Winds from a Bayesian Hierarchical Model: Computation for Atmosphere-Ocean Research

Timothy J. HOAR, Ralph F. MILLIFF, Douglas NYCHKA, Christopher K. WIKLE, and L. Mark BERLINER

Advances in computing power are allowing researchers to use Bayesian hierarchical models (BHM) on problems previously considered computationally infeasible. This article discusses the procedure of migrating a BHM from a workstation-class implementation to a massively parallel architecture, indicative of the current direction of advances in computing hardware. The parallel implementation is nearly 500 times larger than the workstation-class implementation from the data perspective. The BHM in question combines the information from a scatterometer on board a polar-orbiting satellite and the result of a numerical weather prediction model and produces an ensemble of high-resolution tropical surface wind fields with physically realistic variability at all spatial scales.

Key Words: Fortran 90; Gibbs sampling; Parallel computing; Spatio-temporal processes.

1. INTRODUCTION

Recent advances in Bayesian methodology and hierarchical models have created a new era in statistical methods for the geosciences. However, geophysical data of scientific import are often large, complex, and pose difficulties for a statistical scientist who is interested in transferring nontrivial methods to a substantial problem. This article traces through the computational process of combining two very different and large wind field datasets to estimate a surface wind field at six-hour intervals for a three-year period for a region of the tropical ocean. The result is a "data product" that is more suitable for scientific research than the original observations and can be distributed in a compressed and convenient format. We

Timothy J. Hoar is Associate Scientist, Geophysical Statistics Project, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307 (E-mail: thoar@ucar.edu). Ralph F. Milliff is Research Scientist, Colorado Research Associates, Division of Northwest Research Associates, 3380 Mitchell Lane, Boulder, CO 80301 (E-mail: milliff@cora.nwra.com). Douglas Nychka is Senior Scientist, Geophysical Statistics Project, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307 (E-mail: nychka@ucar.edu). Christopher K. Wikle is Associate Professor, Department of Statistics, University of Missouri, 146 Middlebush, Columbia, MO 65211 (E-mail: wikle@stat.missouri.edu). L. Mark Berliner is Professor, Department of Statistics, The Ohio State University, 1958 Neil Avenue, Cockins Hall, Columbus, OH 43210 (E-mail: mb@stat.ohio-state.edu).

Journal of Computational and Graphical Statistics, Volume 12, Number 4, Pages 781–807
DOI: 10.1198/1061860032616

use this focused project to illustrate some general concepts that are useful for other statistical groups tackling large geophysical problems or spatio-temporal datasets.

One motivation for these ideas comes from the interdisciplinary workshop on Massive Data Streams, convened by the Committee of Applied and Theoretical Statistics of the National Academy of Sciences December 2002. The workshop identified a range of massive data stream applications from such disparate fields as experimental particle physics, Internet communications, and Earth science. Applications based on the massive data streams implied by Earth-observing satellite systems differ from applications in particle physics or Internet communications in a fundamental way that we will explore here. In the case of the Earth-observing satellite data stream, data are rarely discarded and indeed, value-added datasets of the kind to be described here can *increase* the volume of data to be archived and reused. From a practical perspective, this project combines a large spatio-temporal input data stream with a computationally intense statistical model that results in a very large output data stream. Because we have chosen a simple format of drawing realizations from the posterior, the output data stream may be many times larger than the comparable input streams.

Another salient difference between geophysical data streams and other examples from the workshop is the spatio-temporal structure of the streams. Because geophysical data are often observations of a continuous process, the statistical models, at least formally, must have global extent and do not readily break into independent pieces. We have found it useful to characterize a *massive* dataset based on its context. Although our datasets are smaller than those of some other problems, the process models for the ocean surface wind require that many parts of the data are related to each other and these interactions add much more computational burden. This overhead is typical in spatial data problems where objects such as covariance matrices nominally have sizes that grow by the square of the number of observations.

Understanding how physical, chemical, and biological processes, as well as human activities, influence the complete Earth system is a grand scientific challenge for this century. Coupled with this integrated study is a rapid increase in Earth observations from spaceborne platforms. This includes new sensor systems, new types of target observations and finer resolutions and will result in exponentially increasing data volumes. A fruitful area for statistical science is to work with increasing observation dataset sizes without compromising the integrity of the statistical analyses. For example, often problems of "data fusion"—combining information from several sources or instruments—can be solved by hierarchical statistical models that relate the different forms of data to the geophysical quantities of interest. With large datasets and limited resources, however, it may not be possible to implement a hierarchical model in a straightforward manner.

1.1 Combining Data

We propose little new statistical methodology. We believe that Bayesian hierarchical models (BHM) are rapidly becoming a mature and flexible tool for handling complex geophysical processes. In contrast to the richness of the statistical methodology, we feel there is a deficit in principles for adapting the methods for large and interesting problems. The

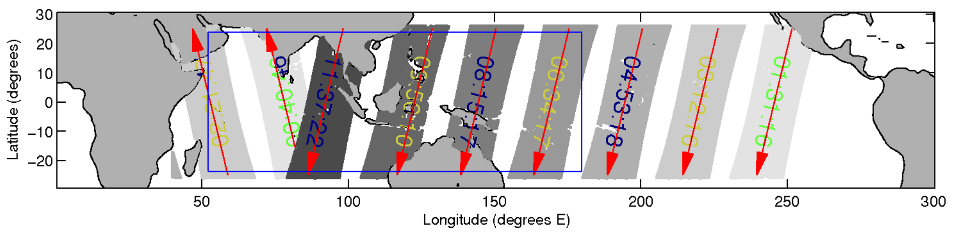


Figure 1. Seven consecutive orbits of SWS on QSCAT observation locations. The equator-crossing time (GMT) and direction are indicated. There are 114,552 observation locations in this domain in these orbits. Some of the ascending portions of the orbits are not in this domain. Some patchy areas of data dropout are also visible.

goal of this article is to outline the many ways in which running jobs on a workstation is fundamentally different from using a shared, high-performance computing environment. We illustrate some general ideas by tracing through the production of a value-added surface wind dataset for the tropical Indian and western Pacific ocean, based on combining scatterometer observations from the *SeaWinds* system on the NASA *QuikSCAT* satellite (*SWS* on *QSCAT*) and a more conventional numerical weather prediction (NWP) data source from the National Centers for Environmental Prediction (NCEP). The work was done at the supercomputer facility at the National Center for Atmospheric Research (NCAR) and made use of a massively parallel system (called *blackforest*) coupled to a mass storage device. To our knowledge this project is larger, by perhaps two orders of magnitude, than other BHM applications.

Our basic statistical task is to blend or fuse two complementary datasets to produce a product that is superior to either one alone. The scatterometer data provide observations with unprecedented density over the global oceans, but with irregularly spaced gaps in space and time (see Figure 1). The numerical weather prediction (NWP) winds are on regular spatio-temporal grid but lack sufficient detail (e.g., Liu, Tang, and Polito 1998 and Figure 5, p. 800) to fully investigate important physical phenomenon, nor do they provide any estimate of uncertainty in the winds at the espoused locations and times. Our goal is to produce wind fields that blend the high-resolution observations from QSCAT with the NWP-estimated state of the atmosphere at large scales. The final format is a regular spatio-temporal grid of sufficient resolution to address interesting scientific questions. Unlike more standard analyses of geophysical data, we sample the posterior of the space-time wind process to generate realizations of the wind fields that are physically consistent. For the geophysical community this is a novel product because we explicitly produce an ensemble (i.e., a random sample) of 50 realized wind fields whose variation quantifies the uncertainty in the estimates. The realizations also provide excellent input fields for perturbation studies. We will come back to this and other uses in Section 6.

1.2 ALGORITHM MIGRATION

The experience we gained from this project can be categorized in three ways: adapting to hardware constraints, managing input and output streams, and leveraging the advantages of several programming environments. The first involves balancing software design with available computing resources and storage. For a massively parallel but shared resource, this turns out to be a nontrivial effort and essentially dictates the scope of the domain that can be accommodated in an individual run. The second category involves the coordination of data flow from the input data stream to the computational core and the subsequent handling of the output data stream. We should note that throughout this article we use the term "data" in a cavalier way that may be confusing to some readers. Quite simply, we view any quantitative information as "data" and, for example, this will include the highly processed QSCAT observations described in the next section. Finally, the development of the computational core was simplified by the flexible combination of several software environments. The development of an efficient FORTRAN 90 (F90) computational engine is facilitated by incremental comparisons to the pilot version coded in Matlab®, an interactive, high-level language. The control of the computation in a batch environment is achieved by UNIX shell scripts and provides the ability to pass run-time information to a precompiled F90 executable. Although these problems might be solved in different ways, given the configuration of high-performance computing facilities and the nature of geophysical data, we feel that similar issues will need to be addressed for any analysis of a BHM with a large dataset.

