Trabajo Práctico 1

Conway's Game of Life

Organización de computadoras

Participantes	Nº padrón	Mail
CAVAZZOLI, Federico	98533	fcavazzoli@fi.uba.ar
ROCCHI, Tomás Guillermo Camilo	99541	trocchi@fi.uba.com

Índice

1. Introducción

Este trabajo consiste en estudiar como es que funcionan las funciones implementadas en asembler de MIPS32 y su performance. Además se mostrara como es que se estructura la memoria del programa al momento de llamar a las funciones implementadas en asembler.
Para lograr esto se implementó el desarrollo del conocido juego El juego de la vida el cual se encuentra gran parte hecho con C y desde este código se invocara a distintas sentencias implementadas en asembler.

2. Diseño e Implementación

2.1. Análisis de la linea de comandos

La aplicación se ejecuta principalmente de tres formas:

- `./conway i M N arch_entrada <-o archivo_output><-p>` Donde:
  - i: Entero representando la cantidad de iteraciones
  - M: Entero representando la cantidad de filas de la matriz de la simulación
  - N: Entero representando la cantidad de columnas de la matriz de simulación
  - arch_entrada: Ruta del archivo de entrada para la simulación
  - Parámetros opcionales
    La aplicación permite ejectutarse con los siguientes parámetros opcionales:
    -o Output., [-o 'archivo_output'] permite especificarle al programa el nombre de los archivos para generar a la salida. De no especificarse ninguno, el nombre es 'default'.
    -p Print. Permite indicar al programa que no se genere archivo a la salida y que tampoco imprima por salida std (utilizado principalmente para evitar generar muchos archivos durante las pruebas internas / benchmarking).

- `./conway -v`

  -v Flag de version.Devuelve por salida estándar la versión del programa.

- `./conway -h`

  -h Flag de ayuda (help). Devuelve por salida estándar la información básica sobre como ejecutar el programa.

2.2. Desarrollo del código fuente

Para simplificar el desarrollo y el proceso de debugging, al ser un programa no tan complejo, optamos por incluir todo el código fuente junto.

Acorde al enunciado, se deben generar dos versiones del programa: una con soporte para arquitecturas genéricas, hecho completamente en lenguaje C, y otro con soporte para la arquitectura MIPS32, desarrollado también en C, pero con una función (vecinos) desarrollada en assembler para MIPS 32.

Para lograr esto, contamos con un archivo `main.c`, que contiene la aplicación con soporte genérico; el archivo `main_mips.c` que contiene el mismo código fuente, pero utiliza la función en assembler `vecinos`, tomada del archivo `vecinos.S`.

3. Proceso de compilación

Se cuenta con un archivo Makefile que indica las reglas de compilación ejecutadas por el comando make.

El código fuente se compila ejecutando el comando make con alguna de las posibles reglas. En todos los casos se compila con los flags de warnings y de debbugging prendidos.

make Compila todo el programa hecho en lenguaje C

make mips Compila todo el programa hecho en C con la función 'vecinos' implementada en assembler

make clean Borra todos los archivos compilados del directorio y también los archivos .pbm generados por el programa

make opt1 Compila el programa usando gcc con nivel de optimización 1

make opt2 Compila el programa usando gcc con nivel de optimización 2

make opt3 Compila el programa usando gcc con nivel de optimización 3

make opt1_mips Compila el programa usando gcc con nivel de optimización 1 con la función vecinos implementada en assembler MIPS

make opt2_mips Compila el programa usando gcc con nivel de optimización 2 con la función vecinos implementada en assembler MIPS

make opt3_mips Compila el programa usando gcc con nivel de optimización 3 con la función vecinos implementada en assembler MIPS

4. Portabilidad

Dado que se utilizó el lenguaje de programación C, sin hacer uso de funciones específicas de sistemas operativos, o de bibliotecas comerciales, los programas

pueden ser compilados en varios de ellos. De todas formas, el Makefile presentado fue hecho particularmente para sistemas de tipo UNIX.

Los casos de prueba presentados fueron llevados a cabo en una arquitectura MIPS emulada. El programa de emulación utilizado fue QEMU y el sistema operativo en la maquina emulada fue Debian MIPS stretch.

Los programas también fueron probados en Ubuntu-Linux corriendo en una arquitectura Intel x86 satisfactoriamente.

5. Casos de prueba

Para realizar las pruebas de esta aplicación se utilizaron 3 archivos bases. Donde cada uno indica como es inicializada la matriz del programa.

Cada linea del archivo corresponde con una coordenada de una fila y una columna en este respectivo orden, se asume que el valor inicial de la matriz es 0 y solo se establecerán en valor 1 aquellos indicados por el archivo en cuestión.

A continuación se procede a indicar el contenido de cada archivo y su respectivo resultado.

Pento:

X	Y
3	5
3	6
4	4
3	6
4	5
5	5

Cuadro 1: Tabla de estados iniciales en 1

Lo que nos genera un tablero inicial tal como se ve en la siguiente imagen.

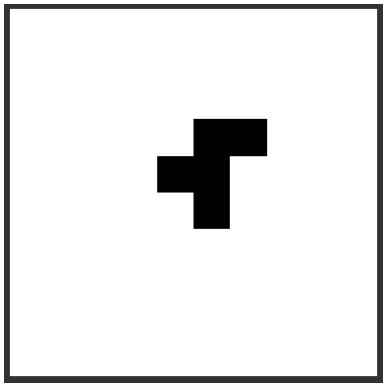


Figura 1: Inicialización Pento

Sapo:

X	Y
5	3
5	4
5	5
4	4
4	5
5	6

Cuadro 2: Tabla de estados iniciales en 1

Lo que nos genera un tablero inicial tal como se ve en la siguiente imagen.

