```
//Thread scheduling on a multi-machine system


#include <string.h>
#include <time.h>
#include "layer.cpp"
#define TRAINING_FILE    "training.txt"
#define WEIGHTS_FILE "weights.txt"
#define OUTPUT_FILE "output.txt"
#define TEST_FILE   "test.txt"

//***********************************************
struct sche{
  float val1;
  float val2;
  float val3;
  float val4;
  float val5;
  float val6;
  float val7;
  float val8;
  float val9;
  float val10;
  float val11;
  float val12;
  float deadpr; //for getting job priority from deadline
  float startpr;//for getting job priority from starttime
  char job1[5];//for operation names
  char job2[5];
```

```
  char job3[5];
  char job4[5];//for operation names
  char job5[5];
  char job6[5];
  char jobname[60];//for job names
  char ope[50];
  };
void main()
{
sche sche_val[100]; //for 4 operation parameters
char valoper1[4], valoper2[4];
char opee[30][30];
int flag = 1;
float error_tolerance=0.1;
float total_error=0.0;
float avg_error_per_cycle=0.0;
float error_last_cycle=0.0;
float avgerr_per_pattern=0.0; // for the latest cycle
float error_last_pattern=0.0;
float learning_parameter=0.02;
float alpha; // momentum parameter
float NF; // noise factor
float new_NF;

unsigned temp, startup, start_weights;
long int vectors_in_buffer;
long int max_cycles;
```

```
long int patterns_per_cycle=0;


long int total_cycles, total_patterns;

int i,j,x,y,n,k;

int exi, value;

float value1,value2, value3;

float earliest[100], dura[100],duration[100], deadline[100],
critical[100], critical1[100];

float array[100],
duration1[100],duration2[100],deadline1[100], array7[4],
array8[4],array9[4];

float array1[4], array2[4], array3[4], array4[4], array5[4],
array6[4], array10[4], array11[4];

float array12[4];

int opno,mno,jobno;

clrscr();

// create a network object

network backp;


FILE * training_file_ptr, * weights_file_ptr, *
output_file_ptr;

FILE * test_file_ptr, * data_file_ptr;


// open output file for writing

if ((output_file_ptr=fopen(OUTPUT_FILE,"w"))==NULL)

          {

          cout << "Problem opening output file\n";

          exit(1);

          }
```

```
cout << "=======================================\n";

cout << "                                       \n";

cout << "THREAD SCHEDULING ON A MULTI-MACHINE SYSTEM \n";

cout << "=======================================\n\n";

cout << "Enter 1 for NETWORK TRAINING or 2 for SCHEDULING : ";

cin >> temp;

if(temp ==2)

{

clrscr();

flag = 0;

cout << endl;

//cout << "OPERATION-ATTRIBUTES ACCUSITION SECTION\n";

//cout << "=======================================\n\n";

cout << "An Operation has 4 attributes such as EARLIST_START,
DURATION, DEADLINE, and ";

cout << "CRITICAL_TYPE.\n";

cout << "\n";

ofstream fout;

fout.open("test.txt");

cout << "For a Job shop with 6 jobs (24 operations) and 4
machines. \n" ;

//cin >> opno >> mno;

   opno = 24;

   mno = 4;

 jobno = opno/4; //Getting jobs


j=0;
```

```
float maxi[96];
srand(time(0));
     for(int u = 0; u < 96; u++)
     maxi[u] =  ((rand()%9) * 0.11) + 0.1;

     for(int l = 0; l < 96; l += 4)
           {    fout << maxi[l] << "  ";
             earliest[j] = maxi[l];
             fout << maxi[l+1] << "  ";
             duration[j] = maxi[l+1];
             fout << maxi[l+2] << "  ";
             deadline[j] = maxi[l+2];
             fout << maxi[l+3] << "  ";
             critical[j] = maxi[l+3];
             fout << endl;
             ++j;
             }
fout.close();
 temp = 0;
 }
cout << endl;
backp.set_training(temp);

if (backp.get_training_value() == 1)
     {
     cout << "Training mode is ON and weights will be saved in
the file \n";
```

```
     cout << "weights.txt at the end of the training.\n\n";
     }
else
     {
     cout << "\n\n";
     cout << "Weights will be loaded from the file weights.txt
and the \n";
     cout << "current attribute values will be trained.\n\n";
     }
if (backp.get_training_value()==1)
     {
     cout << "Enter the Error Tolerance value: ";
     cin >> error_tolerance;
     cout << endl;
     cout << "Enter the learning rate: ";
     cin >> learning_parameter;
     cout << endl;
     cout << "Enter the momentum value<0 - 1.0>: ";
     cin >> alpha;

     if ((training_file_ptr=fopen(TRAINING_FILE,"r"))==NULL)
          {
          cout << "Problem opening training file\n";
          exit(1);
          }
     data_file_ptr=training_file_ptr; // training on

     cout << "Enter number cycles: ";
```

```
     cin >> max_cycles;