The remainder of the article is organized to describe the details of these concepts. Section 2 gives an overview of the data streams and a brief review of the BHM for this project. The BHM and an initial application were developed in detail by Wikle, Milliff, Nychka, and Berliner (2001) (hereinafter WMNB) and served as the pilot project referred to below. Section 3 describes the NCAR computing resources and how they dictate our design, including the management of the data streams. The development of the software computational core is described in Section 4. Section 5 discusses production Gibbs sampling and Section 6 contains a discussion that also includes some examples of scientific projects based of the ensemble of surface wind fields.

2. DATA STREAMS, MODEL, AND METHOD

This section reviews the data sources that provide the input streams and a description of our target output data stream. As mentioned in the introduction, we use a BHM to model the tropical surface wind process and the output stream is generated as random draws from the posterior distribution. The statistical methodology is also reviewed here to the extent that it is needed to understand our computational strategies and choice of algorithms.

As an introduction to the extent of this project it is useful to outline the spatial and temporal domains that were considered. The tropical Pacific Ocean is a region of important dynamics involving atmosphere/ocean interactions and encompasses a key region in the formation of tropical storms that have unusual periodicity/predictability (Madden and Julian 1971). In order to resolve interesting features and produce a useful scientific resource, our target prediction domain has a time interval of six hours and a spatial resolution of 0.5°. The spatial domain spans the equatorial Indian and Western Pacific (52.25° E to 179.75° E, 23.75° S to 23.75° N) and the period is approximately three years (25-Dec-1999 to 28-Feb-2003, 1,162 days). These specifications include the WMNB region as a subset, but the complete space-time domain at the target resolution is twice as large, four times more dense, and 83 times longer.

2.1 WIND DATA FROM SCATTEROMETERS

Scatterometers emit radar pulses of known frequency, polarization, and incidence angles toward the Earth surface. The radar pulses are scattered by sea surface roughness elements (which are caused by the wind), and a portion of this backscatter is detected at the spacecraft. The final scatterometer data that we use is the result of several processing steps with a corresponding decrease in size of the data stream. Because this sequence is common to remotely sensed geophysical data, we outline the major steps.

Packed frames of raw sensor output representing the detected backscatter intensity are telemetered to surface ground stations (NASA calls this level 0 data—*L0*). In the first ground processing stage, the frames are unpacked and converted to backscatter units (called σ_0 for the radar backscatter cross-section—this is "level 1" data). In addition ancillary data is associated with these measurements including satellite ephemeris (the exact postion information of the satellite), calibration data, and time tags. The σ_0 are grouped in 25 km^2 wind vector cells (*WVC*) and a set of likely surface wind vectors (speed and direction) are estimated for each WVC using a geophysical model function (Freilich 1997) and median filter process for the inversion. The process of estimating geophysical quantities from the instrument measurements is termed a retrieval. The wind retrievals in geophysical units (e.g., m/s and degrees from North) are representative of what is called level 2B data (*L2B*); the grouped σ_0 are called *L2A* data. Producing a reliable wind retrieval is a large scientific and statistical effort; we have chosen to take the retrievals at face value, particularly since our observation model includes measurement error.

This article is concerned only with the science data product which represents the 10m winds over the ocean on a grid of WVCs corresponding to the sub-satellite swath. This is called the L2B product, and averages 111Mb per day. The following table gives an idea of the reduction in data volumes for a single SWS (D.J. Collins, personal communication 2003) and the size of one of the input streams that we used for this project.

Data processing stage	*Mb/orbit*	*Gb/year*
raw telemetry	35.35	183.4
level 0	33.27	137.2
level 1	180.92	941.5
level 2B	8.71	45.3
level 2B	1.1 million obs/day	

To put these numbers in perspective, SWS on QSCAT is a moderate-resolution single-sensor system. *Terra* and *Aqua* are recent multisensor platforms designed to measure suites of related geophysical properties of the Earth surface and atmosphere. The total L2 data volume from these systems is about 1 Terabyte per day. The data from SWS on QSCAT are archived and made publicly available via the Internet (see podaac.jpl.nasa.gov).

A feature that is intrinsic to orbital data is its awkward irregularity over both space and time. These are dictated by the orbital parameters of the satellite and sensor characteristics. QSCAT was launched on June 19, 1999, and carries a scatterometer which operates with a conically scanning geometry that sweeps out an 1,800-km wide swath centered on the sub-satellite point of the orbital path (see Figure 1). SWS provides 90% coverage of the global oceans every 24 hr. QSCAT orbits at an altitude of about 800 km and takes about 101 minutes to complete an orbit, in which time the earth has rotated about 25°. The satellite crosses the equator heading north at very nearly the same local time every day—6 a.m. ± 30 min. The recurrent period of the satellite (a repeat cycle) is 57 orbits (4 days).

2.2 Wind Data From Numerical Weather Prediction

The other source of information is the output of a Numerical Weather Prediction (*NWP*) model. The prediction process has two basic steps. Given new observations (reports from weather stations, weather balloons, etc.), the NWP model estimates a new current state of the atmosphere, called the *analysis.* A forecast is made using the analysis as the initial condition and advancing the NWP model. We are interested in the analysis fields; the best estimate of the state of the atmosphere at a given time. The combination of these data with the atmospheric model results in a physically consistent representation of the atmosphere at large scales. Typically, the smallest scales are the most difficult to estimate accurately. Our choice of NWP surface winds is the National Centers for Environmental Prediction (NCEP) Final Analyses (*FNL*) and represents a sophisticated determination for the state of the atmosphere system at the Greenwich Mean Times : 00Z, 06Z, 12Z, and 18Z. (Synoptic times are referenced to the local time at 0° longitude, the Prime Meridian passing through Greenwich, UK.) The surface winds of interest are bundled in the analyses, which contains estimates of 20 variables on a global 1° by 1° grid with many of the variables existing on multiple vertical levels. The daily volume of this compact GRIB-format (Dey 1996) dataset is 90Mb.

2.3 Output Dataset

In its raw form, the output data stream consists of the space-time fields generated by the Markov chain Monte Carlo computation. This chain is sampled after burn-in at 50 points, separated to ensure minimal dependence. The details of monitoring the MCMC are given in Section 5. We also reiterate that the output "data" stream is simply a 50-member ensemble intended to approximate the posterior distribution of surface winds.

2.3.1 Specifications for the Output Fields

There is some flexibility in the extent of the output stream. The wind fields are predicted for the same synoptic times as are available in the NWP data, simply because this is a common sampling interval for the intended audience and should provide sufficient detail for perturbational studies. The spatial domain is sampled at 0.5° resolution as this is near the information content of the satellite footprint. The exact spatial domain is chosen to give a dyadic (or near-dyadic, $2^5 \times 3$) number of grid points to allow for the use of fast and recursive discrete wavelet transforms. The choice of physical model constrains the prediction domain to within 30° of the equator (Gill 1982). It is neither feasible nor desirable to generate the entire dataset in one computation. The temporal domain (which we call an *epoch*) of each computation is dictated by the amount of memory available on blackforest. Currently, the maximum epoch duration is 14 days. The entire output dataset is created by assembling the result of multiple computations. More details on this aspect will be given in Section 5.1.1. These choices lead to the output stream specifications in the following table.

longitudes	128° @ 0.5° →	256
latitudes	48° @ 0.5° →	96
variables $[u, v]$		2
output realizations		50
# predictions every 6 hours		2,457,600
# predictions every day		9,830,400
# predictions in 14 day epoch		137,625,600

Representing each wind vector as a pair of 16-bit integers, and packing minimal metadata into a *netCDF* (network common data form) file, each file (representing a 14-day epoch) requires 262.5Mb. Storing the 83 files from 25 Dec 1999 through 28 February 2003 necessitates 21.28 Gb. The number of output realizations is somewhat arbitrary, and represents a compromise between the desire to fully explore the posterior distribution of the wind fields and concessions to concerns about data volume and distribution.

2.3.2 Output File Design

The final value-added product must be portable and entirely self-describing. NetCDF is a succinct, platform-independent binary format that may be read with the UNIDATA netCDF library (available from http://www.unidata.ucar.edu/packages/netcdf) as well as with software from many free and proprietary vendors. The netCDF F90 interface allows one to initialize a file, write the metadata, and then dynamically add to the existing structure. In our application, all the metadata is written by the initializing routine and, as each wind field realization is achieved, it is added to the netCDF file, along with any new metadata.

