

Online: [http://dust.ess.uci.edu/ppr/ppr\\_ZeM07.pdf](http://dust.ess.uci.edu/ppr/ppr_ZeM07.pdf)

Updated: Fri 25<sup>th</sup> May, 2007, 16:22

Manuscript in press in *Int. J. High Perform. Comput. Appl.*

# Scaling Properties of Common Statistical Operators for Gridded Datasets

by Charles S. Zender and Harry Mangalam  
University of California, Irvine

Department of Earth System Science  
University of California, Irvine  
Irvine, CA 92697-3100

[zender@uci.edu](mailto:zender@uci.edu)  
Voice: (949) 824-2987  
Fax: (949) 824-3256

## Abstract

An accurate cost-model that accounts for dataset size and structure can help optimize geoscience data analysis. We develop and apply a computational model to estimate data analysis costs for arithmetic operations on gridded datasets typical of satellite- or climate model-origin. For these dataset geometries our model predicts data reduction scalings that agree with measurements of widely-used geoscience data processing software, the netCDF Operators (NCO). I/O performance and library design dominate throughput for simple analysis (e.g., dataset differencing). Dataset structure can reduce analysis throughput ten-fold relative to same-sized unstructured datasets. We demonstrate algorithmic optimizations which substantially increase throughput for more complex, arithmetic-dominated analysis such as weighted-averaging of multi-dimensional data. These scaling properties can help to estimate costs of distribution strategies for data reduction in cluster and grid environments.

Keywords: computational model; geoscience; scaling; data analysis; netCDF;

## 1 Introduction

Scientific advances in geosciences increasingly depend on large scale computing (e.g., [NRC, 2001](#); [NSF, 2003](#)). Appropriately, much attention has been given to the computational resources, sometimes called cyberinfrastructure, required to produce such massive numerical experiments (e.g., [UCAR, 2005](#)). Analysis and post-processing of the resulting tera-scale geoscience datasets presents its own set of problems. The solutions to these problems include seamless or virtual data grids (e.g., [Foster et al., 2002](#); [Cornillon et al., 2003](#)) and middleware which optimizes the distribution of data analysis across the available computing resources (e.g., [Woolf et al., 2003](#); [Chen and Agrawal, 2004](#)).

Geoscience datasets typically have characteristics distinct from tera-scale datasets from other scientific disciplines. In contrast to computational biology, where datasets are often irregular in size and stored in databases, and particle physics, where data reduction often consists of sifting records for “events”, geoscientists often work with gridded data which facilitates simple storage and memory access patterns. In particular, we are interested in data analysis optimization for

geoscience datasets stored on rectangular grids rather than, e.g., polygonal meshes common in GIS applications. The size and coordinate values of each dimension in a rectangular dataset are independent of all other dimensions. Often in geoscience datasets the rectangular coordinates include latitude, longitude, and time. Rectangular datasets are well suited to parallel analysis because their mutually independent coordinates facilitate decomposition into smaller datasets of finer granularity, e.g., chunking (*Li et al., 2003; Drake et al., 2005*).

The volume of data and storage space in a rectangular dataset scales exactly 1 : 1 with the total number  $N$  of gridpoints (neglecting compression) and the computational requirements for analyzing the dataset usually scale as  $\mathcal{O}(N)$  for simple arithmetic analyses. However, the constant of proportionality lurking in  $\mathcal{O}(N)$  depends strongly on the dataset rank and arithmetic operation type. As we show below, dataset rank  $R$  can change the computational requirements by more than an order of magnitude for typical geoscience datasets. To make on-line data-processing cost-estimators more efficient, processing costs must be estimated with higher accuracy than this.

Here we develop a framework for predicting resources required to analyze and reduce geoscience datasets stored in rectangular gridded files. This framework enables us to quantitatively estimate data analysis costs associated with the increasing resolution, length, and structure of geoscience datasets. Our analysis focuses on the scaling properties of fundamental statistical operations (differencing, averaging) in uniprocessor environments. These scaling properties can be used in cost-models to allocate and optimize resources in more complex data analysis workflows in parallel and distributed environments (e.g., *Wolf et al., 2003; Chen and Agrawal, 2004*).

To place this research in context, groups such as the Intergovernmental Panel on Climate Change (IPCC) and the Program for Climate Model Diagnosis and Intercomparison (PCMDI) coordinate intercomparisons of dozens of General Circulation Model (GCM) simulations (e.g., *Cubasch and Meehl, 2001; Fiorino and Williams, 2002*), each of which may occupy several terabytes (TB) of netCDF data. Post-processing data reduction for current and previous IPCC assessment reports relies heavily on the netCDF Operators used here (*Zender, 2006*) and other toolkits like the Climate Data Analysis Tools (CDAT) (*Fiorino and Williams, 2002*), the Climate Data Operators (CDO; <http://www.mpimet.mpg.de/fileadmin/software/cdo>), the Grid Analysis and Display System (GrADS; <http://www.iges.org/grads/grads.html>), and the NCAR Command Language (NCL; <http://www.ncl.ucar.edu>). The ability to model the workflow costs will help analysts optimize workflow distribution amongst available computational resources (*Foster et al., 2002; Chen and Agrawal, 2004*) with their toolkit-of-choice.

## 2 Methods

This section first presents a framework for describing the geometry of gridded scientific datasets. The descriptors include dataset shape and size, as well as nomenclature for how shape and size change during data reduction. Next, we enumerate the algorithmic costs of the fundamental data reduction operations. Finally, we present the aggregate costs of high-level data reduction operations such as dataset differencing and weighted averaging.

## 2.1 Framework for Dataset Geometry

The gridded geophysical data of interest are assumed to obey

$$N \gg D \gg R > 1 \quad (1)$$

where  $N$  is the size (number of elements) of a variable,  $D$  is the length of a dimension (e.g., spatial or temporal), and  $R$  is the rank (number of dimensions) in the variable. (Table 4 in the appendix contains a symbol list). In our experience, this assumption applies to most gridded data generated or stored on high performance computers.

The exact size  $N$  of a rank  $R$  variable is

$$N \equiv \prod_{k=1}^{k=R} D_k \quad (2)$$

The cost to process  $N$  elements depends on the arithmetic algorithm which may explicitly depend on the  $D_k$ , i.e., on the variable shape. Computational grids are approximately hypercubes if their dimensions satisfy

$$D_1 \approx D_2 \approx \dots \approx D_{R-1} \approx D_R \quad (3)$$

For such cases (2) simplifies to

$$N = \bar{D}^R \quad \text{where} \quad (4)$$

$$\bar{D} \equiv \sqrt[R]{N} \quad (5)$$

which defines the mean dimension size  $\bar{D}$ . A rank  $R$  hypercube with  $N$  elements has  $R$  dimensions each of the same geometric mean size  $\bar{D}$ . Current Earth System Models are approaching a spatial  $\bar{D} \sim 128$  (Cubasch and Meehl, 2001). Equation (4) suffices to estimate storage costs for changing resolution. For example, doubling resolution in all three spatial dimensions increases  $N$ , i.e., storage, by  $2^3$  or eight-fold. However, estimating data analysis costs requires knowledge of  $N$  and rank  $R$  (and other, analysis-specific parameters) on which we now focus.

Data analysis often involves transforming variables from an initial rank  $R_i$  to a final rank  $R_f$ . Rank-reducing operations have  $R_i > R_f$ , while rank expansion operations have  $R_f > R_i$ . A typical example of rank reduction is distilling raw data down to one or two meaningful statistics, e.g., mean and standard deviation. A form of rank expansion called broadcasting is often a required intermediate data processing step.

A prime example of broadcasting occurs when gridded quantities are aggregated. Geophysical computational grids are (typically) rectangular coordinate grids where the number of longitudes per latitude is constant. Such rectangular grids on the sphere (e.g., Earth) are not equi-areal grids. Hence aggregation and manipulation of conservative physical quantities (e.g., fluxes, concentrations, state variables) requires area-weighting the grid data in order to conserve mass, energy, tracers, and momentum. The weights (relative areas) are one-dimensional for many popular geophysical grids such as equi-angular and Gaussian grids. Thus it is common to “broadcast” weights from  $R_i = 1$  to the variable rank  $R$  prior to processing variables.