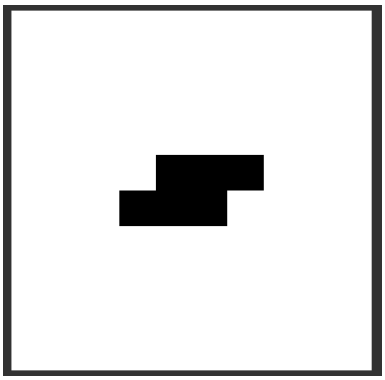


Figura 2: Inicialización Sapo

Glider:

X	Y
5	3
5	4
5	5
3	4
4	5

Cuadro 3: Tabla de estados iniciales en 1

Lo que nos genera un tablero inicial tal como se ve en la siguiente imagen.

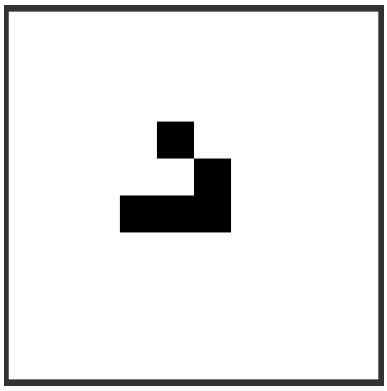


Figura 3: Inicialización Glider

5.1. Funcionamiento de la aplicación

A continuación se procederá a describir a grandes rasgos el algoritmo implementado.

Como toda aplicación en código C se cuenta con una función main la cual, en este caso, se encarga del flujo general del programa.

En un primer momento se llama a la función 'validar_datos' la cual establece si se tienen los parámetros mínimos para la ejecución del programa, de no ser así se interrumpe la ejecución con un mensaje de error por salida stderr amigable al usuario.

Si los parámetros de ejecución son validos se pasa a inicializar las variables locales con los mismos y crear el mapa del juego. Para luego dar paso al bucle principal del programa.

Llegado este punto se entra en un bucle while en el que en un primer momento se verifica el modo de ejecución de la aplicación, de modo que si se es ejecutada con el flag -p, esta no generara una salida por std ni creara archivos .pbm.

Luego se procede a avanzar de un estado del mapa a otro. Esto se logra mediante la llamada a la función 'avanzar'. La función avanzar contiene toda la lógica del juego, sus reglas y estados. Para su funcionamiento, la función 'avanzar' crea un segundo mapa a fin de no hacer una modificación in-place del mapa original.

Se recorre cada celda del mapa utilizando una doble iteración y por cada una de las celdas se llama a la función 'vecinos' la cual se encarga de verificar el estado de las 8 celdas aledañas a la celda indicada. Esta función devuelve la cantidad exacta de vecinos 'vivos' que tiene una celda para que la función avanzar se encargue de aplicar las reglas de supervivencia, muerte o resurrección sobre la misma.

La función vecinos se encuentra implementada tanto en lenguaje C como en lenguaje assembler de MIPS y posee una firma de la forma:

```
int vecinos(unsigned char *mapa, int x, int y,unsigned int filas,unsigned int cols);
```

El mapa esta representado por un array de chars, es decir una sucesión de bloques de 8bits cada uno. Este array representa una matriz donde sus índices son mapeados de la siguiente manera (suponiendo una matriz de 3x3):

0	1	2
3	4	5
6	7	8

Cuadro 4: Matriz del tablero

0	1	2	3	4	5	6	7	8

Cuadro 5: Matriz del tablero representada en forma de array

Para obtener el número de celda correspondiente a una posición [f,c], podemos simplemente utilizar la siguiente fórmula:

$$pos = f * cant\_cols + c \tag{1}$$

Esto se debe a que avanzamos f veces $cant\_cols$ para llegar hasta la celda de inicio de la fila deseada y nos movemos c lugares sobre esa fila para llegar a la posición buscada.

Una vez actualizado el estado del mapa en el bucle while principal, a menos que sea indicado lo contrario se llama a la función 'dump' cuyo único propósito es imprimir por salida std el estado del mapa como así también en el archivo de salida .pbm para que se pueda ver una imagen del tablero una iteración a la vez.

5.2. Funcionamiento de Vecinos

Como se dijo anteriormente, la función vecinos se encarga de devolver la cantidad de celdas adyacentes a una celda específica que están prendidas/vivas. Para hacer esto, se parte de una posición base `[x,y]` y se procede a recorrer con un doble ciclo, las 8 celdas aledañas a la misma. Por cada celda viva visitada, se incrementa el valor de un contador, y al finalizar las iteraciones, se devuelve el valor de este contador, indicando la cantidad de vecinos vivos.

Como se explicita en el enunciado, se debe tomar una hipótesis sobre los ejes del mapa. En nuestro caso, optamos por elegir la 'Hipótesis del mundo Toroidal', en la cual si nos paramos en el borde del mapa, y procedemos a salir del mismo, 'saltamos' al extremo opuesto del mismo. Para poder implementar esta solución, utilizamos el operador módulo `%` que permite obtener el resto de una división. Como sabemos, el resto de `a/b` resulta un número entre 0 y b - 1. De esta forma, si nos 'pasamos' de un extremo del mapa, volveremos al incio. Sin embargo, también puede pasar que nos encontremos 'antes del inicio' o en una

posición negativa del mapa. En muchos compiladores, el resultado de aplicarle módulo a un número negativo es indeterminado, por lo tanto optamos por incluir un paso previo donde chequeamos la negatividad del número en cuestión y, de ser negativo, es mappeado al extremo contrario del mapa.