The netCDF format allows one to choose a parsimonious data type for each variable being stored. The coordinate variables `lat`, `lon` may be stored as 32-bit reals, the `time` coordinates may be stored as 64-bit reals, and the winds may be stored as 16-bit integers (that can preserve winds with a fidelity of 0.01 m/s). Each variable may be shifted and scaled by an offset and scale factor, which are automatically applied by routines which follow certain standards. Part of the global metadata of a netCDF file is which, if any, standard applies.

Useful metadata also includes the version of the software, the author, the creation date, the user input, bibliographies, and so on.

2.4 Statistical Model

We provide a brief review of the WMNB BHM to set the stage for the computational considerations, particularly those that pertain to the use of sparse matrices and the ability to distribute the computation. The interested reader will find full details of the model design in WMNB. For clarity, we discuss only one component of the wind vector, the other component is treated similarly. The Bayesian hierarchical model as implemented, has three parts:

(a) a data stage or likelihood distribution, [data|process, θ_1];
(b) a process model stage [process|θ_2]; and
(c) prior distributions for model parameters, [θ_1, θ_2].

Data Stage: This stage accounts for the measurement error (Σ_t), given the true wind ($\mathbf{u}_t$).

$$\mathbf{U}_t|\mathbf{u}_t, \mathbf{\Sigma}_t \sim \text{Gau}(\mathbf{K}_t\mathbf{u}_t, \mathbf{\Sigma}_t),$$

where $\mathbf{U}_t$ are the surface wind estimates from the scatterometer and the NWP models which we treat as observations and $\mathbf{K}_t$ is a sparse matrix that maps the prediction grid locations to the expected value of the observations. As described in WMNB, Gaussian assumptions were found to be appropriate for these distributions, with parameters for the SWS on QSCAT measurement error model being taken from Freilich (1997).

Process Model Stage: A key innovation in WMNB was the implementation of process models based on dynamical balances and turbulence properties appropriate for the tropical atmosphere. Large-scale propagating spatial patterns (columns of $\mathbf{\Phi}$) are implemented consistent with the normal modes of the equatorial β-plane approximation (e.g., Matsuno 1966; Gill 1982). Here the columns of $\mathbf{\Phi}$ index different basis functions and the rows index the grid of spatial locations (256 $\times$ 96). Given basis functions coefficients $\mathbf{a}_t$ the field can be evaluated at the output grid through the multiplication $\Phi\mathbf{a}_t$. In the larger domain, the set of relevant normal modes has expanded to included larger scales excluded from the WMNB implementation. The additional parameters were obtained from Wheeler and Kiladis (1999), as they were in WMNB. Small-scale variability is modeled using nested wavelet bases $\mathbf{\Psi}$ and prior variances on coefficient $\mathbf{b}_t$ imply a power-law dependence of surface wind kinetic energy on spatial wavenumber (see Wikle, Milliff, and Large 1999). With respect to WMNB, the wavelet basis is extended to an additional level to account for higher resolution (0.5°) in the posterior realizations of the tropical surface wind fields. The process model is

$$\begin{aligned}
\mathbf{u}_t &= \boldsymbol{\mu} + \mathbf{\Phi}\mathbf{a}_t + \Psi\mathbf{b}_t \\
\boldsymbol{\mu} &= \mathbf{P}\boldsymbol{\gamma} \\
\boldsymbol{\gamma} &\sim \text{Gau}(\boldsymbol{\gamma}_0, \mathbf{\Sigma}_\gamma)
\end{aligned}$$

$$\begin{aligned} \mathbf{a}_t &\sim \text{Gau}(\mathbf{H}_a \mathbf{a}_{t-1}, \Sigma_{\eta_a}) \\ \mathbf{b}_t &\sim \text{Gau}(\mathbf{H}_b \mathbf{b}_{t-1}, \Sigma_{\eta_b}), \end{aligned}$$

where $\boldsymbol{\mu}$ is the mean wind, $\mathbf{P}$ is a $\{0, 1\}$ land/sea mask design matrix, and $\mathbf{H}_a$, and $\mathbf{H}_b$ are diagonal, autoregressive propagator matrices. ($\mathbf{H}_a$ is block diagonal to allow for interaction between the real and imaginary components for each wave mode.)

Parameters: The θ_1 θ_2 here are placeholders for a variety of parameters necessary to specify data stage (θ_1) and process model (θ_2) distributions. WMNB demonstrated that hierarchies of relatively simple conditional distributions could achieve physically realistic complexity in spatio-temporal structure for marginal distributions that would otherwise be unachievablein an "all-at-once" model specification. Model sensitivitiesto parameter values and distributions have been investigated and/or discussed for many of the specifications in WMNB, and this remains an area of interest. The efficient computational implementation to be described here permits more extensive sensitivity testing than was practical for WMNB.

Based on these distributions, the basic calculation is to characterize the posterior:

$$[\text{process}, \theta_1, \theta_2 | \text{data}].$$

Through the choice of conjugate priors, one can derive closed form full conditional distributions for groups of parameters and the wind field. Thus, we use a Gibbs sampler to generate a Markov chain whose stationary distribution corresponds to the posterior. A key step in the MCMC is sampling the full conditional of $\mathbf{b}_t$. Although not obvious from this presentation, both of these steps involve a computation that is equivalent to solving a large linear system ($\mathbf{b}$ has dimension 24,576). The linear system has little symmetry due to the irregular spatial sampling of the QSCAT observations, however, one can take advantage of the sparsity of $\mathbf{K}_t$ and also the ability to multiply $\boldsymbol{\Psi}$ and $\boldsymbol{\Psi}^T$ with a vector efficiently. These factors lead to the use of a simple iterative method, the *conjugate gradient* algorithm (Golub and Van Loan 1996, sec. 10.2), to find an approximate solution to the linear system.

3. PRACTICAL IMPLEMENTATION

For algorithm development, a state-of-the-art SUN® workstation running Matlab was sufficient. However, even for the smaller domain in WMNB, it took about three days to exhaustively complete a single 14-day period, one epoch. A rough calculation shows that the full production targets are about 500 times larger, implying more than 1,500 CPU *days* to complete the calculations. This estimate also does not take into account the companion demands of managing larger input and output data streams. Clearly, moving to a bigger, faster computer was in order! This section describes the ramifications of the available hardware and software. Two of the fundamental tenets are that we want to avoid estimating the entire three-year period as a single BHM and we want to distribute the computation to many processors. The following section describes how we adapted this problem to the specific hardware and batch environment available at NCAR and created a software and data management framework that made the production runs possible.

3.1 HARDWARE

NCAR's computing environment is very diverse and dynamic. We chose to use an IBM symmetric multiprocessor (*SMP*) platform running the AIX operating system. NCAR has two distinct, but nearly identical, machines of this nature; one for interactive use and debugging, and one for production. The production machine (blackforest) has 293 Winterhawk II RS/6000 nodes, each with four 64-bit POWER3 CPUs and 2Gb of memory shared among the four processors. Each of the 1,172 375MHz POWER3 CPUs is capable of executing up to four instructions per clock cycle. This capability can be contrasted against the CPU speed of a typical workstation where multiple clock cycles are required to execute a single instruction. The machine for interactive use simply has a much smaller number of nodes and separate disk space. There are about 13 Tb of disk available to blackforest, shared among many projects.

The primary data archive at NCAR consists of a series of robotic silos that contain tape cartridges, the mass storage system *MSS*. When a request is made, the appropriate cartridge is retrieved and placed in one of the tape drives to be read. Each cartridge is a serial device with a capacity of about 6Gb. If the desired file is near the end of the cartridge, the entire cartridge must be read The process is not instantaneous, and significant wall-clock time can elapse for many requests.

3.1.1 Choices Dictated by the Hardware

The BHM calculations necessitate all the data for an epoch to be accessible throughout a Gibbs iterate. Retaining the data in memory is crucial for performance, swapping to disk is unacceptably slow. Therefore, our definition of "big" is defined by the memory available. The epoch length was chosen such that the memory high-water mark is less than 512Mb so that we can maximally use the 2Gb of memory shared between the 4 processors on a node. The high-water memory depends on the number of scatterometer data, typical high-water marks are about 460Mb per epoch.

Blackforest is a compute-server where jobs are submitted to a queue where they wait to be executed. At any time, multiple jobs are distributed throughout the nodes. The queueing protocol mandates that no single execution can accumulate more than six hours of time (on a single CPU). This means the job must be structured such that it can be resubmitted multiple times and pick up from where it left off. This is good practice in general and provides a convenient restart mechanism.