     start_weights = 0;
     }
else
     {
     if ((test_file_ptr=fopen(TEST_FILE,"r"))==NULL)
          {
          cout << "Problem opening test file\n";
          exit(1);
          }

     data_file_ptr=test_file_ptr; // training off
     }
total_cycles=0; // a cycle is once through all the input data
total_patterns=0; // a pattern is one entry in the input data
new_NF=NF;
backp.get_layer_info();
backp.set_up_network();
if ((backp.get_training_value()==1)&&(start_weights!=1))
     {

     backp.randomize_weights();
          backp.set_NF(new_NF);
     }
else
     {
```

```
     if ((weights_file_ptr=fopen(WEIGHTS_FILE,"r"))
               ==NULL)
          {
          cout << "problem opening weights file\n";
          exit(1);
          }
     backp.read_weights(weights_file_ptr);
     fclose(weights_file_ptr);
     }

startup=1;
vectors_in_buffer = MAX_VECTORS; // startup condition
total_error = 0;

while (             ((backp.get_training_value()==1)
               && (avgerr_per_pattern
                         > error_tolerance)
               && (total_cycles < max_cycles)
               && (vectors_in_buffer !=0))
               || ((backp.get_training_value()==0)
               && (total_cycles < 1))
               || ((backp.get_training_value()==1)
               && (startup==1))
               )
{
startup=0;
```

```
error_last_cycle=0; // reset for each cycle
patterns_per_cycle=0;
backp.update_momentum(); // added to reset
     while ((vectors_in_buffer==MAX_VECTORS))
     {
    vectors_in_buffer=
          backp.fill_IObuffer(data_file_ptr); // fill buffer
          if (vectors_in_buffer < 0)
               {
cout << "Error in reading in vectors, ABORTING...\n";
               exit(1);
               }
               // process vectors
          for (i=0; i<vectors_in_buffer; i++)
               {
               backp.set_up_pattern(i);
               total_patterns++;
               patterns_per_cycle++;
                         backp.forward_prop();
               if (backp.get_training_value()==0)
                    backp.write_outputs(output_file_ptr);
               if (backp.get_training_value()==1)
                    {
                  backp.backward_prop(error_last_pattern);
                    error_last_cycle +=

     error_last_pattern*error_last_pattern;
                    avgerr_per_pattern=
```

```
     ((float)sqrt((double)error_last_cycle/patterns_per_cycle)
);
                    if ((avgerr_per_pattern
                         > error_tolerance)
                         && (total_cycles+1 < max_cycles))

     backp.update_weights(learning_parameter,alpha);
                              }
                         }
          error_last_pattern = 0;
     }
total_error += error_last_cycle;
total_cycles++;
if (total_cycles>0.7*max_cycles)
          new_NF = 0;
else if (total_cycles>0.5*max_cycles)
               new_NF = 0.25*NF;
          else if (total_cycles>0.3*max_cycles)
                         new_NF = 0.50*NF;
                   else if (total_cycles>0.1*max_cycles)
                                   new_NF = 0.75*NF;
backp.set_NF(new_NF);
cout << total_cycles << "\tMSE : " << avgerr_per_pattern <<
"\n";
fseek(data_file_ptr, 0L, SEEK_SET); // reset the file pointer
          vectors_in_buffer = MAX_VECTORS; // reset


} // end main loop
```

```
clrscr();
if (backp.get_training_value()==1)
     {
     if ((weights_file_ptr=fopen(WEIGHTS_FILE,"w"))
               ==NULL)
          {
          cout << "problem opening weights file\n";
          exit(1);
          }
     }

if (backp.get_training_value()==1)
     {
     backp.write_weights(weights_file_ptr);
     backp.write_outputs(output_file_ptr);
     avg_error_per_cycle=(float)sqrt((double)total_error/total
_cycles);
     error_last_cycle=(float)sqrt((double)error_last_cycle);
     fclose(weights_file_ptr);

cout << "\n\n ";
cout << "weights saved in file weights.txt\n";
cout << "Error last cycle = " << error_last_cycle << "\n";
cout << "Mean Squared Error in the last pattern = " <<
avgerr_per_pattern << " \n";
     }
cout << "Total cycles = " << total_cycles << "\n";
cout << "Total patterns = " << total_patterns << " \n";
```

```
fclose(data_file_ptr);
fclose(output_file_ptr);
cout << endl;
if(flag == 0)
{
     ifstream fin;
     fin.open("output.txt");
     if(!fin)
     cout << "The file, output.txt is corrupted!\n";
     else
     {
      for(i=0; i < opno; i++) //For getting all parameters
from output.txt
      fin >> array[i];    //and stores into array[i]
