A Parallel Distributed Genetic Algorithm Application for Feature Selection in Classification Problems
What is this long title about??

- What is feature selection?
- What is a genetic algorithm (GA)?
  - Selection, Crossover, Mutation, Fitness
  - Furthermore what is a distributed GA (DGA)?
- How do you classify given a data set? ...Well, in our case we use a Support Vector Machine, specifically the program \textit{SVM}^\text{light}
- \texttt{MPI} as the message passing service...but why? ...and what implementation to use?
Problem to solve

Goal:
- Create a program such that given a data set as input can output a subset of the features which provides a better accuracy during classification and reduces the data set size.

Approach:
- Use MPI to parallelize a DGA which will search for a solution to the subset problem across a network of machines increasing accuracy of our solution as well as improved running time.
- We have used 11 machines in the SunLab to run our program in which they communicate with each other while running in parallel.
Feature Selection

- **Definition:** a process used in machine learning for selecting a subset of the features that reduces number of features while increasing the accuracy

- **Why is it useful?**
  - Data reduction - saves resources collecting/storing data
  - Improves prediction accuracy - by removing noise features
  - Reduces training time and algorithm execution time
  - Can provide insight to the problem

- Other approaches based on Greedy forward/backward search, mutual information, simulated annealing, etc.
Message Passing Interface (MPI)

- We chose to use MPICH (Version 3.0)
  - Superior performance than Open MPI [3]
  - Updated to the MPI 3.0 standard which was introduced in late 2012 (which Open MPI has not done yet)
  - Exclusively used on 9 of the top 10 supercomputers including the world’s fastest Tianhe-2 [1]
- Used to provide communication, a virtual topology, and synchronization between a set of processes
- Basic operations: MPI_Send, MPI_Recv
Background for GAs

- **Chromosome** - encoding of a solution to the given problem (i.e. a subset of features)
- **Genes** of a chromosome - encode whether or not we include a feature from our data set in our classification process
  - Ex. If C1 has genes = [1, 1, 0, 0, 1] this means that out of 5 features for the given data set this chromosome will only use features 1, 2, and 5 for classification
- **Fitness** of a chromosome - the measure of how fit the given chromosome is in relation to the problem
  - Fitness = accuracy value returned by SVM\textsuperscript{light} after running the modified data set which corresponded to the chromosome’s genes
Generic Scheme for a GA

1. Create a population $P$ of chromosomes
2. Compute the fitness of each chromosome in $P$
3. WHILE(Stopping condition not met) DO
   1. Select Parents $P_1$ and $P_2$ from $P$
   2. Crossover: combine $P_1$ and $P_2$ to create 2 offspring $O_1$ and $O_2$
   3. Mutation: Mutate $O_1$ and $O_2$
   4. Replacement: Attempt to place $O_1$ and $O_2$ in $P$, while keeping $|P|$ fixed
4. Return the best chromosome in $P$

Referenced From:
Dr. Bui
COMP511
We sum up the fitness values then assign each chromosome a proportion of the wheel based on their individual fitness.
GA- Crossover & Mutation

- Best explained in an example:
  
  Selected parents $P_1$ and $P_2$
  
  $P_1=[1,0,1,0,1,0]$ & $P_2=[0,0,0,1,1,1]$
  
  Offspring $O_1$ and $O_2$ after one-point crossover
  
  $O_1=[1,0,1,1,1,1]$ & $O_2=[0,0,0,0,1,0]$
  
  Offspring $O_1$ and $O_2$ after mutation
  
  $O_1=[1,0,1,1,1,0]$ & $O_2=[0,1,0,0,1,0]$
GA- Replacement

- We want to possibly bring the offspring into our population, but while keeping the overall size fixed.

- One possible replacement is as follows:
  - If an offspring is more fit than any of the chromosomes in our current population, we replace them.
Implementation – Coded in C

- Master/Worker relationship with 11 computers
  - 10 worker processes (islands) & 1 master process
- Use DGA such that each island runs a separate GA and occasionally send chromosomes to each other to increase diversity
- Bank Marketing Data Set from the UCI repository[4]
  - Number of Instances: 45,211  Number of Features: 16
  - Binary classification: Y/N whether a client will open a term deposit (data taken from a Portuguese bank)
- How to determine fitness for chromosome C?
  - Create a temporary data set which only contains the subset of features that C represents
  - Train a model based on the temporary data set using SVM \text{light} and obtain a test accuracy
  - C’s fitness = obtained accuracy value
Structure of our DGA