3.2 SOFTWARE

The use of Matlab for coding the pilot study was very successful. Matlab has easy, natural syntax, a great facility to improve efficiency, and has native sparse matrix support. (The $\mathbf{K}_t$ matrices that map the data to the prediction grid are huge, but sparse.) Unfortunately, there is no native parallel support, which precludes the use of a network of workstations and/or the efficient use of Matlab on large, parallel architectures.

3.2.1 Pilot Versus Production Implementation

The decision was made to implement the Gibbs sampler in a low-level, portable, non-proprietary language. F90 was chosen for many reasons, some of which are: syntactic similarity to Matlab, the long history of excellent Fortran support on supercomputers, fast libraries, the parallel-processing capability (using the message passing interface—MPI), and the ability to declare user-defined types and overload operators. The main drawback to F90 is that there are no native sparse matrix algebraic routines.

3.2.2 Code Verification

Coding tasks were logically compartmentalized and components were developed and checked offline before inserting new code into the main routine. Then, the integration of the components was verified. The Matlab debugger allows breakpoints, so the Matlab code was advanced to the desired point, and the binary result of the F90 code was read into Matlab and the results were compared.

The verification of a successful port to F90 was, in itself, an interesting endeavor but was simplified by the precursor implementation in Matlab. The Matlab applications programming interface (*API*) provides the ability to replace routines on a component-by-component basis by writing Matlab wrappers around low-level source code. Thus, F90 modules could be developed and tested individually in the context of the Matlab code.

The biggest obstacle was trying to get both Matlab and F90 routines to use the same predictable set of random numbers from multiple distributions. Ultimately, identical streams of random numbers were created by setting known seeds for the Matlab random number routines (different distributions had their own seeds) and generating machine-specific binary output files (one for each distribution) that were then read by the F90 routines in place of the F90 library random numbers. Using binary files was essential for limiting numerical roundoff differences caused by the conversion to ASCII. Preprocessor directives were used to conditionally compile different blocks of code in the same source, providing a means for the same random number routines to either call the library routines or read from the binary file. This is much easier than maintaining separate routines, files, and interfaces. See the Appendix for an example.

The number of iterations required for convergence in the conjugate gradient routine (Wikle et al. 2001, appendix) proved to be a very useful run-time diagnostic. Often the first indication that an error was encountered was an inordinate number of trips through the conjugate gradient routine. A particularly sinister error was detected in this manner. Due to a change in data format, the NWP data was ingested "upside-down" relative to the scatterometer data and required a large number of conjugate gradient iterations.

3.3 DATA MANAGEMENT

Because each production run requires substantial "real" time, efforts are made to ensure the availability of the necessary files throughout the duration of the job. As is common on shared resources, the available filesystem runs a scrubber and the existence of the files cannot be guaranteed for the lifetime of the entire series of job submissions. The necessary files must be copied to an archive for safekeeping, should the successive script need to recover them to continue processing. At NCAR, the archive is the MSS. The files are only retrieved if they do not exist on the local disk. However, they always need to be archived, which causes some wall-clock delay in processing.

NCAR's vast data holdings (including our NWP and scatterometer data) exist on the MSS, which is accessible to many compute-servers. Preprocessing the data minimizes the wait during the production jobs and condenses the input data to a more manageable size. This can be done offline and not impact the highly competitive resources of the production machine. Preprocessing the data also provides an opportunity to discover and correct any problems with the incoming data files, be they corrupt or simply missing. The scrubbers are much less of a problem, and any data problems that may have stalled a production run have been removed.

4. SOFTWARE DESIGN

In this section we take the reader through the software framework in the order of execution. We divide the computation into two parts: (1) initialize the Gibbs sampler, and (2) advance the sampler. The time limits on the job queues require step (2) to be executed repeatedly to advance the sampler to the desired state. This is controlled through the use of two shell-level scripts, which are described in Section 4.1. Each execution of each script requires many CPUs, each CPU (nearly) independently working on a separate epoch. This distribution of the computational burden is covered in Section 4.2.

It is desirable to move as much of the programming burden as possible to the compiler. To this end, we employ user-defined types, their operators, and generic interfaces to underlying routines as described in Section 4.3. All of these mechanisms improve code readability and reduce typographical and logical errors. Despite being conceptually simple, ingesting the data and user input is usually one of the most difficult tasks. Section 4.4 describes this process. Section 4.5 describes the scenario when those jobs have input and output files that depend on the number of CPUs being used and achieving a graceful termination under the threat of the job scheduler.

4.1 Batch Scripts

The job control is separated into two scripts (written in `ksh`) that initialize the Gibbs sampler and then advance the sampler. A *cookie* file that records the progress of the sampler is tracked throughout the resubmission process. The parallelization occurrs within the F90 executable via MPI directives, not by submitting nearly a hundred single-processor jobs (each of which would have to resubmit themselves).

The initializing script prepares the output data directory and cookie files, checks for the existence of the input data, and converts all user input into appropriate input files as

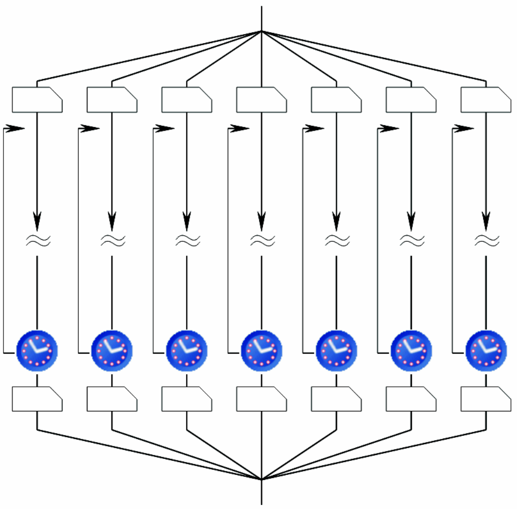


Figure 2. Illustration of our SPMD implementation. A single executable forks multiple tasks, each of which is autonomous until completion. If a task has enough time to complete another Gibbs iterate, it does.

described in Section 4.4 before finally running the F90 executable. Finally, the restart files are copied for safekeeping and the second script is submitted. The (second) script to advance the Gibbs sampler simply reinstates the restart files if necessary and starts the F90 executable. At the end of an execution, the restart files are archived and the cookie file is queried and decremented. Should another execution be necessary, the script resubmits itself.

4.2 PARALLEL COMPUTING ISSUES

Under the assumption that each epoch is processed independently, one can recognize our problem as embarrassingly parallel. Computation for each epoch is distributed to a separate processor (an MPI *task*) and no subsequent interprocess communication is necessary. To be precise, we are using a single *executable* with many processors, each with their own data streams and disjoint memory use, as illustrated in Figure 2. Technically, the F90 code is written under the single program multiple data (SPMD) paradigm. Although there are more CPUs available (1,172) than epochs (83), we choose to submit smaller batches of epochs and verify the results before submitting the next batch. This strategy is *necessary* for the Gibbs coupler described in the next section. Executing the job a handful of times is much simpler than executing 83 separate jobs. This SPMD strategy is very efficient from the CPU perspective; there is no downtime waiting for another CPU to gather/scatter information, except for once at the beginning and again at the end of the entire process. Each task needs its own control file, coding the task ID (`myid`—available through the MPI library) into the control file name was an effective way to ensure this. A version of the code is under development that also uses multiple processors to share a single computationally intense calculation, the conjugate gradient algorithm.

4.3 GENERIC, PORTABLE INTERFACES

To improve code clarity, reduce coding errors, and improve portability, generic interfaces to common routine calls are defined. For example, a call to invert a matrix with a routine called `inv` is replaced by the compiler with a call to `r8inv` if called with a 64-bit matrix, or `r4inv` if called with a 32-bit matrix, and the appropriate library calls are made to perform the inverse. Furthermore, preprocessor switches (defined at compile-time) in `r8inv` (and `r4inv`) signal which library routines are available. All available platforms are supported with *one* source code, an example of which is supplied in the Appendix.

