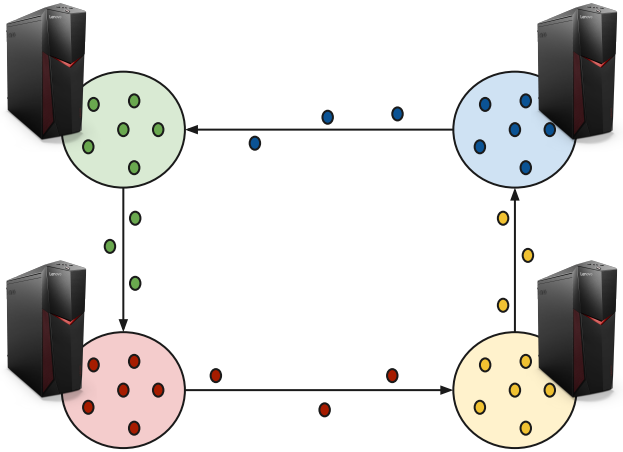
Proyecto Final

Cómputo Concurrente

Garcia Godoy Victor Saul 2183042802
Hernandez Flores Aldo Enrique 2193035908
Ledesma Vazquez Hector Raciel 2193035999
Soto Zarate Luis Alberto 2193035819

Explicación de las partes más importantes

Método Migración



Método Migración.

```
113         if (gen % tamEpoca == 0)
114         {
115             migracion(newpop);
116         }
```

```
268
269 void GeneticoSimple::migracion(Individuo *pop)
270 {
271     // Enviaremos el 1) arreglo x de cada individuo y 2) su aptitud (un solo doble).
272     cout << "Estoy en el metodo de migracion" << endl;
273     int position = 0;
274     vector<int>elegidos(nMigrantes);
275     obtenElegidos(elegidos, nMigrantes); //-------> REVISAR
276     printf("ANDAMOS SALIENDO DE ELEGIDOS");
277     for (int i = 0; i < nMigrantes; i++)
278     {
279         MPI_Pack(pop[elegidos[i]].x.data(), problema->numVariables(), MPI_DOUBLE, buffer, bufSize,
280                  &position, MPI_COMM_WORLD);
281         MPI_Pack(&(pop[elegidos[i]].aptitud), 1, MPI_DOUBLE, buffer, bufSize,
282                  &position, MPI_COMM_WORLD);
283     }
```

Método Migración.

1. Obtener los elegidos: En la primera parte de la migración se hace la llamada a la función ‘obtenElegidos()’, donde se utiliza para obtener el vector de los índices de los individuos que se seleccionan para la migración. Los índices se van a almacenar en el vector de “elegidos” y, el número total de individuos a migrar se especifica mediante el parámetro de “nMigrantes”.

```
void GeneticoSimple::migracion(Individuo *pop)
{
   // Enviaremos el 1) arreglo x de cada individuo y 2) su aptitud (un solo doble).
   cout << "Estoy en el metodo de migracion" << endl;
   int position = 0;
   vector<int>elegidos(nMigrantes);
   obtenElegidos(elegidos, nMigrantes); //-------> REVISAR
```

Método Migración.

2.Yendo al primer bucle, este va a iterar sobre los individuos seleccionados y empaqueta sus datos en la variable del “buffer”. Utilizando la función ‘Pack’ de MPI, que en este caso empaqueta el arreglo x del individuo en la posición de los elegidos en el buffer. En el segundo empaquetado (‘MPI_Pack’), funciona para poder empaquetar la aptitud del individuo en el buffer

```
for (int i = 0; i < nMigrantes; i++)
{
   MPI_Pack(pop[elegidos[i]].x.data(), problema->numVariables(), MPI_DOUBLE, buffer, bufSize,
            &position, MPI_COMM_WORLD);
   MPI_Pack(&(pop[elegidos[i]].aptitud), 1, MPI_DOUBLE, buffer, bufSize,
            &position, MPI_COMM_WORLD);
}
```

Método Migración.

3. Comenzando con el if anidado; estas líneas gestionan el envío y la recepción de datos entre las diferentes islas, esto mediante la comunicación de punto a punto utilizando las funciones Send y Recv de MPI.

