Mirror-Image Round Robin Spatial Data Partitioning : a Case Study With Parallel SEUS

A. A. El Haddi[1,2] S. Shekhar[2] S. Carroll[3]

August 13, 1996

Abstract

The National Weather Service interpolates snow conditions over numerous hydrologic basins to obtain snow water equivalent estimates and associated errors for gridded fields with 30 arc second resolution. Solving problems of this scale involves an enormous number of computations and data input and output operations. Using sequential implementation to obtain gridded estimates may require execution times ranging from hours to days. Thus it becomes infeasible to solve large problems of national scale interactively or in a time frame suitable for near real time forecasts and flood warnings.

In this paper we discuss some techniques used to speed-up the computation by partitioning physical space into sub-domains. Spatial operations relating to each sub-domain are executed over a distributed heterogeneous memory architecture. Results from various strategies for distributing computations are discussed. A performance analysis is presented for the parallel implementation of snow estimation and updating system of the National Weather Service. A new Mirror-Image Round-Robin(MIRR) data partition technique is introduced. Many of these techniques have applications in developing high performance and distributed geographic information systems(DGIS).

[1]National Remote Sensing Center, Office of Hydrology, NWS, 1735 Lake Drive West, Chanhassen, MN 55317 elhaddi@nws.gov http://www.nohrsc.nws.gov

[2]Department of Computer Science, University of Minnesota, 200 Union Street, Minneapolis, MN 55455 {elhaddi,shekhar}@cs.umn.edu

[3]Department of Economics, Arizona State University, TEMPE, AZ 85287-3806 scarroll@nws.gov

1 Introduction

You are watching your favorite TV program when an unexpected voice interrupts with the following message "The National Weather Service(NWS) has issued a severe flood warning for Fort Wayne. A crest of 9.50 feet above flood stage for the Maumee River at Anthony Boulevard is expected. The Red Cross, Civil Defense, Lutheran Services and the Salvation Army are preparing for major ..., please go to the nearest ...". This could have been a real scenario, according to a study of flood damages at Fort Wayne, Indiana [CM85].

NWS maintains a set of applications to estimate snow–water equivalent (SWE) that are used to issue water forecasts and flood warnings for the country. SWE is used by NWS hydrologists to quantify the stream flow, to forecast water supplies for the United States and to manage this resource for various competing needs (e.g., domestic use, irrigation, hydro-power, etc). Government and state agencies rely on these forecasts to prepare for flood disasters and issue early flood warnings.

NWS applications use multi–source data to interpolate snow–water equivalent over large areas [HRA95, Car95]. These data are obtained by satellites, low flying aircraft, ground based sensors and sampling crews. A spatial model is used to obtain a gridded SWE over larger areas and to obtain areal snow–water equivalent estimates over a river basin. Each areal estimate has an associated uncertainty, expressed in terms of its associated mean-squared prediction error. The accuracy with which areal snow–water equivalent is estimated is critical since lives are involved, economic losses due to floods are enormous and the need to manage water resources for the competing needs of irrigation, domestic use and hydro-power are crucial. It is also imperative that the snow–water equivalent estimation is done accurately in near real time. The National Operational Hydrologic Remote Sensing Center (NOHRSC) maintains a

Snow Estimation and Updating System (SEUS) that is used to compute gridded SWE. SEUS is a software system developed by NWS to operate on the Geographical Resources Analysis Support System (GRASS). Readers can find more details on SEUS and snow estimation problems in [CDCC95, Day90, MSH$^+$93, MHH95, FSS$^+$95].

In this paper we will focus on the spatial computational aspects of SEUS and propose a new mirror-image round-robin data partition technique and a next generation Parallel SEUS (PSEUS). Section 2 describes the application domain and the data types used by SEUS and the spatial interpolation methodology. Section 3 describes the computational problems associated with gridded snow estimation using SEUS. Section 4 describes data partitioning, a parallel version of SEUS (PSEUS). Section 5 presents an analytic and experimental evaluation. Section 6 is a summary and discussion of current and future research.

2 Application Domain: Snow Estimation

The NWS uses a spatial prediction model to derive gridded snow–water equivalents over many hydrologic basins in the United States. The model relies on spatial correlation among the data and geostatistical techniques to estimate SWE where no observed data exist. A detailed review of the SEUS methodology can be found in [Day90]. The gridded snow–water equivalent is used as input to a snow ablation and accumulation model that uses observed temperature and precipitation to simulate snow cover conditions [And73]. [MSH$^+$93] and [CDCC95] describe the use of ground and airborne data to compute gridded estimates of snow–water equivalent and associated standard deviates.

2.1 Data types

From a modeling perspective, the data used to interpolate gridded snow–water equivalent fall into two categories: point and line data. From a computational point of view however, the data can be classified into three types: areal, point and line data.

2.1.1 Areal Data

The NWS divides the conterminous US into 12 hydrologic regions served by NWS River Forecast Centers (RFC). Each RFC region is sub–divided into as many as 800 hydrological basins.

Each basin in this study has a certain number of ground data collection points and airborne snow survey flight lines. Figure 1 depicts the relationship between all data types. The Colorado Basin RFC, for example, has over 200 NWS forecast basins. We will use notation $R_i (i = 0, ..., r-1)$ to represent RFCs, where r is the total number of regions. Basins will be represented using $A_k (k = 0, ..., nb - 1)$, where nb is the total number of basins in region i.

2.1.2 Line Data

Line data are SWE observations obtained from a network of over 1800 flight lines. The NOHRSC uses low–flying aircraft (operated by NOAA's commissioned Officer Corps) to measure natural terrestrial gamma radiation before the snow season and during the snow season. The technique is based on $^{40}K,^{238}U,^{208}Tl$ isotopes radiation attenuation due to the water mass in the snow cover [Fri82, PCV80, CV80, CVG85]. Each flight line is approximately 16 km long, 300 m wide. Each flight line is subdivided into one or more segments.

The technique used to estimate SWE for each basin A_k uses flight lines $F_l (l =$

$0, ..., nfl - 1)$, where nfl is the total number of flight lines that are within the basin itself or within adjacent basins.