5.3. Estructura del stack de Vecinos

A continuación se muestra la forma que tiene el stack para la función vecinos cuando esta es implementada en assembler MIPS, siguiendo la ABI presentada por el curso:

A3	Stack Padre [Argument Building Area]
A2	Stack Padre [Argument Building Area]
A1	Stack Padre [Argument Building Area]
A0	Stack Padre [Argument Building Area]
gp	Stack Propio [Saved Registers Area]
fp	Stack Propio [Saved Registers Area]
LTA_CONT	Stack Propio [Local Temporary Area]
LTA_I	Stack Propio [Local Temporary Area]
LTA_J	Stack Propio [Local Temporary Area]
LTA_X	Stack Propio [Local Temporary Area]
LTA_Y	Stack Propio [Local Temporary Area]
LTA_POS	Stack Propio [Local Temporary Area]

Acorde a lo indicado por nuestra ABI, la función `vecinos` posiciona los primeros 4 argumentos de la función (mapa, x, y , filas) en el ABA de su padre, localizado inmediatamente sobre su propio stack. El 5to parámetro (cols) es seteado por el caller en la posición inmediatamente superior a A3 y levantado directamente desde el código de `vecinos`

Por otro lado, notamos que la función es una función hoja, por lo tanto su región de Saved Registers solamente contiene los registros `fp (frampe pointer )` y `gp (global pointer)`.

Por ultimo, vemos que contamos con 6 variables temporales en nuestra función:

- contador [LTA_CONT]: Variable utilizada para contar la cantidad de vecinos vivos.
- i [LTA_I]: Variable utilizada como índice del ciclo externo.
- j [LTA_J]: Variable utilizada como índice del ciclo interno.
- _x [LTA_X]: Variable utilizada para almacenar la posición corregida de x luego de desplazarse.
- _y[LTA_Y]: Variable utilizada para almacenar la posición corregida de y luego de desplazarse

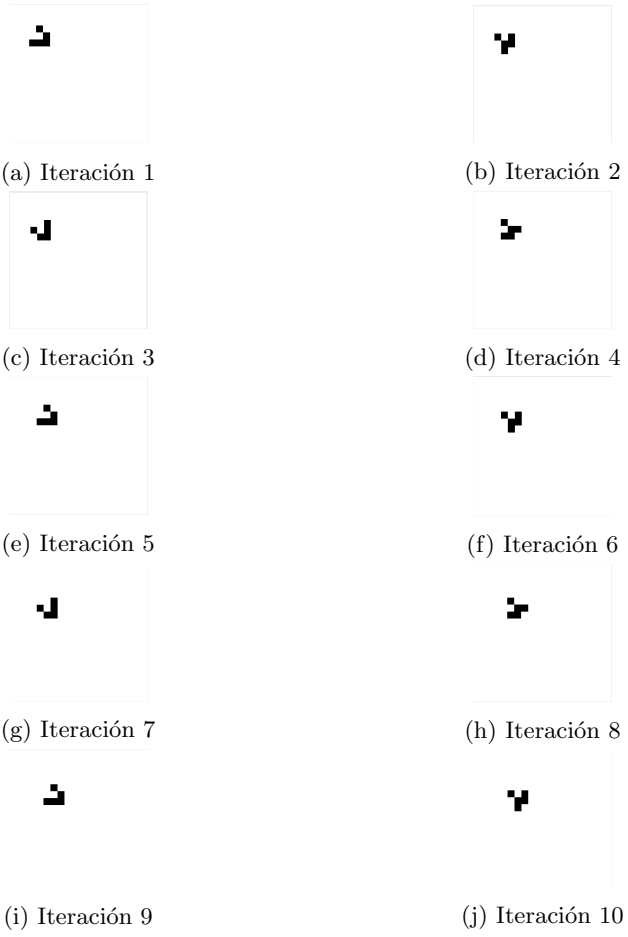
- posición [LTA_POS]: Variable utilizada para almacenar la posición calculada de un vecino de la celda

6. Resultados

6.1. Corridas de prueba

A continuación se muestra la evolución de la matriz una iteración a la vez para cada uno de los archivos de entrada. La matriz en cada caso es de 20 filas por 20 columnas.

Glider:



(a) Iteración 1 (b) Iteración 2

(c) Iteración 3 (d) Iteración 4

(e) Iteración 5 (f) Iteración 6

(g) Iteración 7 (h) Iteración 8

(i) Iteración 9 (j) Iteración 10

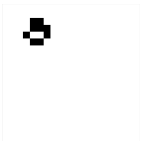
Figura 4: Evolución de Glider

Pento:

(a) Iteración 1

(b) Iteración 2

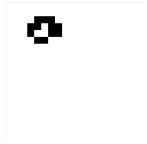
(c) Iteración 3



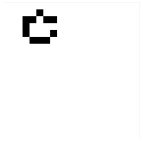
(d) Iteración 4

(e) Iteración 5

(f) Iteración 6

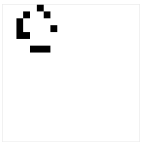


(g) Iteración 7



(h) Iteración 8

(i) Iteración 9



(j) Iteración 10

Figura 5: Evolución de Pento

Sapo:

(a) Iteración 1 (b) Iteración 2

(c) Iteración 3 (d) Iteración 4

(e) Iteración 5 (f) Iteración 6

(g) Iteración 7 (h) Iteración 8

(i) Iteración 9 (j) Iteración 10

Figura 6: Evolución de Sapo

6.2. Resultados de performance

Para esta sección, se procederá a mostrar los resultados de dos maneras distintas.

