Optimization and Scalability
Hands-on Lab

Carlos Rosales
Carlos@TACC.UTexas.EDU

October 23th, 2012
Introduction to Parallel Computing
Setup

• Login to Lonestar:
  – `ssh username@lonestar.tacc.utexas.edu`

• Untar the lab files:
  – `cd`
  – `tar xvf ~train00/parallel_opt_lab.tar`

• Change directories and ls to see the files:
  – `cd parallel_opt_lab`
  – `ls`

• You should see both C and F90 versions of the code
A simple 2D problem

\[ f'(x,y,t) = \frac{df(x,y,t)}{dx} = \frac{f(x+\Delta x,y,t) - f(x-\Delta x,y,t)}{2\Delta x} \]

\[ f(x,y,t+1) = f(x,y,t) + \varepsilon f'(x,y,t) \]

- No particular physical process
- Structure is similar to many explicit codes
  - Calculate the derivative of \( f \)
  - Update \( f \)
  - Update neighbor boundary values
  - Start again
The Partitioning Scheme

- For clarity we will use a 1D partitioning scheme
- Lines 35-48 (C) and 38-50 (F90) define the 1D virtual topology we will use
- Periodic boundary conditions are embedded in the Cartesian topology
- This allows us to employ “left” and “right” as well defined directions for the MPI exchange
Data Exchange Optimization

• Focus on the main loop starting on line 69 (C, F90) of the code.

• There are several ways to optimize the data exchange between tasks

• Think back to the concepts presented and find at least one way to improve the overall execution time of the code

• Make any changes you need to the code to improve its current performance, but always keep a copy of the original

• Towards the end of the Lab I will explain two different ways to speed up the exchange, but give it your best shot!

• Extra points if your best code is better than mine 😊
Getting started

• Choose the C or the F90 version of the lab

• Make a personal copy that you will modify later
  
  cp ./exchange_1d.c ./exchange_opt.c
  cp ./exchange_1d.f90 ./exchange_opt.f90

• Compile the current version of the code
  
  mpicc ./exchange_1d.c -o original
  mpif90 ./exchange_1d.f90 -o original

• Start an interactive session in lonestar
  
  idev

• Run the code using 10 processors and record the timings it gives you when done
  
  ibrun -n 10 -o 0 ./original

• Now try to beat that time by modifying exchange_opt.c!
I feel kind of lost with this lab…

• Don’t panic! And keep reading…

• If you are not used to coding with MPI this lab can be a little hard.

• There are two proposed solutions already coded in the “proposed_solutions” directory.

• The next four slides explain what was done in the two proposed solutions and why.

• Feel free to simply compile the “solution” versions and compare the execution timings.

• Make sure you understand WHY the proposed solutions are faster than the original.
Optimization and Scalability

PROPOSED SOLUTION
// Send to right, receive from left
for( j = 0; j < NY; j++ ){
    sendBuf[0] = f[NX][j];
    MPI_Irecv( recvBuf, 1, MPI_DOUBLE, left, …);
    MPI_Send( sendBuf, 1, MPI_DOUBLE, right, …);
    MPI_Wait( &request, &status );
    f[0][j] = recvBuf[0];
}

// Send to left, receive from right
for( j = 0; j < NY; j++ ){
    sendBuf[0] = f[1][j];
    MPI_Irecv( recvBuf, 1, MPI_DOUBLE, right, …);
    MPI_Send( sendBuf, 1, MPI_DOUBLE, left, …);
    MPI_Wait( &request, &status );
    f[NX+1][j] = recvBuf[0];
}

- One message for each data item to exchange in each direction
- Message size is 8 Bytes

Tiny effective bandwidth !!!
Optimized Code (1)

// Send to right, receive from left
for( j = 0; j < NY; j++ ) sendBuf[j] = f[NX][j];

MPI_Irecv( recvBuf, NY, MPI_DOUBLE, left, ... );
MPI_Send( sendBuf, NY, MPI_DOUBLE, right, ... );
MPI_Wait( &request, &status );

for( j = 0; j < NY; j++ ) f[0][j] = recvBuf[j];

// Send to left, receive from right
for( j = 0; j < NY; j++ ) sendBuf[j] = f[1][j];

MPI_Irecv( recvBuf, NY, MPI_DOUBLE, right, ... );
MPI_Send( sendBuf, NY, MPI_DOUBLE, left, ... );
MPI_Wait( &request, &status );

for( j = 0; j < NY; j++ ) f[NX+1][j] = recvBuf[j];

- Pack data to be sent to the right
- Single exchange with packed data
- Unpack data received from left
- Repeat for the left to right exchange
- Message size is 4 KB

Large effective bandwidth increase
Optimized Code (2)

```c
for( j = 0; j < NY; j++ ){
    sendBufRight[j] = f[NX][j];
    sendBufLeft[j] = f[1][j];
}

MPI_Irecv( recvBufLeft, NY, MPI_DOUBLE, left,…);
MPI_Irecv( recvBufRight, NY, MPI_DOUBLE, right,…);
MPI_Isend( sendBufRight, NY, MPI_DOUBLE, right,…);
MPI_Isend( sendBufLeft, NY, MPI_DOUBLE, left …);
MPI_Waitall( 4, request, status );

for( j = 0; j < NY; j++ ){
    f[0][j] = recvBufLeft[j];
    f[NX+1][j] = recvBufRight[j];
}
```

- Pack data to send to both left and right
- Non-blocking data exchange
- Unpack data
- Uses bi-directional capability of IB

Significant effective bandwidth increase

![Graph showing MPI Bidirectional Bandwidth](image)

**The University of Texas at Austin**

**Texas Advanced Computing Center**