2.1.3 Point Data

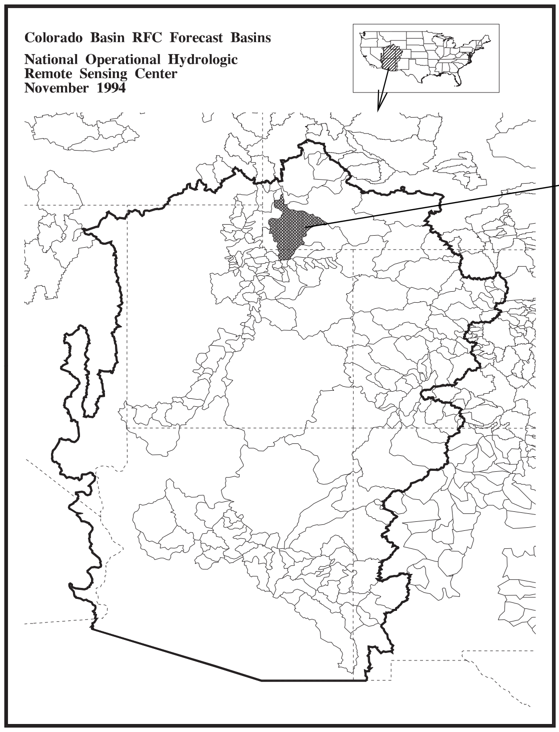
Point data are SWE observations obtained from a ground-based network composed of snow course and automated snow data telemetry (SNOTEL) sites. [MSH+93] reported that the Natural Resources Conservation Service (NRCS) collects these ground data at over 2000 locations across the west. Snow course data are obtained by sampling snow and measuring its water content. We will use $G_m(m = 0, ..., ng - 1)$ to denote the ground points in each basin A_k, with ng being the total number of ground points used in a basin's SWE computation. Each basin A_k is divided into a grid of points $GR_s(s = 0, ..., npt - 1)$ where npt is the total number of points in the grid. These are the points for which snow–water equivalent is to be computed. They will be referred to as grid points. The number of grid points in each basin is proportional to the basin size.

2.2 SWE Interpolation

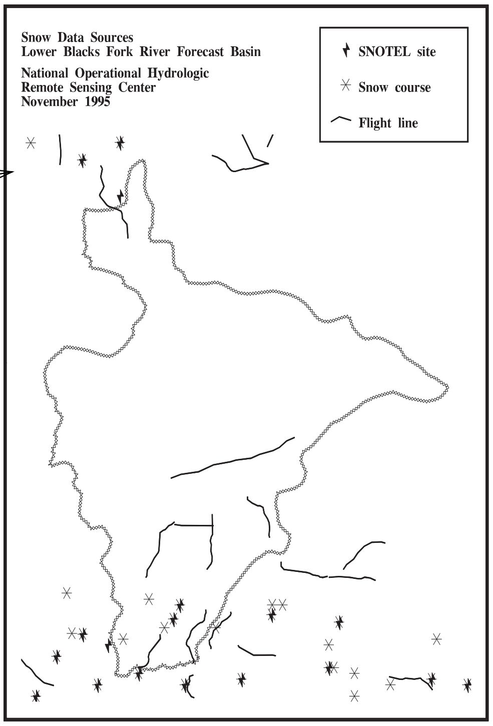
Snow–water equivalent is estimated using simple kriging. [Cre91] describes ordinary kriging, or optimal prediction, as making inferences on unobserved values of a random process $\mathbf{Z}(\mathbf{s_i})$. These values are modeled using

$$\mathbf{Z}(\mathbf{s_i}) = (\mathbf{Y}(\mathbf{s_i}) - \mu(\mathbf{s_i}))/\sigma(\mathbf{s_i}), \tag{1}$$

where $\mathbf{Y}(\mathbf{s_i})$ is a non-standardized SWE value, $\mathbf{Z}(\mathbf{s_i})$ is a standardized SWE for both ground point and flight data. $\mu(\mathbf{s_i})$ and $\sigma(\mathbf{s_i})$ are the mean and standard deviation of SWE at location $\mathbf{s_i}$.


Colorado Basin RFC Forecast Basins
National Operational Hydrologic
Remote Sensing Center
November 1994

(a)



(b)

Figure 1: **US map and relationship between data types (a) Basins within one RFC (b) Ground and Flight line data (Courtesy Don Anderson, NWS, NOHRSC).**

Ground and airborne data are used to compute the best linear predictor of $\mathbf{Z}(\mathbf{s_u})$:

$$\hat{\mathbf{Z}}(\mathbf{s_u}) = \sum_{\mathbf{i=0}}^{\mathbf{npt-1}} \lambda_{\mathbf{i}} \mathbf{Z}(\mathbf{s_i}) \tag{2}$$

where λ_i are the simple kriging coefficient estimates. The vector λ is computed as follows:

$$\lambda = \Sigma^{-1} \mathbf{C}_{\mathbf{s_u}}, \tag{3}$$

where

$$C_{\mathbf{s_u}} = (\text{cov}(\mathbf{Z}(\mathbf{s_u}), \mathbf{Z}(\mathbf{s_0})), \ldots, \mathbf{cov}(\mathbf{Z}(\mathbf{s_u}), \mathbf{Z}(\mathbf{s_{npt-1}})) \tag{4}$$

and Σ is the $n \times n$ matrix whose $(i,j)^{th}$ element is $\mathrm{cov}(\mathbf{Z}(\mathbf{s_i}), \mathbf{Z}(\mathbf{s_j}))$. The covariances are modeled using a distance weighted function.

For a basin A_k whose grid points are $GR_s, s = 0, ..., npt-1$, the mean squared prediction error of the basin areal SWE estimate is derived by

$$\hat{\sigma}^2_{Y_s k} = \frac{\sum_{i=0}^{npt-1} \sum_{j=0}^{npt-1} \hat{\sigma}(\mathbf{s_i})\hat{\sigma}(\mathbf{s_j})(\mathbf{cov}(\mathbf{Z}(\mathbf{s_i}), \mathbf{Z}(\mathbf{s_j})) - \mathbf{C_{s_i}} \sum^{\mathbf{-1}} \mathbf{C_{s_j}})}{(npt)^2}, \tag{5}$$

where $\hat{\sigma}(\mathbf{s_k})$ is the estimate of the standard deviation for a grid point $\mathbf{s_k}$. More details about the interpolation can be found in [CDCC95] and [Cre91].

3 Serial Implementation of SEUS: SSEUS

The computation of the estimated SWE at each grid point can be broken down into seven steps, S1 through S7, shown in figure 2. From these computations we also obtain the total basin SWE and associated uncertainty for each basin A_k. These steps are representative of serial implementation such as serial SEUS(SSEUS).

To determine the run time of the SEUS model as the characteristics of its instances change, we have used profiling, analysis, and measurement. Table 1 shows the number of iterations for each of the modules S1 through S7. We have analyzed the code and expressed the worst case number of iterations as a function of the number of ground points (ng), the total number of flight lines (nf), the total number of segments per flight line (ns_i), and the total number of grid points (npt) for each basin A_i.