WORKER-Island

//START WORKER
1. WHILE (Master says Work) DO:
   a. Run 5 Generations of GA:
      * Selection
      * Crossover
      * Mutation
      * Fitness
      * Replacement
   b. Migration:
      * Send migrants to Master
      * Receive immigrants
      * Replacement of immigrants
//END WHILE
2. Send Best Chromosome to Master
//END WORKER

MASTER

//START MASTER
1. Randomize data set
2. DO
   * Tell Workers to Work
   * Wait for migrants from all islands
   * Mutate migrants
   * Shuffle migrant ordering
   * Randomly send migrants to each island
   WHILE ( ! Stopping Condition) 
3. Tell Workers to STOP working
4. Receive Best Chromosome from each island
5. Output the best overall Chromosome
//END MASTER
Results & Analysis

Running times and Results:

- 10 migration rounds, 5 GA generations between migration rounds, each island has population 50:
  - Full data set (∼45k): Estimated 4-5 days (currently running)
    - Unknown results
  - Small data set (∼4.5k): ~8-10 hours
    - Produces a few different subsets with same accuracy ∼ 91%

- 3 migration rounds, 2 GA generations between migration rounds, each island has population 25:
  - Very small data set (50): < 10 minutes
    - You will see in the upcoming demo
```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#define MESSAGE_TAG 1
#define NUM_FEATURES 16
#define NUM_MIGRANTS 1
typedef struct {
    char genes[NUM_FEATURES];
    double fitness;
} Chromosome;

int main (int argc, char *argv[]) {
    int rank, size, t;
    MPI_Init (&argc, &argv); /* Initialization of MPI operations */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* To get the rank of the process */
    MPI_Comm_size (MPI_COMM_WORLD, &size); /* To get the number of processes */
    MPI_Status status; /* Stores the message tag, error status and source of message being sent */

    /* Creating Derived Data Type */
    MPI_Datatype migrant;
    MPI_Datatype types[2]= {MPI_DOUBLE, MPI_CHAR};
    int blocklengths[2]= {1, 16}; /* Number of variables of each MPI data type */
    MPI_Aint displacements[2]; /* Holds the starting position of each variable of the struct */
    MPI_Aint displacements[2]= {1, 16};
    MPI_Aint doubleex;
    MPI_Type_extent(MPI_DOUBLE, &doubleex);
    displacements[0]= (MPI_Aint) 0;
    displacements[1]= doubleex;
    /* Set the displacement of fitness to 0 since it's the first variable of the struct */
    displacements[0]= 0;
    displacements[1]= doubleex;
    MPI_Type_struct(2, blocklengths, displacements, types, &migrant); /* Constructing the above components to create our migrant derived data type */
    MPI_Type_commit(&migrant); /* Committing the data type so it can be used in our program */

    if (rank==0) /* Master Process */ {
        Chromosome wiggle;
        MPI_Recv(&wiggle, NUM_MIGRANTS, migrant, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Chromosome fitness is %f and genes are: ", wiggle.fitness);
        for(t=0; t < NUM_FEATURES; t++)
            printf("%c", wiggle.genes[t]);
        printf("\n");
    } else /* Worker Processes - ranks other than zero */ {
        Chromosome jiggle;
        jiggle.fitness = 0.314159;
        for(t=0; t < NUM_FEATURES; t++)
            if (t%2)
                jiggle.genes[t] = '0';
            else
                jiggle.genes[t] = '1';
        MPI_Send(&jiggle, NUM_MIGRANTS, migrant, 0, MESSAGE_TAG, MPI_COMM_WORLD);
    }
    MPI_Finalize();
```
Our Struct and Initializing MPI

```c
#define MESSAGE_TAG 1
#define NUM_FEATURES 16
#define NUM_MIGRANTS 1

typedef struct
{
    char genes[NUM_FEATURES];
    double fitness;
} Chromosome;

int main (int argc, char *argv[])
{
    int rank, size, i;
    /* Initialization of MPI operations */
    MPI_Init (&argc, &argv);
    /* To get the rank of the process */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    /* To get the number of processes */
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    /* Stores the message tag, error status
    and source of message being sent */
    MPI_Status status;
```
MPI Derived Data Type

/* Creating Derived Data Type */
/* Name of new data type */
MPI_Datatype migrant;
MPI_Datatype types[2]={MPI_DOUBLE,MPI_CHAR};
/* Number of variables of each MPI data type */
int blocklengths[2]={1,16};