Broadcasting algorithms transform smaller-ranked variables into larger ranks by duplicating existing values into the new dimensions. For broadcasting to work, the initial and final variable

shapes must conform. Variable shape is the vector representation of its dimensionality. A rank  $R$  variable has shape  $\mathbf{S}$

$$\mathbf{S} = [D_1, D_2, \dots, D_{k-1}, D_k, D_{k+1}, \dots, D_{R-1}, D_R] \quad (6)$$

Consider two variables with shapes  $\mathbf{S}_1$  and  $\mathbf{S}_2$  where  $R_1 < R_2$ . Shapes  $\mathbf{S}_1$  and  $\mathbf{S}_2$  conform only if  $\mathbf{S}_1$  is a subset of  $\mathbf{S}_2$  ( $\mathbf{S}_1 \subset \mathbf{S}_2$ ) so that all axes (dimensions) in  $\mathbf{S}_1$  appear in  $\mathbf{S}_2$ . Broadcasting algorithms are more efficient if the common dimensions, i.e., all  $\mathbf{S}_1$  dimensions, appear in the same order in  $\mathbf{S}_2$ . However,  $\mathbf{S}_2$  may contain any number of additional dimensions arbitrarily interspersed among the common  $\mathbf{S}_1$  dimensions.

The preceding discussion of shapes  $\mathbf{S}_1$  and  $\mathbf{S}_2$  applies equally to two other important tasks in geophysical data reduction: rank reduction and re-ordering. The most common form of rank reduction is averaging, which geophysicists often perform on spatio-temporal datasets. Variables of shape  $\mathbf{S}$  (6) may be reduced (e.g., averaged) over a rank- $R_A$  averaging space  $\mathbf{S}_A$  if  $\mathbf{S}_A \subset \mathbf{S}$ . The  $R_A$  dimensions in  $\mathbf{S}_A$  are enumerated from one to  $R_A$  so that

$$\mathbf{S}_A = [A_1, A_2, \dots, A_{R_A-1}, A_{R_A}] \quad (7)$$

$$N_A = \prod_{\substack{k=1 \\ A_k \in \mathbf{S}_A}}^{k=R_A} A_k \quad (8)$$

where  $N_A$  is the number of elements in an input variable reduced to a single element of the output variable. Hence  $N_A$  is the product of all  $R_A$  reduced dimensions (8).

Data reduction algorithms must be flexible enough to handle not only weights, but also masks and invalid values. Masks are boolean flags  $m$  which indicate whether a datum should be included or excluded from the arithmetic operation. For instance, spatial masks are used to restrict arithmetic operations to specific geographic regions. In gridded datasets, invalid or missing data are indicated by a special value often called the missing value (*Rew and Davis, 1990*). The boolean missing value flag  $\mu$  indicates whether a datum should be included or excluded from the arithmetic operation. The average  $\bar{x}$  of a variable  $x$  weighted by  $w$  with mask  $m$  and missing value  $\mu$  is

$$\bar{x}_j = \sum_{i=1}^{i=N_A} \mu_i m_i w_i x_i \bigg/ \sum_{i=1}^{i=N_A} \mu_i m_i w_i \quad (9)$$

where the subscripts  $i$  and  $j$  range over the input and averaged hyperslabs, respectively. The denominator of (9) makes clear that the weight  $w$  must be arithmetically reduced like the variable  $x$  to ensure proper normalization. The remainder of this paper neglects  $\mu$  and  $m$  since our investigations showed that their presence introduces negligible computational overhead in modern compilers.

## 2.2 Operation Counts

Data reduction operations occur in five phases: input, pre-processing, arithmetic, post-processing, and output. Input and output refer to reading and writing, respectively. In this paper, we restrict our attention to local data reduction, where I/O occurs from/to local disks rather than across networks. Pre-processing may also include allocation of intermediate buffers, data broadcasting, and type

**Table 1: Operation Count Notation**

	Parameter Description
$F$	Floating point operations
$I$	Total integer operations $= I_A + M_u + M_s$
$I_A$	Integer arithmetic operations
$M_u$	User memory operations
$M_s$	System memory operations

promotion. Post-processing often includes similar operations in the reverse order, e.g., buffer de-allocation, type demotion. Pre- and post-processing include, when necessary, byte-swapping to convert between disk and memory byte-ordering. The arithmetic phase encompasses those computations strictly required by the specific analysis (e.g., differencing and averaging).

To quantify the computational complexity of data-processing, we count the number of operations the netCDF binary operator `ncbo` and netCDF weighted averager `ncwa` use for data reduction. These programs are two of the twelve netCDF Operators (NCO) (Zender, 2006) commonly used to analyze gridded geoscience data stored in the self-describing netCDF format (Rew and Davis, 1990). We counted four basic types of operations: floating point arithmetic (addition, division), integer arithmetic, user memory management, and system memory management. (Table 1). Integer arithmetic operations  $I_A$  include pointer arithmetic and all arithmetic between non-floating point values. User memory operations  $M_u$  include memory fetches and writes without system calls. System memory management  $M_s$  includes all memory manipulation performed with system calls such as `memcpy()`.

Counting the three sub-categories ( $I_A$ ,  $M_u$ ,  $M_s$ ) of integer operations separately provided us useful information on algorithmic design. However, we can only measure the total number of integer operations  $I = I_A + M_u + M_s$ . For brevity, we report only  $I$  for our computational model. Contact the authors directly for the individual relations for  $I_A$ ,  $M_u$ , and  $M_s$ .

### 2.2.1 Byte-Swapping

netCDF3 imposes a byte-swapping overhead on little-endian machines (Rew and Davis, 1990). A byte-swap per datum translates the on-disk netCDF format data (big-endian) to/from the native machine type (often little-endian) prior to arithmetic and storage, respectively. The netCDF3 byte-swapping implementation scales as

$$I(\text{byte-swap}) \approx N(W + \epsilon) \tag{10}$$

where  $W$  is the number of bytes per datum and  $\epsilon$  is an implementation-defined constant. Geoscience data are usually stored as single precision ( $W = 4$ ) floating point numbers so we use single precision in our tests. netCDF3 library versions  $\leq 3.6.1$  implement an  $\epsilon = 2$  algorithm. We propagate  $\epsilon = 2$  to the cumulative operation counts described below. Future byte-swapping implementations for the netCDF3 library (i.e., versions  $\geq 3.6.2$ ) may use loop-unrolling techniques to achieve  $\epsilon \lesssim 0.5$ . This could reduce the arithmetic cost of byte-swapping by up to 25%.

All benchmarks reported here were performed on an Opteron (little-endian) system. Hence our predicted (and observed) operation counts account for byte-swapping (10) during data pre-processing (after disk-reads) and post-processing (before disk-writes).

### 2.2.2 Binary Operations

Simple data reduction often involves straightforward binary (i.e., dual operand) arithmetic operations such as subtraction, addition, or comparison. For binary operations,

$$F(\text{binary}) = N = \bar{D}^R \quad (11)$$

We do not explicitly report zero-valued counts so the absence of a formula for  $I$  in Equation 11 implicitly indicates that binary operations require no integer operations that scale with  $N$ . Hence, the computational model sets  $I = 0$ .

### 2.2.3 Broadcasting

The integer operations required to broadcast a variable (such as a weight) of rank  $R_w$  to conform to a variable of rank  $R$  and size  $N$  are

$$I(\text{broadcast}) \approx N(6R + 8R_w + 2) \quad \text{for } R > R_w \quad (12)$$

Broadcasting uses a large number of arithmetic operations and no floating point arithmetic.

Our broadcast algorithm deconstructs each one-dimensional offset of the target buffer to fill into the  $R$  subscripts of the target shape. These subscripts are then mapped to the address of the source element in the array to broadcast. This mapping requires a loop over the  $R_w$  dimensions of the variable to broadcast so increasing  $R_w$  is expensive (12). For example, it requires about 25% more integer operations to broadcast into an  $R_w = 3$  shape  $\mathbf{S}_3 = [D_1, D_2, D_3]$  from an  $R_w = 2$  shape such as  $\mathbf{S}_2 = [D_1, D_3]$  than from an  $R_w = 1$  shape such as  $\mathbf{S}_1 = [D_2]$ .

### 2.2.4 Intra-file Reduction

Three sequential algorithms are employed to rank-reduce a variable over an arbitrary subset of its dimensions. Averaging is the most common rank reduction procedure performed, so we describe the scaling of our multi-dimensional averaging algorithms. First, the hypercube of input values associated with each averaged value (9) is placed in a one-dimensional array called an averaging block. This is the collection step. It places data in sequential, non-strided, memory that chips may cache during arithmetic. Collecting together all input values which contribute to a given output value demands many integer arithmetic, pointer manipulation, and memory re-arrangement operations. After collection, each averaging block is summed and a tally made of the valid input values contributing to each output value (missing values, e.g., are invalid data). This step is called reduction since it condenses all the values from the  $R_A$ -averaged input dimensions into a single (summed) output value, thus reducing the variable rank. Reducing the averaging blocks, even though it involves floating point arithmetic, can be an order of magnitude faster than the collection step, depending on the variable rank. After reduction, the normalization step divides the sum of each averaging block by the tally to obtain the mean.