The time complexity shows the run time as a function of the number of ground points(ng), the number of flight lines (nf), the number of segments per flight line (ns), and the number of grid points (npt) for which snow–water equivalent is to be computed. In our set–up, npt is always very large compared to nf, ns_i and ng.

```
S0: do S1, S2, S3, S4, S5, S6, S7 where:
S1: Input
    [S1.1] input ground points data.
    [S1.2] input flight line data.
    [S1.3] input grid points data.
S2: Standardize Means
    [S2.1] standardize ground point means.
    [S2.2] standardize flight line means.
    [S2.3] standardize grid point means.
S3: Compute covariance Matrix ($\Sigma$)
    [S3.1] compute Cov(Flight lines , Flight lines).
    [S3.2] compute Cov(Flight lines , Ground points).
    [S3.3] compute Cov(Ground points , Ground points).
S4: Compute inverse of covariance matrix($\Sigma^{-}1$)
S5: Compute Covariance Matrix with Grid points.
    [S5.1] Compute Cov(Flight Lines, Grid points).
    [S5.2] Compute Cov(Ground points, Grid points).
S6: Compute Gridded and total SWE estimates.
S7: Compute Total standard deviation
```

Figure 2: **Pseudo algorithm to compute a single Basin (A_i) SWE and total variance.**

For example, the small Animas river basin used by [CDCC95], contained 2681 grid points, 13 ground points and 4 flight lines. The numbers of segments per flight line were 47, 62, 65 and 61 for flight lines 1, 2, 3 and 4 respectively. The number of grid points is proportional to the basin surface size. The relation $npt \gg ns_i \gg ng \gg nf$,

Table 1: **Serial SEUS worst and average time complexities for each module. Worst case is expressed as a function of actual variable ranges. Average case is expressed as a function of n which is a combination of all variable ranges for a given basin A_k or a region R_i, $n \equiv min(npt, ns_i, nf, ng)$.**

Module	Worst case	Average (n)	Average (npt)
S1	$g(\sum_{i=1}^{nf} ns_i + ng + npt)$	$O(n^2)$	$O(npt)$
S2	$g(\sum_{i=1}^{nf} ns_i + ng + npt)$	$O(n^2)$	$O(npt)$
S3	$g(\frac{1}{2}\sum_{i=1}^{nf} ns_i^{\;2} + ng\sum_{i=1}^{nf} + ng^2)$	$O(n^4)$	$O(1)$
S4	$g(\sum_{i=1}^{nf} ns_i + ng + npt)$	$O(n^2)$	$O(npt)$
S5	$g(npt\sum_{i=1}^{nf} ns_i + (ng)(npt))$	$O(n^3)$	$O(npt)$
S6	$g(npt(nf + ng))$	$O(n^2)$	$O(npt)$
S7	$g(\frac{1}{2}((npt)^2(ng + nf)^5)$	$O(n^7)$	$O(npt^2)$

holds for all basins having flight lines. Table 1 shows the time complexity only for a single basin in one RFC. Using the same algorithm for multiple basins and multiple regions, the time complexity becomes an $O(npt^3)$ problem.

Serial SEUS executable was profiled using gprof on an HP 735-99Mhz running HP–UX 9.05. Execution profiles enabled us to see the number of calls to each module and its descendant functions. It also showed an approximation of the execution time for each module and descendants. A partial listing of the results where nf and ng were held constant is shown in table 2. The results suggest that computing the total variance represents the largest portion of the execution time. Our analysis of the time complexity concurs with the results of profiling.

Examining the function that computes the total standard deviates (S7 is implemented as **compute_area_std_deviate()** as shown in figure 3), we found that we have a time complexity largely dependent on the total number of grid points. Our

Table 2: **Results of gprof on the serial version of SEUS for small grid sizes (times are in seconds). Each value is a mean of 20 repetitions. A 300 second sleep time follows each run to take into account different system loads.**

	npt=100		**npt=200**		**npt=1000**		**npt=2681**	
Module	**T(sec)**	**%Total**	**T(sec)**	**%Total**	**T(sec)**	**%Total**	**T(sec)**	**%Total**
S1	0.186	16.87	0.189	6.23	0.203	0.33	0.200	0.05
S2	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00
S3	0.186	16.87	0.192	06.32	0.185	0.30	0.179	0.04
S4	0.003	0.27	0.004	0.12	0.002	0.00	0.002	0.00
S5	0.121	10.93	0.231	7.59	1.166	1.87	3.160	0.72
S6	0.011	00.95	0.020	0.66	0.123	0.20	0.319	0.07
S7	0.590	53.47	2.396	78.92	60.705	97.31	436.171	99.12
Total	1.102	100.00	3.036	100.00	62.385	100.00	440.039	100.00

experimental data show that this is actually an $O(npt^2)$ problem, since ng and nf are very small compared to npt. We can take advantage of the symmetry and loop only on $\frac{1}{2}npt^2$. Also Since Σ^{-1} is a constant for any combination i, j, we can pre-compute $C_j\Sigma^{-1}$ or $C_i\Sigma^{-1}$ before starting to loop on i or j.

4 Parallel implementation of SEUS: PSEUS

In section 3, we have shown that the module S7, which is implemented by the **compute_area_std_deviate()** function, is the slowest of all modules. This function computed $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ for each basin. The function was slightly modified to run on a network of workstations, using a master/slave model.

This model is used to execute **compute_area_std_deviate()** in a distributed

```
// Compute $\hat{\sigma} = sumsig$ .
double **compute_area_std_deviate**(double c, double *plat, double *plon,
        double *sig, double **sigmainv, double *$C_i$, double *$C_j$)
{
double sumsig,rho,dist;
int i,j;
sumsig=0
    for ( j=0; j < npt; j++ ) {
        // mul_mat_vec() is a $matrix \times vector$ routine.
        lambda = **mul_mat_vec**(sigmainv,Cj);          // $\lambda = C_j \sum^{-1}$
        for ( i= j+1; i < npt ; i++ ) {
            // Compute Distance between two points i with a latitude plat[i]
            // and a longitude plon[i] and a point j with latitude plat[j]
            // and a longitude plon[j].
            dist = **distance**( plat[j],plon[j], plat[i]; plon[i]);
           // Compute the modeled $cov(Z(s_i), Z(s_j))$
            rho = c*exp(d*dist);
            if ( rho == c ) rho=1;
            // mul_vec_vec() is a $vector \times vector$ routine.
            sumsig = sumsig + sig(i)*sig(j)*(rho - **mul_vec_vec**(lambda,$C_i$));
        }
    }
    sumsig = 2*sumsig/n*n;
return sumsig;
}
```

Figure 3: **Serial Version of stub computing total variance for a basin.**

scheme on a network of workstations. In this model, one host was designated as the master, while the rest of the hosts were designated as slaves executing tasks on behalf of the master. Figure 5 depicts the data flow among workstations. The algorithm used by the master consists of the serial modules S1, S2, S3, S4, S5, S6, S8, S9. S1 through S6 are as explained in the serial implementation of gridded SWE. S8 and S9 are shown in figure 4.