Dentro del condición anidada (if-else), este determinará el comportamiento de cada isla dependiendo de su rango entre el comunicador y el número total de islas.

Si “myRank” es igual a 0, la isla envía al buffer al rango siguiente → “myRank+1”, y recibe en el buffer de la última isla → “numIslas-1”.

En caso de que “myRank” sea igual a “numIslas-1”, la isla envía al buffer a la isla 0 y recibe el buffer de la isla anterior → “myRank-1”. En caso de que sea otro valor del rango, la isla envía el buffer al rango siguiente → “myRank+1” y recibe el buffer de la isla anterior → “myRank-1”.

.

Topología de Anillo

```
if (myRank == 0)
{
   MPI_Send(buffer, position, MPI_PACKED, myRank + 1, 0, MPI_COMM_WORLD);
   MPI_Recv(buffer, position, MPI_PACKED, numIslas - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); // provisional
}
else if (myRank == numIslas - 1)
{
   MPI_Send(buffer, position, MPI_PACKED, 0, 0, MPI_COMM_WORLD);
   MPI_Recv(buffer, position, MPI_PACKED, myRank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); // provisional
}
else
{
   MPI_Send(buffer, position, MPI_PACKED, myRank + 1, 0, MPI_COMM_WORLD);
   MPI_Recv(buffer, position, MPI_PACKED, myRank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); // provisional
}
```

Método Migración.

4. Ahora se pasa al paso del desempaquetado, como primer punto volveremos a llamar a la función para obtener a los elegidos, esto para obtener a los individuos seleccionados después de la migración y almacenarlos en el vector de elegidos. Seguido de eso se iguala a 0 la variable “position”, esto para poder desempaquetar los datos.

Se utiliza un bucle para iterar sobre los individuos seleccionados y realizar el desempaquetado de datos; Con la función “Unpack” se desempaqueta el arreglo “x” del individuo en el buffer y restaurarlo en “pop[elegidos[i]].x.data()” y, seguido de eso se vuelve a utilizar el Unpack para desempaquetar la aptitud del individuo y restaurarla en “pop[elegidos[i]].aptitud”.

Por último, llamamos al método “x2cromosoma()” en cada individuo en donde es seleccionado para para realizar alguna operación con el arreglo “x”.

```
obtenElegidos(elegidos, nMigrantes); // -----> REVISAR 271 UNPACK
position = 0;
for (int i = 0; i < nMigrantes; i++)
{
   MPI_Unpack(buffer, bufSize, &position, pop[elegidos[i]].x.data(), problema->numVariables(), MPI_DOUBLE, MPI_COMM_WORLD);
   MPI_Unpack(buffer, bufSize, &position, &(pop[elegidos[i]].aptitud), 1, MPI_DOUBLE, MPI_COMM_WORLD);
   pop[elegidos[i]].x2cromosoma(problema);
}
```

ObtenerElegidos

```cpp
void GeneticoSimple::obtenElegidos(vector<int> &elegidos, int nMigrantes)
{
    cout << "Metodo obtenElegidos" << endl;
    int nElegidos = elegidos.size();
    if (nElegidos > nMigrantes)
    {
        return;
    }

    vector<int> indices(nMigrantes);
    for (int i = 0; i < nMigrantes; i++)
    {
        indices[i] = i;
        // cout << "Arreglo inicial de migrantes: " << indices[i] << endl;
    }

    srand(time(nullptr));

    for (int i = 0; i < nElegidos; i++)
    {
        int aleatorio = rand() % nMigrantes;
        elegidos[i] = indices[aleatorio];
        indices[aleatorio] = indices[nMigrantes - 1];
        nMigrantes--;
    }
```

ObtenerElegidos.

