Accelerating CMIP Data Analysis with Parallel Computing in R

Daniel Milroy
Sophia Chen
Brian Vanderwende
Dorit Hammerling

NCAR Technical Notes
NCAR/TN-534+CODE

National Center for Atmospheric Research
P. O. Box 3000
Boulder, Colorado
80307-3000
www.ucar.edu

NCAR TECHNICAL NOTES

http://library.ucar.edu/research/publish-technote

The Technical Notes series provides an outlet for a variety of NCAR Manuscripts that contribute in specialized ways to the body of scientific knowledge but that are not yet at a point of a formal journal, monograph or book publication. Reports in this series are issued by the NCAR scientific divisions, serviced by OpenSky and operated through the NCAR Library. Designation symbols for the series include:

EDD – Engineering, Design, or Development Reports
Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

IA – Instructional Aids
Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

PPR – Program Progress Reports
Field program reports, interim and working reports, survey reports, and plans for experiments.

PROC – Proceedings
Documentation or symposia, colloquia, conferences, workshops, and lectures. (Distribution maybe limited to attendees).

STR – Scientific and Technical Reports
Data compilations, theoretical and numerical investigations, and experimental results.

The National Center for Atmospheric Research (NCAR) is operated by the nonprofit University Corporation for Atmospheric Research (UCAR) under the sponsorship of the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

National Center for Atmospheric Research
P. O. Box 3000
Boulder, Colorado 80307-3000

NCAR/TN-534+STR
NCAR Technical Note

2017-06

Accelerating CMIP Data Analysis with Parallel Computing in R

Daniel Milroy
Department of Computer Science,
University of Colorado, Boulder, CO
Sophia Chen
Fairview High School, Boulder, CO
Brian Vanderwende
Operations and Services Division,
National Center for Atmospheric Research, Boulder, CO
Dorit Hammerling
Institute for Mathematics Applied to Geosciences,
National Center for Atmospheric Research, Boulder, CO

Computational and Information Systems Laboratory (CISL)
Institute for Mathematics Applied to Geosciences (IMAGe)

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
P. O. Box 3000
BOULDER, COLORADO 80307-3000
ISSN Print Edition 2153-2397
ISSN Electronic Edition 2153-2400

Accelerating CMIP Data Analysis with Parallel Computing in R

Daniel Milroy [*1], Sophia Chen[2], Brian Vanderwende [†3], and Dorit Hammerling [‡3]

[1]University of Colorado, Boulder
[2]Fairview High School
[3]National Center for Atmospheric Research

June 28, 2017

Abstract

In this Technical Note we examine eight schemes for parallelizing Extreme Value Analysis (EVA) on Coupled Model Intercomparison Project data via `R` `foreach`, `doParallel`, and `doMPI` packages. We perform strong scaling studies to delineate the performance impacts of factors such as `R` cluster type (TCP/IP sockets and MPI), communication protocol (Ethernet, IP over InfiniBand, and MPI), loop parallelization (outer or inner loop), and approaches to reading data from the NCAR GLADE parallel filesystem. We elucidate peculiarities of `R` memory management and overhead associated with interprocess communication and discuss broadcast limitations of `Rmpi`. The best performing scheme parallelizes the outer EVA loop across latitude and reads only the subset of the data operated on in the inner loop over longitude; the different cluster types and communication protocols all perform about equally for this scheme. This configuration represents a parallel speedup of 50 with 96 `R` workers, and is scalable for EVA on larger problem sizes than those presented here.

Keywords: Extreme Value Analysis, CMIP, `R`, `foreach`, `doParallel`, `doMPI`, strong scaling, parallelization, internode communication, interprocess communication, InfiniBand, TCP/IP, RDMA, IPoIB, communication overhead, memory overhead, Yellowstone supercomputer, GLADE

[*]daniel.milroy@colorado.edu
[†]vanderwb@ucar.edu
[‡]dorith@ucar.edu

Contents

1 Introduction

The Coupled Model Intercomparison Project (CMIP) began as a comparison of a small number of coupled climate models, but has evolved into a large organization of 21 individual Model Intercomparison Projects (MIPs). CMIP now forms an integral part of international evaluations of climate change, such as the Intergovernmental Panel on Climate Change (IPCC) [1]. In particular, the IPCC Fifth Assessment Report Evaluation of Climate Models "draws heavily on model results collected as part of the Coupled Model Intercomparison Projects as these constitute a set of coordinated and thus consistent and increasingly well-documented climate model experiments" [2]. Beyond the IPCC, CMIP has contributed tremendously to advance the field of climate science. The goal of CMIP is "to better understand past, present, and future climate change arising from natural, unforced variability or in response to changes in radiative forcings in a multi-model context" [1]. To that end, CMIP makes model outputs publicly available and standardizes their formats to facilitate community analysis [1]. This charge itself is formidable: the CMIP Phase 5 (CMIP5) archive contains 1.8 PB (petabytes) of data, and the current CMIP6 is expected to produce up to 40 PB.

The CMIP Analysis Platform hosted at NCAR provides researchers with access to CMIP data on the GLobally Accessible Data Environment (GLADE), thus allowing users to employ NCAR's Yellowstone, Geyser, and Caldera high performance computing systems for analysis of CMIP data. This eliminates the need for researchers to transfer large datasets to local storage and perform analysis on personal machines.

Effective parallelization is crucial for many types of analysis on CMIP data due to the large number of grid cells and data points in question. If performed efficiently, analyzing datasets in parallel can substantially reduce computation time. The goal of this TechNote is to determine the optimal parallelization scheme for conducting grid point-wise statistical analysis in a parallel computing environment– in this case the Yellowstone supercomputer. We adopt a typical strong scaling testing approach, where the problem size remains constant and the number of processors is varied. If the application exhibits perfect strong scaling, doubling the number of processors will halve the runtime. Extreme Value Analysis (EVA) is the statistical analysis performed on the data set, but the strong scaling investigation is suitable for any parallel application. This study is based on the statistical computing language `R` and employs `R`'s `foreach` package to parallelize data analysis. `R` is designed for statistics and graphics, and provides a large variety of packages for myriad statistical analyses. `R` is also free, open-source, and regularly maintained, making it widely available and accessible.

An ancillary objective of this TechNote is to enhance the reader's appreciation for the important roles both the computing hardware and software, and the code itself play in effective parallelization. Indeed, understanding of both is necessary for performance optimization. Ideally, a researcher should be aware of the networks available on a particular cluster to select an appropriate internode communication protocol. Memory limitations

and CPU properties are also important details that have implications for parallelization. Filesystem I/O (the manner of reading from and writing to files) is perhaps the most frequently overlooked factor when considering parallelization and optimization; adroit access to storage can facilitate code scalability. We examine the effects of all these components on runtime in this TechNote.

2 Dataset and Analysis

Data used in this TechNote (`pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc`) consist of daily precipitation climate model data spanning 35 years of 365 day years (ignoring leap years), from January 1, 1955 through December 31, 1989. Each data point is associated with a specific latitude and longitude coordinate, forming a grid of 288 longitude and 192 latitude coordinates, resulting in 55,296 coordinate grid cells. Each data point is recorded at the specific latitude and longitude coordinate every day for 35 years, resulting in 12,775 time coordinates. Thus, this dataset is a 288 by 192 by 12,775 array of time series data, for a total of 706,406,400 data points. The dataset occupies approximately 2.7 GiB of filesystem space. The dataset and code from this work can be found at `https://doi.org/10.5065/D6JW8CK2` [3]. Note that throughout this TechNote units of gigabytes (GB, or 10^9 bytes), gibibytes (GiB, or 2^{30} bytes), and gigabits (Gb, or 10^9 bits) are used according to system reporting.

A schematic representation of the 3D structure of the dataset is shown in Figure 1. The other datasets in the CMIP library have similar structures containing a grid of values with climate data through different time periods. Figure 2 represents the global precipitation flux on a single day of the dataset examined in this TechNote, and Figure 3 represents the precipitation flux time series for the grid cell containing Boulder, CO.

CMIP data is stored in netCDF format. NetCDF (network Common Data Form) "is a set of interfaces for array-oriented data access and a freely distributed collection of data access libraries for C, Fortran, C++, Java, and other languages" [4]. Individual files are self-describing (contain metadata), portable, scalable, appendable, shareable, and archivable [4].

