COMP 696: Advanced Parallel Computing HW 4: MPI Datatypes, File I/O, and scaling studies.

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 11/08/15
Posted: 10/29/15
Updated: 12/02/15

Table of Contents

Problem 4A: MPI File I/O

- You will use different File I/O mechanisms to store your data
- For different PE and ProbSize distributions, store data using the following mechanisms
- You may need multiple files to store data: velocity, time, etc.
  - Gather data to Root, save as single ASCII file (standard method)
  - Each PE writes to its own file (use MPI_File methods)
  - All PEs write to one file (use MPI_File methods)

Problem 4b: NetCDF Data Storage

- Modify your code to save velocity data at different time slices into a single NetCDF file
- You will need to define your own meta data and arrays to store. Examples include:
  - data stored in structs, velocity, time data
  - create names for data, and scalings (min/max temp, min/max velocity, etc.)
- Visualize 3 or more timeslices using software that can read your netCDF file.

HW 4 Instructions

- For this HW, you will work with your 2D Iterative solver to modernize and optimize the code.
- Code features will include:
  - Dynamic/command line arg processing for both processor and problem distributions.
  - Using 1D communicator groups for subarray data exchange
  - Use of MPI datatypes, including MPI_Type_structs, MPI_Type_vector, and MPI_Type_create_subarray
- You will use different MPI_File I/O and netCDF protocols to store your data.

Configuration for all Problems

- Modify code to process command line arguments for PE's and ProbSize
- Define an MPI_Type_structs to hold configuration information for each node.
- Types of structs & data would include:
  - distribution: rank, problem size,
  - grid: problem size, start and end index
  - time: total time, number of measurements
- consider a local and global data structure.

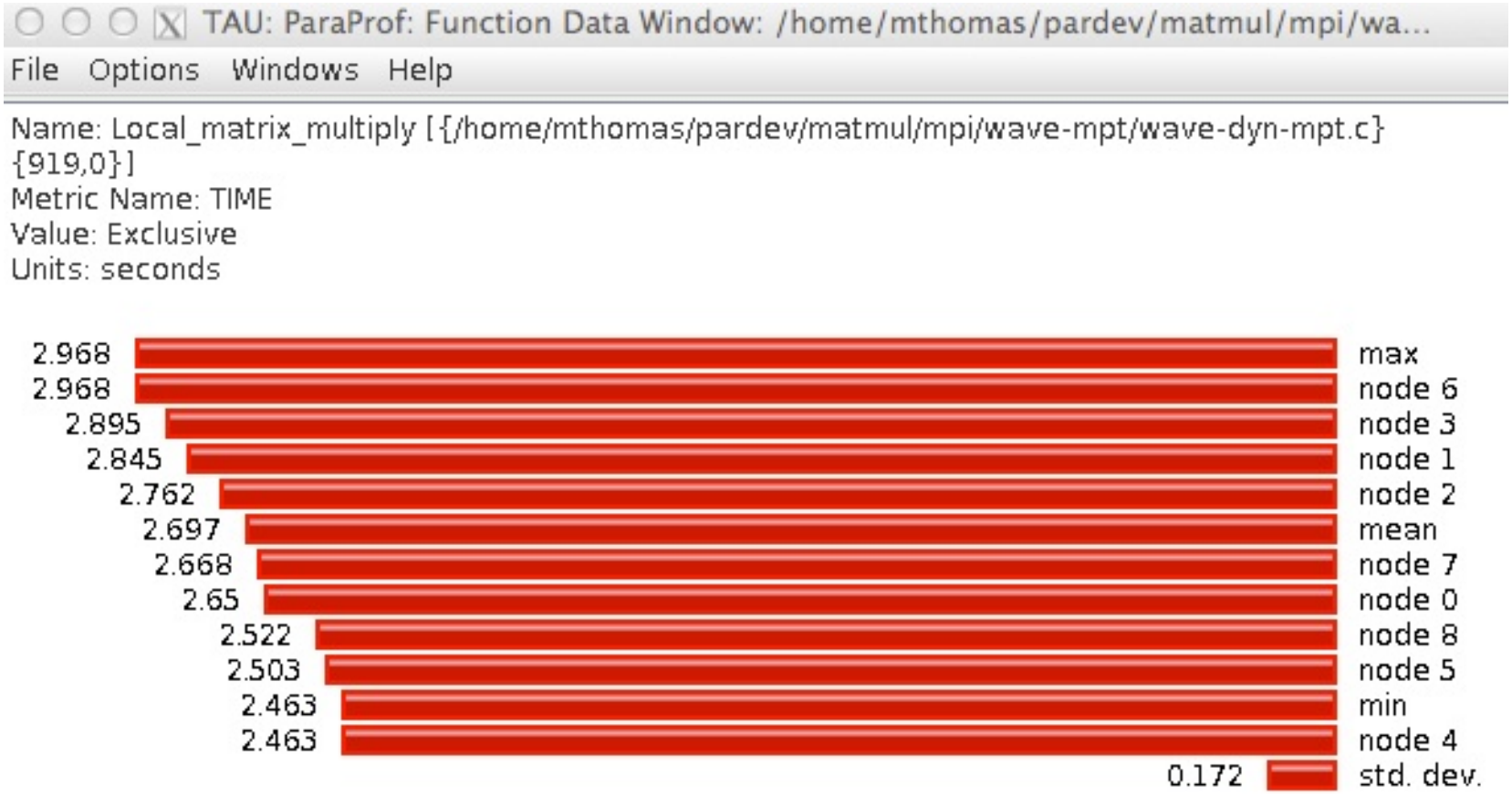
Configuring the Problem Size and Processor Distribution