El primer caso de estudio es una comparativa entre los distintos niveles de optimización de `gcc` y el programa optimizado con `-o0` pero con la función vecinos implementada en assembler de MIPS.

Para este análisis se realizaron 3 ejecuciones distintas del programa, 100 iteraciones del tablero, luego 1.000 iteraciones, luego 10.000 y finalmente 100.000. A fines de no romper con la escala del gráfico, la ejecución con 100.000 fue puesta en un gráfico a parte dado que sus valores son mas de 10000 % mayores a los que se observan en 100 corridas.

Para todos los casos se utilizo la misma matriz de entrada de tamaño de 10

filas por 10 columnas y los valores iniciales corresponden a los indicados por el archivo *glider*

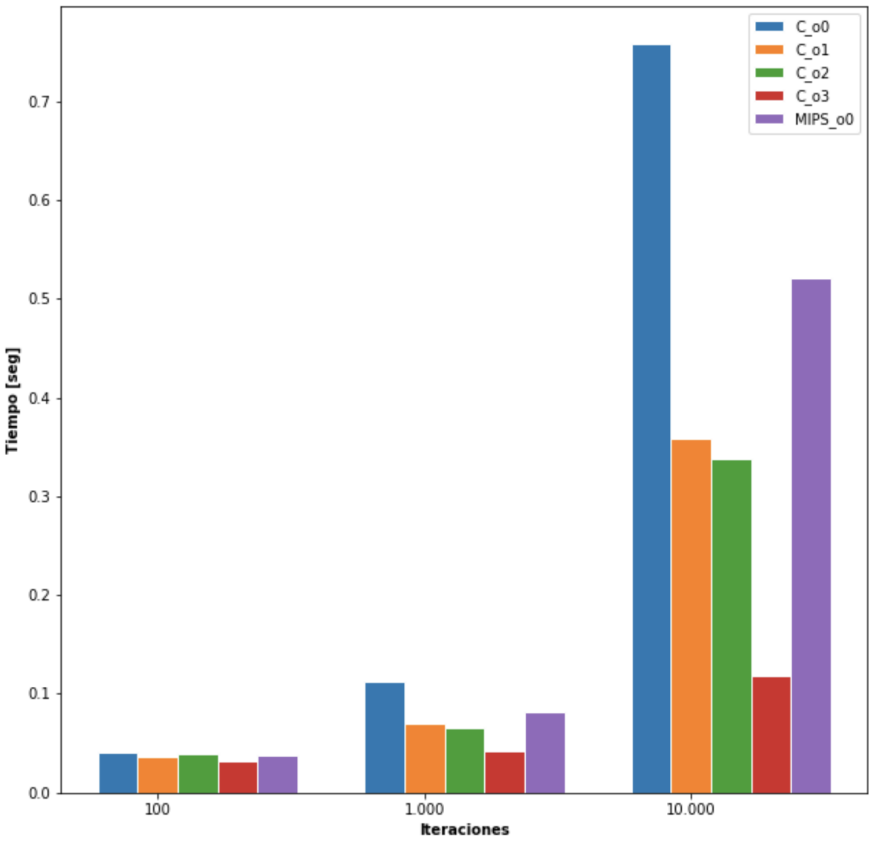


Figura 7: Comparativa de optimizaciones según la cantidad de iteraciones

Se puede ver claramente como los programas comienzan a mostrar una mejora significativa en la performance conforme se aumenta el nivel de optimización al momento de compilación. Puede verse claramente como con una optimización de nivel 0 el programa demora el doble de tiempo en ejecutarse que cuando se ejecuta con optimización o3. Además se puede agregar que la implementación en MIPS es mejor en cada uno de los casos versus la versión con optimización o0 de su contra parte completamente en C. Sin embargo no resulta lo suficientemente eficiente como para superar a las versiones o1, o2 y o3.

También se realizo una prueba en la cual se llevo el numero de accesos a la matriz al extremo respecto de las pruebas anteriores, se realizo una comparación entre cada una de las optimizaciones aunque para solo un valor de iteraciones puesto que cada corrida del programa demora aproximadamente unos 10 minutos utilizando un procesador i7 a 3GHz. La prueba consiste en ejecutar el programa con una matriz de 100 filas por 100 columnas y 100.000 iteraciones. Esto resulta en la siguiente cantidad de accesos a la tabla:

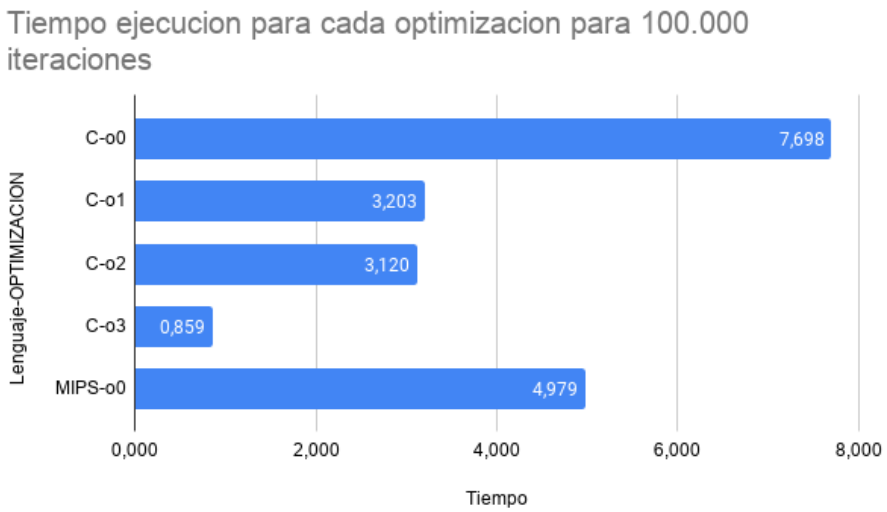


Figura 8: Comparativa de optimizaciones para 100.000 iteraciones

$$cantidad\_de\_accesos = 100x100x100{,}000 = 1{,}000{,}000{,}000 = 1x10^9 accesos \quad (2)$$