Extreme Value Analysis (EVA) models extreme values from observations using specialized statistical distributions. In this TechNote we explore the parallelization of the Peaks Over Threshold approach, which involves fitting the time series data grid cell-wise to a Generalized Pareto Distribution. Since each grid cell's time series is fit independently, interprocess communication is not necessary for this stage. In this case, parallelization simply involves choosing an efficient way to distribute grid cells across a group of processors. The EVA parallelization scheme developed is applicable to all CMIP files, which have a similar netCDF structure as the one used in this TechNote.

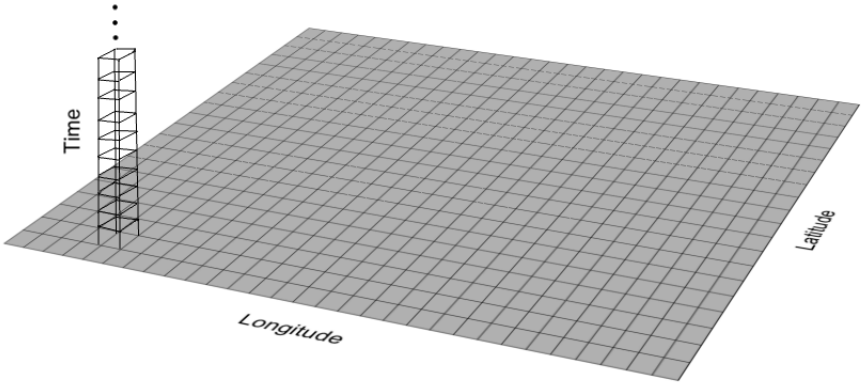

Figure 1: Visualization of a subset of the 3D dataset

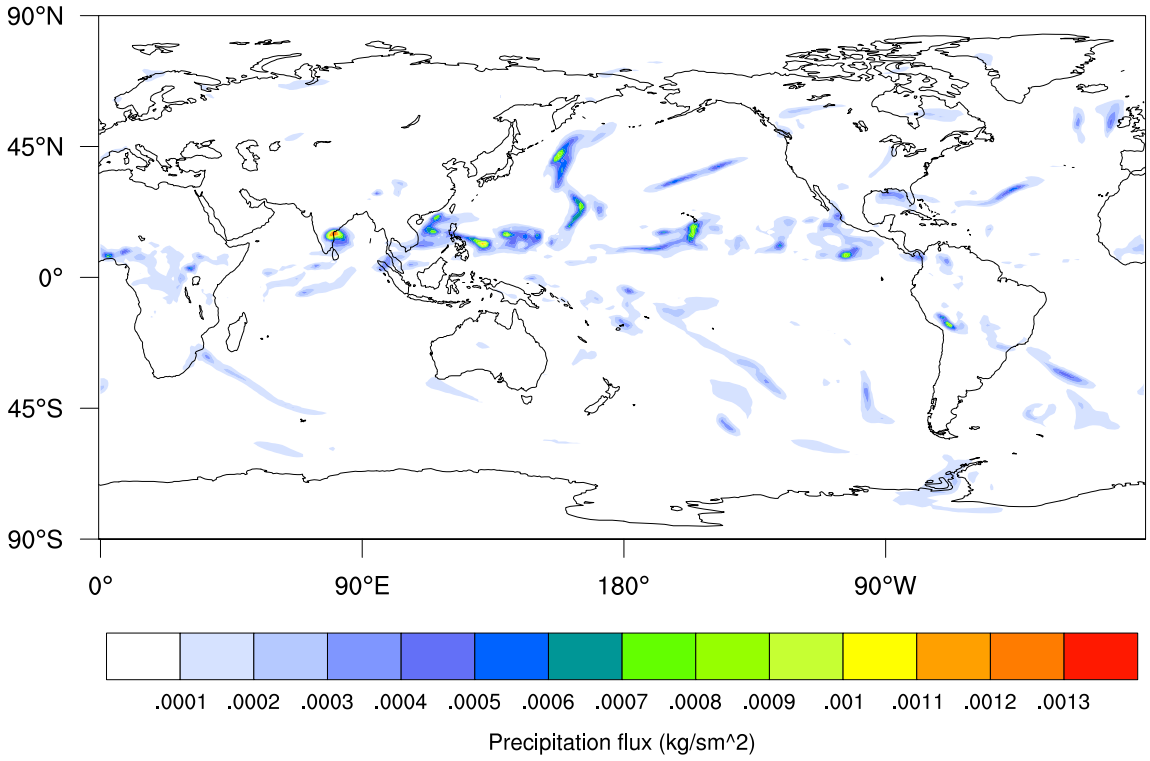

Figure 2: Global precipitation flux ($\frac{kg}{m^2 s}$) on October 3, 1983

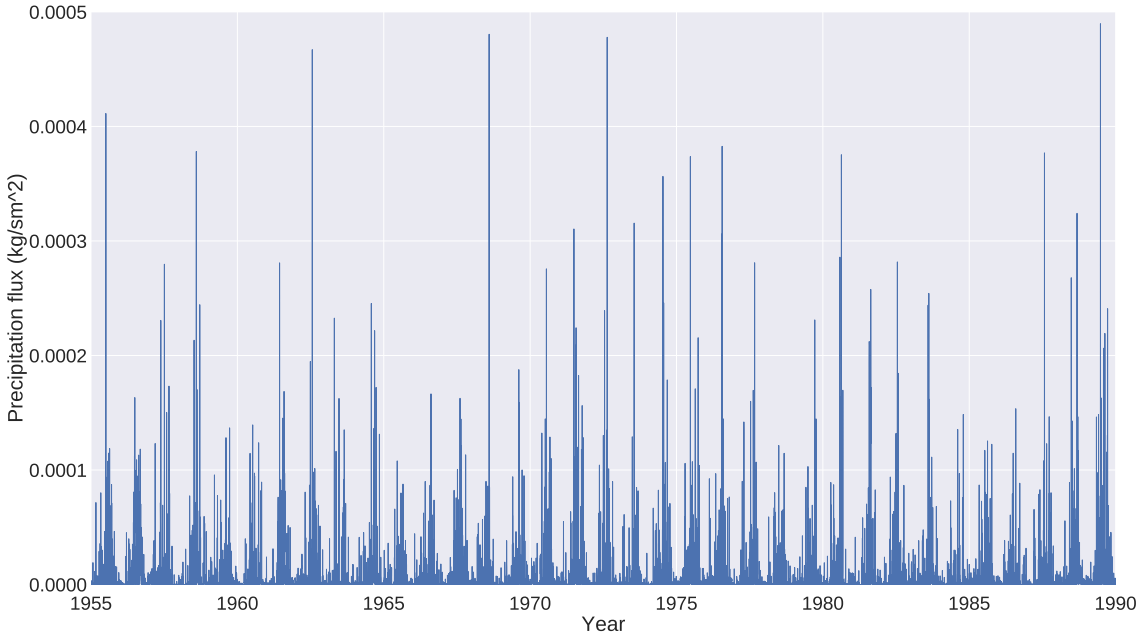

Figure 3: Precipitation flux ($\frac{kg}{m^2 s}$) from Jan 1, 1955 through December 31, 1989 for 40.05° N and 105.0° W (grid cell containing Boulder, CO).

3 Parallel Processing Environment

3.1 Parallel `R`

`R` contains several packages with parallel computing capabilities which facilitate inter-node communication and the concomitant efficient processing of large datasets. The main parallel `R` package investigated in this TechNote is parallel `foreach`. The `R` `foreach` package allows the user to parallelize a `for` loop with the `dopar` command, which is placed after the first line in the `for` loop. `Foreach` requires an initialized `R` cluster, which is a set of `R` processes that run in parallel and communicate using different protocols and networks. It is loaded via the `doParallel` package, which also includes `registerDoParallel` for registering the parallel cluster backend. `R` clusters are broadly divided into two different protocols: TCP/IP sockets and Message Passing Interface (MPI). `R` MPI clusters depend on `R`'s `doMPI` package, which in turn requires the `Rmpi` package. The `Rmpi` package is an `R` interface to the lower-level MPI library.

3.2 `R` Built on Yellowstone

To use the `Rmpi` package on Yellowstone, `R` (v3.3.2) was installed with parallelism and performance in mind. The latest versions of bzip2 (v1.0.6), libzma (xz-v5.2.3), and PCRE

(v8.40) provide compression and regular expression functionality, while matrix math operations are optimized by using the Intel compilers (v16.0.3) with the threaded Intel Math Kernel Library (v11.3.3). The `Rmpi` package itself (v0.6-6) is built using Open MPI (v1.8.8), as that MPI library increased `Rmpi` stability relative to other tested MPI options (IBM POE, Intel MPI, and newer Open MPI versions). Open MPI is configured with the --disable-dlopen option, which prevents dynamic loading of plugins, a feature that is incompatible with `Rmpi`.