- 3D block-block data distribution: $M(n_i, n_j, n_k)$
  - where $1 <= n_i <= M$;
  - $M = n_i \star n_j \star n_k$
  - set default to be $n_i = n_j = n_k$
- 3D cartesian processor arrangment: $P(p_i, p_j, p_k)$
  - where $1 <= p_i <= NP$;
  - $NP = p_i \star p_j \star p_k$
  - set default to be $p_i = p_j = p_k$
- Save as MPI struct datatypes; use to communicate to other cores

Analysis

- Compare serial to parallel as a function of PE configurations, problem sizes, and File I/O mechanisms.
- Plot speedup
- Use TAU/ParaProf to identify bottlenecks; you can include images to show what you did.
- See notes at: `http://www-rohan.sdsu.edu/faculty/mthomas/courses/f15/comp696/topics/perf/`

TAU: Using ParaProf to visualize profile files.



Matlab code to read NetCDF Data (pg 1/3)

```
% pres_temp_4D matlab code
% Created by Mary Thomas San Diego State University
% Matlab code based on tutorial at:
% http://www.public.asu.edu/~hhuang38/matlab_netcdf_guide.pdf
%
clear all;
%%
home='/Users/mthomas/working/pardev/svn.assembla/comp696/trunk/netcdf/ser/simple';
ncfile=[home,'/','pres_temp_4D.nc'];
fprintf('Analyzing data file: %s \n', ncfile);

%% netCDF has 3 header sections:  dimensions; variables; data
% dump the data to stdio
ncdisp ( 'pres_temp_4D.nc' );

%% Step 1: Open the file
ncid1 = netcdf.open(ncfile,'NC_NOWRITE');
fprintf('netCDF data file has been opened. \n');

%% Steps 2-4 are often not necessary if one has already obtained
% the needed information about specific variables by running ncdisp in Step 0.

%% Step 2: Inspect the number of variables and number of dimensions, etc., in the file
fprintf(' \n');
[ndim, nvar, natt, unlim] = netcdf.inq(ncid1);
fprintf('netCDF file insp: ndims=[%d], nvars=[%d], natt=[%d], unlim=[%d] \n', ...
          ndim, nvar, natt, unlim);
fprintf(' \n');

%% Step 3: Extract further information for each "dimension"
for i=0:ndim-1
   [dimname, dimlength] = netcdf.inqDim(ncid1, i);
   fprintf('netcdf.inqDim[%d]: dimname=[%s], dimlength[%d] \n',i, dimname, dimlength);
end
fprintf(' \n');
```

Matlab code to read NetCDF Data (pg 2/2)

```
%% Step 4: Extract further information for each variable
for i=0:nvar-1
    [varname, xtype, dimid, natt] = netcdf.inqVar(ncid1, i);
   %dimid
    fprintf('netcdf.inqVar[%d]: varname=%s, xtype=%f, natt=%d\n', i,varname, xtype, natt);
end
fprintf(' \n');

%% read var 1 == latitude
lat1 = netcdf.getVar(ncid1,0,0,6);
lon1 = netcdf.getVar(ncid1,1,0,12);
[n,lev1] = netcdf.inqDim(ncid1, 0);
%%  pres_out[nlat][nlon]
%   #define NLVL 2
%   #define NLAT 6
%   #define NLON 12
%
% = netcdf.getVar(ncid1, varid, start, count)
fprintf('Setting up to read pressure data \n');
pres_varid=2;

% this is a 4D variable
pressure = netcdf.getVar(ncid1,pres_varid);

%massage/extract 2D data
for p = 1:12   %lon
for q = 1:6   %lat
 map1(q,p) = double(pressure(p,q));
end
end
```

