadCM3.

8. Code availability

Our examples of this implementation have been released under the name Employing Message Passing Interface for Researching Ensembles (EMPIRE). Step-by-step implementation guides and minimal codes to test the implementation are available at http://www.met.reading.ac.uk/~darc/empire, along with software that implements various sequential data assimilation methods according to the design specified in Section 4.

9. Conclusions

In this paper we have proposed a technique for coupling a dynamical model with a data assimilation code by using MPI. This technique has the advantage of not writing files to disk, as well as not changing the flow of the model program. This means that the changes to the model source code are small, reducing the effort required to adapt the model to fit the data assimilation system.

We have given a specific example of the changes needed in a simple model to elucidate the abstract concepts of sending and receiving data. The technique has been applied to much larger models, specifically HadCM3 and the current UK MetOffice operational global forecast model. Timings for running the models with the proposed MPI technique have been shown not to significantly impede performance and have excellent scaling properties.

Acknowledgements

The authors would like to thank PJ van Leeuwen for his discussion on the data assimilation test cases to apply this framework. This work was supported by NERC grant NE/J005878/1. This work used the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk) .

Appendix A. Supplementary data

Supplementary data related to this article can be found at http://dx.doi.org/10.1016/j.envsoft.2015.02.003.

References

Ades, M., van Leeuwen, P., 2014. The equivalent-weights particle filter in a high dimensional system. Q. J. R. Meteorol. Soc. http://dx.doi.org/10.1002/qj.2370.

Ades, M., van Leeuwen, P.J., 2012. An exploration of the equivalent weights particle filter. Q. J. R. Meteorol. Soc. 139 (672), 820–840.

Allen, J., Siddorn, J.R., Blackford, J.C., Gilbert, F.J., 2004. Turbulence as a control on the microbial loop in a temperate seasonally stratified marine systems model. J. Sea Res. 52 (1), 1–20.

Anderson, J., Hoar, T., Raeder, K., Liu, H., Collins, N., Torn, R., Avellano, A., 2009. The data assimilation research testbed: a community facility. Bull. Am. Meteorol. Soc. 90 (9), 1283–1296.

Barth, A., Vandenbulcke, L., 2013. Ocean Assimilation Kit. http://modb.oce.ulg.ac.be/mediawiki/index.php/Ocean_Assimilation_Kit.

Best, M.J., Pryor, M., Clark, D.B., Rooney, G.G., Essery, R.L.H., Ménard, C.B., Edwards, J.M., Hendry, M.A., Porson, A., Gedney, N., Mercado, L.M., Sitch, S., Blyth, E., Boucher, O., Cox, P.M., Grimmond, C.S.B., Harding, R.J., 2011. The joint UK land environment simulator (JULES), model description Part 1: energy and water fluxes. Geosci. Model Dev. 4 (3), 677–699.

Brankart, J., Testut, C., Parent, L., 2002. An Integrated System of Sequential Assimilation Modules: Sesam Reference Manual. LEGI/MEOM, Grenoble, France. Tech. Rep., Office Note. http://lgge.osug.fr/meom/Outils/SESAM/sesam.html.

Browne, P., Charlton-Perez, C., Dance, S.L., 2014. RMetS special interest group Meeting: high resolution data assimilation. Atmos. Sci. Lett. 15 (4), 354–357.

Chorin, A.J., Morzfeld, M., Tu, X., 2010. Implicit Particle Filters for Data Assimilation arXiv preprint arXiv:1005.4002.

Desouza, J., Kuhn, B., de Supinski, B.R., 2005. Automated, scalable debugging of MPI programs with Intel message Checker. In: SE-HPCS '05 Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications, pp. 78–82.

Errico, R., 1997. What is an adjoint model? Bull. Am. Meteorol Soc. 78 (11), 2577–2591.

Evensen, G., 1994. Sequential data assimilation with a nonlinear quasi geostrophic model using Monte Carlo methods to forecast error statistics. J. Geophys. Res. Oceans (1978–2012) 99 (C5), 10143–10162.