3.3 Yellowstone Supercomputer

The NCAR Yellowstone machine is an IBM iDataPlex High Performance Computing (HPC) cluster. It consists of 4,536 compute nodes each with two eight-core Intel Sandy Bridge CPUs and 32 GB memory (25 GB usable). This translates to 72,576 total cores and 144.58 TB total system memory. The high performance system interconnect is Mellanox InfiniBand, which provides 13.6 GBps of bidirectional bandwidth per node. Each compute node is also connected to a 1Gb Ethernet network. Yellowstone is connected to the GLADE high performance parallel storage system, which provides the cluster with more than 90GBps bandwidth to disk and tens of petabytes of storage space. Note that we report the units per the Yellowstone documentation on the NCAR Computational and Information Systems Lab (CISL) website: `https://www2.cisl.ucar.edu/resources/computational-systems/yellowstone`.

4 Experimental Design

The design of the `foreach` experiments is shaped by our goal of minimizing EVA runtime on CMIP datasets. This analysis is spatially independent and involves looping over latitude and longitude values in the dataset. Since either coordinate can be iterated upon in the inner or outer loop, we assess both options. While it is generally advantageous to parallelize an outer loop due to parallelization overhead, we also test parallelizing the inner loop. We explore two additional factors that impact runtime and memory usage: the method of reading data from storage and the communication between the parallel workers. We acknowledge that a job "load-balancing" parallelization approach can also be taken; in this scheme the independent tasks are divided into separate jobs and the execution order is determined by the cluster job scheduler. While this method is effective under limited circumstances, coordinating job submission and analyzing the output of many jobs (perhaps thousands) can be a formidable undertaking in its own right. Moreover, this tactic presents greater exposure to cluster resource contention, thus predicting when all jobs will complete can be very difficult. Ultimately, this tactic requires a greater total number of I/O calls as each job must read the relevant subset of the data and write its result subset to disk.

4.1 Description of Experiments

We first choose a dimension to loop through in parallel– either latitude or longitude. For example, if the experiment parallelizes across longitude in the outer loop, the sequential inner loop will iterate over latitude coordinates, while the outer longitude loop will be parallelized such that each `R` process receives a subset of the longitude values. Eight parallelization schemes are examined, with the major differences being parallelization across latitude or longitude, inner or outer loop parallelization, and the method of reading the data. A summary of the parallelization schemes is presented in Table 1. The following sections describe the factors that are varied in the eight parallelization schemes.

4.1.1 Parallelization Across Longitude or Latitude

The CMIP dataset used here contains more longitude values than latitude values, so comparing parallelization schemes across latitude or longitude can illustrate the performance impact of communication overhead. Experiments 1, 2, 7, and 8 parallelize across longitude values, implying that the latitude coordinates were looped through sequentially. Experiments 3, 4, 5, and 6 parallelize across latitude values, while the longitude coordinates are looped through sequentially. Figure 4 visualizes iteration across latitude and Figure 5 across longitude. See Appendix B for code examples.

4.1.2 Placement of Parallel Loops

If the inner loop is parallelized, the outer loop sequentially iterates through one dimension, while within each iteration the inner loop is distributed to the `R` workers. However, if the outer loop is parallelized, the inner loop sequentially iterates over the second dimension. Comparing these two approaches can reveal useful properties of `R` foreach communication overhead.

4.1.3 Reading Data

The method of reading data is also explored– retrieving only the relevant data set for each iteration or reading the entire dataset into memory at once. The first scheme involves each worker extracting subsets of the data according to the latitude and longitude coordinates of the sequential loop. In this case, each worker executes a `getData` function which takes a latitude or longitude argument (depending on whether the experiment's inner loop is across latitude or longitude), the value of which is received from the master. Therefore the workers engage in independent reads of different slices of the dataset before beginning the inner loop. See Appendix B.3.2 for an example in `R`. In the second scheme the master reads the entire dataset before starting the EVA. Depending on whether the inner or outer loop is parallelized, the master will either send a slice or the entire dataset to each worker. See Appendix B.1.2 for an example in `R`. Under specific circumstances this approach can

Experiment No.	Parallel loop		Sequential loop		
	Inner or outer	**Across**	**Inner or outer**	**Across**	**Data read**
1	inner	288 longitude	outer	48 latitude	only read data used in each iteration
2	inner	288 longitude	outer	48 latitude	read all at once at the beginning
3	inner	192 latitude	outer	72 longitude	only read data used in each iteration
4	inner	192 latitude	outer	72 longitude	read all at once at the beginning
5	outer	48 latitude	inner	288 longitude	only read data used in each iteration
6	outer	48 latitude	inner	288 longitude	read all at once at the beginning
7	outer	72 longitude	inner	192 latitude	only read data used in each iteration
8	outer	72 longitude	inner	192 latitude	read all at once at the beginning

Table 1: Summary of the experiments

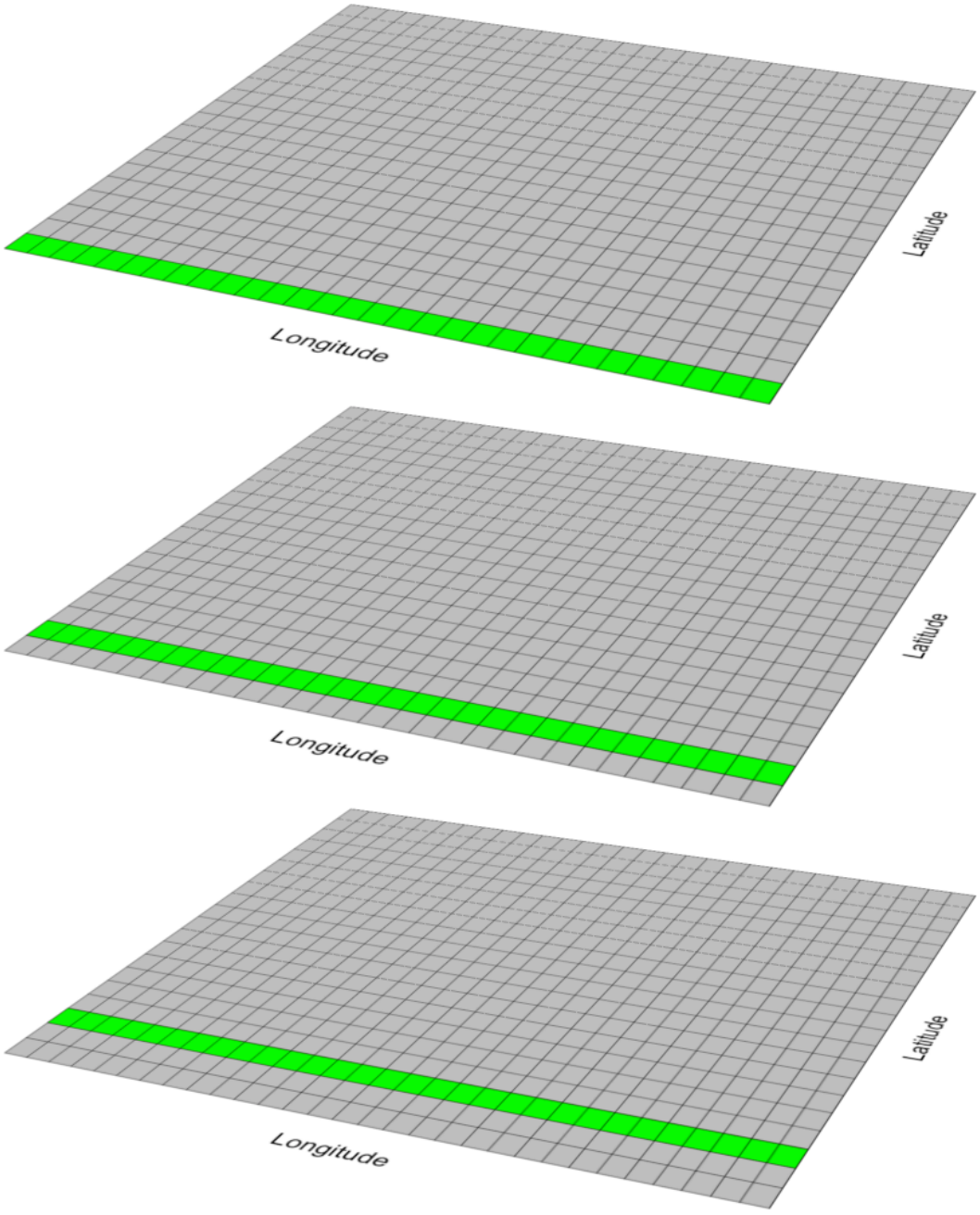

Figure 4: This example represents a parallelized outer loop across latitude, where the inner loop across longitude is executed in serial by each `R` process. The green grid cells highlight the subsets that the first three `R` processes iterate across. In this case the first `R` process performs EVA on the first latitude grid (top), and the third `R` process on the bottom grid.