The collection step for a variable of rank  $R > 1$  and size  $N$  to be averaged over  $R_A$  dimensions requires

$$I(\text{collect}) \approx N(14R + 4) \quad \text{for } R > 1 \quad (13)$$

operations. Our collection algorithm deconstructs the one-dimensional offset of each element in the source buffer to be averaged into its  $R$ -subscripts. The subscripts for the  $(R - R_A)$ -non-averaged dimensions determine the address of the appropriate averaging block, while the subscripts for the  $R_A$ -averaged dimensions determine the offset within the averaging block. Since the inner loop traverses all  $R$ -dimensions for each element, the collection step is independent of  $R_A$ . The collection algorithm and its integer operation count are similar to broadcasting (12) with  $R_w = R$ . In practice, masking and weighting (9) may precede collection to minimize the number of buffers that require collection.

Rank  $R = 1$  data (aka unstructured data, arrays) are stored in averaging block order and so never require collection. Quantifying the change in data analysis cost due to the dataset structure—by which we mean variable size, shape, and rank—is a central objective of this work. It is worth noting that perhaps the most significant difference between single and multi-dimensional data reduction is that the former has no collection cost.

The collection procedure (13) for creating averaging blocks always works though it is unnecessary and therefore should be omitted in an important class of averages (including  $R = 1$ ). Variables are already stored in averaging block order if the averaging space  $\mathbf{S}_A$  (7) comprises the most rapidly varying (MRV) dimensions of  $\mathbf{S}$  (6). The averaging dimensions are the MRV dimensions if

$$\mathbf{S}_A = [A_1, A_2, \dots, A_{R_A-1}, A_{R_A}] \in [D_{R-R_A+1}, D_{R-R_A+2}, \dots, D_{R-1}, D_R] \quad (14)$$

The collection step (13), therefore, is unnecessary for averages over MRV dimensions.

Many geophysical applications order data output so that frequently reduced dimensions are MRV dimensions (14). For example, the widely used Climate and Forecast (CF) metadata convention (Gregory, 2003) recommends ordering geophysical variables in  $(t, z, y, x) = (\text{time, level, latitude, longitude})$  order. This assumes the right-most dimension varies most quickly in storage order. The C-language adopts this convention. Fortran defines the left-most dimension to vary most rapidly in storage order so that the CF-recommended storage order in Fortran notation is  $(x, y, z, t)$ . Hence commonly requested longitude-averages (i.e., zonal means) and longitude/latitude averages (i.e., area means) reduce MRV dimensions of CF-compliant datasets.

The reduction step for averaging has

$$F(\text{reduce}) = N \quad (15a)$$

$$I(\text{reduce}) \approx N(6 + N_A^{-1}) \quad (15b)$$

The reduction step implements the floating point operations specific to the requested arithmetic. For example, reduction of  $N$  elements requires  $N$  additions for determining averages,  $N$  compares for determining extrema, and  $N$  adds and  $N$  multiplies for determining standard deviations. Reduction (15) is much quicker than collection (13) since one of the purposes of collection is to place all averaging blocks in contiguous memory. Reduction then accesses this sequential, non-strided, memory which often causes values to be stored in fast cache before averaging.

The factor  $N_w = N/N_A$  in (15) is the number of elements remaining in the rank-reduced (i.e.,

output) variable. Exact and approximate values for  $N_W$  come from (8) and (5), respectively

$$N_W = N/N_A = \prod_{\substack{k=R-R_A \\ D_k \notin \mathcal{S}_A}}^{k=R-R_A} D_k \approx \bar{D}^{R-R_A} \quad (16)$$

The number  $N_W$  of output elements to write decreases geometrically as  $R_A$  increases (16) until  $N_W = 1$  (i.e., a scalar) for  $R_A = R$  (i.e., complete averaging).

The operation counts for the final step, normalization, are

$$F(\text{normalize}) = N/N_A \quad (17a)$$

$$I(\text{normalize}) \approx 4N/N_A \quad (17b)$$

Normalization does not occur when the output is independent of the number of contributing values. Integration, minimization, and maximization, for example, do not require normalization.

## 2.2.5 Multi-file Reduction

Computing statistics (e.g., averages) of an arbitrary number of input files results in significantly different memory and disk access patterns than the intra-file averaging described above. Geoscience data producers (e.g., GCMs, satellites) typically truncate files after a sufficient number of records have been stored. We will describe scaling and optimization of multi-file data reduction in future work.

## 2.3 Total Operation Counts

### 2.3.1 Binary Operations

The total operation counts for a binary operation on a variable of size  $N$  is the sum of the counts from byte-swapping (10) each datum three times (once when reading each input buffer and once when writing the output buffer) and of the binary operation (e.g., subtraction) itself (11)

$$F(\text{binary}) = N \quad (18a)$$

$$I(\text{binary}) \approx 3N(W + 2) \quad (18b)$$

Hence binary operations do not depend on variable rank  $R$ .

### 2.3.2 Averaging

The total operation counts to average (without weighting) a variable of rank  $R > 1$  and size  $N$  are the sum of the counts from byte-swapping (10), collection (13), reduction (15), and normalization (17) of the variable.

$$F(\text{average}) = N(1 + N_A^{-1}) \quad (19a)$$

$$I(\text{average}) \approx N[14R + 13 + W + (W + 6)N_A^{-1}] \quad (19b)$$

Hence averaging (and similar reduction operations) scales linearly with variable rank  $R$ . The averaging cost depends on the number  $R_A$  of averaged dimensions through the  $N_A^{-1}$  terms (19).

Interestingly,  $R_A$  has a negligible effect on arithmetic time since  $N_A^{-1} \ll 1$  (1). The more significant impact of  $R_A$  on processing time occurs through its effect on  $N_W$  (16) and the time to write the output to disk.

As described in Section 2.1 above geophysicists often compute weighted moments of variables. The averaging costs summarized in (19) can be extended to weighted averaging by taking into account the additional costs imposed by processing the weights (9). Weighted averages incur collection costs (13) for both the numerator and denominator (9). Accounting for byte-swapping, broadcasting, collection, reduction, and normalization of the weights and values in (9) leads to

$$F(\text{wgt. avg.}) = N(3 + 2N_A^{-1}) \quad (20a)$$

$$I(\text{wgt. avg.}) \approx N[34R + 8R_w + 25 + W + (W + 11)N_A^{-1}] + (W + 2)N_w \quad (20b)$$

where  $R_w$  and  $N_w$  are the rank and number of elements in the weight, respectively, and  $R > R_w$ . Weighting a variable more than doubles the operations required to reduce it.

Other statistical and arithmetic rank reduction procedures such as minimization, maximization, standard-deviations may be performed with very similar algorithms to averaging. The collection step is identical to (13) for many simple rank-reduction procedures, and it suffices to replace the reduction step (15) with the appropriate operator (e.g., minimization, summation of squares). Hence many rank reduction operations scale like averaging, and allow the same optimizations.

## 2.4 Optimizations

Two optimizations to weighted rank reduction are sufficiently general and powerful as to warrant special mention. First, MRV optimization (Section 2.2.4) eliminates twice the collection cost indicated by (13), a substantial fraction of (20). Throughput can increase by an order-of-magnitude in such cases (cf. Figure 5).

Second, weights are often broadcast prior to rank reduction (9) when  $R_w < R$  to facilitate fast, non-strided “dot-product” arithmetic which maximizes caching efficiencies. Reduction never need alter the broadcast weight which may, therefore, be re-used to average other variables of the same shape. This Weight Re-Use (WRU) technique can eliminate all but the first weight broadcast for a set of identically shaped variables. To obtain the full value of WRU with minimum cost, identically shaped variables should be averaged consecutively. Otherwise a variety of broadcast weights (of different shapes) must be maintained. NCO (Zender, 2006) implements a simple WRU form in which the last broadcast weight is always retained and re-used, if possible, until it is destroyed when a differently shaped weight is required. This is a compromise between optimization complexity and memory use.

These optimizations significantly reduce the total integer operations required by weighted rank reduction from (20) to

$$I(\text{Un-optimized}) \approx N[34R + 8R_w + 25 + W + (W + 11)N_A^{-1}] + (W + 2)N_w \quad (21a)$$

$$I(\text{WRU}) \approx N[28R + 0 + 23 + W + (W + 11)N_A^{-1}] + (W + 2)N_w \quad (21b)$$

$$I(\text{MRV}) \approx N[6R + 8R_w + 17 + W + (W + 11)N_A^{-1}] + (W + 2)N_w \quad (21c)$$

$$I(\text{WRU+MRV}) \approx N[0 + 0 + 15 + W + (W + 11)N_A^{-1}] + (W + 2)N_w \quad (21d)$$

for the WRU and MRV optimizations, and their combination, respectively (floating point operations are unaffected).