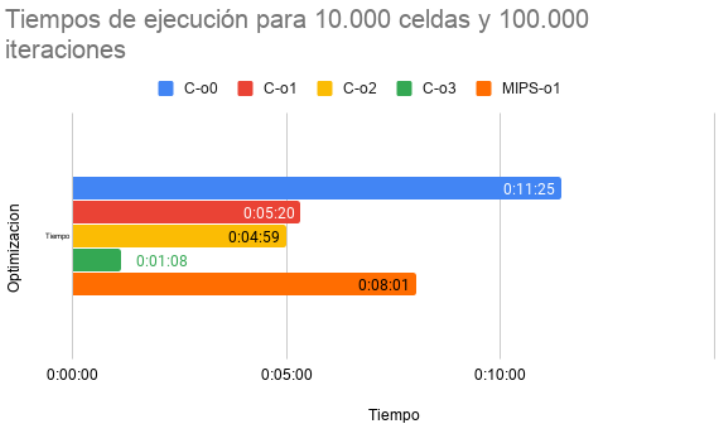


Figura 9: Comparativa de optimizaciones para 100.000 iteraciones con una matriz de 100x100

Se puede observar como nuevamente la optimización o0 resulta la menos performante de todas siendo el tiempo de ejecución 100 % mayor que la optimización o3

Además se incluyo un análisis de las distintas optimizaciones en C comparadas directamente con ese mismo nivel de optimización pero en la versión que incluye la implementación de `vecinos` en Assembler MIPS.

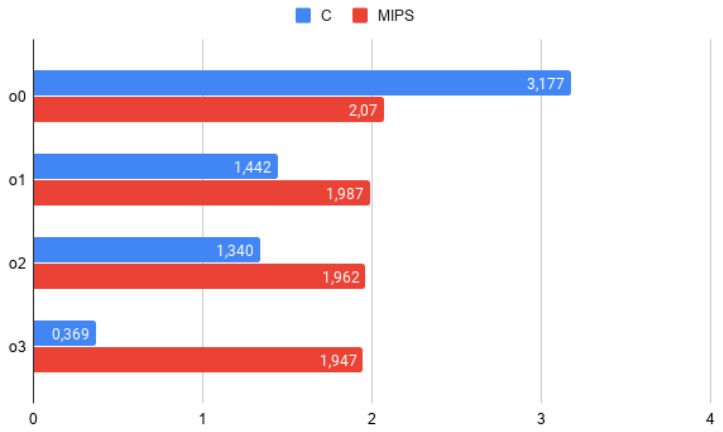


Figura 10: Comparativa de las implementaciones para cada nivel de optimización disponible

Para realizar este análisis se corrió el programa (en la versión de compilación correspondiente) con los parámetros 10.000 20 20 glider. Lo que indica que el valor inicial de la matriz es lo que indique el archivo glider, se corren 10.000 iteraciones y la matriz tiene un tamaño de 20 filas por 20 columnas. Podemos observar como sin optimizaciones, la versión con `vecinos` implementada en MIPS reduce 1 segundo el tiempo de ejecución con respecto a su contraparte en C.

A partir de la primera optimización en adelante, se ve muy poca ganancia en la versión de MIPS (menor a 0.1 % por nivel de optimización), pero una gran ganancia en la versión en C, reduciendo los tiempos de ejecución en un 50 % para o1, y 90 % para o3.

7. Conclusión

En este trabajo practico se estudio el desempeño un mismo algoritmo compilado con distintos niveles de optimización y además una implementación del mismo hecha en assembler MIPS.

Referente a los resultados de performance obtenidos para cada una de las optimizaciones y la implementación en MIPS se puede afirmar que el compilador gcc sin ningún grado de optimización (es decir optimización -o0) resulta el menos performante de todas las opciones ya que es la opción que mas tiempo demoró en ejecutar la totalidad del programa. Esto se debe a que el flag -o0 no optimiza absolutamente ninguna instrucción de C, el mismo realiza una traducción plana sin tener en cuenta la cantidad de accesos a memoria ni tampoco intenta reducir el tamaño del programa. Este resultado se observo en cada una de las opciones de ejecución que fueron planteadas.

La optimización o1 resulto ser claramente mas performante que o0 (y la implementación en MIPS que sera discutida mas adelante) el primer nivel de optimización puede justificarse luego de ver detenidamente como es que el stack de la función vecinos es armado para cada una de las optimizaciones. En el caso de o0 el stack de vecinos es 40b bytes, tiene 134 lineas y su implementación involucra un doble loop (observable en el archivo main0.s), lo que se puede ver que conlleva muchos mas accesos a memoria que o1. Respecto al stack de o1 puede decirse que ocupa 16 bytes, significativamente menor que el stack de o0 (que es mas del doble).

De los resultados podemos observar como al optimizar nuestra version con la función `vecinos` implementada en MIPS32 ganamos performance, pero la misma es mínima con respecto a la optimización del programa con la función implementada en C.