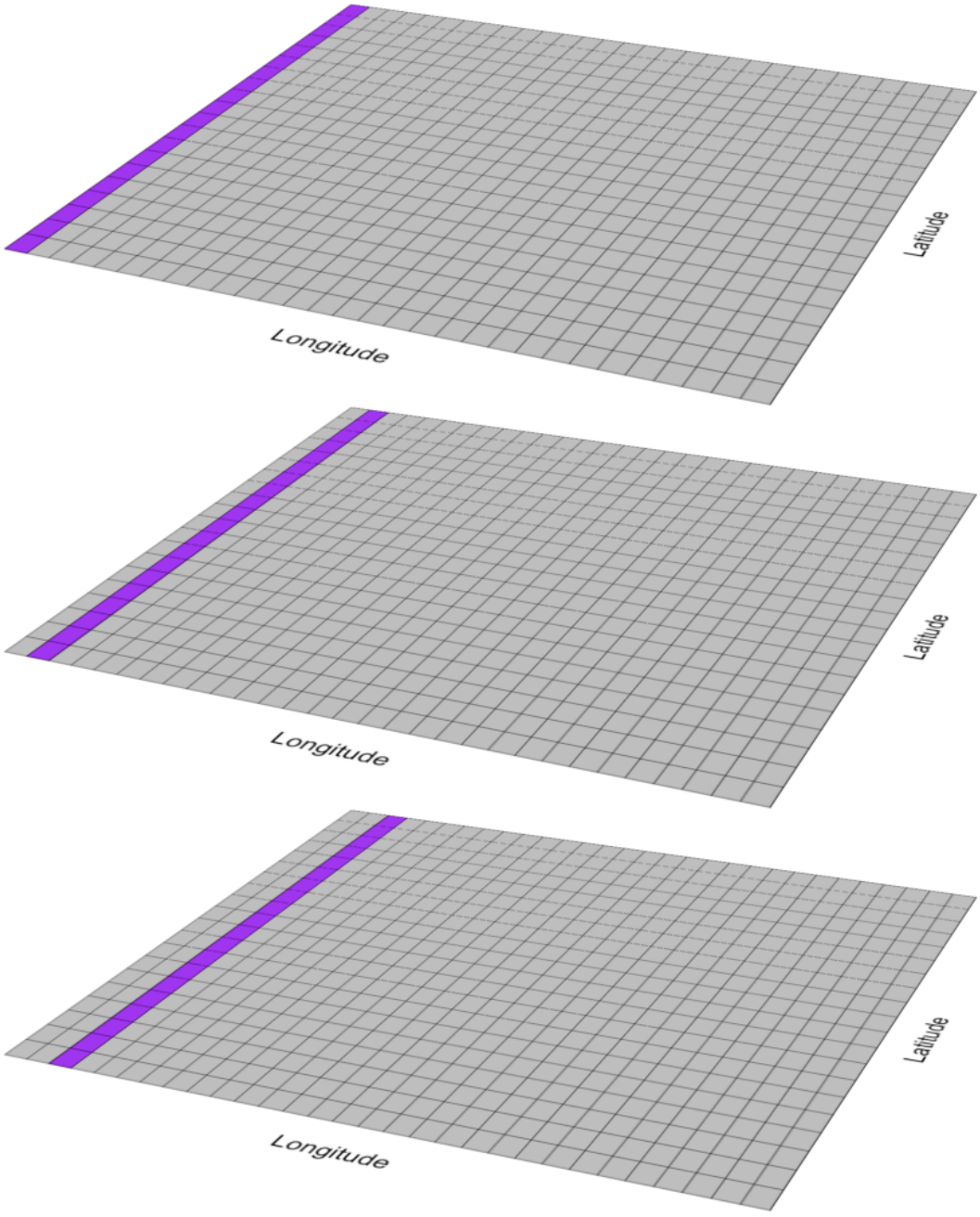

Figure 5: This example represents a parallelized outer loop across longitude, where the inner loop across latitude is executed in serial by each `R` process. The purple grid cells highlight the subsets that the first three `R` processes iterate across. In this case the first `R` process performs EVA on the first longitude grid (top) and the third `R` process on the bottom grid.

be fast at the expense of requiring more memory per `R` process. See Tables 7, 9, 11, and 13 for details.

Ultimately, the most efficient method of reading netCDF data is for each worker to read only its slice of the data once, before the outer loop. To accomplish this, each worker needs to access its subset of latitude and longitude values before entering the `dopar` construct, which is not possible in `foreach`. Pure `Rmpi` must be used to access this functionality, which requires considerably more expertise and development time for effective parallelization of EVA.

4.1.4 Communication Protocol

We examined the use of two different clusters: PSOCKcluster, known as sockets here, and MPI. Ethernet and InfiniBand physical networks were also tested. Therefore each experiment was run with sockets clusters via the Ethernet network (Eth), sockets clusters over the InfiniBand network (Internet Protocol over InfiniBand: IPoIB), and MPI clusters over the InfiniBand network. MPI over Ethernet is slow in comparison to MPI over InfiniBand due to InfiniBand's support of Remote Direct Memory Access (RDMA). In order to approach InfiniBand speeds using MPI over Ethernet, specialized hardware and a protocol known as RDMA over Converged Ethernet (RoCE) are necessary. This setup was not available, and we did not test this combination.

Testing different protocols and clusters was done to provide a reference for researchers with access to varied parallel computing infrastructures and software packages. Ethernet is ubiquitous in parallel computing, but InfiniBand is restricted to High Performance Computing clusters. `Rmpi` is not commonly installed on HPC clusters; if a researcher has access to a cluster with InfiniBand it is quite possible that the `Rmpi` package will not be available, hence the investigation into IPoIB.

4.2 Computing Resource Requirements

For the strong scaling study two resource requirements are listed in each experimental test– the number of `R` workers and the number of nodes. For every experiment except 8, the number of `R` workers is set equal to the number of cores per node so each core will receive one `R` process. In experiment 8, parallelization of the outer loop combined with the master process' initial read of the full dataset into memory results in each worker process receiving the entire dataset. For ptile=16 this requires more than the approximately 25GB of usable physical memory on a Yellowstone compute node. In this case we set the LSF ptile option to 8, which allows `R` to spawn a maximum of 8 processes on each node for the MPI cluster. For sockets clusters the LSF ptile option controls the number of `R` processes per node via the $LSB_HOSTS environment variable. This variable contains a list of hostnames with a hostname multiplicity equal to the ptile value. See Appendix B for details of how this environment variable is used to start a sockets cluster with the correct number of

`R` processes per node. Note that requesting 16 CPUs with a ptile of 8 instructs LSF to allocate two nodes for the job. In the other seven experiments, the number of tasks per node is kept constant at 16, while the number of nodes increases sequentially from 1 to 3. Therefore the number of cores increases from 16 to 48 in multiples of 16.

4.3 Test Data Size and Timing Variability Assessment

To more rapidly identify the best performing schemes, each experiment is run on one quarter of the data. If one loop uses all 288 longitude values, the other loop iterates over only 48 latitudes (13,824 grid cells). Analogously, if one loop iterates across 72 longitudes, the other loop will use 192 latitudes (13,824 grid cells). Hence the number of grid cells is fixed across experiments and only the fastest configurations are then evaluated on the full dataset.

We also perform an analysis of variability for the best performing experiments. Each of these experiments is run 10 times for each protocol and node count, and the means, standard deviations, and ranges are computed. It is important to recognize that the runtime variability is strongly influenced by resource contention. Yellowstone, like most HPC clusters, has tens or hundreds of concurrent user jobs running at any time. These jobs have many varied workloads that require access to shared infrastructure. For example, nodes from two different jobs may make concurrent resource requests from one of the storage servers, which it satisfies more slowly than if only one request was made. Furthermore, unless a network is topologically fully-connected it will have shared links that transmit data from multiple sets of nodes to one or many destinations. These links can become congested, affecting performance.