S8: Send the data to all hosts

[S8.1] Configure the virtual machine

[S8.2] Start all slaves with **compute_area_std_deviate()**

[S8.3] Broadcast all data required to compute $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$

S9: Receive partial results from each host

[S9.1] Wait until all results are in

[S9.2] Reassemble $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ from all hosts

Figure 4: **Master modules involved in parallelism.**

Modules S1 through S6 output is used as input for module S7. Once the master is done executing modules S1 through S6, it farms out S7 to the slaves. Each individual slave determines the physical domain that will constitute its computation domain and runs S7. The master and the slaves constitute a virtual machine. At any one time, all slaves run exactly the same copy of the code (S7). This task has an identifier that we will use interchangeably with processor identifier. The task identifier for processor i will be represented using P_i.

Our goal is to compute $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ as shown in section 3. We will decompose $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ to obtain equation(6)

$$\hat{\sigma}^2_{Y_{\mathbf{s_k}}} = \frac{2 \sum_{P_i=0}^{P-1} \hat{\Psi}(P_i)}{npt^2}. \tag{6}$$

$\hat{\Psi}(P_i)$ is the non weighted portion of $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ computed by processor P_i and P is the total number of processors used to compute $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$. $\hat{\Psi}(P_i)$ can be further refined to be expressed in terms of the domain over which it is computed. The grid is partitioned row-wise to compute $\hat{\Psi}(P_i) = \sum_{k=row_i}^{row_n} \sum_{j=k+1}^{npt-1} \hat{\sigma}(\mathbf{s_k})\hat{\sigma}(\mathbf{s_j})(\ \mathrm{cov}(\mathbf{Z}(\mathbf{s_i}), \mathbf{Z}(\mathbf{s_j})) - \mathbf{C_{s_k}} \sum^{-\mathbf{1}} \mathbf{C_{s_j}})$

Rows row_i and row_n are the lower and upper bounds of the row values from the grid. These are not necessarily consecutive as we will show.

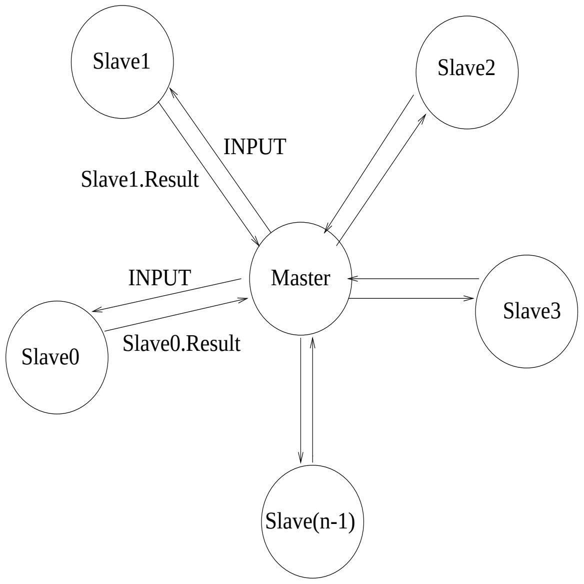


Figure 5: **Master/Slave PSEUS model and data flow.**

Once each processor receives its data from the master, it computes $\hat{\Psi}(P_i)$ only over the portion of the grid which constitutes its partition domain. Three strategies were used to partition data among processors. These strategies are:(1) Contiguous Row Blocking(CRB), (2) Round–Robin(RR) and (3) Mirror–Image Round–Robin(MIRR). The reader can find a survey of other partitioning techniques for GIS data in [SL95, SRT$^+$95, ISS86, KGGK94]. In all of our strategies, the basin is represented as an $npt \times npt$ grid space.

```
// Computes bounds for consecutive rows assigned to a processor.
getrows1(int $P_i$, int npt, int P, int *lower_bound, int *upper_bound )
{
     // compute upper and lower bound range of rows
     // allocated to each processor $P_i$
     if ( npt > 0 && $P_i$ < P ) {
        *lower_bound = $P_i$*(npt/P) ;
        if ( $P_i$ < P - 1 )
              *upper_bound = *lower_bound + npt/P -1;
        else *upper_bound = npt - 1; // last processor gets the rest
     }
}
```

Figure 6: **Contiguous Row Blocking partition strategy: allocates equal rows to all processors.**

4.1 CRB

In CRB strategy , we attempt to allocate an equal number of consecutive rows to each processor using the stub of figure 6. Using this scheme, each processor $P_i\{P_i = 0, ..., P-1\}$ is allocated $(npt \div P)$ rows (where $\div$ is the integer division operator). Processor, P_{P-1}, is allocated an additional $npt - (P-1)(npt \div P)$ rows. Figure 7(a) illustrates this partition method. In this strategy, each processor P_i is allocated rows $R_j, R_j \in [P_i(npt \div P), (npt \div P)(P_i + 1) - 1]$. The number of cells allocated to processor P_i is given by equation (7):

$$P_i c = \frac{((npt \div P)(2npt - (2P_i + 1)(npt \div P) - 1))}{2}. \tag{7}$$

The total number of cells processed by the virtual machine is given by equation

(8).

$$T_c = \sum_{i=0}^{P-1} P_i c = \frac{npt(npt-1)}{2}. \tag{8}$$

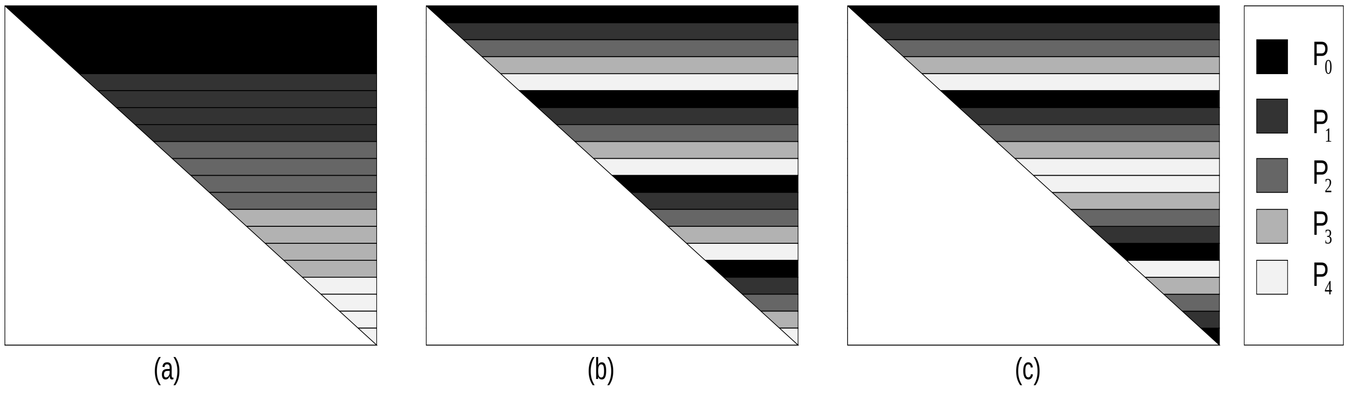


Figure 7: **Partition of a 20 x 20 grid among 5 processors P_0 through P_4(a) Contiguous Row Blocking (b) Round–Robin (c) Mirror–Image Round–Robin.**

We use a grid size of 20 $npt \times 20$ npt to illustrate this partition strategy. We obtain the results shown in table 3. The results are also expressed as a proportion of the total number of cells processed ($p = \frac{P_i c}{T_c} \times 100$).

