

CAAM 520

HW2

Blair Christian

27th February 2003

For this assignment, I chose to calculate the sample mean and variance recursively at each step. The algorithm works in the following way. The premise is that if we have np processors, then the mean can be calculated like this:

$$\bar{x}_{i+np} = \frac{1}{i+np} \left\{ i\bar{x}_i + \sum_{j=i+1}^{i+np} x_j \right\}$$

All processors have the \bar{x}_i from the last iteration, and the sum is an Allreduce at each step. For the sample variance, we subtract the top statement from the bottom statement,

$$(i-1)s_i^2 = \sum_{j=1}^i (x_j - \bar{x}_i)^2 = x_1^2 + \dots + x_i^2 - i\bar{x}_i^2$$
$$(i+np-1)s_{i+np}^2 = \sum_{j=1}^{i+np} (x_j - \bar{x}_{i+np})^2 = x_1^2 + \dots + x_{i+np}^2 - (i+np)\bar{x}_{i+np}^2$$

to get:

$$s_{i+np}^2 = \frac{1}{i+np-1} \left\{ (i-1)s_i^2 + i\bar{x}_i^2 + \sum_{j=i+1}^{i+np} x_j^2 - (i+np)\bar{x}_{i+np}^2 \right\}$$

To calculate this quantity, all we need is an Allreduce on x_j^2 at each step, since each processor already has all of the other quantities. Using BLAS and Allreduce on the vectors, the loop looks something like this:

```
for (i=1;i<=reps;i++){  
    /* P0 sends out random v2's, receives v2's, and updates mean, sd */  
    if (my_rank==0){  
        /* generate (# processors) of v2 between SHIFT and SHIFT+RANGE */  
        for (j=0;j<nproc;j++){  
            v2vec[j]= SHIFT+RANGE*(rand()+0.0)/RM;
```

```

    }
  }
  MPIBcast(v2vec,MAXPROC,MPI_FLOAT,0,MPI_COMM_WORLD);
  /* Send the current random numbers to each processor */

  v2=v2vec[my_rank];
  /* generate a random v vector */
  buildm_(&m[0][0],&v1,&v2,velx,vely,&f2v1,&f2v2,f2velx,f2vely);
  buildv_(v,&v1,&v2,velx,vely,&f2v1,&f2v2,f2velx,f2vely);
  cflp_(&m[0][0],v,wv,&v1,&v2,velx,vely,&f2v1,&f2v2,f2velx,f2vely);

  /* Everybody gets the localSum all of the new v[i]'s now */
  MPI_Allreduce(v,localSum,16,MPI_FLOAT,MPI_SUM,MPI_COMM_WORLD);

  /* temp<-oldGlobalMean */
  scopy_(&n,mean,&ione,temp,&ione);

  /* MEAN<- 1/(i*nproc) [(i-1)*nproc*MEAN+localSum] (3 calls) */
  /* 1) mean<- (i-1)*np*mean */
  alpha=(i-1.0)*nproc;
  sscal_(&n,&alpha,mean,&ione);

  /* 2) mean<-mean+localsum */
  alpha=1.0;
  saxpy_(&n,&alpha,localSum,&ione,mean,&ione);

  /* 3) mean<-1/(i*np)*mean */
  alpha=1.0/(i*nproc);
  sscal_(&n,&alpha,mean,&ione);

  /* STDDEV^2<- 1/(i*np-1) [((i-1)*np-1)*STDDEV_old + (i-1)*np*xbar_old^2 +
    + sum(x_i^2) - (i*np)*xbar_new^2 (6 steps) */

  /* 1) sd<-((i-1)*np-1)*sd */
  alpha=(i-1.0)*nproc-1.0;
  sscal_(&n,&alpha,sd,&ione);

  /* 2) square old mean, sd<-((i-1)*np*xbar_old^2 +sd */
  vecsq(temp,n);
  alpha=(i-1.0)*nproc;
  saxpy_(&n,&alpha,temp,&ione,sd,&ione);

  /* 3) v<-v^2, then allreduce */
  vecsq(v,n);

  MPI_Allreduce(v,localSum,16,MPI_FLOAT,MPI_SUM,MPI_COMM_WORLD);

  /* 3) v<-xbar_new^2 */

```

```

scopy_(&n,mean,&ione,v,&ione);
vecsq(v,n);

/* 4) localSum<- -(i*np)*v +localsum */
alpha= -(1.0*i*nproc+0.0);
saxpy_(&n,&alpha,v,&ione,localSum,&ione);

/* 5) sd_new<-sd_old+temp+localSum+v */
alpha=1.0;
saxpy_(&n,&alpha,localSum,&ione,sd,&ione);

/* 6) sd<-1/(i*np-1)*sd */
alpha=1.0/(i*nproc-1.0);
sscal_ (&n,&alpha,sd,&ione);
}

```