Esto se lo atribuímos a que las optimizaciones de la versión MIPS optimizan todo menos la función `vecinos`, la cual es la función más ejecutada del programa. Las optimizaciones realizadas por gdb sobre la función son claramente mas performantes que la versión desarrollada 'a mano' por nosotros. Esto es un claro ejemplo de la Ley de Amdahl donde se puede ver que optimizando el caso mas frecuente, se obtiene una gran mejora de performance.

8. Código fuente

8.1. Código C

```
    #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>

const int MAPA = 0;
const int ITER = 1;
const int FILS = 2;
const int COLS = 3;
const int INPUT_FILE = 4;
const int OPTION = 5;
const int OUT_FILE = 6;

unsigned char* crear_mapa(int filas, int cols) {
  unsigned char *mapa = calloc(filas, cols *sizeof(unsigned char));
  return mapa;
}

int es_numerico(char* param){
  size_t len = strlen(param);
  for (int i=0; i<len; i++){
    if(!isdigit(param[i])){
      return -1;
    }
  }
  return 0;
}

int cargar_mapa(unsigned char* mapa, unsigned int filas, unsigned int cols, char* filename) {
  FILE* archivo = fopen(filename, "r");
  if(archivo != NULL) {
    char linea[256];
    while(fgets(linea, sizeof(linea), archivo)) {
      char* input_f = strtok(linea, " ");
      char* input_c = strtok(NULL, "\n");
      if (!(es_numerico(input_f) == 0 && es_numerico(input_c) == 0)) {
        fprintf(stderr, "Error en el archivo de entrada. La celda [%s, %s] contiene caracteres
        return -1;
      }
      int f = atoi(input_f);
      int c = atoi(input_c);
      if (f < 0 || c < 0 || f >= filas || c > cols) {
        fprintf(stderr, "Error en el archivo de entrada. Celda [%d, %d] fuera del mapa\n", f, c
        fclose(archivo);
```

```
        return -1;
      }
      unsigned int posicion = c + f * cols;
      mapa[posicion] = 1;
    }
  } else {
    fprintf(stderr, "Error abriendo el archivo de entrada");
    return -1;
  }
  fclose(archivo);
  return 0;
}

int vecinos(unsigned char *mapa, int x, int y,unsigned int filas,unsigned int cols){
  unsigned int contador = 0;
  for(int i = -1; i <= 1; i++) {
    for (int j = -1; j <= 1; j++) {

      int _x = (x + i);
      _x = _x < 0 ? filas - 1 : _x%filas;
      int _y = (y + j);
      _y = _y < 0 ? cols - 1 : _y%cols;

      unsigned int pos = _y + _x * cols;

      if (!(i == 0 && j == 0)) contador += mapa[pos] ? 1 : 0;
    }
  }
  return contador;
}

void avanzar(unsigned char *mapa, unsigned int filas,unsigned int cols){
  unsigned char* mapa_tmp = crear_mapa(filas, cols);

  for(int i = 0; i < filas; i++) {
    for (int j = 0; j < cols; j++) {
      unsigned int pos = j + i * cols;
      unsigned int cant_vecinos = vecinos(mapa, i,j, filas, cols);
      int vive = mapa[pos] ? (cant_vecinos == 3 || cant_vecinos == 2) : cant_vecinos
      mapa_tmp[pos] = vive;
    }
  }
  for(int x = 0; x < cols * filas; x++) {
      mapa[x] = mapa_tmp[x];
  }

  free(mapa_tmp);
}

void dump(unsigned char *mapa,unsigned int filas,unsigned int cols, FILE* pgming) {
```

```
  /*Volcar Mapa a archivo*/

  for(int i = 0; i < filas; i++) {
    for (int j = 0; j < cols; j++) {
      int pos = j + i * cols;
      printf("%d ", mapa[pos]);
      fprintf(pgming, "%d ", mapa[pos]);
      if (j == cols - 1){
        printf("\n");
        fprintf(pgming, "\n");
      }
    }
  }
  printf("\n");
  printf("\n");
}

int validar_datos(int argc, char** argv){

  int opt;
  if (argc == 2){
    while((opt = getopt(argc, argv, "hv")) != -1) {
    switch(opt){
        case 'h':
          printf("-V Da la version del programa. -o Prefijo de los archivos de salida.\n Ejempl
          return -1;
        case 'v':
          printf("Version 1.0.0\n");
          return -1;
        default:
          fprintf(stderr, "Parametros incorrectos, use -h o -v si usa un solo argumento.\n");
          return -1;
      }
    }
  }
  if (argc < 4){
    fprintf(stderr, "Argumentos insuficientes, usar flag -h para ver paramentros obligatorios\n
    return -1;
  }
  for (int i=1; i<4; i++) {
    if (es_numerico(argv[i]) != 0 || atoi(argv[i]) <= 0) {
      fprintf(stderr, "Parametros incorrectos, las iteraciones y el tamaño de la matriz deben s
      return -1;
    }
  }
  return 0;
}


void set_filename(char* filename, char** argv, int argc, int iter){
```

```
  if(argc>=7){
    if (strcmp("-o", argv[OPTION])==0){
      strcpy(filename, argv[OUT_FILE]);
    }
  }
  else{
    // Uso nombre default
    strcpy(filename, argv[INPUT_FILE]);
  }

  //Agrego el numero de corrida
  char corrida[10];
  sprintf(corrida, "_%d", iter);

  strcat(filename, corrida);
  // Agrego la extension
  strcat(filename,".pbm");
}

int main(int argc, char** argv){
  if(validar_datos(argc,argv)<0){
    return -1;
  }

  unsigned int num_iter = atoi(argv[ITER]);
  unsigned int filas = atoi(argv[FILS]);
  unsigned int cols = atoi(argv[COLS]);
  // Construir mapa
  unsigned char* mapa = crear_mapa(filas, cols);

  // abrir archivo
  char* filename = argv[INPUT_FILE];

  if(cargar_mapa(mapa, filas, cols, filename) < 0) {
    fprintf(stderr, "Error cargando mapa. Cerrando programa...\n");
    free(mapa);
    return -1;
  }

  // Correr
  FILE* pgming;
  int k = 0;
  char archivo_salida[500];
   while (k < num_iter) {
    int verbose = 1;
    if(argc == 6 ){
      verbose = strcmp(argv[OPTION],"-p");
    }
    if (verbose != 0){
      // Crear archivo de salida
```

```
      set_filename(archivo_salida, argv, argc, k + 1);
      pgming = fopen(archivo_salida, "wb");                 //write the file in binary mode

      // Formateo el achivo de salida
      fprintf(pgming, "P1\n");                              // Writing Magic Number to the File
      fprintf(pgming, "%d %d\n", cols, filas);              // Writing Width and Height into th
      fprintf(pgming, "\n");
      dump(mapa, filas, cols, pgming);
      fclose(pgming);
    }

    avanzar(mapa, filas, cols);
    k++;
  }

  // Limpiar
  free(mapa);
  return 0;
}
```

