497 Application Report | Elad Mazurek, 06/04/2011

On the Parallelization of an Air Temperature Spatial Interpolation and Prediction Program

Contents

Phase One

MASS Polynomial Prediction (PPHandler.java)

Pseudo code

This program takes output files from the air temperature spatial interpolation programs and instantiates MASS threads to process each file and output temperature predictions from two hours after sunset until sunrise.

The way it does this is:

- Receive the number of files to be processed, and MASS threads to instantiate
- Initialize MASS with desired number of threads
- Create a MASS grid of PPHandlers
- Perform MASS.callAll() to have each handler instantiate MassPolynomialPrediction to perform the polynomial prediction work

How to use

The program runs with the following command line:

```
Java polynomialprediction.PPHandler [numFiles] [numThreads]
```

numFiles – refers to the number of output files this program should process
numThreads – refers to the number of MASS threads to have working on these files

Program usage will show when starting the program with no arguments.

Design notes

- Input files
    - Currently looks for grid output files in the home directory of the code
    - The program is currently hardcoded to look for "gridOutput_" followed by an index number. When MASS is invoked, each MASS thread will look for a gridOutput file that matches its MASS id. For example:
        - MASS thread 0 will look for "gridOutput_0.txt"
    - First line of file has to have the height and width of the grid that was output into the file in the following format:
        - height<space>width<\r\n>
    - The rest of the file follows this format:
        - airTemperature<Tab>latitude<Tab> longitude<Tab>elevation<\r\n>
- output files
    - will be named "predictions_" plus the thread ID
- number of predictions per hour

- is controlled by the global variable “PREDICTIONS_PER_HOUR”
- is currently set to 6 (every 10 minutes)

Phase Two

MASS Artificial Neural Network (NNPHandler.java)

Pseudo Code

This program uses MASS to initialize a grid of ANN applications that take in sensor data files to predict and output temperature changes.

The way it does this is:

- Receive the number of files to be processed, and MASS threads to instantiate
- Initialize MASS with desired number of threads
- Create a MASS grid of NNPdrivers
- Perform MASS.callAll() to have each handler instantiate NeuralNetworkPrediction to initialize the artificial neural network and predict future temperature changes

How to use

The program runs with the following command line:

```
Java neuralnetwork.NNPHandler [numFiles] [numThreads]
```

numFiles – refers to the number of output files this program should process
numThreads – refers to the number of MASS threads to have working on these files

Program usage will show when starting the program with no arguments.

Design notes

- Initializing ANN from a previously saved state is currently disabled
- Run parameters currently hardcoded in the run() method within NNPDriver.java. Parameters are set as following:
    - Args[0] == true
        - Set to first time run
    - Args[1] == input file name
        - Looks for a file whose start number matches the current MASS ID + 1. This file is set as the current input file name.
            - Ex: MASS thread 0 will select '1_frost_ff03' as its input file
    - Args[2] == output file name
        - Set to 'outputFile_' plus current MASS ID
    - Program defaults are accepted for the rest of the parameters
    - List of all parameters:

```
Every time arguments:
false  if you have already initialized and do not want to reset       required
-if    inputFileName          required
-bf    backupFileName         optional
-of    outputFileName         optional
```

```
▪
▪    First time through arguments:
▪    true   if it is the first time running for the location or want to reset the prediction variables
     required
▪    -if    inputFileName           required
▪    -bf    backupFileName          optional
▪    -of    outputFileName          optional
▪    -fA    functionA               optional
▪    -fB    functionB               optional
▪    -fC    functionC               optional
▪    -sh    startHour               optional
▪    -eh    endHour                 optional
▪    -pph   minuteIncrament         optional
▪    -w     minWeight, maxWeight    optional
```