5 Timing Results

We present experimental results for the quarter-size dataset in Appendix Tables 6-13 and for the full dataset in Tables 2-5. Only the loops containing the EVA are timed; cluster startup and variable assignment are not included in the runtime. The two best performing experiments (numbers 5 and 7) are considered in more detail via runtime variability studies and experimental runs using the full dataset. Experiments 6 and 8 exhibit excellent runtimes on the quarter-size dataset but scale poorly. Results for experiments 6 and 8 for worker counts from 64-96 on the full dataset are excluded due to excessive memory and node resource requirements. See Section 5.2 for details. Experiments 2, 4, 6, and 8 make salient the effect of the InfiniBand network's much greater bandwidth: the runtimes increase much more dramatically for Ethernet in comparison to IPoIB as the number of workers increases. This is caused by the master sending the dataset to each worker, which exposes the physical network bottleneck. Note that the MPI cluster and protocol performs poorly for experiments 1-4, which may be due to an improper implementation of the MPI library in `Rmpi`.

5.1 Variability Results

Experiments 5 and 7 both perform best, so we selected them for further analysis. Both experiments conform to the general rule that parallelizing outer loops is preferable to inner loop parallelization for minimizing communication overhead. In Tables 10 and 12, variability characteristics are also included. For both of these tables, each experiment was run 10 times to compute mean runtime, standard deviation, and runtime range. It is important to note that filesystem caching effects in Linux buffer cache can contribute to decreased run times beginning with the second execution. This can occur if each compute node reads the same subset of the data for each execution. It is assumed that the caching effects are similar between experiments 5 and 7 on the quarter dataset. Moreover, as mentioned in Section 4.3, resource contention can have a tremendous impact on job runtime, especially when the jobs are run sequentially as was done here. Some of the runtime spread is due to resource contention from other users' jobs. For a better representation of job runtime variability jobs should run at random times over the course of several days.

The variability characteristics are similar between the two experiments, and due to their higher number of filesystem reads (since both experiments perform a read before each inner loop), differences between them are attributed to transient filesystem performance decreases. Experiment 5 performs slightly better than 7, which we attribute to the lower frequency of parallel communication necessary for distribution of tasks. In other words, the inner loop in Experiment 5 contains more work, which reduces the number of tasks sent to each `R` worker process and lowers the frequency of communication with the `R` master process. For purposes of comparison, we also present the variability results for experiment 5 with a single process in Table 10.

5.2 Memory Footprint and Scalability

While experiments 6 and 8 with the MPI cluster achieve the shortest runtimes of all experiments on the quarter-size dataset, they do so with an inherently flawed approach to parallelization. The data distribution approach of experiments 6 and 8 results in each `R` process sequentially receiving the entire dataset. Running the experiments with ptile=8 on the full dataset fails due to over-allocation of memory. Despite the full dataset occupying about 2.7 GiB on the filesystem, it occupies closer to 5.3 GiB of memory per `R` process. Upon reading the file, the dataset values are cast from single to double precision, nearly doubling the memory consumption. Furthermore, `R` allocates a surplus of memory when the master process reads the dataset from the filesystem. The memory occupancy for the dataset balloons to approximately 10 GiB, and surpasses 14 GiB on the master process when sending the dataset to the worker processes. While this can be mitigated momentarily by forcing `R` to perform garbage collection (deallocating unused memory areas) via `gc()`, the memory footprint rapidly increases as memory addresses are mapped but unused (or disused).

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	1	514.6
Eth	32	2	263.9
Eth	48	3	186.8
Eth	64	4	139.3
Eth	80	5	127.4
Eth	96	6	102.5
IPoIB	16	1	519.4
IPoIB	32	2	271.4
IPoIB	48	3	194.1
IPoIB	64	4	153.6
IPoIB	80	5	135.4
IPoIB	96	6	106.0
MPI	16	1	617.3
MPI	32	2	320.8
MPI	48	3	225.3
MPI	64	4	174.8
MPI	80	5	152.1
MPI	96	6	135.8

Table 2: **Experiment 5, full dataset** - outer loop parallelization across latitude for 192 latitude values, only reading data used in each iteration.

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	8	972.2
Eth	32	16	1604.0
Eth	48	24	3182.8
IPoIB	16	8	552.8
IPoIB	32	16	691.0
IPoIB	48	24	927.7

Table 3: **Experiment 6, full dataset** - outer loop parallelization across longitude for 192 longitude values and ptile=2.

Protocol	R workers	Nodes	Runtime (seconds)
Eth	16	1	612.7
Eth	32	2	318.4
Eth	48	3	218.4
Eth	64	4	175.0
Eth	80	5	145.2
Eth	96	6	115.0
IPoIB	16	1	608.2
IPoIB	32	2	323.6
IPoIB	48	3	233.9
IPoIB	64	4	178.2
IPoIB	80	5	146.2
IPoIB	96	6	125.6
MPI	16	1	605.4
MPI	32	2	315.4
MPI	48	3	207.0
MPI	64	4	167.2
MPI	80	5	139.8
MPI	96	6	124.7

Table 4: **Experiment 7, full dataset** - outer loop parallelization across longitude for 288 longitude values, only reading data used in each iteration.

Protocol	R workers	Nodes	Runtime (seconds)
Eth	16	8	981.3
Eth	32	16	1651.9
Eth	48	24	2337.4
IPoIB	16	8	569.8
IPoIB	32	16	712.0
IPoIB	48	24	948.0

Table 5: **Experiment 8, full dataset** - outer loop parallelization across longitude for 288 longitude values and ptile=2.

On the quarter-size dataset, experiments 6 and 8 MPI far surpass Eth and IPoIB. This is due to MPI's optimized tree broadcast algorithm that runs in near constant time [5]. In contrast, TCP/IP broadcast (Eth and IPoIB protocols) uses a simple one-to-all communication pattern that shows its limitations as the number of workers increases. Network saturation occurs rapidly for Eth and even IPoIB.

To run experiments 6 and 8 effectively on the full dataset, a ptile of 2 must be chosen, as the master process overhead drives the memory utilization beyond 25 GB with ptile=3. The MPI cluster fails to complete the EVA on the full dataset with ptile=2. It produces the following error:

```
Error in mpi.bcast(data, 4, 0, cl$comm) :
  long vectors not supported yet: memory.c:3441
```

`R` long vectors contain more than $2^{31} = 2,147,483,648$ elements; the entire dataset consists of approximately 700,000,000, so this may be an indication of a bug in `Rmpi`. These experiments neatly illustrate the importance of scalability in parallel processing, as they cannot be run with an input dataset of greater than approximately 6 GiB (even with ptile=1) on a Yellowstone compute node.

6 Discussion and Conclusions

Experiment 5 abides by conventional recommendations for parallelization: the outer loop is parallelized and the amount of work per inner loop iteration is maximal, which decreases communication frequency. Furthermore, it does not engage in redundant communication (sending either a slice or the entire dataset to every worker) since each worker only reads its subset of the data. Our tests indicate that Experiment 5 runs well on any available protocol and physical network, but IPoIB with sockets cluster is our recommended configuration due to the greater bandwidth available from an InfiniBand network. Experiment 5 for the full dataset achieves a speedup of approximately 50 when run with 96 tasks on six nodes. See Figure 6 for a graphical representation of strong scaling for Ethernet, IPoIB, and MPI protocols in comparison with perfect scaling and the projected runtime for a serial `R` script on the full dataset (based on the serial runtime on the quarter dataset: $5419.2s = 1354.8s \times 4$). Note that EVA on a single grid cell for the full temporal range takes $4.2s$ (average of five runs).