8.2. Version con Extern

```
    #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>

extern int vecinos(unsigned char *mapa, int x, int y,unsigned int filas,unsigned int cols);

unsigned char* crear_mapa(int filas, int cols) {
  unsigned char *mapa = calloc(filas, cols *sizeof(unsigned char));
  return mapa;
}

int es_numerico(char* param){
  size_t len = strlen(param);
  for (int i=0; i<len; i++){
    if(!isdigit(param[i])){
      return -1;
    }
  }
  return 0;
}

int cargar_mapa(unsigned char* mapa, unsigned int filas, unsigned int cols, char* filename) {
  FILE* archivo = fopen(filename, "r");
  if(archivo != NULL) {
    char linea[256];
```

```
    while(fgets(linea, sizeof(linea), archivo)) {
      char* input_f = strtok(linea, " ");
      char* input_c = strtok(NULL, "\n");
      if (!(es_numerico(input_f) == 0 && es_numerico(input_c) == 0)) {
        fprintf(stderr, "Error en el archivo de entrada. La celda [%s, %s] contiene c
        return -1;
      }
      int f = atoi(input_f);
      int c = atoi(input_c);
      if (f < 0 || c < 0 || f >= filas || c > cols) {
        fprintf(stderr, "Error en el archivo de entrada. Celda [%d, %d] fuera del map
        fclose(archivo);
        return -1;
      }
      unsigned int posicion = c + f * cols;
      mapa[posicion] = 1;
    }
  } else {
    fprintf(stderr, "Error abriendo el archivo de entrada");
    return -1;
  }
  fclose(archivo);
  return 0;
}

void avanzar(unsigned char *mapa, unsigned int filas,unsigned int cols){
  unsigned char* mapa_tmp = crear_mapa(filas, cols);

  for(int i = 0; i < filas; i++) {
    for (int j = 0; j < cols; j++) {
      unsigned int pos = j + i * cols;
      unsigned int cant_vecinos = vecinos(mapa, i,j, filas, cols);
      int vive = mapa[pos] ? (cant_vecinos == 3 || cant_vecinos == 2) : cant_vecinos
      mapa_tmp[pos] = vive;
    }
  }
  for(int x = 0; x < cols * filas; x++) {
      mapa[x] = mapa_tmp[x];
  }

  free(mapa_tmp);
}

void dump(unsigned char *mapa,unsigned int filas,unsigned int cols, FILE* pgming) {
  /*Volcar Mapa a archivo*/

  for(int i = 0; i < filas; i++) {
    for (int j = 0; j < cols; j++) {
      int pos = j + i * cols;
      printf("%d ", mapa[pos]);
```

```
      fprintf(pgming, "%d ", mapa[pos]);
      if (j == cols - 1){
        printf("\n");
        fprintf(pgming, "\n");
      }
    }
  }
  printf("\n");
  printf("\n");
}

int validar_datos(int argc, char** argv){

  int opt;
  if (argc == 2){
    while((opt = getopt(argc, argv, "hv")) != -1) {
    switch(opt){
        case 'h':
          printf("-V Da la version del programa. -o Prefijo de los archivos de salida.\n Ejempl
          return -1;
        case 'v':
          printf("Version 1.0.0\n");
          return -1;
        default:
          fprintf(stderr, "Parametros incorrectos, use -h o -v si usa un solo argumento.\n");
          return -1;
      }
    }
  }
  if (argc < 4){
    fprintf(stderr, "Argumentos insuficientes, usar flag -h para ver paramentros obligatorios\n
    return -1;
  }
  for (int i=1; i<4; i++) {
    if (es_numerico(argv[i]) != 0 || atoi(argv[i]) <= 0) {
      fprintf(stderr, "Parametros incorrectos, las iteraciones y el tamaño de la matriz deben s
      return -1;
    }
  }
  return 0;
}


void set_filename(char* filename, char** argv, int argc, int iter){
  if(argc>=7){
    if (strcmp("-o", argv[5])==0){
      strcpy(filename, argv[6]);
    }
  }
  else{
```

```
    // Uso nombre default
    strcpy(filename, "default");
  }

  //Agrego el numero de corrida
  char corrida[10];
  sprintf(corrida, "_%d", iter);

  strcat(filename, corrida);
  // Agrego la extension
  strcat(filename,".pbm");
}

int main(int argc, char** argv){
  if(validar_datos(argc,argv)<0){
    return -1;
  }

  unsigned int num_iter = atoi(argv[1]);
  unsigned int filas = atoi(argv[2]);
  unsigned int cols = atoi(argv[3]);
  // Construir mapa
  unsigned char* mapa = crear_mapa(filas, cols);

  // abrir archivo
  char* filename = argv[4];

  if(cargar_mapa(mapa, filas, cols, filename) < 0) {
    fprintf(stderr, "Error cargando mapa. Cerrando programa...\n");
    free(mapa);
    return -1;
  }

  // Correr
  FILE* pgming;
  int k = 0;
  char archivo_salida[500];
  while (k < num_iter) {
    int verbose = 1;
    if(argc == 6 ){
      verbose = strcmp(argv[5],"-p");
    }

    if (verbose != 0){
      // Crear archivo de salida
      set_filename(archivo_salida, argv, argc, k);
      pgming = fopen(archivo_salida, "wb");                //write the file in binar

      // Formateo el achivo de salida
      fprintf(pgming, "P1\n");                             // Writing Magic Number t
```

```
      fprintf(pgming, "%d %d\n", cols, filas);              // Writing Width and Height into th
      fprintf(pgming, "\n");
      dump(mapa, filas, cols, pgming);
      fclose(pgming);
    }

    avanzar(mapa, filas, cols);
    k++;
  }

  // Limpiar
  free(mapa);
  return 0;
}
```

