

```

#include <stdio.h>
#include <stdlib.h>      /* Example to illustrate O( log(p) ) reduction */
#include <omp.h>

#define MAX_NUMS 2048
#define FNAME "nums"

/* ----- M A I N ----- */
int main()
{
/* Shared variables. */
    int * A ;
    int n ;
    int * totals ;

/* Private variables. */
    int chunk_size, start, ptotal ;
    int j, r, x, jend ;
    int id, nprocs, n_mod_nprocs ;
    int ix, k, np ;
    FILE * fp ;

#pragma omp parallel shared(n,A,totals) \
    private(chunk_size,start,j,jend,id,nprocs,ptotal,fp,r,x,n_mod_nprocs,ix,k,np)
    {
        nprocs = omp_get_num_threads() ;
        id = omp_get_thread_num() ;

#pragma omp master
        {
            /* The master thread does this. */
            A = (int *) malloc( MAX_NUMS * sizeof(int) ) ;
            if ( A == NULL ) {
                fprintf(stderr,"malloc failed !!\n" ) ; exit(1) ;
            }
            fp = fopen( FNAME, "r" ) ;
            if ( fp == NULL ) {
                fprintf(stderr,"unable to read file '%s'\n", FNAME ) ; exit(2) ;
            }
            n = 0 ;
            do {
                r = fscanf( fp, "%d", &x ) ;
                if ( ( r != EOF ) && ( n < MAX_NUMS ) ){
                    A[n] = x ;
                    n++ ;
                }
            } while ( r != EOF ) ;
            fclose( fp ) ;
            printf("%d numbers found.\n", n ) ;
            printf("%d procs.\n", nprocs ) ;

            totals = (int *) malloc( nprocs * sizeof(int) ) ;
            if ( totals == NULL ) {
                fprintf(stderr,"malloc failed !!\n" ) ; exit(3) ;
            }
        }
    } /* end omp master: allocate memory and read data. */

```

```

/* Every process waits until the master thread has updated A, n and totals. */
#pragma omp barrier

/* Every process computes its chunk size and its start position in A. */
n_mod_nprocs = n % nprocs ;
if ( id < (n_mod_nprocs) ) {
    chunk_size = n / nprocs + 1 ;
    start = id * chunk_size ;
}
else {
    chunk_size = n / nprocs ;
    start = (chunk_size + 1) * n_mod_nprocs +
            ( id - n_mod_nprocs ) * chunk_size ;
}
/* printf( "(%d): chunk_size = %d, start = %d\n",id,chunk_size,start ); */

/* Every thread computes a local sum. */
ptotal = 0 ;
jend = start + chunk_size ;
for ( j = start ; j < jend ; j++ ) {
    ptotal += A[j] ;
}
/* printf( "(%d): local sum = %d\n", id, ptotal ) ; */

totals[id] = ptotal ;

/* Do not enter the reduction loop until every thread has updated totals[] */
#pragma omp barrier

/* Half the threads add and continue, the other half "drop out". */
/* But, all threads must stay in the loop until it is done, */
/* otherwise, some threads will not reach the barrier and */
/* the threads that do reach the barrier will wait forever. */
np = nprocs ;
while ( np > 1 ) {
    k = np/2 + np%2 ;
    if ( id < k ) {
        ix = id + k ;
        if ( ix < np ) totals[id] += totals[ix] ;
    }
    np = k ;
}
#pragma omp barrier
}

/* Wait for every process to finish updating 'total'. */

if ( id == 0 ) {
    printf("Total = %d\n", totals[0] ) ;
}

} /* End parallel section. */

} /* End main() */

```

```
## ----- Makefile -----  
##  
CC=gcc  
CFLAGS=-fopenmp -O3  
LNK=gcc  
#  
reduce_demo: reduce_demo.o  
$(LNK) -o reduce_demo $(CFLAGS) reduce_demo.o  
#  
reduce_demo.o: reduce_demo.c  
$(CC) -c $(CFLAGS) reduce_demo.c  
#  
clean:  
/bin/rm -f reduce_demo reduce_demo.o  
###
```

```
----- Sample Session -----  
cosmos% make  
gcc -c -fopenmp -O3 reduce_demo.c  
gcc -o reduce_demo -fopenmp -O3 reduce_demo.o  
cosmos%  
cosmos% reduce_demo  
1024 numbers found.  
11 procs.  
Total = 519632
```



Yaba daba dooo ...