Table 3: **Total number of cells and proportion allocated to each slave**($npt = 20, P = 5, T_c = 190$).

	CRB strategy		**RR strategy**		**MIRR strategy**	
Processor	**# of Cells**	**% Total**	**# of Cells**	**% Total**	**# of Cells**	**% Total**
$\mathbf{P_0}$	70	36.84	46	24.21	38	20.0
$\mathbf{P_1}$	54	28.42	42	22.11	38	20.0
$\mathbf{P_2}$	38	20.00	38	20.00	38	20.0
$\mathbf{P_3}$	22	11.58	34	17.89	38	20.0
$\mathbf{P_4}$	6	3.16	30	15.79	38	20.0

We note that this allocation scheme leads to a load imbalance where processors do not get equal work. Processors with a low identifier have a high number of cells allocated to them. In this example, when the virtual machine is configured with 5 hosts, P_0 processes 37 percent of the total cells while P_2 processes only 3 percent. Thus a few processors do most of the computations while the others spend most of the time waiting for the other processors with a larger cell allocation to finish computing their portion $\hat{\Psi}(P_i)$ of $\hat{\sigma}^2_{\mathbf{Y_{s_k}}}$.

4.2 RR

To distribute the load among all slaves, CRB partition strategy was refined to obtain Round–Robin partition strategy. Figure 7(b) shows how the grid space is partitioned using the algorithm of figure 8. In this strategy, each processor P_i is allocated rows $P_i + j(P), j = 0, 1, \ldots, (npt \div P) - 1$. Each of the remaining $r = (npt$ **modulus** $P)$ rows are allocated to processors 0 to $r-1$. Thus, each processor P_i is allocated the total number of cells given by equation (9):

$$P_i c = d(npt - 1 - P_i) - \frac{P(d-1)d}{2} + \alpha(npt - 1 - P_i - d(P)), \tag{9}$$

where $\quad P_i = 0, \ldots, P-1$

$d = npt \div P$

$\alpha = 1, \forall P_i < r$

$\alpha = 0, \forall P_i \geq r.$

Table 3 shows an example of this allocation scheme for npt=20. We note a significant improvement in load balance. The difference in cell allocation and thus in load of all processors in the virtual machine has been reduced but not eliminated.

```
// Round--Robin Partition strategy:  Given npt grid points and
// P processors, get rows assigned for processor $P_i$.  Rows
// are stored in the array rows.
getrows2(int $P_i$,int npt, int P,int *rows)
{
     int d,r,j;
     if (P == 0 ) P = 1;
     d= (npt) / P ;
     r= (npt) % P ;
     for ( j=0; j < d; j++ ) {
       rows[j] = $P_i$ + j * P;
     }
     if ( r > 0 && $P_i$ < r)
     rows[j++] = $P_i$ + j*P ;
     rows[j]=-1;
}
```

Figure 8: **Round–Robin partition strategy.**

4.3 MIRR

We further refine RR partition strategy to achieve further load balance with Mirror–Image Round–Robin partition strategy. MIRR allocates each processor P_i one row from the top and its corresponding complement from the bottom of the grid. The stub of figure 9 shows the algorithm used for this strategy. Figure 7(c) shows the cell allocation to each processor. MIRR has effectively achieved the load balance as shown in table 3.

```
// MIRR partition strategy:  Given npt grid points and P processors,
// get rows assigned for processor $P_i$.  Rows are stored in the
// array rows.
getrows3(int $P_i$,int npt, int P,int *rows)
{
    int d,r,top,bottom, j=0, s=0, thisrow;
    if (P == 0 ) P = 1;
    d= (npt) / P ; r= (npt) % P ;
    top=$P_i$-P ; bottom=npt - 1 - $P_i$ + P;
    rows[j]=-1;
    while ( s < d ) {
      if ( s%2 == 0 ) { top += P; thisrow=top; }
      else { bottom -= P; thisrow=bottom; }
      rows[j]=thisrow;
      j++; s++;
    }
    if ( r !=0 && $P_i$ < r ) rows[j++]= top+P;
    rows[j]=-1;
}
```

Figure 9: **MIRR partition strategy.**

5 Evaluation

In this section, we will evaluate the effects of the various data partitioning strategies on the computational and communication costs for parallel SEUS. We use speedup metric (ratio of serial run times to the parallel run times) and the communication overhead as a measure of performance.

5.1 Analytic Evaluation

We will restrict our analysis only to the slowest module (S7) which is the only module that was parallelized. Let T_{com} and T_{calc} designate the communications and the computations costs respectively for module S7. The computation cost is expressed as a function of the total number of grid cells allocated to each processor P_i. Figure 3 shows the core of the function that models $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ for module S7.

5.1.1 Cost of Serial SEUS

The serial version assigns the entire upper half of the matrix to one processor. The diagonal points (i,i) are not iterated over. The total number of cells over which we iterate is the count of all cells in every row minus the diagonal cells. Thus, for serial SEUS:

$$T_{calc}(SERIAL) = \sum_{row=0}^{npt-1} (npt - 1 - row). \tag{10}$$

Using the summation for arithmetic series, equation(10) yields the cost given by(11):

$$T_{calc}(SERIAL) = \frac{1}{2}(npt^2 - npt). \tag{11}$$

The speedup metric for each of the three partition strategies is given by

$$speedup = \frac{T_{calc}(SERIAL)}{T_{calc}(PARALLEL)}. \tag{12}$$

5.1.2 Cost of PSEUS Using CRB

Using CRB data partition method as shown in figure 7, the worst load is assigned to processor P_0. In this case T_{calc} is given by equation 13:

$$\begin{aligned} T_{calc}(CRB) &= \sum_{row=0}^{\frac{npt}{P}-1} (npt - 1 - row) & (13) \\ &= \frac{(2P-1)npt^2 - P\ npt}{2P^2}. & (14) \end{aligned}$$