8.3. Código Assembler MIPS

```
#include<sys/regdef.h>

#define SS 32

/* SRA */
#define O_GP 28
#define O_FP 24

/* LTA */
#define O_LTA_CONT 0
#define O_LTA_I 4

#define O_LTA_J 8
#define O_LTA_X 12

#define O_LTA_Y 16
#define O_LTA_POS 20

/* ABA */
#define O_A0 (SS)
#define O_A1 (O_A0 + 4)
#define O_A2 (O_A1 + 4)
#define O_A3 (O_A2 + 4)

/* Extra Regs*/
#define O_A4 (SS + 16)


#define END_LOOP 2
```

```
.text
.align 2


.globl vecinos
.ent vecinos

vecinos:
    subu sp, sp, SS

    /* Saved Register Area 8B */
    sw   gp, O_GP(sp)
    sw   fp, O_FP(sp)
    move fp, sp

    /* Store in Argument Building Area */
    sw   a0, O_A0(fp)
    sw   a1, O_A1(fp)
    sw   a2, O_A2(fp)
    sw   a3, O_A3(fp)

    /* Local and temporary area 16B (2 int, 2 uint) */
    /* contador = 0*/
    li t0, 0
    sw t0, O_LTA_CONT(fp)

    /* i = -1*/
    li t0, -1
    sw t0, O_LTA_I(fp)

    /* j = -1*/
    li t0, -1
    sw t0, O_LTA_J(fp)

    /* _x */
    li t0, 0
    sw t0, O_LTA_X(fp)

    /* _y */
    li t0, 0
    sw t0, O_LTA_Y(fp)

    /* pos */
    li t0, 0
    sw t0, O_LTA_POS(fp)

    /* Planteo de loop */

    lw t0, O_LTA_I(fp)   # i = -1
    li t1, END_LOOP   # i_j_fin = 2
```

```
start_loop_1:
beq t0, t1, end_loop_1
lw t2, O_LTA_J(fp)   # j = -1

start_loop_2:
   beq t2, t1, end_loop_2

/* ========= CUERPO LOOP ========== */
_x:
lw t3, O_LTA_X(fp)   # t3 = _x
lw t4, O_A1(fp) # t4 = x
lw t5, O_A3(fp) # t5 = filas
add t3, t4, t0       # _x = x + i

bltz t3, jump_map_x

divu t3, t5
mfhi t3 #_x = _x % filas
b _y

jump_map_x:
subu t3, t5, 1 # _x = filas - 1
b _y


_y:
lw t6, O_LTA_Y(fp) # t6 = _y
lw t7, O_A2(fp) # t7 = y
lw t8, O_A4(fp) # t8 = cols
add t6, t7, t2      # _y = y + j

bltz t6, jump_map_y

divu t6, t8
mfhi t6 #_y = _y % cols
b pos

jump_map_y:
subu t6, t8, 1 # _y = cols - 1
b pos

pos:
multu t3, t8
mflo t9 # pos = _x * cols
add t9, t9, t6 # pos = pos + _y

mapa:
lw t4, O_A0(fp) # libero la ref a x (t4) y t4 = mapa
or t5, t0, t2 # libero la ref a filas (t5) y t5 = i | j
```

```
beq t5, 0, end_loop_body # si estoy en la celda actual salteo al fin del loop

addu t7, t4, t9 # t7 = mapa + pos
lbu t7, 0(t7) # t7 = *t7

beq t7, 0, end_loop_body # si la celda vecina está apagada, no la cuento como vecino

lw t7, O_LTA_CONT(fp)
addi t7, t7, 1 # contador ++
sw t7, O_LTA_CONT(fp)    # persisto contador para futuras iteraciones



/* ========= FIN CUERPO LOOP ========== */

end_loop_body:

   addi t2, t2, 1    # j++
   b start_loop_2

end_loop_2:
addi t0, t0, 1    # i++
b start_loop_1

end_loop_1:
    /* return contador */
    lw v0, O_LTA_CONT(fp)

    /* Stack unwinding */
    lw fp, O_FP(sp)
    lw gp, O_GP(sp)
    addiu sp, sp, SS

    jr ra
.end vecinos
```