Evensen, G., 2007. Data Assimilation. Springer.

Evensen, G., van Leeuwen, P., 2000. An ensemble Kalman smoother for nonlinear dynamics. Mon. Weather Rev. 128 (6), 1852–1867.

Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S., 2004. Open MPI : goals, concept, and design of a next generation mpi implementation. In: 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary.

Gordon, C., Cooper, C., Senior, C., Banks, H., Gregoire, L.J., Johns, T., Mitchell, J., Wood, R., 2000. The simulation of SST, sea ice extents and ocean heat transports in a version of the Hadley Centre coupled model without flux adjustments. Clim. Dyn. 16, 147–168.

Gordon, N., Salmond, D., Smith, A., 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEE Proc. F Radar Signal Process. 140, 107–113.

Gropp, W., Lusk, E., Doss, N., Skjellum, A., 1996. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Comput. 22 (6), 789–828.

Hervouet, J.-M., 2000. TELEMAC modelling system: an overview. Hydrol. Process. 14 (13), 2209–2210.

Lei, J., Bickel, P., Snyder, C., 2010. Comparison of ensemble Kalman filters under non-Gaussianity. Mon. Weather Rev. 138 (4), 1293–1306.

Lorenz, E.N., 1963. Deterministic nonperiodic flow. J. Atmos. Sci. 20, 130–141.

Martone, M., Filippone, S., Tucci, S., Paprzycki, M., Ganzha, M., 2010. Utilizing recursive storage in sparse matrix-vector multiplication - preliminary considerations. In: Philips, T. (Ed.). ISCA, pp. 300–305. CATA.

Nerger, L., 2004. Parallel Filter Algorithms for Data Assimilation in Oceanography. Phd, Universität Bremen.

Nerger, L., Hiller, W., 2013. Software for ensemble-based data assimilation systems – implementation strategies and scalability. Comput. Geosci. 55, 110–118.

Nerger, L., Hiller, W., Schröter, J., 2005. Pdaf – the parallel data assimilation framework: experiences with kalman filtering. In: Zwieflhofer, W., Mozdzynski, G. (Eds.), Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology. World Scientific, Reading, UK, pp. 63–83.

Nielsen, J., 1989. Coordinating user interfaces for consistency. ACM SIGCHI Bull. 20 (3), 63–65.

OpenDA, 2013. The OpenDA data-assimilation toolbox. http://www.openda.org.

Pieterse, V., Black, P.E. (Eds.), 1999. Algorithms and Theory of Computation Handbook, Chapter Single Program Multiple Data. CRC Press LLC.

Pitt, M.K., Shephard, N., 1999. Filtering via simulation : auxiliary particle filters. J. Am. Stat. Assoc. 94 (446), 590–599.

Snyder, C., Bengtsson, T., Bickel, P., Anderson, J., 2008. Obstacles to high-dimensional particle filtering. Mon. Weather Rev. 136 (12), 4629–4640.

Thainuruk, R., 2013. Using the Equivalent Weights Particle Filter to Predict Storm Surges for a Finite-element Model of the North Sea (MSc. thesis). University of Reading.

Tippett, M.K., Anderson, J.L., Bishop, C.H., Hamill, T.M., Whitaker, J.S., 2003. Ensemble square root filters. Mon. Weather Rev. 131 (7), 1485–1490.

van Leeuwen, P.J., 2009. Particle filtering in geophysical systems. Mon. Weather Rev. 137 (12), 4089–4114.

van Leeuwen, P.J., 2010. Nonlinear data assimilation in geosciences: an extremely efficient particle filter. Q. J. R. Meteorol Soc. 136 (653), 1991–1999.

Verlaan, M., 1998. Efficient Kalman Filtering Algorithms for Hydrodynamic Models (PhD thesis). Technische Universiteit Delft.

Williams, M., Schwarz, P.A., Law, B.E., Irvine, J., Kurpius, M.R., 2005. An improved analysis of forest carbon dynamics using data assimilation. Glob. Change Biol. 11 (1), 89–105.