1. Tenemos la condición if, donde indicamos primero que si nElegidos es mayor a nMigrantes la función termina ya que nElegidos debe ser nMigrantes.
2. Creamos una vector llamado índice con tamaño nMigrantes y con el ciclo for lo rellenamos de i < nMigrantes, con valores consecutivos.
3. Inicializamos la semilla para generar números aleatorios utilizando el tiempo actual como semilla → srand(time(nullptr)).
4. El for, declaramos un entero llamado “aleatorio” donde se almacena un número aleatorio en el Rango de nMigrantes. Seguido de eso, se asigna el i-ésimo elemento del vector “elegidos” el valor del elemento correspondiente en el vector “indices” obtenido aleatoriamente.
5. En la siguiente línea intercambiamos el valor del elemento seleccionado aleatorio con el último elemento no seleccionado en el vector “indices” y, por último decrementamos nMigrantes.

Unión

En esta parte final del programa, se realiza la unión de todas las poblaciones de las islas. Todas las islas envían su población a la isla cero y, ésta las coloca en un arreglo global que contendrá la solución final del algoritmo.

Unión.

1. Se inicializa el bufferUnion donde contendrá todas las poblaciones de las demás islas.

2. Si es la isla 0 primero tendrá realizar una copia de su poblacion “oldpop” a la la poblacion global “globlapop”, después tendrá que hacer una copia de la tarea de desempaquetar el vector x que representa los pesos de la red neuronal, la evaluación del individuo “globalpop[i].eval” este cuenta el tiempo de terminación de la carrera y “globalpop[i].cons[0]” que es la distancia restante para terminar la carrera.

Unión.

```
// for fin
int bufSizeUnion = popSize * (problema->numVariables() + 2) * sizeof(double);
char *bufferU = new char[bufSizeUnion];
int fin;
if (myRank == 0)
{
   // copia de oldpop slide 39 "implementacion_final"
   for (fin = 0; fin < popSize; fin++) {
      globalpop[fin] = oldpop[fin];
   }
   // debe recibir la poblacion de todas las islas, menos de la isla cero.
   for (int k = 1; k < numIslas; k++)
   {
      MPI_Recv(bufferU, bufSizeUnion, MPI_PACKED, k, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      int position = 0;
      for (int i=0; i < popSize; i++)
      {
         MPI_Unpack(bufferU, bufSizeUnion, &position, globalpop[fin].x.data(), problema->numVariables(), MPI_DOUB
         MPI_Unpack(bufferU, bufSizeUnion, &position, &(globalpop[fin].eval), 1, MPI_DOUBLE, MPI_COMM_WORLD);
         MPI_Unpack(bufferU, bufSizeUnion, &position, &(globalpop[fin].cons[0]), 1, MPI_DOUBLE, MPI_COMM_WORLD);
         fin++;
      }
```

Unión.

1. Después se crearán los dos archivo “evals_pop.txt y pesos_pob.txt” en la carpeta “salidafinal”. Se escribirá la población global “globalpop” de cada Isla de tamaño numIslas*popsize.

2. En caso de que no ser la isla 0 se entrará en el else del if, en esta parte las islas restantes tendrán la tarea de empaquetar el vector x que representa los pesos de la red neuronal, la evaluación del individuo “oldpop[i].eval” este cuenta el tiempo de terminación de la carrera y “oldpop[i].cons[0]” que es la distancia restante para terminar la carrera.

Unión.

```
    // Para dejar la evaluación (tiempo y distancia restante) de la población.
    ofstream archEvaluacion("./salidafinal/evals_pob.txt", std::ofstream::out);
    cout<<"Imprimiendo archivo  de salida: "<< popSize*numIslas <<endl;
    stats.writeVariables(archVariables, globalpop, popSize*numIslas);
    stats.writeEvaluation(archEvaluacion, globalpop, popSize*numIslas);
    archVariables.close();
    archEvaluacion.close();
    //
}
else
{
    // parametros a enviar. vector c, pop[i].eval y pop[i].cons[0].
    int position = 0;
    for (int i = 0; i < popSize; i++)
    {
        MPI_Pack(oldpop[i].x.data(), problema->numVariables(), MPI_DOUBLE, bufferU, bufSizeUnion,
                 &position, MPI_COMM_WORLD);
        MPI_Pack(&(oldpop[i].eval), 1, MPI_DOUBLE, bufferU, bufSizeUnion,
                 &position, MPI_COMM_WORLD);
        MPI_Pack(&(oldpop[i].cons[0]), 1, MPI_DOUBLE, bufferU, bufSizeUnion,
                 &position, MPI_COMM_WORLD);
    }
    MPI_Send(bufferU, bufSizeUnion, MPI_PACKED, 0, 0, MPI_COMM_WORLD);
}

delete[] bufferU;
```

Es importante notar que se creó un buffer especial para empaquetar y desempaquetar los valores de la población correspondiente a cada isla.