**Table 2: Machine/Compiler-dependent Parameters<sup>a</sup>**

Sym.	Description [units]	Value
$v_F$	Floating point (averaging)	$153 \times 10^6$
$v_F$	Floating point (subtraction)	$353 \times 10^6$
$v_I$	Integer (averaging)	$200 \times 10^6$
$v_I$	Integer (subtraction)	$1386 \times 10^6$
$v_R$	Read	$63.4 \times 10^6$
$v_W$	Write	$57.9 \times 10^6$
IPO <sub>I</sub>	Averaging	0.38
IPO <sub>I</sub>	Subtraction	2.0
IPO <sub>F</sub>	Floating point	7.0

<sup>a</sup>Parameters used in figures based on Opteron 246 system with GCC 4.0. Operation speeds  $v_I$  and  $v_F$  in operations per second. I/O speeds  $v_R$  and  $v_W$  in bytes per second. IPO in PAPI-measured instructions per operation.

## 2.5 Elapsed Time

The execution time  $T$  is the sum of integer and floating point operation times  $T_I$  and  $T_F$ , respectively, and the times  $T_R$  and  $T_W$  required to read and write data, respectively.

$$T = T_I + T_F + T_W + T_R \quad (22a)$$

$$= I/v_I + F/v_F + N_W W/v_W + N_R W/v_R \quad (22b)$$

Right hand side terms are in decreasing order of importance for typical data reduction operations. Here  $v_I$  and  $v_F$  are the (machine-specific) rates that integer and floating point operations are performed, respectively. Our software performs I/O and arithmetic sequentially (without overlap), as indicated in Equation 22 (potential improvements to this design are discussed below). We assume that I/O times are constant read and write speeds  $v_R$  and  $v_W$ , respectively, times the number of input and output data,  $N_R$  and  $N_W$ , respectively. For instance, binary operations have  $N_R = 2N$  and  $N_W = N$ , while complete rank reduction has  $N_R = N$  and  $N_W = 1$ . We determined the machine and compiler-dependent speed parameters in (22) from these assumptions and empirical testing (Table 2).

## 2.6 File Geometry

We picked two file geometries to try to represent resolutions, file sizes, and file complexities of two common types of geoscience datasets, designated Satellite and GCM, respectively. The Satellite dataset mimics storage requirements typical of global satellite imagery at about 5 km resolution (near the equator). Eight variables are stored on a regular  $2160 \times 4320$  (latitude  $\times$  longitude) grid. A real world analog of this synthetic Satellite test file is a 5 km set of images from the seven spectral channels of the NASA MODIS instrument. The GCM dataset represents output from a relatively high resolution general circulation model (i.e., climate model). The Community

**Table 3: File Geometries**

	Satellite	GCM
Max. Rank $R$	2	4
Variables	8 <sup>a</sup>	128 <sup>b</sup>
Time	—	8
Level	—	32
Latitude	2160	128
Longitude	4320	256
Mean <sup>c</sup> $\bar{D}$	3055	55
Elements $N$ [#]	$75 \times 10^6$	$285 \times 10^6$
Total Size [MB]	299	1143

<sup>a</sup>All eight variables are rank  $R = 2$ .

<sup>b</sup>Eight variables are scalars ( $R = 0$ ), eight variables contain the time dimension only ( $R = 1$ ), sixteen variables contain only latitude and longitude ( $R = 2$ ), sixty-four variables have time, latitude, and longitude ( $R = 3$ ), thirty-two variables have contain all four dimensions ( $R = 4$ ).

<sup>c</sup>Weighted by variable size and number.

Atmosphere Model (*Drake et al., 2005; Collins et al., 2006*) at T85xL32 resolution produces a file of approximately this geometry once per simulated-day when data are archived every three simulated hours. Table 3 summarizes the geometries in the Satellite and GCM datasets tested. The Satellite and GCM datasets are examples of ranks  $R = 2$  and 4 geometries, respectively. Datasets with  $R = 5$  might contain, e.g., spatio-temporal distributions of spectral information, or multiple chemical species information, at each grid point. Geoscience datasets with  $R > 5$  are too rare to warrant further consideration in this study. Geoscience data are usually stored as single precision (four-byte) floating point numbers. In our tests, coordinate axes are stored in double precision and all other data are stored in single precision.

## 2.7 Measurements

The netCDF Operators (NCO) (*Zender, 2006*) are a suite of command-line operators widely used for analyzing geoscience datasets stored in the self-describing netCDF format (*Rew and Davis, 1990*). NCO implements the data reduction algorithms described above. Many institutions (including, to our knowledge, all climate modeling centers) use NCO for data post-processing, hyper-slabbing and serving. This manuscript compares the data processing metrics defined above against measurements of NCO processing standard netCDF3 datasets.

Modern CPUs all have special registers to monitor performance. We use the Performance Application Programming Interface (PAPI) (*Browne et al., 2000*) as interpreted by the HPCToolkit (*Mellor-Crummey et al., 2002*) to count CPU instructions executed. HPCToolkit provides a convenient way to select the counters to monitor, record them, and attribute counts to particular source code blocks. Compiler optimization and CPU-level instruction re-ordering renders analysis beyond the level of a logical code block meaningless. PAPI samples our NCO applications running

at nearly full speed without significant interference.

We present our results below in terms of floating point and integer operation counts and total elapsed time-to-completion. It is necessary to convert floating point and integer instructions (measured by PAPI) to operations for comparison with the computational model described above. To do this we sampled large numbers of operations and imputed a conversion factor from PAPI-measured instructions to model-estimated operations. The resulting number of Instructions Per Operation (IPO) depends on the instruction type and, for integer instructions, the algorithm employed (Table 2).

PAPI cannot distinguish between instructions generated by memory operations (e.g., fetching from memory) and integer arithmetic (e.g., pointer arithmetic). PAPI counts the instructions (or cycles) for each such operation equivalently. Hence we compare PAPI’s measurement of total integer instructions to  $I$ . PAPI counts floating point operations unambiguously so we directly compare measured vs. predicted  $F$ .

All measurements were performed on an Opteron Model 246 processor with 4 GB RAM, 1 MB Level 1 cache (Table 2). The RAM was adequate to hold all data memory-resident in all tests (i.e., no swap space was used). The test results were highly reproducible and the standard error shown is often less than 5% using only four replicates. Automatic caching led to significant performance increases which are unlikely to be met in real-world data processing. Hence we flush all RAM with random data prior to each test except where explicitly noted.

The IPO ratio for all floating point operations on the benchmark system was  $\text{IPO}_F \approx 7.0$  (Table 2). However, the integer IPO ratio increased from  $\text{IPO}_I \approx 0.38$  for the dataset averaging algorithms to  $\text{IPO}_I \approx 2.0$  for the dataset subtraction algorithms. The difference in integer IPO ratios may stem from the algorithmic differences between subtraction and averaging which lead to different compiler optimizations, data locality, and pipelining efficiencies. Moving data to and from main memory is very expensive relative to accessing registers or local cache. In situations where the data are retained local to the CPU registers or primary cache, the Opteron can dispatch up to six instructions per clock cycle, consistent with the six-fold increase in  $\text{IPO}_I$  from averaging to subtraction.