It is notable that the runtimes for experiments 5 and 7 are very similar across protocols and physical networks, with sockets clusters performing slightly better overall. This is somewhat surprising as sockets clusters rely on encrypted communication between the master and remote worker processes, and `Rmpi` is using Remote Direct Memory Access for low latency. It is likely that the higher ratio of computation time to communication time present in the outer loop parallelizations compensates for the higher bandwidth and lower latency MPI protocol over the InfiniBand network. It is also possible that the small

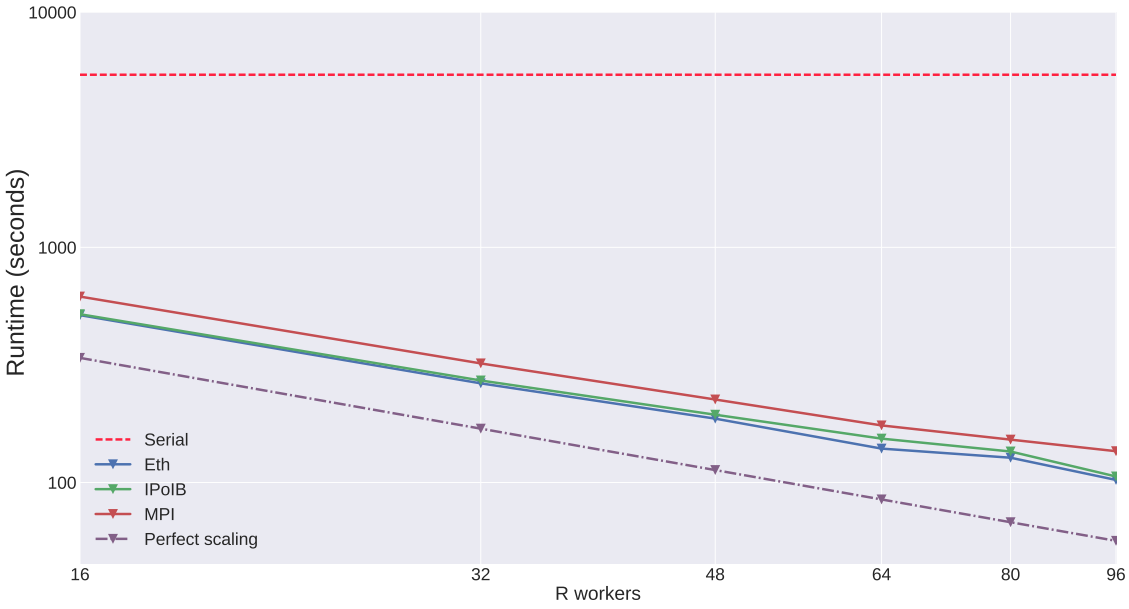

Figure 6: Experiment 5 scaling for the full dataset. Runtimes for the listed protocols are compared to perfect scaling, and the projected serial runtime on the full dataset is extrapolated from the runtime of the quarter-size dataset. Note the logarithmic scale of both axes.

performance penalty for MPI clusters in these cases is due to an inefficient implementation of MPI in the underlying `Rmpi`. In either case, the results suggest that any protocol can be used effectively; this indicates that experiments 5 and 7 are well suited to nearly any parallel computing environment. If a researcher has a choice of physical networks and protocols, IPoIB should be used due to InfiniBand's large bandwidth advantage over 1 and 10 Gb Ethernet. While we see no evidence of this phenomenon in our testing, Ethernet networks' lower bandwidth makes them more prone to saturation than InfiniBand, which could dramatically increase runtime. This can occur if data transfers (e.g. from an external storage system accessed across Ethernet) to compute nodes are using the same upstream Ethernet links.

Another reason experiments 5 and 7 scale so successfully is Yellowstone's high performance parallel filesystem. GLADE is designed for low latency concurrent file access, which occurs after the completion of each inner loop. Running these experiments on filesystems that are not designed for this access pattern will likely result in less optimal scaling than the results reported here.

Appendix A Quarter Dataset Test Results

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	1	373.9
Eth	32	2	541.5
Eth	48	3	682.8
IPoIB	16	1	374.5
IPoIB	32	2	421.9
IPoIB	48	3	480.5
MPI	16	1	1968.6
MPI	32	2	2046.1
MPI	48	3	2107.4

Table 6: **Experiment 1, quarter dataset** - inner loop parallelization across longitude for 48 latitude values, only reading data used in each iteration.

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	1	202.2
Eth	32	2	351.7
Eth	48	3	539.0
IPoIB	16	1	204.5
IPoIB	32	2	248.2
IPoIB	48	3	291.3
MPI	16	1	1777.1
MPI	32	2	1833.0
MPI	48	3	1906.9

Table 7: **Experiment 2, quarter dataset** - inner loop parallelization across longitude for 48 latitude values, all data read at once in the beginning.

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	1	293.3
Eth	32	2	397.8
Eth	48	3	502.1
IPoIB	16	1	298.3
IPoIB	32	2	308.4
IPoIB	48	3	351.2
MPI	16	1	1437.3
MPI	32	2	1494.0
MPI	48	3	1547.5

Table 8: **Experiment 3, quarter dataset** - inner loop parallelization across latitude for 72 longitude values, only reading data used in each iteration.

Protocol	`R` workers	Nodes	Runtime (seconds)
Eth	16	1	141.4
Eth	32	2	237.0
Eth	48	3	370.6
IPoIB	16	1	136.4
IPoIB	32	2	156.6
IPoIB	48	3	193.1
MPI	16	1	1225.4
MPI	32	2	1265.2
MPI	48	3	1361.0

Table 9: **Experiment 4, quarter dataset** - inner loop parallelization across latitude for 72 longitude values, all data read at once in the beginning.

Protocol	R workers	Nodes	Mean runtime (seconds)	Runtime stddev	Runtime min, max
Serial	1	1	1354.8	37.0	1290.9, 1382.1
Eth	16	1	142.0	3.0	136.4, 144.8
Eth	32	2	89.3	2.5	85.4, 93.7
Eth	48	3	56.6	1.7	53.9, 59.1
IPoIB	16	1	140.0	2.1	137.4, 143.1
IPoIB	32	2	90.2	2.9	86.9, 95.2
IPoIB	48	3	55.2	1.6	51.5, 56.7
MPI	16	1	148.9	3.5	144.5, 156.3
MPI	32	2	81.0	1.5	78.4, 82.8
MPI	48	3	71.0	0.7	69.9, 72.3

Table 10: **Experiment 5, quarter dataset** - outer loop parallelization across latitude for 48 latitude values, only reading data used in each iteration. The mean, standard deviation, and range of runtimes are reported for 10 successive runs of each protocol and number of tasks. Results for the serial version for 5 runs are also reported for comparison.

Protocol	R workers	Nodes	Runtime (seconds)
Eth	16	2	219.6
Eth	32	4	377.7
Eth	48	6	540.8
IPoIB	16	2	147.0
IPoIB	32	4	195.1
IPoIB	48	6	245.3
MPI	16	2	93.7
MPI	32	4	55.9
MPI	48	6	54.8

Table 11: **Experiment 6, quarter dataset** - outer loop parallelization across latitude for 48 latitude values, all data read at once in the beginning and ptile=8.

Protocol	R workers	Nodes	Mean runtime (seconds)	Runtime stddev	Runtime min, max
Eth	16	1	167.5	3.9	161.9, 173.8
Eth	32	2	99.3	2.7	95.4, 104.1
Eth	48	3	68.6	2.2	64.6, 73.1
IPoIB	16	1	168.9	3.1	161.8, 172.4
IPoIB	32	2	99.9	1.9	96.5, 102.9
IPoIB	48	3	69.1	1.6	65.9, 72.0
MPI	16	1	154.6	3.3	151.0, 163.0
MPI	32	2	92.7	2.7	89.1, 98.2
MPI	48	3	67.5	1.4	65.4, 69.9

Table 12: **Experiment 7, quarter dataset** - outer loop parallelization across longitude for 72 longitude values, only reading data used in each iteration. The mean, standard deviation, and range of runtimes are reported for 10 successive runs of each protocol and number of tasks.

Protocol	R workers	Nodes	Runtime (seconds)
Eth	16	2	135.7
Eth	32	4	246.0
Eth	48	6	361.0
IPoIB	16	2	97.7
IPoIB	32	4	128.9
IPoIB	48	6	165.7
MPI	16	2	62.9
MPI	32	4	39.7
MPI	48	6	38.8

Table 13: **Experiment 8, quarter dataset** - outer loop parallelization across longitude for 72 longitude values, all data read at once in the beginning and ptile=8.

Appendix B Code Examples

The following sections are examples of batch job submission scripts and R code for experiments 2 and 5 with both Ethernet and MPI protocols. The batch job scripts were written in Bash for the Yellowstone cluster, which uses the IBM Spectrum LSF workload manager. LSF directives (#BSUB lines) will need to be modified for the appropriate workload manager. Furthermore, Yellowstone uses Lmod for environment management (e.g. module

load intel/16.0.3), so these lines should be adapted to the appropriate environment. The R script contains references to the CMIP dataset used in this TechNote; the path to the desired dataset must be modified accordingly. In general, mutatis mutandis.