- Input files
  - Practice data read has only been in SQL format
  - Program is hardcoded to only read in data for 05/21/2011 19:00
    - This can be altered by changing the global date & time parameters within NeuralNetworkPrediction.java
      - Variable names: yr, mo, dy, hr
  - SQL format example

sensorFF01data.txt - Notepad

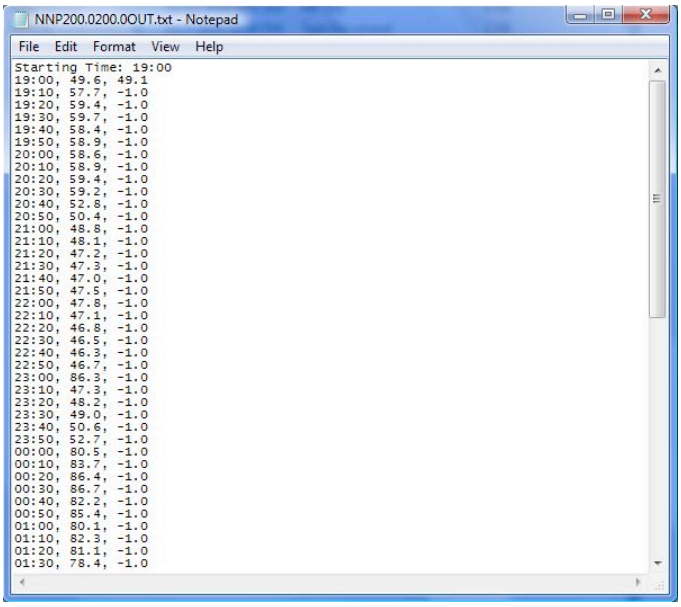
File Edit Format View Help

```
<<< Log from medusa started October 14, 2010, 13:10:47 >>>

mysql> select * from frost_ff01 where station_id = 'ff01' and station_name = 'ff01';
```

station_id	station_name	sensortypes	sensor1	sensor2	battery_voltage	rssi	location_x	location_y	alarm_thresh	alarm_dir	date_time
ff01	ff01	900	69.00000	77.97000	6.30000	-89.00000	200	200	35.00000	B	2007-03-27
ff01	ff01	900	68.50000	74.76000	6.30000	-89.00000	200	200	35.00000	B	2007-05-28
ff01	ff01	900	62.70000	65.20000	6.20000	-82.00000	200	200	35.00000	B	2007-05-28
ff01	ff01	900	76.60000	77.97000	6.60000	-86.00000	200	200	35.00000	B	2007-07-18
ff01	ff01	900	27.10000	29.25000	6.20000	-76.00000	200	200	35.00000	B	2007-11-23
ff01	ff01	900	40.30000	38.53000	5.50000	-79.00000	200	200	35.00000	B	2007-12-19
ff01	ff01	900	29.80000	28.85000	6.60000	-74.00000	200	200	35.00000	B	2007-12-27
ff01	ff01	900	34.00000	35.44000	6.20000	-78.00000	200	200	35.00000	B	2008-02-19
ff01	ff01	900	41.70000	43.14000	4.50000	-76.00000	200	200	35.00000	B	2008-03-17
ff01	ff01	900	38.30000	39.82000	6.00000	-78.00000	200	200	35.00000	B	2008-06-11
ff01	ff01	BOF	0.00000	0.00000	2560.70000	-15118.00000	200	200	35.00000	B	2008-07-04
ff01	ff01	900	80.00000	83.88000	6.50000	-88.00000	200	200	35.00000	B	2008-07-28
ff01	ff01	900	71.00000	71.44000	5.90000	-92.00000	200	200	35.00000	B	2008-08-13
ff01	ff01	900	50.50000	51.53000	6.60000	-86.00000	200	200	35.00000	B	2008-09-18
ff01	ff01	900	84.50000	97.83000	6.20000	-87.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.80000	97.83000	6.20000	-83.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.50000	97.77000	6.30000	-101.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.20000	97.66000	6.20000	-91.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	83.90000	97.60000	6.30000	-85.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.50000	97.44000	6.30000	-92.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.80000	97.38000	6.20000	-91.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.80000	97.55000	6.20000	-84.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.50000	97.60000	6.20000	-101.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	85.10000	97.77000	6.20000	-91.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	85.10000	97.89000	6.20000	-89.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.50000	97.94000	6.20000	-85.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.20000	98.11000	6.20000	-84.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.20000	98.22000	6.20000	-86.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.50000	98.50000	6.20000	-84.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.80000	98.78000	6.20000	-85.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	83.60000	99.07000	6.20000	-93.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	83.60000	99.01000	6.30000	-93.00000	200	200	35.00000	B	2003-12-04
ff01	ff01	900	84.20000	98.90000	6.30000	-92.00000	200	200	35.00000	B	2003-12-04