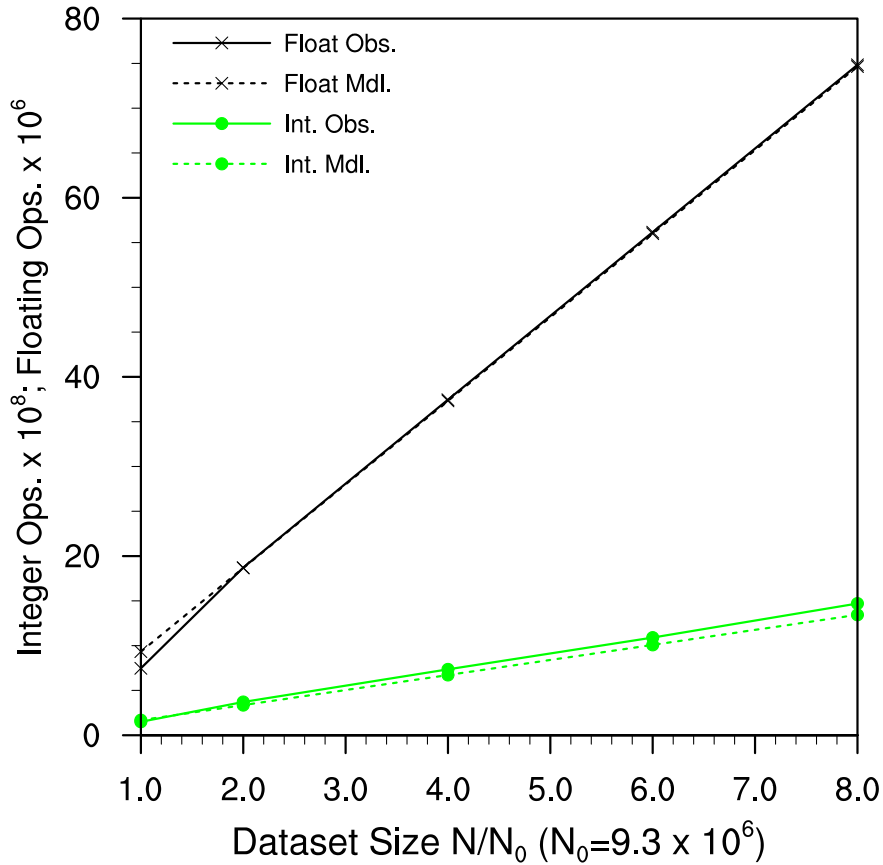
The algorithm-specific changes in  $v_F$  and  $v_I$  (Table 2) may be due to algorithm-specific variation in wait states due to data locality. The compiler and CPU can optimize with block copies, speculative execution, and procedures which are beyond our expertise to count. Accounting for these optimizations would significantly increase the architecture-dependence and complexity of the computational model, and is beyond the scope of this paper.

The experiments measure and model the operation counts and times to process Satellite and GCM datasets (Table 3). The one-eighth increments in dataset size were achieved by processing a fraction of the variables, or by hyperslabbing the dataset dimensions.

### 3 Results

We measured the scaling of typical data reduction operations on the Satellite and GCM data files (Table 3) which are distributed with NCO (Zender, 2006). The numbers of integer and floating point operations necessary to perform a binary operation between datasets with  $N$  elements are simple to predict (18), and agree well with the observed (with PAPI) operation counts (Figure 1) for dataset differencing with the NCO `ncbo` program (Zender, 2006). The integer operations

## Binary Operation Counts Satellite Dataset



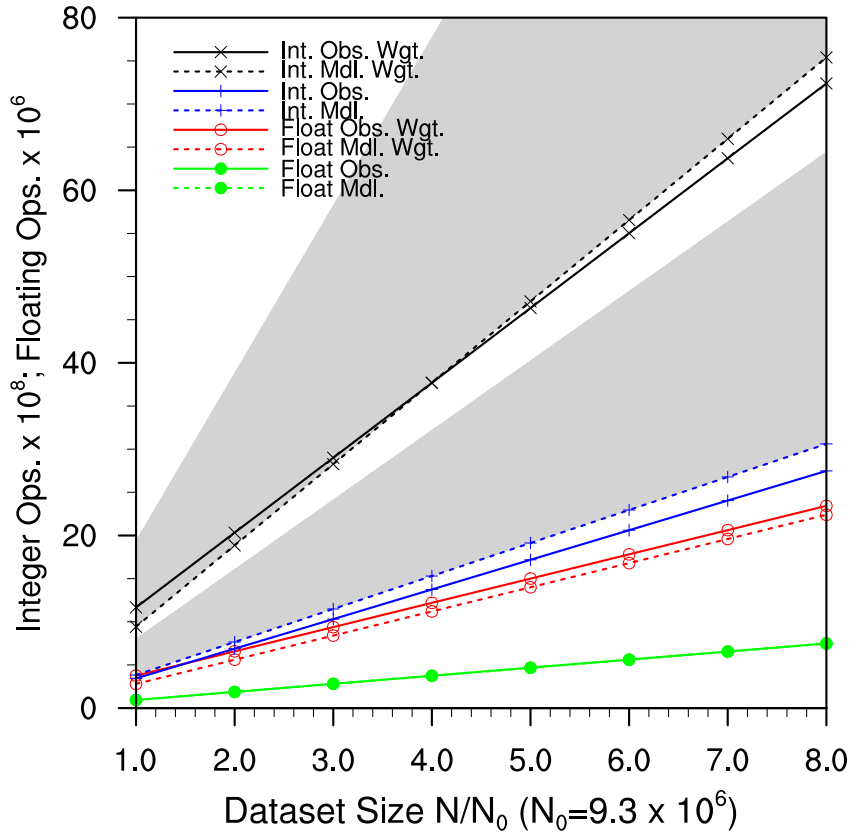
**Figure 1:** Observed (solid) and predicted (dashed) integer  $I$  and floating point  $F$  operations necessary to difference datasets with  $N$  elements. Horizontal axis scaled to units of  $N_0 = 9.3 \times 10^6$  elements. Note different scales for integer and floating point operations.

arise from byte-swapping overhead imposed by the netCDF library which translates IEEE (big-endian) netCDF data to/from native Opteron (little-endian) order before and after arithmetic (18). There is exactly one floating point operation per datum. Some discrepancy between observed and predicted counts is inevitable since our model neglects the number of operations devoted to program overhead and metadata manipulation. However, this discrepancy is small relative to the number of arithmetic operations.

Figure 1 shows results for the Satellite dataset (Table 3). The results for the GCM dataset (not shown) look identical since binary operation counts depend only on  $N$ . Dataset-differencing costs are independent of dataset geometry and rank  $R$  (18).

Dataset structure plays a strong role in data reduction operations such as averaging. The computational model (19) predicts that averaging the Satellite dataset without weights requires approximately forty times more integer than floating point operations, and this is consistent with measurements (Figure 2). Most of the integer operations occur in the collection step (13) required prior to averaging any variable with rank  $R > 1$ . Because algorithms like collection become more expensive with increased rank, it is less expensive to average the Satellite dataset, with  $R = 2$  (Ta-

## Averaging Operation Counts Satellite Dataset



**Figure 2:** Observed (solid) and predicted (dashed) integer  $I$  and floating point  $F$  operations necessary to average an  $N$  element dataset with (upper two sets of curves) and without (lower two sets of curves) weighting. Grey areas indicate  $I$  predicted for rank  $R = 2-5$  datasets. Other markings as in Figure 1.

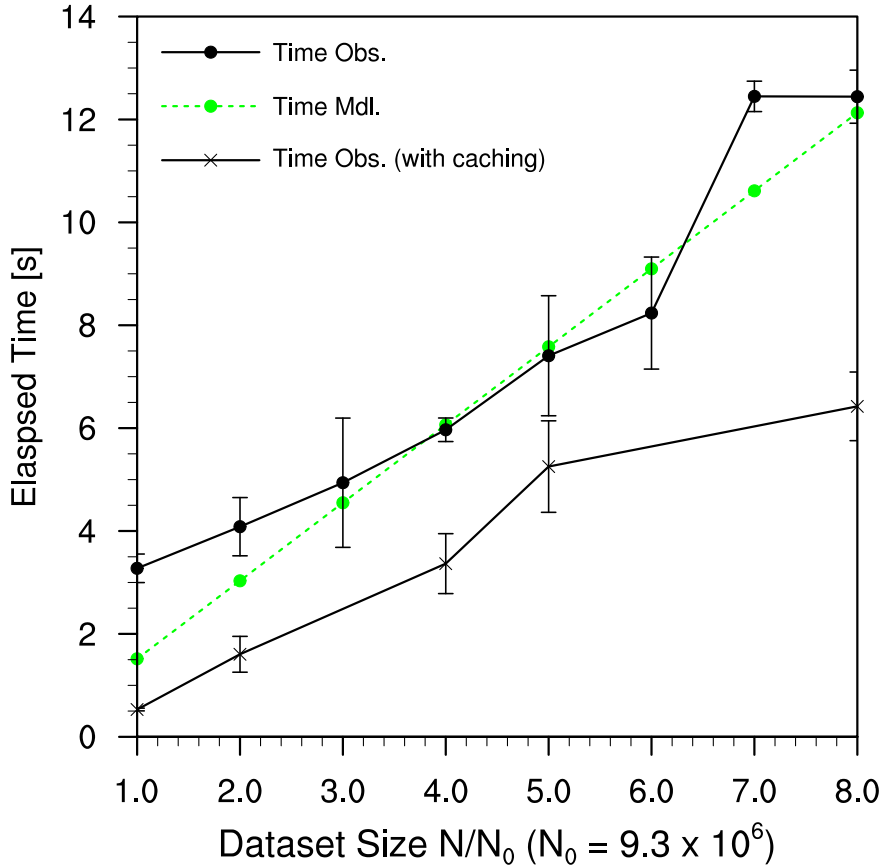
ble 3), than the same amount of data stored in variables with greater rank. The shaded areas shows the integer operations required for datasets with rank ranging from  $2 \leq R \leq 5$ . Floating point operations are very nearly rank-independent (19).