Using equations (11) and (14) , we obtain the speedup of equation (15) for CRB.

$$speedup(CRB) \approx \frac{P^2}{2P-1} \approx \frac{P}{2}. \tag{15}$$

5.1.3 Cost of PSEUS Using RR

In the case of RR, each processor is allocated rows $P_i + P\ j$ where $j = 0, \ldots, \frac{P}{N} - 1$. The maximum load is seen for processor $P_0 = 0$. Thus

$$\begin{aligned} T_{calc}(RR) &= \sum_{row=0}^{\frac{npt}{P}-1} (npt - 1 - P\ row) & (16) \\ &= \frac{1}{2}\frac{npt}{P}(npt - 1 + (npt - 1) - P(\frac{npt}{P} - 1)) & (17) \\ &= \frac{1}{2P}(npt^2 + (P-2)npt). & (18) \end{aligned}$$

If P does not divide npt exactly (npt **modulus P** $\neq$ **0**), P_0 is allocated an additional $P - 1$ cells:

$$T_{calc}(RR) = \frac{1}{2P}(npt^2 + (P-2)npt) + (P-1). \tag{19}$$

Applying this computation cost, we obtain the speedup for RR as given by equation (20):

for npt **modulus P = 0**

$$speedup(RR) = \frac{npt^2 - npt}{npt^2 + (P-2)npt}P. \tag{20}$$

for npt **modulus P $\neq$ 0**

$$speedup(RR) = \frac{npt^2 - npt}{npt^2 + (P-2)npt + 2P(P-1)} P. \tag{21}$$

In both cases, $speedup(RR) \approx P$ for very large npt.

5.1.4 Cost of PSEUS Using MIRR

In the case of MIRR, row allocation to each processor progresses from both the top and the bottom of the grid to the mid-point. Thus

$$T_{calc}(MIRR) = T_{calc}(bottom) + T_{calc}(top). \tag{22}$$

where $T_{calc}(bottom)$ and $T_{calc}(top)$ are equal to the number of cells assigned to P_0 from the bottom and top of the grid respectively. From the top, rows $P_0 + P\ j (j = 0, \ldots, \frac{npt}{P} - 1)$ are assigned. The rows allocated to P_0 from the bottom are $npt - 1 - P_0 - P\ j$. We derive equations (23), (**??**) and (25) for $P_0 = 0$.

- npt **modulus 2 P = 0**:

$$T_{calc}(MIRR) = \frac{npt^2 - npt}{2\ P}. \tag{23}$$

- npt **modulus 2 P $\neq$ 0** and npt **modulus P $\neq$ 0**:

$$T_{calc}(MIRR) = \frac{npt^2 - npt - 2\ P}{2\ P}. \tag{24}$$

- npt **modulus 2 P $\neq$ 0** and npt **modulus P = 0**:

$$T_{calc}(MIRR) = \frac{npt^2 + (P-1)npt - 2\ P}{2\ P}. \tag{25}$$

The speedup obtained for each case is given by equations (26), (27) and (28).

- npt **modulus 2 P = 0**:

$$Speedup = P. \tag{26}$$

- npt **modulus 2 P $\neq$ 0** and npt **modulus P $\neq$ 0**:

$$Speedup = \frac{P(npt^2 - npt)}{npt^2 - npt - 2\ P}. \tag{27}$$

- npt **modulus 2 P $\neq$ 0** and npt **modulus P = 0**:

$$Speedup = \frac{P(npt^2 - npt)}{npt^2 + (P-1)npt - 2\ P}. \tag{28}$$

In all cases, for large npt, $speedup(MIRR) \approx P$.

Our analysis shows that both MIRR and RR achieve a better load balance and thus a higher speedup when compared to CRB. MIRR outperforms RR for small even $\frac{npt}{P}$. MIRR and RR perform the same for $\frac{npt}{P} >> 1$. Table 4 summarizes the results of the computational cost and speedup resulting from using each of the data partitionning strategies with PSEUS.

Table 4: **Costs and speedup of PSEUS for each partition strategy.**

Strategy	**Cost**	**speedup** $\frac{npt}{P} >> 1$	**speedup** $\frac{npt}{P} \longrightarrow 2$
SERIAL	$\frac{1}{2}(npt^2 - npt)$	1	1
CRB	$\frac{(2P-1)npt^2 - P\ npt}{2P^2}$	$\frac{P}{2}$	$\frac{P}{2}$
RR	$\frac{1}{2P}(npt^2 + (P-2)npt)$	P	$\frac{2P}{3}$
MIRR	$\frac{npt}{2P}(npt - 1)$	P	P

5.2 Communication Overhead

In this parallel implementation of SEUS, there are communications of data only between the master and its slaves. No communications take place among the slaves.

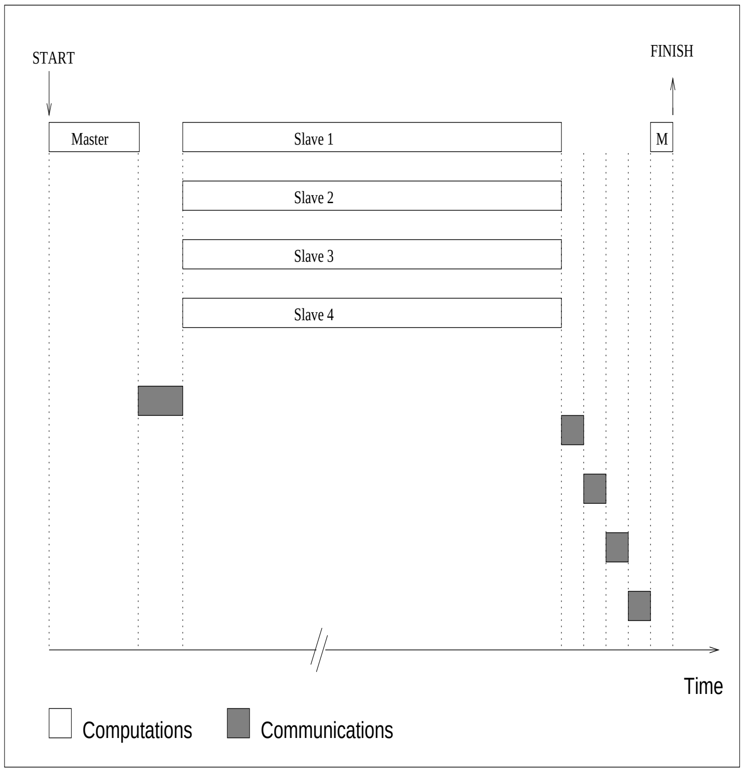


Figure 10: **Communications and computations timing for PSEUS.**

The master performs the initial computations then broadcasts the data to all slaves which in turn perform other computations and send back the results to the master. The master assembles the results into $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$. Hence the master is said to be blocking: it can not progress until all the results from all slaves have come back.
Figure 10 illustrates the timing of events in this type of cooperative processing. Each of the three partition strategies has the same basic communication cost $T_{com} = npt$. Since the computation cost is $O(npt^2)$ and the communication cost is $O(npt)$, the communication overhead is $O(\frac{1}{npt})$. Thus as npt becomes large, the overhead drops significantly.