- Output files
  - Are named ‘outputFile_’ plus the MASS thread ID
    - Ex: MASS thread 0 will output prediction data into ‘outputFile_0’
  - Format
    - First line shows the sensor time received
    - Following lines:
      - Time of day, predicted temp, real temp (if available)
  - Output format example


NNP200.0200.0OUT.txt - Notepad
File Edit Format View Help
Starting Time: 19:00
19:00, 49.6, 49.1
19:10, 57.7, -1.0
19:20, 59.4, -1.0
19:30, 59.7, -1.0
19:40, 58.4, -1.0
19:50, 58.9, -1.0
20:00, 58.6, -1.0
20:10, 58.9, -1.0
20:20, 59.4, -1.0
20:30, 59.2, -1.0
20:40, 52.8, -1.0
20:50, 50.4, -1.0
21:00, 48.8, -1.0
21:10, 48.1, -1.0
21:20, 47.2, -1.0
21:30, 47.3, -1.0
21:40, 47.0, -1.0
21:50, 47.5, -1.0
22:00, 47.8, -1.0
22:10, 47.1, -1.0
22:20, 46.8, -1.0
22:30, 46.5, -1.0
22:40, 46.3, -1.0
22:50, 46.7, -1.0
23:00, 86.3, -1.0
23:10, 47.3, -1.0
23:20, 48.2, -1.0
23:30, 49.0, -1.0
23:40, 50.6, -1.0
23:50, 52.7, -1.0
00:00, 80.5, -1.0
00:10, 83.7, -1.0
00:20, 86.4, -1.0
00:30, 86.7, -1.0
00:40, 82.2, -1.0
00:50, 85.4, -1.0
01:00, 80.1, -1.0
01:10, 82.3, -1.0
01:20, 81.1, -1.0
01:30, 78.4, -1.0

Phase Three

Real-Time Processing (RealtimeProcessing.java)

Pseudo Code

This program uses the previously mentioned temperature prediction programs, and utilizes them to predict temperature changes based on incoming sensor information.

The way it does this is:

- Receive sensor information in the form of a text file
- Parse in sensor data into a 2d array (data[numSensors][numArguments])
- Feed data array to the artificial Neural Network to get predictions
    - Utilizes newly created methods that process real-time data within the NeuralNetworkPrediction class
    - Entry point is the NeuralNetworkPrediction.realTimeNNPRun(data, numSensors) method
- Determine if incoming sensor data is two hours past sunset for current location
    - Compare sensor report time with current sunset information for area as reported by Yahoo weather services
- If sensor time is within the two hour time frame, it runs Polynomial Prediction to get 10 minute predictions from sunset + 2 hours until sunrise
    - Feed data array to MassPolynomialPrediction to get predictions
        - Utilizes newly created methods that process real-time data within the MassPolynomialPrediction class
        - Entry point is the MassPolynomialPrediction.realTimePP() method
- (not yet implemented) if PP has executed, compare ANN output with that of PP

How to use

The program runs with the following command line:

```
Java RealtimeProcessing [name of sensor data file] [WOEID of location]

WOEID – is the location ID used by Yahoo weather services to retrieve weather
related info.  If no ID is given, location defaults to Bothell, WA.
Instructions are given below on how to determine WOEID of a location
```

Program usage will show when starting the program with no arguments.

Design notes

- Weather location
    - Location is currently hardcoded for Bothell Washington. To get sunset information for other locations, the numeric portion of the XML link (WOEID) needs to change:
    - Example: to get the sunset information for San Diego:
        1) Go to http://weather.yahoo.com/
        2) Enter 'san diego' in the city or zip code
        3) Copy the WOEID portion directly from the hyper link
            - http://weather.yahoo.com/united-states/california/san-diego-**2487889/**
            - WOEID is **2487889** in this example
        4) start the program from the command line with the following syntax (assuming sensor data file is called 'test.txt'):
            - java RealtimeProcessing test.txt 2487889
- Input file
    - Needs to conform to this format:
        - First line: <number of sensors reporting>
        - Second line: <year-day-month hour-minute-second>
        - Following line per sensor:
            - Temp </t> latitude </t> longitude </t> elevation </t>
- Output file(s)
    - The program outputs two files. One from ANN, and another from PP (if sensor time received is 2 hours after sunset)