The computational model (19) predicts about 10% more integer operations than observed (Figure 2). This bias scales with  $N$  and is consistent with processor and compiler efficiencies specific to unweighted averages.

Computing weighted averages of the Satellite dataset increases the integer and floating point operations by about 250% and 300%, respectively (20). A small bias in predicted integer operations changes sign from negative to positive near  $N = 4N_0$ . A small and nearly constant offset between predicted and observed floating point operations required for weighted averages is also evident in Figure 2. The offset is consistent with a fixed cost procedure that is not included in the model.

The remaining figures are shown in terms of elapsed time  $T$  rather than operation counts. Elapsed times may be thought of as operation counts times operation rates (22), including I/O times. The elapsed time  $T$  required to perform a binary operation between datasets scales closely with the operation counts (Figure 3). The increase in  $T$  with  $N$  is roughly linear as predicted (22).

### Time to Difference Satellite Datasets

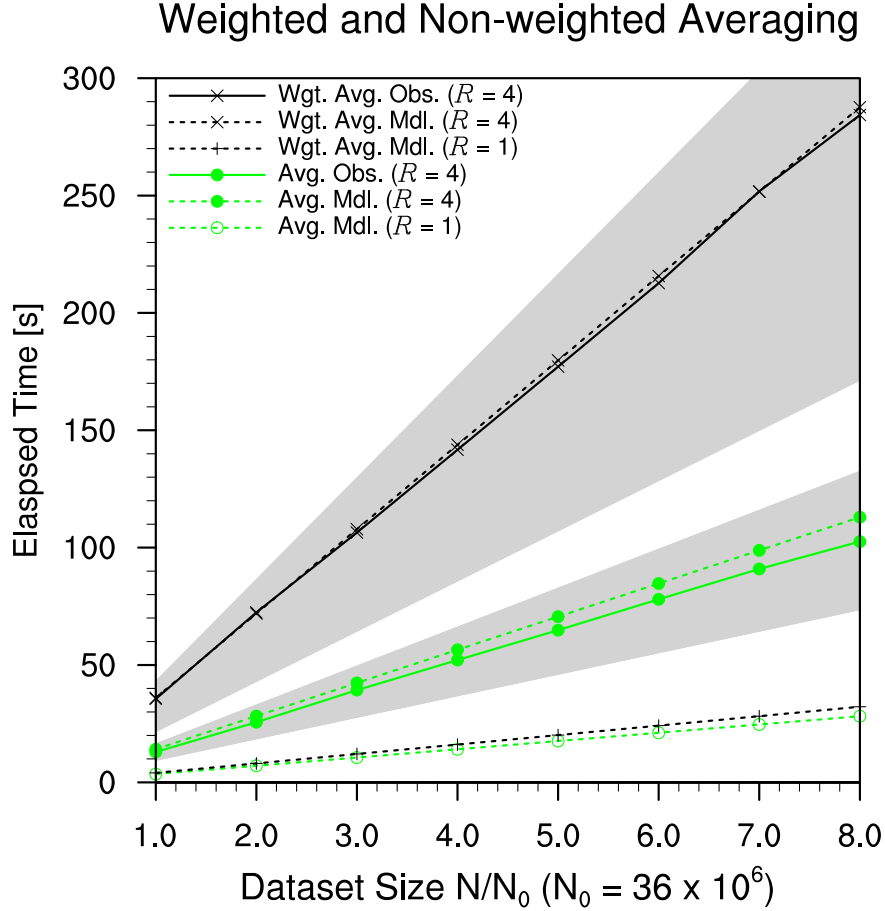


**Figure 3:** Elapsed time  $T$  [s] to difference two datasets each with  $N$  elements. Measurements (solid) are for non-cached and cached datasets, model (dashed) is for non-cached only. Vertical bars span range of four replicate measurements. Other markings as in Figure 1.

Measurements show that  $T$  is roughly 100% worse than predicted for  $N/N_0 = 1$ , scales better-than-expected for  $2 \leq N/N_0 \leq 6$ , and plateaus for  $7 \leq N/N_0 \leq 8$ . Timings are consistent for  $N/N_0 = 1, 4, 7$  but show up to about 30% variability in successive replicates for  $N/N_0 = 3, 5, 6$ . This variability is difficult to interpret systematically. Most of the variability occurs in the netCDF library during the definition and allocation of the output file, which is the same size as the input files. We think this indicates a combinations of hardware- and software- specific efficiencies that are triggered by different size I/O requests. This variability is greatly reduced when the output file is small such as after rank reduction of the input files (cf. Figure 4).

To illustrate the effect of automatic caching we repeated the subtraction operation in Figure 3 many times consecutively. The resultant caching improves throughput by roughly a factor of two. As mentioned previously, real-world data analysis rarely repeats operations on identical datasets so we used a cache-flushing protocol to obtain all other results reported here.

Weighted averaging takes about about three times longer than non-weighted averaging when no optimizations are applied (Figure 4). While disabling optimizations to measure these worst-case times, we found that the measured cost of broadcasting weights was approximately 80% larger



**Figure 4:** Observed (solid) and predicted (dashed) elapsed time to perform weighted and non-weighted averages on  $N$  element GCM-geometry ( $R = 4$ ) and unstructured ( $R = 1$ ) datasets. Grey areas indicate prediction range for rank  $R = 2-5$  datasets. Horizontal axis scaled to units of  $N_0 = 36 \times 10^6$  elements. Other markings as in Figure 1.

than predicted (12). We increased the computational model cost of broadcasting by 80% to better fit the non-WRU (Section 2.4) measurements in Figures 4 and 5.

The tripling in processing time of the structured ( $R > 1$ ) data due to weighting is approximately the ratio of  $I(\text{Un-optimized})$  (21a) to  $I(\text{average})$  (19) for rank  $R = 4$  datasets. Note that the GCM dataset (Table 3) averaged here is more complex, and about four times larger, than the satellite dataset used in Figures 1–3.

For both weighted and non-weighted averages, the distinction between processing times for structured data (Figure 4 grey areas show  $2 \leq R \leq 5$ ) and unstructured data are striking. The unstructured ( $R = 1$ ) curves represent the costs of averaging the full GCM dataset stored as  $R = 1$  arrays. These costs are assumed to equal to the costs of averaging the full structured GCM dataset with WRU+MRV optimizations (21d), i.e., with no collection or broadcasting. This is permissible because the only structure-dependent costs of averaging are due to collection and broadcasting. We find that averaging an equivalent amount of unstructured ( $R = 1$ ) data reduces operation counts and elapsed time by nearly an order of magnitude. Weighted averaging of unstructured data costs nearly the same as non-weighted averaging because collection and broadcasting are unnecessary

and the weighting itself (i.e., the additional floating point multiply) costs relatively little.

The weighted-average timings have about  $\pm 1$  second variability in four replicate experiments. This is less variability than that generated by dataset differencing (Figure 3), even though the numbers of file-layout and disk geometry factors in the GCM dataset are greater than those in the Satellite dataset. We attribute most of the reduced variability with averaging to the reduced output file requirements. Averaging all output variables to scalars makes the output file trivially small and easy for the netCDF output layer to allocate. Moreover, the fraction of time spent on I/O decreases from about 90% with dataset differencing to about 10% with weighted averaging (Figures 3 and 4, respectively), while arithmetic uses the remaining time. Hence the I/O variability is less than about 1% of elapsed time for weighted averaging, compared to about 30% variability for differencing.

Observed throughput is about 10% better than predicted (Figure 4) for the unweighted case. While attempting to reconcile the predictions and observations, we learned that the broadcasting and weighting algorithms reduce throughput disproportionately to their predicted arithmetic requirements, (12) and (20), respectively. A single empirically determined integer operation speed  $v_I$  (Table 2) does not fit both weighted and unweighted averages. Figure 4 uses one speed  $v_I$  which best fits the weighted average data and leads to the  $\sim 10\%$  over-estimate for non-weighted averages.

