Parallel Optimization for HPC
Hands-on Lab

Kent Milfeld
milfeld@tacc.utexas.edu

TACC Summer Supercomputing Institute
June 16-20, 2014
Setup

• Login to Stampede:
  – `ssh username@stampede.tacc.utexas.edu`

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

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

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

- 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
  
  \[
  \text{cp} \ ./\text{exchange}_1\text{d.c} \quad ./\text{exchange}_\text{opt.c} \\
  \text{cp} \ ./\text{exchange}_1\text{d.f90} \quad ./\text{exchange}_\text{opt.f90}
  \]

• Compile the current version of the code
  
  \[
  \text{mpicc} \ ./\text{exchange}_1\text{d.c} \quad -o \text{original} \\
  \text{mpif90} \ ./\text{exchange}_1\text{d.f90} \quad -o \text{original}
  \]

• Start an interactive session in stampede
  
  \[
  \text{srun} \quad \text{or} \quad \text{idev} \quad \# \text{ refer to userguide for syntax}
  \]

• Run the code using 10 processors and record the timings it gives you when done
  
  \[
  \text{ibrun} \quad -n \quad 10 \quad -o \quad 0 \quad ./\text{original}
  \]

• Now try to beat that time by modifying exchange_opt.c!
Parallel Optimization for HPC Hands-on

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