The incidence matrices K_t are huge, but sparse. Implementing these as a user-defined type and using F90 interface operators (operator overloading) enables the programmer to treat these objects much the same as any other object. There is no symmetry to exploit in these matrices, hence the operators are for the general case. One can foresee symmetry-specific types with operators that can exploit the symmetry. Because of the compact representation, performance on cache-based machines will be enhanced. The following stylized interface block for the addition operator generic shows how the compiler sorts out which routine to call for a specific scenario of input argument types.

```
TYPE sparse                          ! Defining the type
integer(kind=int32)                              :: nrows,ncols,nelems
integer(kind=int32), DIMENSION(:), POINTER :: irow
integer(kind=int32), DIMENSION(:), POINTER :: jcol
real(kind=real64),   DIMENSION(:), POINTER :: valu
END TYPE sparse

INTERFACE OPERATOR( + )              ! A = B + C   matrix addition,
                                       no visible loops
MODULE PROCEDURE spmatPr4mat         ! sparse matrix + 32bit real matrix
MODULE PROCEDURE spmatPr8mat         ! sparse matrix + 64bit real matrix
MODULE PROCEDURE spmatPi4mat         ! sparse matrix + 32bit integer matrix
MODULE PROCEDURE spmatPi8mat         ! sparse matrix + 64bit integer matrix
MODULE PROCEDURE r4matPspmat         ! 32bit real matrix + sparse matrix
MODULE PROCEDURE r8matPspmat         ! 64bit real matrix + sparse matrix
...
END INTERFACE
```

4.4 INITIALIZATION OF THE GIBBS SAMPLER

Multiple jobs are likely to be running simultaneously, and resubmitted automatically, it is undesirable and unnecessary to compile a unique executable for each job sequence. It is better to have a single executable whose behavior is controlled by flexible input files. The F90 standard formalizes the use of the `namelist` mechanism for flexible I/O. (A sample namelist and concomitant `read` are in the Appendix.) The variables may be supplied in any order and the values may be either scalars or arrays, as appropriate for the declared type. Default values may be set before reading the namelist, so not all variables need to be present in the namelist. Furthermore, everything past the namelist terminator is ignored by the program; we find it useful to include comments documenting the input. The control files for the next job submission are namelists written by the F90 program as it completes. The use of namelists is key to automatic generation of job control files for a subsequent execution.

```
!-----------------------------------------------------------------------
!  The 'state' namelist will simply contain the active filenames.
!-----------------------------------------------------------------------
...
character(len=80)    :: logfname, statefname, netCDFfname
NAMELIST / stateinfo / logfname, statefname, netCDFfname
...
write(fnm,'(''State_TaskID_'',i2.2,''.namelist'')')  myid
open( NMLfid+myid, file=fnm)
read( NMLfid+myid, NML=stateinfo)
close(NMLfid+myid)
call GetState()         !  and now I have everything ...
...
```

4.5 RESTARTS

Providing a restart mechanism is standard practice for CPU-intense methods. Our hardware essentially requires it by limiting the CPU-hours available to any one job. Moreover, the user is expected to know how long a job will run so it can be submitted it to the proper queue. The job schedulers use the queues to balance the load on the computer, so simply using the longest queue results in suboptimal machine usage. Exceed the expected time and the schedulers simply halt the job, without regard for the consequences. Files are usually lost or corrupt. Timing routines in the F90 code are required to ensure a timely, graceful termination. Before each iteration of the Gibbs sampler, the remaining CPU time is estimated and compared to the recent (i.e., over the last few Gibbs iterates) mean iterate time. If sufficient time remains, the iterate proceeds (as depicted in Figure 2). Experience dictates how much time is necessary to write the control and restart files. With the SPMD model described previously, each task reads its own control namelist, which contains the names of the restart, output, and diagnostic files. The restart files represent the state of the program memory and are written in binary to avoid loss of precision upon being reinstated. Because each task is running independently, each task is free to complete as many Gibbs iterations as possible in the allotted time.

5. PRODUCTION GIBBS SAMPLING

Formal Gibbs sampler convergence is still an area of active research, especially for high-dimensional state-spaces. This section describes some practical decisions we made to quantify convergence, complete the sampling, and compare the (approximated) posterior wind fields to an independent dataset.

5.1 Burn-in and Dependence Among Gibbs Iterates

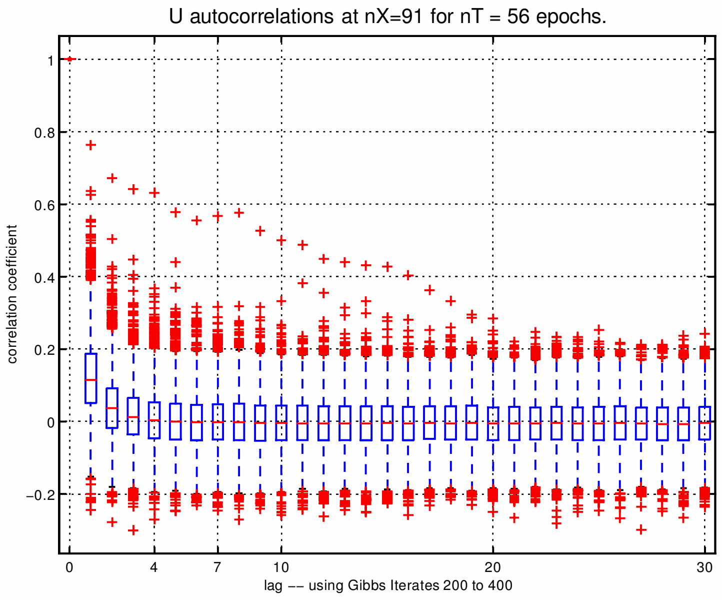
Knowing when the Gibbs sampler has sufficiently stabilized and one can start drawing samples is an open question. We employ a practical method that balances scientific demands with computational expense. To assess convergence, we computed a 1,000-iteration chain on a test epoch and divided the iterates into batches of 50 space-time fields. (In the pilot study, the convergence criterion of Gelman and Rubin (1992) was met after 700 iterations.) The means and variances for the batch (201–250) and batch (951–1000) were similar enough to limit the production burn-in to 200 iterations.

Having established a benchmark for burn-in we then determined a useful sampling frequency for the Markov chain. To make this determination we focused on the realized wind fields rather than the unobserved parameters in the hierarchy. Figure 3 is a plot of the autocorrelations across Gibbs iterates 200–400 at 91 select grid locations (see Figure 4) for all 56 time steps in a given epoch. The mean autocorrelation is essentially zero after three lags. Consequently, we skip three Gibbs iterates between samples. The Gibbs iterate of the sample is also recorded in the netCDF output file.

5.1.1 Temporal Continuity of Epochs

We use disjoint prediction epochs in our attempt to span the three-year period of interest with the rationale that it is the most computationally efficient possibility. This represents a naive approach to sampling over the entire time period. The autoregressive part of the model, of course, connects the current time with the previous time. Based on this dependence, the interior points in the epoch have full conditionals, say for $\mathbf{a}_t$, that depend on the states at both $t-1$ and $t+1$. If the calculations are kept separate, the full conditional distributions for the states at the endpoints are not conditioned on states outside the epoch. At first this appears to be a poor approximation to the complete posterior over time. States in adjacent epochs are calculated without ensuring continuity over the epoch boundaries. However, when we scrutinize the time series from simply concatenating the MCMC results for separate epochs we *cannot* detect the epoch boundaries! (For such a timeseries, see Figure 4.) Such surprising results are fortuitous in that we do not see a need to add any additional complexity to our sampling to facilitate splicing epochs together. Certainly treating each epoch individually is the easiest strategy on the available multiprocessor hardware.

We conjecture that the temporal component of the model does not have a strong influence due to the regularity of the NWP wind data and the temporal sparsity of the QSCAT



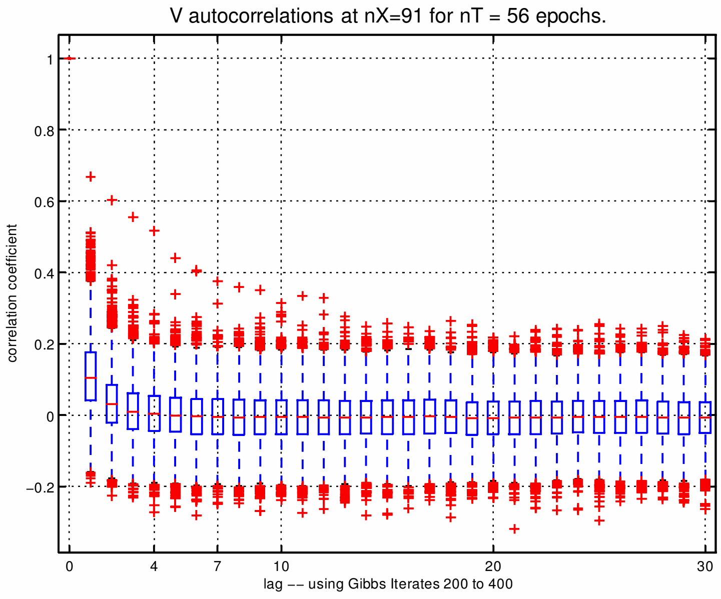


Figure 3. Autocorrelations over Gibbs iterates 200–400 of the U (top) and V (bottom) components of wind at the 91 test locations for each of the 56 epochs, all 50 ensemble members.

observations. The NWP fields essentially provide the temporal backbone and the small-scale features associated with the QSCAT observations were infrequent enough in time that in most cases they were not propagated by the autoregressive component in the model. For this reason adjacent time points would tend to be correlated due to the continuity of the NWP data even if these times straddled an epoch boundary.