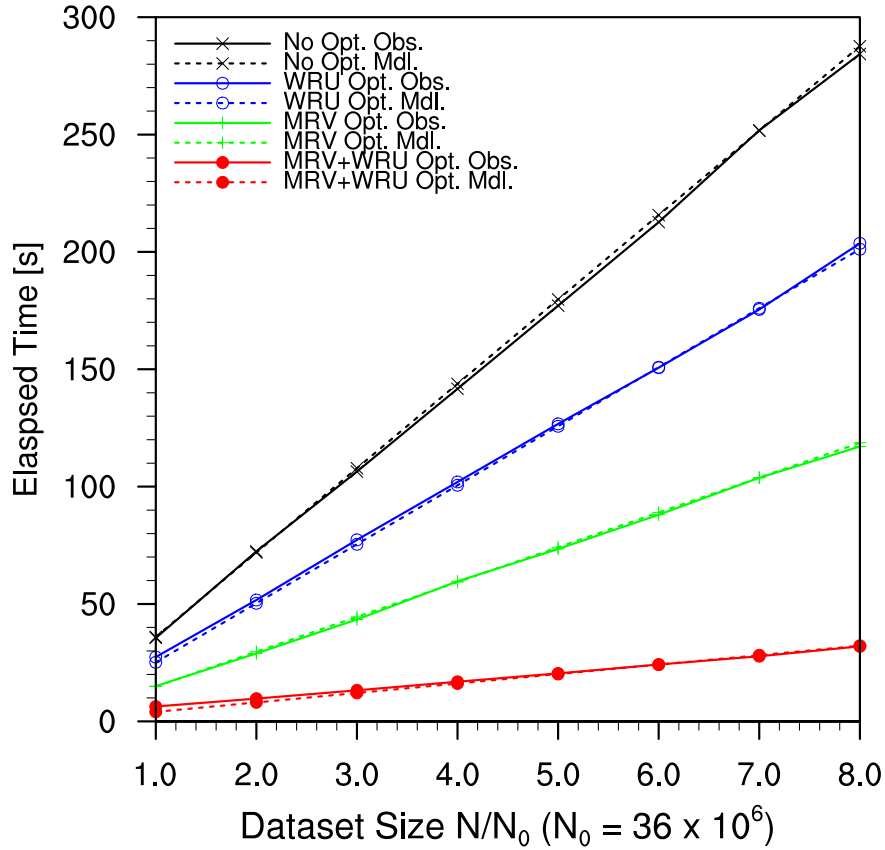
Our model of averaging algorithms suggests that averaging over all dimensions takes no longer, and may even be slightly faster, than averaging over a subset of dimensions because of the  $N_A^{-1}$ -dependence of reduction (15) and normalization (17) and the larger output size of partially averaged variables. We tested this prediction by measuring the elapsed time ( $T$ ) to average the Satellite dataset over latitude-only, longitude-only, and both dimensions: 32 s, 26 s, and 26 s, respectively. Averaging over latitude-only takes about 15% longer than completely averaging each variable over longitude-only or over both dimensions. This asymmetry in averaging behavior appears to arise from the location of longitude as the outermost or most rapidly varying (MRV) dimension. When the dimensions to be averaged are MRV, the average accumulates values from consecutive memory locations which are likely already cached by the processor. This increases memory performance significantly, and, we think, leads to the the observed 15% increase in averaging throughput for MRV dimensions. Note that this modest performance boost occurs automatically when averaging MRV dimensions thanks to hardware-implemented caching.

The MRV dimension optimization described in Section 2.2.4 is a software-implemented algorithm acceleration which boosts performance even more (Figure 5). The worst case times to perform weighted averages occur when neither the WRU nor the MRV optimizations apply or, equivalently, if the averaging software implements “brute force” techniques rather than the WRU and/or MRV optimizations.

The WRU optimization alone reduces the elapsed time to average the GCM dataset from 285 s to 200 s. This is a throughput increase of about 40%. However, not all datasets are as amenable to WRU as the GCM-dataset. For instance, averaging variables which alternated in shape rather than averaging groups of identically shaped-variables (as we did) could significantly degrade the WRU benefits.

The MRV optimization alone increases GCM dataset averaging throughput by about 140%. Figure 5 shows that MRV causes impressive gains when averaging over all dimensions (i.e.,  $R_A = R$ ). In fact, MRV optimization is nearly as effective for partial averages ( $R_A < R$ , not shown). The MRV improvement is distinct from and in addition to the acceleration which hardware-based caching provides for accessing contiguous non-strided datasets, such as MRV averages.

## Optimized Averaging of GCM Datasets



**Figure 5:** Observed (solid) and predicted (dashed) elapsed times to perform a weighted average of an  $N$  element GCM-geometry ( $R = 4$ ) dataset with and without WRU and MRV and both optimizations. Un-optimized curves same as Figure 4. Other markings as in Figure 1.

The MRV and WRU optimizations are often applicable in tandem. The MRV and WRU combination reduces the elapsed time to average the GCM dataset from 285 s to 30 s. This order-of-magnitude throughput increase agrees well with the ratio  $I(\text{Un-optimized})/I(\text{WRU+MRV})$  (21) for the GCM dataset (Table 3) which has  $R = 4$  and  $R_w = 1$ . The combination of MRV and WRU optimizations shifts the I/O time for weighted averaging of the GCM dataset from  $< 10\%$  to about 50%.

## 4 Discussion

Our computational model of arithmetic throughput agrees well with throughput measurements for simple analyses and file geometries typical of geoscience studies. The computational model requires about a half-dozen machine-specific parameters (Table 2). These results demonstrate the feasibility of constructing accurate cost-predictors for more complex and extensive data analyses. In most cases it should suffice to estimate machine-specific parameters on new hardware by performing a simple least squares fit between the measured and predicted elapsed times (22).

Algorithmic optimizations can significantly increase analysis throughput. Two optimizations were shown here. First, the most-rapidly-varying (MRV) optimization eliminates the collection step for data reduction across contiguous most-rapidly-varying dimensions. Second, weight re-use (WRU) eliminates the broadcasting step when a weight has already been broadcast to the desired shape for a previous variable.

Predicting dataflow throughput on dedicated uniprocessor workstations is a necessary first step to understanding, and eventually exploiting, distributed data reduction and analysis environments. In future work we will use this information to optimize evaluation of (9) when  $\bar{x}_j$  depends on input elements  $x_i$  distributed across a network. Some statistical procedures require multiple rank-reduction operations each of which scales like (19) or (20). For example, standard deviations require sum and sum-of-squares operations over each averaging block. Retaining and re-using the mapping information derived in the expensive collection step (13) is a promising strategy to accelerate such procedures. Such techniques are already used on-line in numerical model couplers (*Jacob et al., 2005*) which decompose, analyze (e.g., average), and re-map model data among multiple fixed grids each timestep (*Craig et al., 2005*).

Opportunities to further improve data processing efficiency exist at many levels from software to hardware. At the file level, netCDF3 imposes unnecessary byte-swapping penalty on little-endian machines which, we have shown, can dominate total execution time. The forthcoming netCDF version, netCDF4, will not perform unnecessary byte-swapping (*Rew et al., 2006*). Significant improvements to the current synchronous netCDF I/O library (*Rew and Davis, 1990*) are possible, as are client optimizations such as pre-staging data or working with partial results. MPI-IO can further reduce I/O bottlenecks (*Gropp et al., 1998*), although it requires specialized file-systems to fully exploit. Parallel netCDF (pnetCDF) (*Li et al., 2003*) currently offers an MPI-IO implementation of the netCDF3 API and file format, and netCDF4 will also support MPI-IO via its HDF5 back-end (HDF; <http://hdf.ncsa.uiuc.edu>). Exploiting these and other technologies to optimize data analysis throughput are topics of our current and future research.

## 5 Conclusions

Our computational model of arithmetic throughput on gridded geoscience datasets agrees well with throughput measurements. The datasets tested represent file geometries typical of satellite sensor data and weather/climate models. The data reduction tested represented simple arithmetic operations which resulted in I/O-bound processing (differencing files), to arithmetic-dominated processing (weighted averages).

We find that dataset structure can reduce analysis throughput ten-fold relative to same-sized unstructured datasets. Our computational model accurately predicts this reduction. Algorithmic optimizations can substantially increase throughput for more complex, arithmetic-dominated analysis such as weighted-averaging of multi-dimensional data.

This study defined the costs of local data reduction, i.e., analysis of data already stored on a local file-system. This local focus allowed us to characterize the efficiencies of storage and analysis procedures without considering the complexity that networks and client-server models add. Moreover, this study focused on uniprocessor systems and neglected data reduction efficiencies achievable with parallel environments and multi-CPU systems. Although such systems are used operationally in production of geoscience data (e.g., satellite data archival, climate models), data

reduction strategies which take advantage of distributed (or shared) memory parallelism and/or parallel filesystems are rare. More research on and implementation of efficiencies (scaling improvements) to be gained through distributed and server-side data reduction are needed. We designed our analysis framework to be readily extensible in these directions and are currently conducting such studies.