5.3 Experimental Evaluation

5.3.1 Material and Methods

We have implemented PSEUS using the Parallel Virtual Machine Libraries version 3.3.6 [GBD+93]. PSEUS was tested on various platforms (Sun running Sun-OS or Solaris, Intel X86 running linux, IBM RS/6000 running AIX 3.25 and HP 9000 running HP-UX 9.05). PSEUS was tested through a series of 9866 runs on HP 9000/700 series connected to the NWS snow survey Ethernet based local area network. The number of grid points(npt) and the number of processors(P) were among the factors that we have varied during this study. Our measurements consisted of the execution times for various modules and the values of $\hat{\sigma}^2_{Y_{\mathbf{s_k}}}$ that were compared with a version of SEUS implemented in SAS version 9.07 by Steve Carroll [CDCC95]. We have estimated the communication cost T_{com} by subtracting the computation time of σ_k from the total round-trip time which included the communication and the the computation times. We computed the communication overhead, $\tau = \frac{T_{com}}{T_{calc}}$. We also counted the volume of network traffic that was generated and the total number of collisions that occurred during our trial. Data analysis was performed using SAS 9.07 on a SparcStation 10/54.

5.3.2 Results and Discussion

Figure 11 shows the resulting increase in speed when PSEUS is distributed among 1,2,3,4 or 5 processors for a grid size of 10000×10000. RR and MIRR strategies result in a four fold speed increase, compared to the serial version. The increase in speed resulting from partition strategy RR and MIRR is significantly higher than that of CRB partition strategy. We also note that the increase in speed is a function of the problem size and the number of processors used(figure 12).

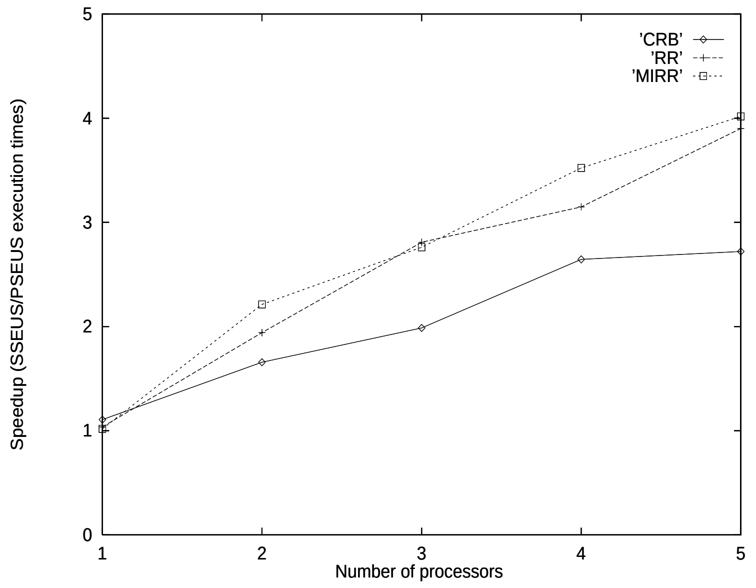


Figure 11: **Relative speedup of PSEUS using three data partition strategies. The speedup is the ratio of PSEUS/SSEUS execution times for npt=10,000.**

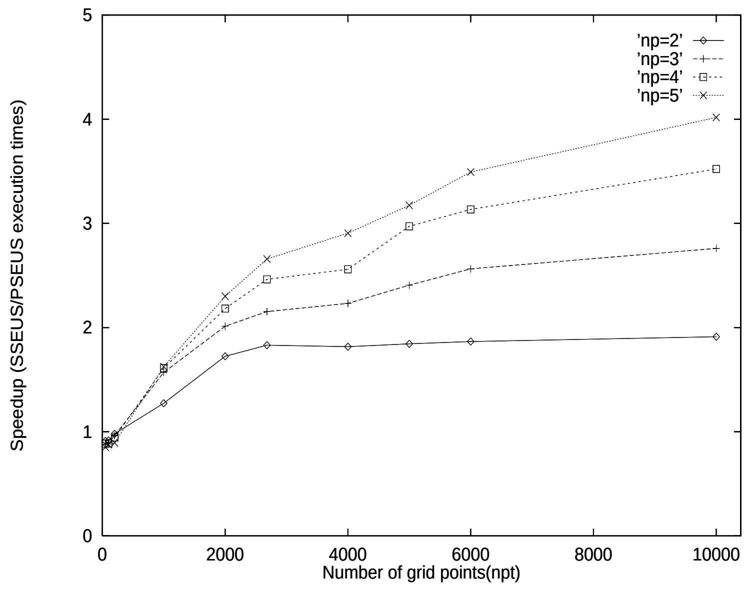


Figure 12: **Speedup as a function of problem size and the number of processors for data MIRR partition strategy.**

Both RR and MIRR strategies however, require additional storage for row bounds since rows are not allocated in consecutive order. MIRR allocates rows in an unsorted order. This may impact locality when non–associative cache is used. The algorithm could be improved further by generating the rows in sorted order without additional overhead of a sort routine. CRB partition method has the disadvantage of load imbalance. However, this method may be useful in heterogeneous architectures. CRB or a variation of RR strategy (i.e., with different stride values) can be used in combination with a weight function to allocate a larger number of cells to faster slaves and a lower number of cells to slower hosts or when executing PSEUS on networks of workstations connected via slower and busier data networks. It may also be used to dynamically change the number of cells allocated to a given slave while it is executing by farming out portions of its computational domain when CPU load exceeds a certain threshold. The communication overhead τ is shown in figure 13 for MIRR when the total number of processors $P = 5$.

We note that τ is very high for small hydrologic basins and extremely small for large basins . However the smallest basin has 2681 grid points. Thus PSEUS scales well as the problem size and the resources allocated to solve it increase.