/* Holds the starting position of each variable of the struct */
MPI_Aint displacements[2];
MPI_Aint doubleex;
MPI_Type_extent(MPI_DOUBLE,&doubleex);
/* Set the displacement of fitness to
0 since its the first variable of the struct */
displacements[0]=(MPI_Aint)0;
/* Set the displacement of the genes array
to be after fitness in our MPI derived data type */
displacements[1]=doubleex;

/* Constructing the above components to create
our migrant derived data type */
MPI_Type_struct(2,blocklengths,displacements,types,&migrant);
/* Committing the data type so it can be used in our program */
MPI_Type_commit(&migrant);
/* END Creating Derived Data Type */
Basic Communication

```c
/* Master Process */
if (rank==0)
{
    Chromosome wiggle;
    MPI_Recv(&wiggle, NUM_MIGRANTS, ngrant, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    printf("Chromosome fitness is %f and genes are: ", wiggle.fitness);
    for(i=0; i<NUM_FEATURES; i++)
        printf("%c", wiggle.genes[i]);
    printf("\n");
}
/* Worker Processes - ranks other than zero */
else
{
    Chromosome jiggle;
    jiggle.fitness = .314159;
    for(i=0; i < NUM_FEATURES; i++)
    {
        if(i%2)
            jiggle.genes[i] = '0';
        else
            jiggle.genes[i] = '1';
    }
    MPI_Send(&jiggle, NUM_MIGRANTS, ngrant, 0, MESSAGE_TAG, MPI_COMM_WORLD);
}
MPI_Finalize();
```

```c
#define MESSAGE_TAG 1
#define NUM_FEATURES 16
#define NUM_MIGRANTS 1
typedef struct
{
    char genes[NUM_FEATURES];
    double fitness;
} Chromosome;
```
Executing the Above Example

- **How to compile MPI programs:**
  - `mpicc example.c –o example`

- **How to execute MPI programs:**
  - `mpiexec –f machinefile –n 2 ./example`
  - **-f** indicates we have a machine file
    - **Machine file:** Contains the IPs for each of the available machines we can connect to. In our case the names of the SunLab computers (i.e. Grace, Pascal, Galois, etc.)
  - **-n** is used to define how many processes to start
Problems We Faced

- For SVM\textsuperscript{light} we used C system(command) function
  - this printed unnecessary statements to the terminal and made it convoluted to obtain the accuracy
  - **Fix:** Use of popen(command, “r”) to read and parse the output of SVM\textsuperscript{light}

- All islands were generating the same chromosomes!!
  - **Fix:** we seeded the rand() function with srand((island\_ID + 1) * time(null)) as they have unique ids

- mpiexec command needing ssh login password for every machine being used on execution:
  - **Fix:** using ssh keygen we created a public key so that we can connect to any machine in the lab without a password

- MPI trying to use ports blocked in the Sun Lab
  - **Fix:** Set Environment Variable
    - MPIEXEC\_PORT\_RANGE = 6000:7000
Future thoughts…

- To create a GUI: Enable users to select a dataset file and choose input parameters such as: number of generations between migration, number of islands, mutation rate, etc.

- Using GPUs…
  - New research is being done to use MPI for communication between nodes, but also using CUDA or OpenACC to create MPI + OpenACC, CUDA-aware MPI, and MVAPICH2 systems
    - Increase performance by processing on several cores of a GPU rather than few of a CPU

- Implement failure handling through backup logs
  - MPI cannot handle machine failures or for you to add new machines during runtime
How OS Class Helped?

- Master/worker architecture (Client/Server)
- The examples from class with pipes in C
- Blocking/non-blocking send, receive, and broadcast
- Concurrency is handled using MPI and the use of MPI_Barrier allows for synchronization points
- Resource sharing- We use a shared folder for all processes
  - Shared data set: used in read mode (allows concurrency)
  - All process specific files append unique ID
- Scalability—no code changes needed…
  - MPI –n command to specify number of machines
  - TCP sockets only created when needed

Thank you Dr. Null 😊
If Time Travel Existed...

- SVM is a superior classifier...however very slow
  - Instead use a faster, less accurate classifier (e.g. Naïve Bayes)

- Choosing another data set for proof of concept
  - Our banking data set seems to be already refined
    - All features appear to be relevant after our investigation
  - Instead choose a more recent data set with more features that hasn’t been refined by researchers
Conclusion

- Implementation of a parallel DGA to solve the feature selection problem encountered in machine learning classification problems
- Uses of MPICH: message passing, synchronization, resource sharing, and scalability
  - Allowed us to get a feel for distributed parallel computing and made it easier
- Learned that decision of data set is crucial
References


Image/Figure sources:
- http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/
- https://computing.llnl.gov/tutorials/mpi/#Abstract
Thank You