On Yellowstone, the experiment 2 Ethernet R script is submitted (requesting 16 CPUs with -n) via the following command:

```
bsub -n 16 < fe_CMIP_2a.lsf
```

The "a" suffix after the experiment number signifies that the R script is Ethernet protocol. We designate IPoIB scripts with "b," and MPI with "c."

Note that we have preserved the path to the dataset residing on GLADE scratch in the R scripts. This is to highlight that the highest performance storage available (GLADE scratch here) should be used for optimal results.

The code for all experiments and the dataset used in this TechNote can be found at https://doi.org/10.5065/D6JW8CK2 [3].

B.1 Experiment 2: Ethernet

B.1.1 LSF submission script

```
#!/bin/bash -f

#BSUB -J fe2
#BSUB -q small
#BSUB -P ACCT
#BSUB -W 00:30
#BSUB -o fe2.%J.stdout
#BSUB -e fe2.%J.stderr
#BSUB -N
#BSUB -x
#BSUB -R "span[ptile=16]"

source /glade/apps/opt/lmod/lmod/init/bash

module load intel/16.0.3
module load R/3.3.2

cd /glade/u/home/$USER/<R script location>

# For Ethernet remove the -ib suffix from hostnames, since
# we want traffic to flow across the onboard
# 1Gb NIC.
```

```
echo $LSB_HOSTS | sed 's/-ib//g' | tr ' ' '\n' > hostfile.$LSB_JOBID
# For IPoIB,  remove "| sed 's/-ib//g'" from the previous expression

# Run this script for 1/4 dataset (48 latitude grid cells).
# For the full dataset, use 192.
Rscript fe_CMIP_2a.R 48 hostfile.$LSB_JOBID

rm -f hostfile.$LSB_JOBID
```

B.1.2 R script

```
library(ncdf4)
library(extRemes)
library(doParallel) # loads foreach, parallel and iterators as dependencies

################################################################################
#### EXPERIMENT 2a: INNER LOOP PARALLELIZED ACROSS LONGITUDE, DATA ONLY READ ONCE  ####
################################################################################

dataPath <- "/glade/scratch/$USER/pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc"

args <- commandArgs(trailingOnly = TRUE)
# Arguments passed to R script
numLat <- as.numeric(args[1])
hostfile <- args[2]

##### Data Read #####
dataHandle <- nc_open(dataPath)
lon <- ncvar_get(dataHandle, "lon")
lat <- ncvar_get(dataHandle,"lat")
tm <- ncvar_get(dataHandle,"time")
dataset <- ncvar_get(dataHandle, "pr", start = c(1, 1, 1),
                     count=c(-1, numLat,-1))
nc_close(dataHandle)

##### User specifications #####
tailProb <- .01 # tail probability used in extremes fitting
returnLevelYear <- 100 # years used for return level
numResults <- 5  # number of result outputs for each gridcell
# empty array to store results
outSummary <- array(NA, c(dim(lat), dim(lon), numResults))
```

```
##### Cluster setup #####
lines <- readLines(hostfile)
hosts <- character()

# This construct comes from an R email list question.
for (line in lines) {
  x <- (strsplit(line[[1]], " "))
  hosts <- c(hosts, rep.int(x[[1]][1], 1))
}

local <- system("hostname", intern=TRUE)
batchMaster <- hosts[1]

for (i in 1:length(hosts)) {
  if (grepl(local, hosts[i])) {
    hosts[i] <- "localhost"
  }
}

if (length(unique(hosts))==1) {
  numWorkers <- length(hosts)
  # remove outfile option to suppress debugging messages
  cl <- makePSOCKcluster(numWorkers, outfile="")
} else {
    workers <- hosts
    # remove outfile option to suppress debugging messages
    cl <- makePSOCKcluster(master=batchMaster, workers, outfile="")
}

registerDoParallel(cl) # Register parallel backend for foreach

numCores <- getDoParWorkers()
print(numCores)

##### EVA fitting #####
p1 <- proc.time()
# outer loop over latitude
for (latindex in 1:numLat) {
  subDataset <- dataset[, latindex, ]
  # start of inner loop over longitude
```

```
  # Note that outSummary assigns the data structure returned from
  # the inner loop due to R's rbind combine statement.
  outSummary[latindex,,] <- foreach (lonindex=1:dim(lon),
                                     .combine=rbind,
                                     .packages=c("extRemes")) %dopar% {
    Y <- subDataset[lonindex,]
    threshold <- quantile(Y, 1-tailProb)
    frac <- sum(Y > threshold)/length(Y)
    # Fit Generalized Pareto Distribution
    GPFit <- fevd(Y, threshold=threshold, type="GP", method="MLE")
    returnLevel <- try(return.level(GPFit, returnLevelYear, do.ci=FALSE))
    # data structure returned from inner loop
    c(threshold, GPFit$results$par, frac=frac, returnLevel)
  }
  print(latindex)  # Counter to monitor progress
}

diffp <- proc.time()-p1
loopTime <- diffp[3]
print(loopTime)

save("outSummary", "loopTime", file=paste("outSummary_exp2a_",
     numCores, ".Rdata", sep=""))

stopCluster(cl)  # Close cluster
```

B.2 Experiment 2: MPI

B.2.1 LSF submission script

```
#!/bin/bash -f

#BSUB -J fe2
#BSUB -q small
#BSUB -P ACCT
#BSUB -W 02:00
#BSUB -o fe2.%J.stdout
#BSUB -e fe2.%J.stderr
#BSUB -N
#BSUB -x
#BSUB -R "span[ptile=16]"
```

```
source /glade/apps/opt/lmod/lmod/init/bash

module load intel/16.0.3
module load R/3.3.2
module load rmpi

cd /glade/u/home/$USER/<R script location>

# Get the number of processors for the mpirun command.
numProcs=$( echo $LSB_HOSTS | tr ' ' '\n' | wc -l )

# We use n-1 MPI workers since the master also performs
# computation.  Run this script for 1/4 dataset (48 latitude grid cells).
# For the full dataset, use 192.
mpirun -np 1 Rscript fe_CMIP_2c.R 48 $(( $numProcs - 1 ))
```

B.2.2 R script

```
library(ncdf4)
library(extRemes)
library(doParallel) # loads foreach, parallel and iterators as dependencies
library(Rmpi)
library(doMPI)

################################################################################################
#### EXPERIMENT 2c: INNER LOOP PARALLELIZED ACROSS LONGITUDE, DATA ONLY READ ONCE  ####
################################################################################################

dataPath <- "/glade/scratch/$USER/R/pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc"

args <- commandArgs(trailingOnly = TRUE)
# Arguments passed to R script
numLat <- as.numeric(args[1])
numCores <- as.numeric(args[2])

##### Data Read #####
dataHandle <- nc_open(dataPath)
lon <- ncvar_get(dataHandle, "lon")
lat <- ncvar_get(dataHandle,"lat")
tm <- ncvar_get(dataHandle,"time")
```

```
dataset <- ncvar_get(dataHandle, "pr", start = c(1, 1, 1),
                     count=c(-1, numLat,-1))
nc_close(dataHandle)

##### User specifications #####
tailProb <- .01 # tail probability used in extremes fitting
returnLevelYear <- 100 # years used for return level
numResults <- 5  # number of result outputs for each gridcell
# empty array to store results
outSummary <- array(NA, c(dim(lat), dim(lon), numResults)

##### Cluster setup #####
cl <- startMPIcluster(numCores) # Create MPI cluster
registerDoMPI(cl)

print(numCores)

##### EVA fitting #####
p1 <- proc.time()
# outer loop over latitude
for (latindex in 1:numLat) {
  subDataset <- dataset[, latindex, ]
  # start of inner loop over longitude
  # Note that outSummary assigns the data structure returned from
  # the inner loop due to R's rbind combine statement.
  outSummary[latindex,,] <- foreach (lonindex=1:dim(lon),
                                     .combine=rbind,
                                     .packages=c("extRemes")) %dopar% {
    Y <- subDataset[lonindex,]
    threshold <- quantile(Y, 1-tailProb)
    frac <- sum(Y > threshold)/length(Y)
    # Fit Generalized Pareto Distribution
    GPFit <- fevd(Y, threshold=threshold, type="GP", method="MLE")
    returnLevel <- try(return.level(GPFit, returnLevelYear, do.ci=FALSE))
    # data structure returned from inner loop
    c(threshold, GPFit$results$par, frac=frac, returnLevel)
  }
  print(latindex)  # Counter to monitor progress
}

diffp <- proc.time()-p1
```

```
loopTime <- diffp[3]
print(loopTime)

save("outSummary", "loopTime", file=paste("outSummary_exp2c_",
     numCores, ".Rdata", sep=""))

closeCluster(cl)  # Close cluster
Rmpi::mpi.quit()
```

B.3 Experiment 5: Ethernet

B.3.1 LSF submission script

The script is identical to experiment 2 Ethernet, with appropriate changes to the script and job names.