6 Conclusion

We have shown that some of the spatial interpolations and geographic information systems analyses can be parallelized efficiently using distributed memory architectures. Various techniques from domain decomposition can be used to partition spatial data among processors to achieve higher performance of applications. A new partition technique called Mirror–Image Round–Robin (MIRR) achieves load balance. Data partitioning algorithms should be considered carefully in order to achieve acceptable

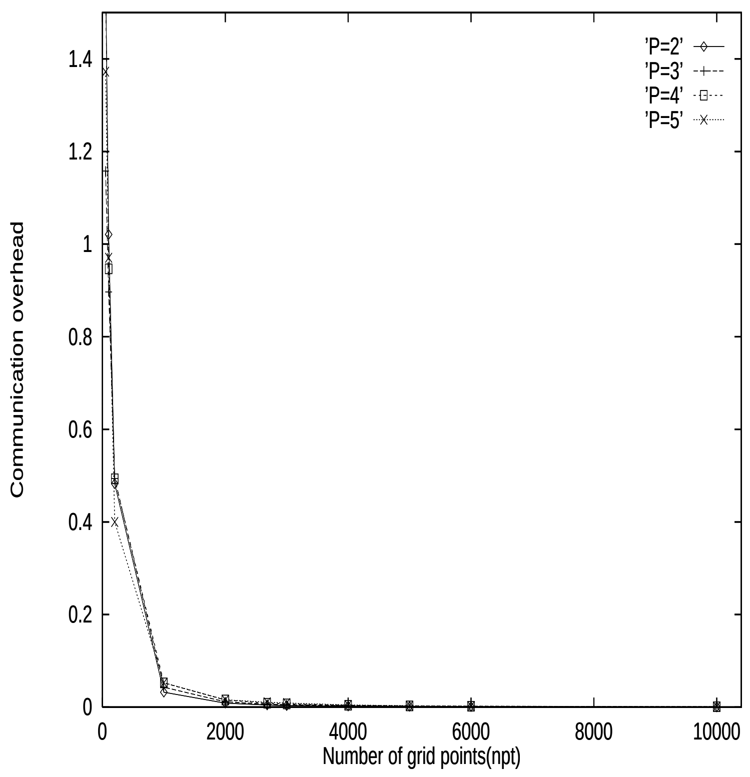


Figure 13: **Communication overhead(τ) as a function of the problem size and the number of processors.**

performance since there seems to be tradeoffs between the partition strategies. MIRR strategy may be suitable for homogeneous architectures, while CRB and RR strategies may be more suitable for heterogeneous architectures.

Our future work will focus on applying some of these techniques to spatial interpolations and neighborhood analysis in an operational environment. A variation of the same techniques used in PSEUS could be applied to some GRASS routines (e.g., to distribute data for CPU and I/O intensive operations). PSEUS is still a serial algorithm, so we will be examining ways to change the computation algorithm itself to exploit the parallelism. General purpose libraries for distributed spatial operations could ease the path towards a high performance distributed GIS architecture.

References

[And73] E. Anderson. National Weather Service River Forecast System–Snow Accumulation and Ablation Model. Technical Report HYDRO-17, NOAA, NWS, 1973.

[Car95] T. Carroll. GIS Used to Derive Operational Hydrologic Products From In Situ and Remotely Sensed Snow Data. In A. Carra and F. Guzzetti, editors, *Geographical Information Systems in Assessing Natural Hazards*, pages 335–342. Kluer Academic Publishers, The Netherlands, 1995.

[CDCC95] S. Carroll, G. Day, N. Cressie, and T. Carroll. Spatial Modeling of Snow Water Equivalent Using Airborne and Ground-Based Snow Data. *Environmetrics*, 6:127–139, 1995.

[CM85] T. Carroll and R. Marshall. Cost–Benefit Analysis of Airborne Gamma Radiation Snow Water Equivalent Measurements Made Before The February 1985 Fort Wayne Flood. In *Sixth Conference on Hydrometeorology, American Meteorological Society, Indianapolis, Indiana*, October 1985.

[Cre91] N. Cressie. *Statistics for Spatial Data* . John Wiley & Sons, Inc., 1991.

[CV80] T. Carroll and K. Vadnais. Operational Airborne Measurement of Snow Water Equivalent Using Terrestrial Gamma Radiation. In *48th Annual Western Snow Conference, Laramie, Wyoming*, April 1980.

[CVG85] T. Carroll, R. Vogel, and R. Gauthier. Operational Airborne Snow Water Equivalent and Soil Moisture Measuements Used in Water Supply and Snowmelt Flood Forecasting. In *Fifth Remote Sensing Symposium, Re-*

mote Sensing Applications For Water Resources Management, Ann Arbor, Michigan, October 1985.

[Day90] G. Day. A Methodology for Updating Conceptual Snow Model With Snow Measurments. Technical Report 43, NOAA, NWS, March 1990.

[Fri82] A. Fritzsche. The National Weather Service Gamma Snow System Physiscs and Calibration. Technical Report 43, EG&G Energy Measurements Group, NWS-8201, December 1982.

[FSS$^+$95] D. Fread, R. Shedd, G. Smith, R. Farnsworth, C. Hoffeditz, L. Wenzel, S. Wiele, J. Smith, and G. Day. Modernization in the National Weather Service River and Flood Program. *Weather and Forecasting*, 10(3):477–484, 1995.

[GBD$^+$93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. PVM 3 Users Guide & Reference Manual ORNL/TM–12 187. Technical report, May 1993.

[HRA95] R. Hartman, A. Rost, and D. Anderson. Operational Processing of Multi-Source Snow Data. In *63rd Annual Western Snow Conference, Reno, Nevada*, April 1995.

[ISS86] I. C. F. Ipsen, Y. Saad, and M. H. Schultz. Complexity of Dense-Linear-System Solution on a Multiprocessor Ring. *LINEAR ALGEBRA AND ITS APPLICATIONS*, 77:205–239, 1986.

[KGGK94] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1994.

[MHH95] A. McManamon, R. Hartman, and R. Hills. Implementation of The Snow Estimation and Updating System (SEUS) In the Clearwater River Basin, Idaho. In *63rd Western Snow Conference, Reno, Nevada*, April 1995.

[MSH+93] A. McManamon, T. Szeliga, R. Hartman, G. Day, and T. Carroll. Gridded Snow Water Equivalent Estimation Using Ground Based and Airborne Snow Data. In *Proceedings of the 61st Western Snow Conference, Quebec City*, pages 75–81, 1993.

[PCV80] E. Peck, T. Carroll, and S. Vanemark. Operational Aerial Snow Surveying in the United States. *Hydrological Sciences-Bulletin-des Sciences Hydrologiques*, 25(3):51–62, 1980.

[SL95] S. Shekhar and D. R. Liu. Partitioning Similarity Graphs: A Framework for Declustring Problems. In *IEEE Intl. conf. Data Eng.*, 1995.

[SRT+95] S. Shekhar, S. Ravada, G. Turner, D. Chubb, and V. Kumar. Load Balancing in High Performance GIS: Partitioning Polygonal Maps. In *Lecture Notes in Computer Science*. Springer Verlag, 1995.