We consider these results peculiar to the particular form of the data being used and,

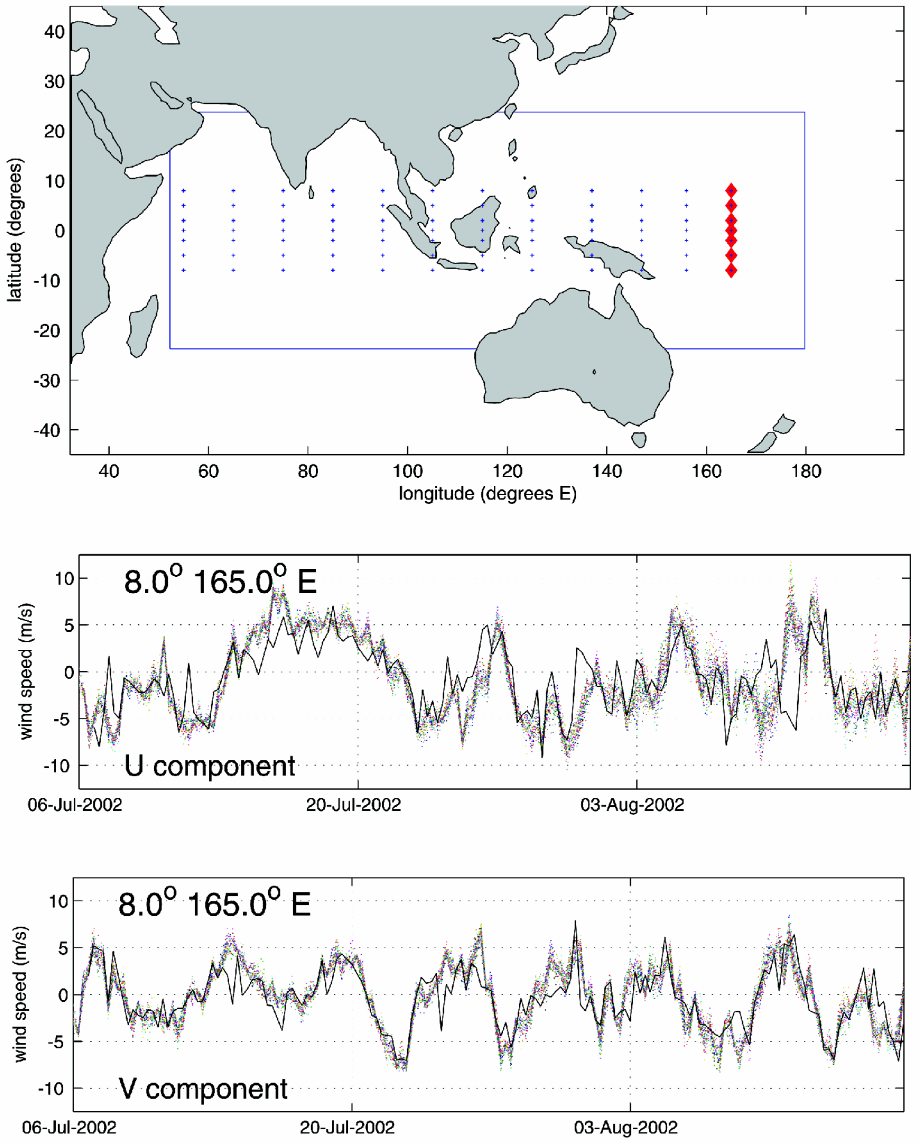


Figure 4. The locations (top) of the TAO buoys and the prediction domain. Also plotted are the 91 test locations used for the autocorrelation study of Section 5.1. The buoy wind (bottom) is plotted as a solid line which overlays 50 dotted lines of the BMH ensemble members. The dates annotated are the edges of the epochs.

Table 1. Schematic of the Gibbs Coupler. The "o" represents the endpoints of the previous (Odd) set of epochs, which are used to condition the states for the Even epochs. The "e" represents the endpoints of the Even epochs, which are used to condition the subsequent calculation of the Odd epochs.

	Independent simulations					
no overlap	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
	Gibbs Coupler					
[Odd]	[ ]		[ ]		[ ]	
[Even | Odd]		o[ ]o		o[ ]o		o[ ]
[Odd | Even]	[ ]e		e[ ]e		e[ ]e	

in general, we would expect epoch boundary effects. We suggest a modification to the sampling to accommodate conditioning between epochs and so reduce any biasing artifacts at epoch boundaries. This strategy uses the same software framework that was built for the independent epoch case, but involves an additional complexity, schematically represented in Table 1. For reasons that will become clear, we term this the *Gibbs coupler.*

The first line of Table 1 is a representation of calculating independent epochs. The subsequent lines are the algorithm we suggest for generating epochs conditional on adjacent endpoints. One first generates every other epoch in an independent manner—for clarity we will call these the *Odd* epochs. One now generates the remaining epochs, the *Even* ones, but uses the endpoints of the Odd epochs for a conditional simulation. That is, the full conditional distribution for the state at the endpoint of an Even epoch is conditioned on the state from the adjacent time point from the Odd epoch. The final step is a natural iteration; given the Even states, one now repeats the sampling of the Odd states while conditioning on the Even boundary states. The reader will recognize this scheme as the alternation of simulating odd and even epochs as just a Gibbs iteration, albeit costly, and one could repeat this process "until convergence." If one stops after the second iteration, the extra computational burden is only increased by about 50%. The first two steps are similar in the amount of CPU time as the independent case. The last iteration, conditioning the odd epochs, requires half again as much computing.

5.2 VERIFICATION AGAINST INDEPENDENT OBSERVATIONS

The Tropical Atmosphere Ocean (TAO) project began deploying a basin-scale array of moored buoys in 1992 to collect observations in the Tropical Pacific Ocean. The array is now called the TAO/TRITON array and consists of approximately 70 moorings which telemeter oceanographic and meteorological data to shore in real-time via the *Argos* satellite system. Of the buoys that archive winds at a 10-min resolution, only seven are in our prediction domain and are indicated in Figure 4.

There are several reasons why we do not expect perfect correlation between the buoys and the predicted winds. The buoys measure winds 30 times per second at a 4m height at a single location and the measurements are averaged over a 2-min sampling interval. The scatterometer winds represent a near-instantaneous representation of the winds at a 10m height over a 25km^2 region. Nonetheless, the buoy winds are the winds most likely to measure something similar to what the scatterometers measure. Figure 4 is an example of a typical timeseries of wind components from the buoys and the GibbsWinds product at the nearest prediction location. Although the prediction grid is actually staggered with respect to the buoy locations, no spatial interpolation was done for the comparison; the buoy winds were linearly interpolated to the prediction times.

5.3 Output Examples

Figure 5 demonstrates the result of the wind blending process for one ensemble member