## Acknowledgments

We thank S. Jenks, D. Wang, and the three anonymous reviewers for helpful comments, and Unidata for creating and maintaining netCDF. This material is based upon work supported by the National Science Foundation under Grants ATM-0231380 and IIS-0431203. Download this manuscript from [http://dust.ess.uci.edu/ppr/ppr\\_ZeM07.pdf](http://dust.ess.uci.edu/ppr/ppr_ZeM07.pdf).

## Author Biographies

*Charlie Zender*, an atmospheric physicist, is Associate Professor of Earth System Science at the University of California, Irvine. He specializes in microphysics including radiative transfer, aerosol-surface interactions, and the atmospheric solar energy budget. He developed the mineral Dust Entrainment And Deposition (DEAD) model, and is co-developer of the SNow-ICe-Aerosol Radiation (SNICAR) model. His interest in improving geoscience data reduction and analysis at the National Center for Atmospheric Research initiated the netCDF Operators (NCO) project in 1994. He received his Ph.D. in Astrophysical, Planetary, and Atmospheric Science in 1996 from the University of Colorado at Boulder.

*Harry Mangalam* received his PhD from UCSD in 1989 and has worked in academic, non-profit, and commercial bioinformatics since then. He currently works in Research Computing Support at UCI's Network and Computing Services, specializing in bioinformatics, especially string and pattern matching, and visualization.

## Bibliography

- Browne, S., J. Dongarra, N. Garner, G. Ho, and P. Mucci (2000), A portable programming interface for performance evaluation on modern processors, *Int. J. High Perform. Comput. Appl.*, 14(3), 189–204. [2.7](#)
- Chen, L., and G. Agrawal (2004), Resource allocation in a middleware for streaming data, in *Proceedings of the 2nd Workshop on Middleware for Grid Computing*, pp. 5–10, ACM Press, New York, NY, USA, doi:10.1145/1028493.1028494. [1](#)
- Collins, W. D., et al. (2006), The formulation and atmospheric simulation of the Community Atmosphere Model: CAM3, *J. Climate*, 19(11), 2144–2161, doi:10.1175/JCLI3760.1. [2.6](#)
- Cornillon, P., J. Gallagher, and T. Sgouros (2003), OPeNDAP: Accessing data in a distributed heterogeneous environment, *Data Science Journal*, 2, 164–174. [1](#)

- Craig, A. P., R. Jacob, B. Kauffman, T. Bettge, J. Larson, E. Ong, C. Ding, and Y. He (2005), CPL6: The new extensible, high performance parallel coupler for the Community Climate System Model, *Int. J. High Perform. Comput. Appl.*, 19(3), 309–327, doi:10.1177/1094342005056117. 4
- Cubasch, U., and G. Meehl (2001), Projections of future climate change, in *Climate Change 2001: The Scientific Basis. Contribution of Working Group I to the Third Assessment Report of the Intergovernmental Panel on Climate Change*, edited by J. T. Houghton, Y. Ding, D. J. Griggs, M. Noguer, P. J. van der Linden, X. Dai, K. Maskell, and C. A. Johnson, chap. 9, pp. 527–578, Cambridge Univ. Press, Cambridge, UK, and New York, NY, USA. 1, 2.1
- Drake, J. B., P. W. Jones, and G. R. Carr, Jr. (2005), Overview of the software design of the Community Climate System Model, *Int. J. High Perform. Comput. Appl.*, 19(3), 177–186, doi:10.1177/1094342005056094. 1, 2.6
- Fiorino, M., and D. Williams (2002), The PCMDI Climate Data Analysis Tools (CDAT)—an open system approach to the implementation of a model diagnosis infrastructure, in *Proceedings of the 18th International Conference on Interactive Information and Processing Systems for Meteorology*, p. J3.22, American Meteorological Society, AMS Press, Boston, MA, January 11–15, Seattle, WA. 1
- Foster, I., et al. (2002), The Earth System Grid II: Turning climate datasets into community resources, in *Proceedings of the 18th International Conference on Interactive Information and Processing Systems for Meteorology*, American Meteorological Society, AMS Press, Boston, MA, January 11–15, Seattle, WA. 1
- Gregory, J. (2003), The CF metadata standard, *CLIVAR Exchanges*, 8(4), 4. 2.2.4
- Gropp, W., S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir (1998), *MPI: The Complete Reference. Volume 2, The MPI-2 Extensions*, 350 pp., MIT Press, Cambridge, MA. 4
- Jacob, R., J. Larson, and E. Ong (2005),  $M \times N$  communication and parallel interpolation in Community Climate System Model version 3 using the Model Coupling Toolkit, *Int. J. High Perform. Comput. Appl.*, 19(3), 309–327, doi:10.1177/1094342005056117. 4
- Li, J., et al. (2003), Parallel netCDF: A high-performance scientific I/O interface, in *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, pp. 39–49, Association for Computing Machinery, IEEE Computer Society, Washington, DC, USA, November 15–21, Phoenix, AZ. 1, 4
- Mellor-Crummey, J., R. J. Fowler, G. Marin, and N. Tallent (2002), HPCVIEW: A tool for top-down analysis of node performance, *J. Supercomput.*, 23(1), 81–104, doi:10.1023/A:1015789220266. 2.7
- NRC (2001), *Grand Challenges in Environmental Sciences*, 96 pp., National Research Council, National Academy Press, Washington, DC. 1

- NSF (2003), *Revolutionizing Science and Engineering Through Cyber-Infrastructure*, NSF 03-2, 84 pp., National Science Foundation, Arlington, VA, D. E. Atkins, Ed. 1
- Rew, R., and G. Davis (1990), NetCDF: an interface for scientific data access, *IEEE Comput. Graph. Appl.*, 10(4), 76–82, doi:10.1109/38.56302. 2.1, 2.2, 2.2.1, 2.7, 4
- Rew, R., E. Hartnett, and J. Caron (2006), NetCDF-4: Software implementing an enhanced data model for the geosciences, in *Proceedings of the 22nd AMS Conference on Interactive Information and Processing Systems for Meteorology*, p. 6.6, American Meteorological Society, AMS Press, Boston, MA. 4
- UCAR (2005), *Establishing a Petascale Collaboratory for the Geosciences: Scientific Frontiers. A Report to the Geosciences Community*, 80 pp., University Corporation for Atmospheric Research/JOSS, Boulder, CO, Ad Hoc Committee and Technical Working Group for a Petascale Collaboratory for the Geosciences. 1
- Woolf, A., K. Haines, and C. Liu (2003), A web service model for climate data access on the grid, *Int. J. High Perform. Comput. Appl.*, 17(3), 281–295. 1
- Zender, C. S. (2006), netCDF Operators (NCO) for analysis of self-describing gridded geoscience data, *Submitted to Environ. Modell. Softw.*, available from [http://dust.ess.uci.edu/ppr/ppr\\_Zen07.pdf](http://dust.ess.uci.edu/ppr/ppr_Zen07.pdf). 1, 2.2, 2.4, 2.7, 3

**Table 4: Notation**

Sym.	Description	Units
$A$	Averaging dimension size	#
$D$	Dimension size	#
$\bar{D}$	Geometric mean dimension size	#
$F$	Floating point operations	#
$I$	Total integer operations	#
$I_A$	Integer arithmetic operations	#
$i$	Input hyperslab element index	ordinal
$k$	Dimension index	ordinal
$j$	Output hyperslab element index	ordinal
$M_s$	System memory operations	#
$M_u$	User memory operations	#
$m$	Mask flag	boolean
$N$	Input hyperslab size	#
$N_0$	Hyperslab scaling size	#
$N_A$	Averaging block size	#
$N_W$	Output hyperslab size	#
$N_R$	Input hyperslab(s) total size	#
$N_w$	Weight size	#
$R$	Rank (number of dimensions)	#
$R_A$	Rank of averaging space	#
$R_B$	Rank of variable to broadcast	#
$\mathbf{S}$	Shape vector	vector
$\mathbf{S}_A$	Shape of averaging space	vector
$T$	Operation time	s
$T_{IO}$	I/O time	s
$T_R$	Read time	s
$T_W$	Write time	s
$v$	Operation speed	# s <sup>-1</sup>
$v_R$	Read speed	# s <sup>-1</sup>
$v_W$	Write speed	# s <sup>-1</sup>
$W$	Float size	byte # <sup>-1</sup>
$w$	Weight	real
$x$	Variable value	real
$\mu$	Missing value flag	boolean