Errores que se presentaron para hacer la implementación

Errores que se presentaron.

1. Instalación incorrecta de la biblioteca MPI. Se instaló la biblioteca mpich, la cual causaba errores en la compilación, se resolvió instalando la biblioteca openmpi-bin.

2. En las líneas en que se realiza MPI_Send, se modificó el parámetro “MPI_ANY_TAG” cambiandolo por el valor de 0.

3. No se había declarado con anterioridad el vector que iba recibir a los elegidos de la función “obtenElegidos”.

4. No se había declarado un segundo buffer donde se almacenan los datos de la unión de todas las islas.

5. Se declaró erróneamente el tamaño del buffer de la unión.

6. Error en la compilación final del proyecto junto con el torcs.

Muestra de la ejecución donde obtuvieron el mejor controlador del auto.

Salida de los archivos “pesos_pob.txt” y “eval_pop.txt”

pesos_pob.txt

```
155.018540 207.810507 -205.285058 -212.328579 -568.178409 -370.293376 360.793217 295.875925 287.025124 -205.812679 560.641884 -719.006070 -54.650157
-657.929197 109.391150 487.567652 -532.989809 226.439429 -198.003968 -360.561231 -532.596443 321.299980 404.489187 -54.474622 -756.275809 -25.624977
-252.130467 621.344568 -165.635695 -168.083013 319.436257 -381.909580 220.200811 -15.752472 688.793150 -413.176596 362.778338 111.964366 -711.643046
167.645584 056.727783 325.652803 419.974812 672.265718 658.696668 -156.305219 -669.950530 072.463422 573.445099 296.541255 -241.632759 051.819613
-647.467064 796.747673 -543.806575 -652.570060 -484.281047 -06.504130 471.929029 -784.992434 487.923635 -617.058881 -576.376652 102.270513 -762.514365
291.852583 -513.451154 402.217508 634.910366 -145.966487 340.706371 -356.362036 -672.226582 025.594202 -396.527726 391.431146 -581.021505 732.054488
-353.958986 -697.047810 -121.306065 041.521900 068.289920 -132.519763 -213.058422 -179.652837 630.037334 569.419145 -589.702485 133.776297 -787.393638
-477.257633 -111.175680 -748.629270 -469.453521 269.715689 -137.708533 686.423121 -241.162926 -117.168908 -558.058395 -326.717292 066.231089 -478.159016
443.117054 -126.993473 267.144278 -104.603588 -522.602204 -344.505678 751.747908 -600.780132 280.616123 502.728566 348.987907 336.379033 522.046500
389.497810 508.025815 -766.770314
```

evals_pob.txt

```
154.766  000000.000
154.786  000000.000
148.730  000000.000
0.000  001645.430
145.876  000000.000
0.000  001627.350
154.786  000000.000
154.786  000000.000
0.000  003210.830
0.000  003210.830
0.000  003030.750
141.948  000000.000
240.648  000000.000
154.756  000000.000
153.818  000000.000
140.084  000000.000
154.766  000000.000
154.662  000000.000
146.492  000000.000
146.470  000000.000
154.766  000000.000
155.626  000000.000
142.798  000000.000
142.770  000000.000
142.770  000000.000
154.766  000000.000
154.766  000000.000
146.594  000000.000
0.000  003210.830
154.624  000000.000
154.714  000000.000
154.780  000000.000
154.766  000000.000
```

Video de muestra de ejecución

A continuación mostramos la ejecución del simulador TORCS con los datos arrojados por el algoritmo genetico_torcs.

La ejecución de la simulación se hizo con el comando siguiente:
./launch_n_th_car.sh ../salidafinal/pesos_pob.txt <n>
donde n es el número de renglón a elegir del archivo “pesos_pob.txt”.

En este caso, la ejecución fue realizada con el renglón 16.