B.3.2 R script

```
library(ncdf4)
library(extRemes)
library(doParallel) # loads foreach, parallel and iterators as dependencies

######################################################################
#### EXPERIMENT 5a: OUTER LOOP PARALLELIZED ACROSS LATITUDE ###
######################################################################

dataPath <- "/glade/scratch/$USER/pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc"

args <- commandArgs(trailingOnly = TRUE)
# Arguments passed to R script
numLat <- as.numeric(args[1])
hostfile <- args[2]

##### Initial data read #####
dataHandle <- nc_open(dataPath)
lon<- ncvar_get(dataHandle, "lon")
lat<- ncvar_get(dataHandle,"lat")
tm<- ncvar_get(dataHandle,"time")
nc_close(dataHandle)

#function to reading data for loop
getData <- function(latnum){
```

```
  dataHandle <- nc_open(dataPath)
  dataSlice <- ncvar_get(dataHandle, "pr",
                         start = c(1, latnum, 1), count=c(-1,1,-1))
  nc_close(dataHandle)
  return(dataSlice)
}

##### User specifications #####
tailProb<- .01 # tail probability used in extremes fitting
returnLevelYear <- 100 # years used for return level

##### Cluster setup #####
lines <- readLines(hostfile)
hosts <- character()

# This construct comes from an R email list question.
for (line in lines) {
  x <- (strsplit(line[[1]], " "))
  hosts <- c(hosts, rep.int(x[[1]][1], 1))
}

local <- system("hostname", intern=TRUE)
batchMaster <- hosts[1]

for (i in 1:length(hosts)) {
  if (grepl(local, hosts[i])) {
    hosts[i] <- "localhost"
  }
}

if (length(unique(hosts))==1) {
  numWorkers <- length(hosts)
  # remove outfile option to suppress debugging messages
  cl <- makePSOCKcluster(numWorkers, outfile="")
} else {
    workers <- hosts
    # remove outfile option to suppress debugging messages
    cl <- makePSOCKcluster(master=batchMaster, workers, outfile="")
}

registerDoParallel(cl) # Register parallel backend for foreach
```

```
numCores <- getDoParWorkers()
print(numCores)

###### EVA fitting #####
p1<-proc.time()
# outer loop over latitude
# Note that outSummary assigns the data structure returned from
# the inner loop due to R's rbind combine statement.
outSummary <- foreach(latindex=1:numLat,
                      .combine=rbind,
                      .packages=c("extRemes", "ncdf4", "foreach")) %dopar% {
  dataset <- getData(latindex)
  foreach(lonindex = 1:dim(lon),
          .combine=rbind,
          .packages=c("extRemes", "foreach")) %do% {
    Y <- dataset[lonindex,]
    threshold <- quantile(Y, 1-tailProb)
    frac <- sum(Y > threshold)/length(Y)
    # Fit Generalized Pareto Distribution
    GPFit <- fevd(Y, threshold=threshold, type="GP", method="MLE")
    returnLevel <- try(return.level(GPFit, returnLevelYear, do.ci=FALSE))
    c(latindex, lonindex, threshold, GPFit$results$par, frac=frac, returnLevel)
  }
}

diffp <- proc.time()-p1
loopTime <- diffp[3]
print(loopTime)

save("outSummary","loopTime", file=paste("outSummary_exp5a_",
     numCores, ".Rdata", sep=""))

stopCluster(cl)  # Close cluster
```

B.4 Experiment 5: MPI

B.4.1 LSF submission script

The script is identical to experiment 2 MPI, with appropriate changes to the script and job names.

B.4.2 R script

```
library(ncdf4)
library(extRemes)
library(doParallel) # loads foreach, parallel and iterators as dependencies
library(Rmpi)
library(doMPI)

#########################################################################
#### EXPERIMENT 5c: OUTER LOOP PARALLELIZED ACROSS LATITUDE       ####
#########################################################################

dataPath <- "/glade/scratch/$USER/pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc"

args <- commandArgs(trailingOnly = TRUE)
# Arguments passed to R script
numLat <- as.numeric(args[1])
numCores <- as.numeric(args[2])

##### Initial data read #####
dataHandle <- nc_open(dataPath)
lon <- ncvar_get(dataHandle, "lon")
lat <- ncvar_get(dataHandle,"lat")
tm <- ncvar_get(dataHandle,"time")
nc_close(dataHandle)

#function to reading data for loop
getData <- function(latnum){
  dataHandle <- nc_open(dataPath)
  dataSlice <- ncvar_get(dataHandle, "pr",
                         start = c(1, latnum, 1), count=c(-1,1,-1))
  nc_close(dataHandle)
  return(dataSlice)
}

##### User specifications #####
tailProb <- .01 # tail probability used in extremes fitting
returnLevelYear <- 100 # years used for return level

##### Cluster setup #####
cl <- startMPIcluster(numCores) # Create MPI cluster
```

```
registerDoMPI(cl)

print(numCores)

###### Fitting #####
p1<-proc.time()
# outer loop over latitude
# Note that outSummary assigns the data structure returned from
# the inner loop due to R's rbind combine statement.
outSummary <- foreach(latindex=1:numLat,
                      .combine=rbind,
                      .packages=c("extRemes", "ncdf4", "foreach")) %dopar% {
  dataset <- getData(latindex)
  foreach(lonindex = 1:dim(lon),
          .combine=rbind,
          .packages=c("extRemes", "foreach")) %do% {
    Y <- dataset[lonindex,]
    threshold <- quantile(Y, 1-tailProb)
    frac <- sum(Y > threshold)/length(Y)
    # Fit Generalized Pareto Distribution
    GPFit <- fevd(Y, threshold=threshold, type="GP", method="MLE")
    returnLevel <- try(return.level(GPFit, returnLevelYear, do.ci=FALSE))
    c(latindex, lonindex, threshold, GPFit$results$par, frac=frac, returnLevel)
  }
}

diffp <- proc.time()-p1
loopTime <- diffp[3]
print(loopTime)

save("outSummary", "loopTime", file=paste("outSummary_exp5c_",
     numCores, ".Rdata", sep=""))

closeCluster(cl)  # Close cluster
Rmpi::mpi.quit()
```

Acknowledgements

We wish to thank Anthony Tracy for his assistance running the numerous experiments for this TechNote. We would also like to acknowledge high-performance computing sup-

port from Yellowstone (ark:/85065/d7wd3xhc) provided by the NCAR Computational and Information Systems Laboratory, sponsored by the National Science Foundation. This work was funded in part by the Intel Parallel Computing Center for Weather and Climate Simulation `https://software.intel.com/en-us/articles/intel-parallel-computing-center-at-the-university-of-colorado-boulder-and-the-national`.

References

[1] V. Eyring, S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor, "Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization," *Geoscientific Model Development*, vol. 9, no. 5, pp. 1937–1958, 2016. [Online]. Available: http://www.geosci-model-dev.net/9/1937/2016/

[2] G. Flato, J. Marotzke, B. Abiodun, P. Braconnot, S. Chou, W. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, C. Forest, P. Gleckler, E. Guilyardi, C. Jakob, V. Kattsov, C. Reason, and M. Rummukainen, *Evaluation of Climate Models*. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press, 2013, book section 9, pp. 741–866. [Online]. Available: www.climatechange2013.org

[3] D. Hammerling, "CMIP 5 dataset and code for R parallelization," Boulder, CO, 2017. [Online]. Available: https://doi.org/10.5065/D6JW8CK2

[4] R. Rew and G. Davis, "NetCDF: an interface for scientific data access," *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76–82, July 1990.

[5] T. Hoefler, C. Siebert, and W. Rehm, "A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–8.