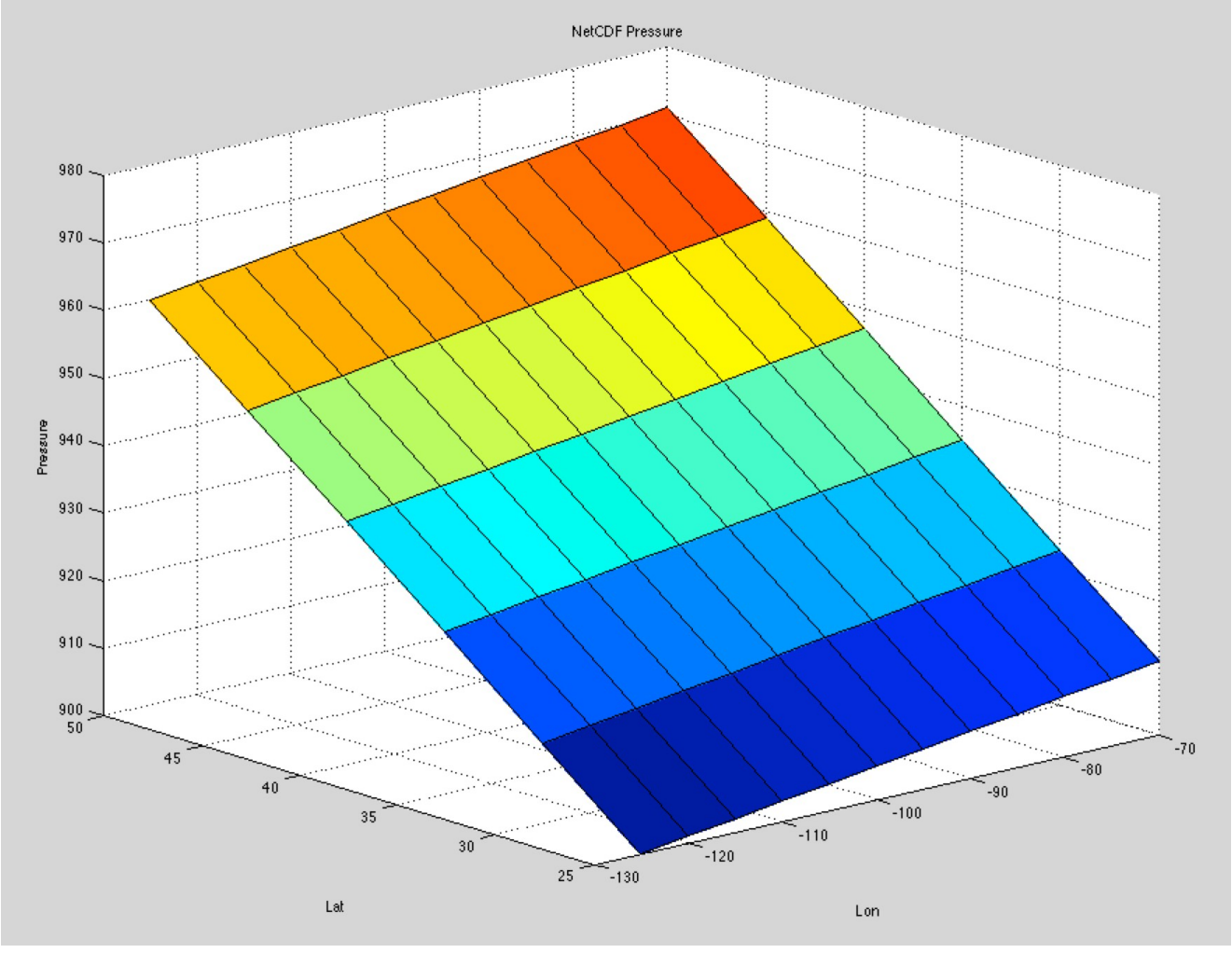
Matlab code to read NetCDF Data (pg 3/3)

```
% Plot the mapped data
pcolor(lon1,lat1,map1);
shading interp
colorbar
hold on
contour(lon1,lat1,map1,'k-')

grid on;

break;
%%
surface(lon1,lat1,map1);
xlabel('Lon');
ylabel('Lat');
zlabel('Pressure');
title('NetCDF Pressure');
grid on;
```

TAU: Using ParaProf to visualize profile files.



Simple plot of pressure data from netCDF file.