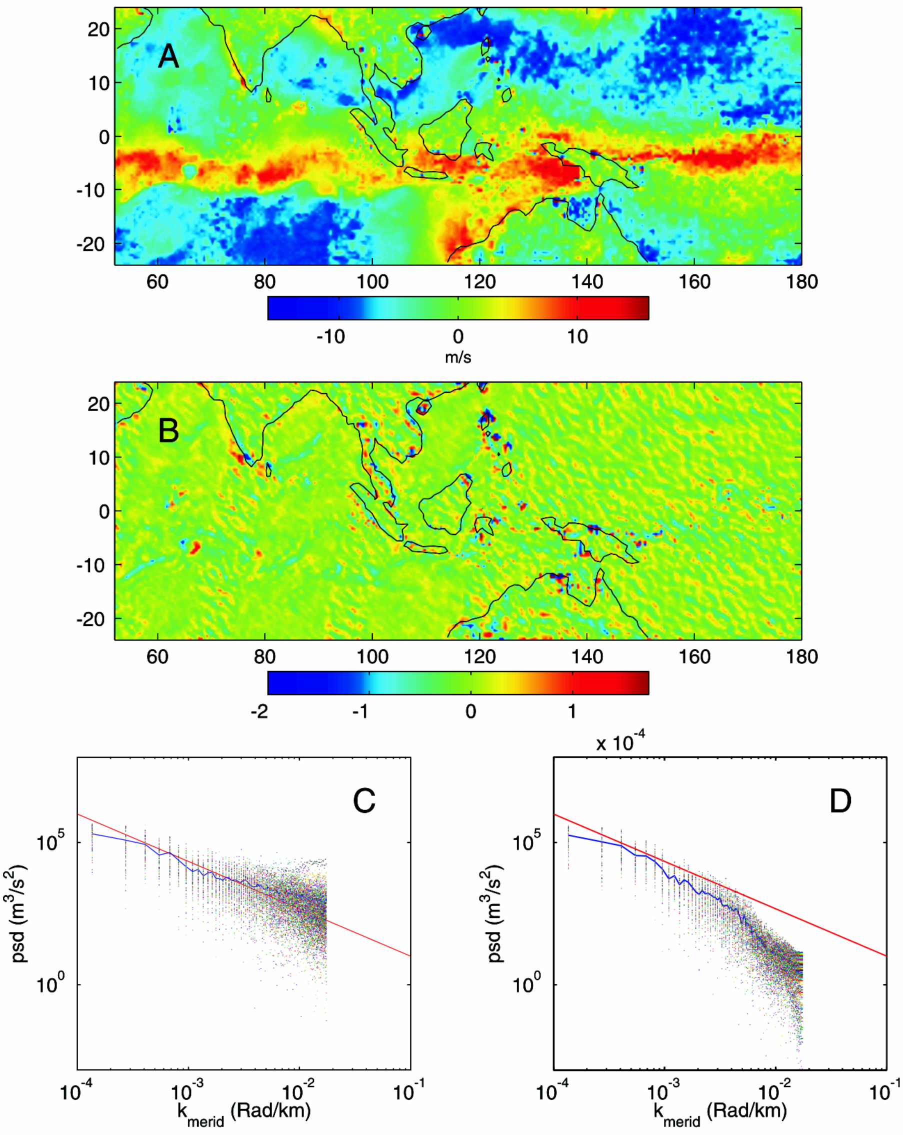


Figure 5. (A) One U wind field ensemble member for 12Z 04-Jan-2003. (B) The divergence ($\partial U/\partial x + \partial V/\partial y$) of an ensemble member at the same time. (C) is the power spectral density of each of the 96 lines of longitude in the domain for the ensemble member in (A). (D) is the power spectral density of the NWP data for the same time, linearly interpolated to match the resolution of the ensemble member. The slope of the line in both C,D is $-5/3$.

at one prediction time. The periodogram (one for each of the 96 latitudes) of the U component of the wind is shown at the bottom left. The same calculation is performed on the NWP wind component linearly interpolated to the same locations. Note the deficiency in energy at the smallest scales, indicated by a departure from the line whose slope is $-5/3$. Notice the improvement in the energy content of the ensemble member.

The resulting output file header demonstrates the self-describing nature of the netCDF format and is as follows:

```
netcdf Gibbs_20020413 {
dimensions:
        lat = 96 ;
        lon = 256 ;
        time = 56 ;
        member = 50 ;
variables:
        float lat(lat) ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:valid_range = -23.75, 23.75 ;
        float lon(lon) ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:valid_range = 52.25, 179.75 ;
        double time(time) ;
                time:long_name = "time" ;
                time:calendar = "julian" ;
                time:cartesian_axis = "T" ;
                time:units = "days since 0000-00-00 00:00:00" ;
                time:example = "730479.5 ==> 1999 Dec 25 12Z" ;
                time:valid_range = 731319., 731332.75 ;
        short member(member) ;
                member:long_name = "ensemble member ID,
                  wind realization ID" ;
                member:valid_range = 1, 50 ;
        short u(time, member, lon, lat) ;
                u:long_name = "zonal wind at 10m" ;
                u:units = "cm/s" ;
        short v(time, member, lon, lat) ;
                v:long_name = "meridional wind at 10m" ;
                v:units = "cm/s" ;
        short iteration(member) ;
                iteration:long_name = "Gibbs Iteration" ;
// global attributes:
              :title = "Blended QuikSCAT JPL DIRTH and GDAS FNL 1x1 winds:
              wind components at 10m." ;
              :reference = "Wikle, Milliff, et.al., Spatiotemporal
                Hierarchical
              Bayesian Modeling, JASA Vol. 96 No. 454, June 2001
                pps 382-397" ;
              :author = "Created by Tim Hoar (thoar@ucar.edu)";
              :SoftwareVersion = "CVS:Winds V1_2" ;}
```

6. DISCUSSION

A successful migration of software from a workstation-class implementation to a server-class implementation requires planning. We have outlined the process in several major steps: (1) consider the hardware, (2) manage the data streams, and (3) customize the software. Frequently, these topics cannot be considered in isolation. For example, the filesystem scrubbers (step 1) have a profound influence on the management of the data streams (step 2), and the queueing structure of the hardware imposes some severe design constraints on the organization of the software. Dividing the tasks (checking if a resubmission is necessary, managing the restart files, creating control files, etc) between the executable and the batch scripts allows for a portable solution that provides failsafe methods for restarting a job should a hardware or software problem halt execution. Furthermore, should a different archive mechanism become available, only the scripts would need to be changed, for example. Good programming practices (using interface procedures, portable F90 modules, user-defined types and operators, conditionally compiled code segments) allow for code reuse when the hardware changes.

6.1 Science Applications in Progress

Bayesian methods contribute formal uncertainty management procedures for data, model, and data-model applications in Earth science. The outputs of Bayesian analyses and BHM are probability distributions that interface well with notions of ensemble forecasting and error analysis that are active areas of Earth science research.

A simple example of the utility of a posterior distribution of tropical surface wind field realizations has to do with an eastward-propagating large-scale feature of the general circulation known as the Madden–Julian oscillation (MJO). For a review of the MJO see Madden and Julian (1994). For our purposes, it is sufficient to know that repeating patterns in the surface wind convergence at the equator propagate eastward in the Indian and western Pacific ocean basins on 40 to 50 day timescales when the MJO is active.

In the absence of a posterior distribution during a single MJO cycle, surface wind

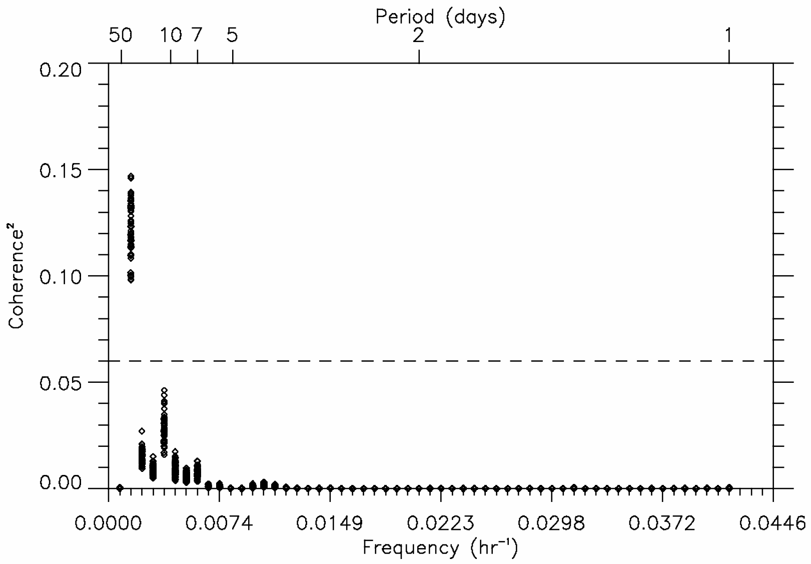


Figure 6. Squared coherence versus frequency for 50 realizations of the meridional wind on the equator at two locations spanning the Indian Ocean (55° E and 75° E), for the period 23 December 2000 through 21 February 2001. The time period has purposefully been chosen to correspond to the active phase for a Madden–Julian oscillation (MJO) in the region. The surface wind realizations from the Gibbs Sampler implementation described in this article permit estimates of uncertainty (i.e., the spread in coherence-squared estimates) in the spectrum without having to composite the MJO over several cycles, or average over several discrete frequencies. The dashed line ($coh^2 = 0.06$) represents a 95% confidence interval with respect to the null hypothesis of no coherence squared (R. Madden, personal communication 2003).

realizations from several cycles have to be composited in coarse-resolution longitude bins for analyses of MJO propagation statistics and mechanisms (e.g., see Madden, Hoar, and Milliff 1999). MJO composites of this kind necessarily combine large-scale background fields that can be quite different from one MJO cycle to the next, and certainly different for MJO cycles that are not members of the same oscillating set (e.g. say from different phases of ENSO). As such, composite analyses of MJO cycles have not been definitive with respect to high-resolution aspects of the propagation mechanism.