      }
     for(i=0; i < 4; i++)
         array1[i] = array[i];
          x=0;
     for(i=4; i < 8; i++)
{
          array2[x] = array[i];
          ++x;      }
     x=0;
     for(i=8; i < 12; i++)         //get 4 start-orders for
job3
       {
          array3[x] = array[i];
```

```
      ++x;
                    }
  x=0;
for(i=12; i < 16; i++)
  {
      array4[x] = array[i];
      ++x;
                    }
 x=0;
for(i=16; i < 20; i++)
  {
      array5[x] = array[i];
      ++x;
                    }
 x=0;
for(i=20; i < 24; i++)
  {
      array6[x] = array[i];
      ++x;
                    }
            for(i=0; i < 4; i++)
    array7[i] = duration[i];

x=0;
for(i=4; i < 8; i++)
  {
      array8[x] = duration[i];
```

```
      ++x;       }
x=0;
for(i=8; i < 12; i++)          //get 4 durations for job3
  {
      array9[x] = duration[i];
      ++x;
                    }
   x=0;
for(i=12; i < 16; i++)
  {
      array10[x] = duration[i];
      ++x;
                    }
  x=0;
for(i=16; i < 20; i++)
  {
      array11[x] = duration[i];
      ++x;
                    }
  x=0;
for(i=20; i < 24; i++)
  {
      array12[x] = duration[i];
      ++x;
                    }

 j=0;
```

```
 for(i=0; i < J-SIZE; i++)
 { sche_val[j].val = array1[i];

            }
//NON SHARING GREEDY ALGORITHM

   for(i=0; i < 4; i++)
       duration[i] = sche_val[i].val7;

   x=0;
   for(i= 4; i < 8; i++)
     {
        duration[i] = sche_val[x].val8;
        ++x;      }
   x=0;
   for(i=8; i < 12; i++)
     {
        duration[i] = sche_val[x].val9;
        ++x;
              }
     x=0;
   for(i= 12; i < 16; i++)
     {
        duration[i] = sche_val[x].val10;
        ++x;      }
   x=0;
   for(i=16; i < 20; i++)               {
```

```
            duration[i] = sche_val[x].val11;
            ++x;
                  }
           x=0;
      for(i=20; i < 24; i++)                {
            duration[i] = sche_val[x].val12;
            ++x;
                  }
        for(i=0; i < 4; i++)
strcpy(sche_val[i].ope,  sche_val[i].job1);


      x=0;
      for(i= 4; i < 8; i++)                       {
            strcpy(sche_val[i].ope , sche_val[x].job2);
            ++x;       }
       x=0;
      for(i=8; i < 12; i++)                 {
            strcpy(sche_val[i].ope,  sche_val[x].job3);
            ++x;
                  }
        x=0;
      for(i= 12; i < 16; i++)                     {
            strcpy(sche_val[i].ope , sche_val[x].job4);
            ++x;       }
       x=0;
      for(i=16; i < 20; i++)                {
            strcpy(sche_val[i].ope,  sche_val[x].job5);
            ++x;
```

```
                }


          ++x;
      x=0;
     for(i=20; i < 24; i++)                {
          strcpy(sche_val[i].ope,  sche_val[x].job6);
          ++x;
                }
float myra = 0, myra1, RE;
          //====Feasibility test===========
          for(i=0; i < 24; i++)
          myra += duration[i];
          myra1 = myra/4.0;


            if(mini > myra1)
            cout << "\nFeasible schedule (sharing).\n";
            else if (mini == myra1)
             cout << "\n Good enough schedule (sharing).\n";
             else
             cout << "\n Schedule is non-reasonable
(sharing).\n";
             RE = (mini - myra1)/ myra1;
             cout << "\n Relative Error (sharing) is " << RE
<< endl;
    cout<< "-------------------------------------\n";
    cout << "The best FT from Non-sharing scheme is " <<
gunna1 << endl;
    if(gunna1 > poora1)
```

```
            cout << "Feasible schedule (non-sharing).\n";
            else if (gunna1 == poora1)
             cout << "Good enough schedule(non-sharing).\n";
             else
             cout << "Schedule is non-reasonable(non-
sharing).\n";
             RE1 = (gunna1 - poora1)/ poora1;
             cout << "Relative Error (non-sharing) is " << RE1
<< endl;
    cout << "\nBest performance scheme:\n";
    cout << "\n----------------------\n";
    if(mini < gunna1)
     cout << "\n The schedule is with FT " << mini << "
(sharing)." << endl;
     else
     cout << "\n The schedule is with FT " << gunna1 << "(Non-
sharing)." << endl;


}
             cout << "  Run the program again to train a new
attribute set.\n";
     }
else
{
cout <<
"\n==========================================================
==========\n";
cout << "Initial Dataset training is over. Run the program
again for scheduling.\n";
cout <<
"=====================================================\n";
```

```
}
getch();
}
```