Given the high-resolution posterior distributions of physically consistent surface wind fields developed here, one can refine mechanistic descriptions of the MJO propagation while avoiding contamination from differing background states of the large-scale circulation. For example, from the distributions of time series for zonal winds at two prediction locations spanning the Indian Ocean (at 55° E and 95° E), we compute coherence and phase spectra for the time period of a single MJO active phase (23 December 2000 through 21 February 2001; see Figure 6). Fourier transforms of the time series at each station lead to co- and quadrature spectra with 2 degrees of freedom (one for each Fourier coefficient) for each discrete frequency, for each of 50 realizations; that is, 100 degrees of freedom for each discrete frequency without frequency binning. The amplitude of coherence-squared between stations, while modest, is significantly different from 0 (null hypothesis), for the discrete frequency corresponding to the period of the MJO.

This simple application makes use of the posterior distributions at two points in our tropical domain. More involved applications of the wind field distributions in forcing ensembles of numerical ocean model simulations are under development and will be reported elsewhere.

6.2 Code and Product Access

Many of the F90 modules and scripts are available from the GSP Web site www.cgd.ucar.edu/stats/GibbsWinds and are also available upon request (send requests to thoar@ucar.edu). Code examples include defining sparse matrix data types, sparse matrix operators, defining conditionally compiled code, the use of generic interface blocks, and more. The scripts to manage the batch submission process are clearly tailored to the NCAR environment, but should provide a template that will readily customize to many installations.

The result of the wind-blending process is a set of DVDs, each of which contains 17 netCDF files, one for each epoch. Each DVD therefore contains wind fields for 238 days, about 4.5 Gb. The DVDs are available upon request, the details are provided on the GibbsWinds Web site.

APPENDIX: CODE FRAGMENTS

A namelist input file demonstrating the free-form input and usage notes.

```
&input
cg_tol  = 0.0005
cg_max  = 300
he      = 25.0
kslope  = 1.666667
wind    = "u"
grdlats = -23.75, 0.5, 23.75
grdlons = 52.25, 0.5, 179.75
pt1 = 2001, 1,  5,  0
ptN = 2001, 1, 18, 18
/
! The parameters and their default values
!-----------------------------------------------------------------------
! cg_tol   0.0005                  ! conj gradient tolerance
! cg_max   100                     ! max # of iterations
! he       25.0                    ! equivalent depth, in km
! kslope   5.0/3.0                 ... the slash is a killer
! wind     "u"
! grdlats  -23.75, 0.5,  23.75     ! latmin, latincrement, latmax
! grdlons   52.25, 0.5, 179.75    ! latmin, latincrement, latmax
! pt1      []                     ! year/month/day/hour of first prediction
! ptN      []                     ! year/month/day/hour of last  prediction
```

Demonstrating the trivial nature of reading a namelist.

```
open( NMLfid+myid,file="input.namelist")    ! common input parameters
read( NMLfid+myid,NML = input)              ! # of iterations, domain, ...
close(NMLfid+myid)
```

The function `r8inv` is the underlying routine to the generic interface `inv`. Depending on which preprocessor directives are set at compilation, different sets of libraries may be used. This provides a simple portable interface which clarifies the code and reduces errors.

```
FUNCTION r8inv(A) RESULT(ainv)
!--------------------------------------------------------------------------
!  find the inverse of a general matrix
!--------------------------------------------------------------------------

real(kind=real64), DIMENSION(:,:), INTENT(IN)     :: A
real(kind=real64), DIMENSION(SIZE(A,1),SIZE(A,2)) :: ainv

real(kind=real64)                                 :: rcond
integer,           DIMENSION(SIZE(A,2))           :: ipvt
real(kind=real64), DIMENSION(2)                   :: det
real(kind=real64), DIMENSION(SIZE(A,1)*SIZE(A,2)) :: aux
integer                                :: lda, n, iopt, naux, info

lda  = SIZE(A,1)
n    = SIZE(A,2)

if ( n /= lda ) then
      print *,'ERROR( r8inv ) matrix not square'
      print *,'   leading dimension is ',lda
      print *,'   order of matrix   is ',n
      STOP
endif

ainv = A             ! copy ...
naux = 0
iopt = 1
```

```
#ifdef CRAY

      call SGEFA(ainv,lda,n,ipvt,info)
      if (info == 0) then
         call SGEDI(ainv,lda,n,ipvt,det,aux,iopt)
      else
         print *,'ERROR(r8inv:SGEFA): cannot factor matrix ... giving up.'
         STOP
      endif

#elif ESSL

      call DGEICD(ainv,lda,n,iopt,rcond,det,aux,naux)  ! document misleading
      IF ( 1.0_real64 + rcond == 1.0_real64 ) THEN
         print *,'WARNING(r8inv:DGEICD): poorly conditioned matrix ...'
         print *,'condition(rcond) = ',rcond
      END IF

#elif LINPACK

      call DGEFA(ainv,lda,n,ipvt,info)
      if (info == 0) then
         call DGEDI(ainv,lda,n,ipvt,det,aux,iopt)
      else
         print *,'ERROR(r8inv:DGEFA): cannot factor matrix ... giving up.'
         STOP
      endif

#elif LAPACK

      naux = lda*n
      call DGETRF(lda, n, ainv, lda, ipvt, info)
      if (info /= 0) then
         print *,'ERROR(r8inv:DGETRF): cannot factor matrix ... '
         print *,'error code is ',info
         STOP
      else
         call DGETRI(lda,ainv,lda,ipvt,aux,naux,info)
         if (info /= 0) then
            print *,'ERROR(r8inv:DGETRI): cannot invert matrix ... '
            print *,'error code is ',info
            STOP
         endif
      endif

#else

   call NEED_A_GENERAL_R8_MATRIX_INVERT_ROUTINE

#endif
END FUNCTION r8inv
```

ACKNOWLEDGMENTS

Support for this research was provided by the National Center for Atmospheric Research (NCAR) Geophysical Statistics Project, sponsored by the National Science Foundation (NSF) under Grants DMS98-15344 and DMS93-12686. Wikle also acknowledges support from NSF Grant ATM 0222057, and DMS 0139903. Milliff acknowledges support from the NASA Ocean Vector Winds Science Team. Berliner was also supported by NSF Grant DMS-0139897.

[Received June 2003. Revised October 2003.]

REFERENCES

Dey, C. H. (1996), "The WMO Format for the Storage of Weather Product Information and the Exchange of Weather Product Messages in Gridded Binary Form," Office Note NCEP Office Note 388, NOAA National Weather Service, U.S. Department of Commerce.

Freilich, M. H. (1997), "Valication of Vector Magnitude Datasets: Effects of Random Component Errors," *Journal of Atmospheric and Oceanic Technology*, 14, 695–703.

Gelman, A., and Rubin, D. B. (1992), "Inference From Iterative Simulation Using Multiple Sequences," *Statistical Science*, 7, 457–472.

Gill, A. E. (1982), *Atmosphere-Ocean Dynamics*, New York: Academic Press.

Golub, G. H., and Van Loan, C. F. (1996), *Matrix Computations* (3rd ed.), Baltimore, MD: Johns Hopkins University Press.

Liu, W. T., Tang, W., and Polito, P. S. (1998), "NASA Scatterometer Provides Global Ocean Surface Wind Fields with More Structures than Numerical Weather Prediction," *Geophyscial Research Letters*, 25, 761–764.

Madden, R. A., and Julian, P. R. (1971), "Detection of a 40–50 Day Oscillation in the Zonal Wind in the Tropical Pacific," *Journal of Atmospheric Science*, 28, 702–708.

——— (1994), "Observations of the 40–50 Day Tropical Oscillation: A Review," *Monthly Weather Review*, 122, 814–837.

Madden, R. A., Hoar, T. J., and Milliff, R. F. (1999), "Scatterometer Winds Composited According to the Phase of the Tropical Intraseasonal Oscillation," *Tellus*, 51A, 263–272.

Matsuno, T. (1966), "Quasi-Geostrophic Motions in the Equatorial Area," *Journal of the Meteorological Society of Japan*, 41, 25–42.

Wheeler, M., and Kiladis, G. N. (1999), "Convectively-Coupled Equatorial Waves: Analysis of Clouds and Temperature in the Wavenumber-Frequencey Domain," *Journal of the Atmospheric Sciences*, 56, 374–399.

Wikle, C. K., Milliff, R. F., and Large, W. G. (1999), "Surface Wind Variability on Spatial Scales from 1 to 1,000 km Observed During TOGA COARE," *Journal of the Atmospheric Sciences*, 56, 2222–2231.

Wikle, C. K., Milliff, R. F., Nychka, D., and Berliner, L. M. (2001), "Spatiotemporal Hierarchical Bayesian Modeling: Tropical Ocean Surface Winds," *Journal of the American Statistical Society*, 96, 382–397.