Relatório de Entrega de Trabalho 1 da disciplina de Programação Paralela

Prof. Marcelo Veiga Neves
Giovanni Caprio (pp12813) e Guilherme Prates

Implementação

O presente trabalho consiste na implementação do jogo Conway's Game Of Life (Joga da Vida) em linguagem C de duas formas distintas: uma serial e outra paralela através das diretrizes OpenMP. O objetivo dessa atividade é avaliar o ganho de desempenho em um nodo multiprocessado do cluster Grad de um programa paralelo. Primeiramente foi definido que o problema alvo será o especificado pelo enunciado do trabalho: o jogo Conway's Game Of Life. Para tanto, foi utilizado uma versão serial pronta do jogo que foi desenvolvida por Ali Ahmed[1].

Apesar de pronto, o código base utilizado para a versão sequencial do trabalho foi alterado. O programa de Ahmed possui uma apresentação do tabuleiro pouco intuitiva, bem como possui em meio ao código algumas estruturas que o tornavam lento (*printf, sleep* e funções de desenho do tabuleiro) que foram removidas e/ou adaptadas para a versão final utilizada. O tabuleiro do código serial, que posteriormente será transformado em paralelo, possui 175 linhas com 40 colunas. Além disso, uma das características dessa implementação é de que o tabuleiro não fica "limpo", pois quando as células avançam para além dos limites da matriz elas automaticamente aparecem na posição oposta. Por exemplo, se uma célula sair pela parte de baixo do *board* a mesma será realocada na parte superior do mesmo. Sendo assim o número de jogadas deve ser especificado e nesse caso será sempre submetido ao total de 10.000 jogadas (iterações).

Por fim, todos os códigos desenvolvidos e/ou adaptados foram avaliados na máquina Atlantica do Laboratório de Alto Desempenho (LAD) da PUCRS e estão disponíveis no github[2]. Foram alocados de forma exclusiva 8 cores físicos, totalizando 16 com *Hyper-Threading*, do cluster Grad. Nessa máquina o desempenho programa serial foi de **5.756345** segundos, valor que servirá como parâmetro para os próximos testes.

Estratégia de paralelismo

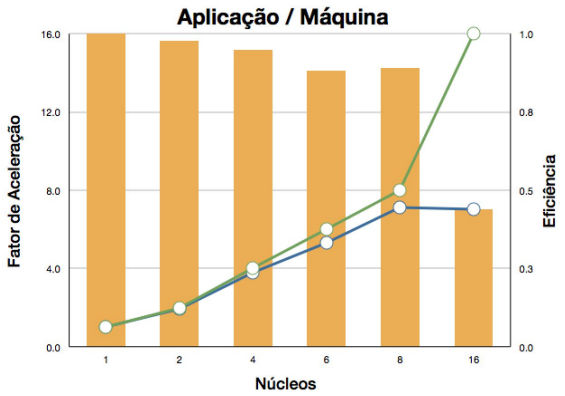
A abordagem utilizada na paralelização do código foi definida após a observação do comportamento do programa serial, cuja forma de atuação baseia-se na iteração por quadrantes de cada elemento do tabuleiro. Cada peça selecionada passa por inspeção de seus vizinhos para a aplicação da regra do jogo e a criação de um novo tabuleiro após o final da rodada. Dessa forma, a estratégia adotado foi paralelizar o código para abranger uma região maior sem que houvesse a necessidade de percorrer peça por peça de forma sequencial. Sendo assim, baseados na ideia de expandir a "visão" do algoritmo sobre o tabuleiro, foram paralelizadas as linhas e as colunas. Assim, cada thread estaria selecionando as colunas que desejasse sem estar presa a qual tupla linha x coluna está sendo processada no momento, criando uma espécie de *spanning tree*.

Essa estratégia se mostrou acertada, vide tabela 1 de tempo de execução e speedup abaixo, na qual é possível perceber que conforme o número de threads cresce, o tempo diminui.

Núcleos	Tempo de Execução (s)	Speed-Up	Speed-Up Ideal	Eficiência
1	6.139934	1.0	1	1.0
2	3.139517	2.0	2	1.0
4	1.617197	3.8	4	0.9
6	1.159302	5.3	6	0.9
8	0.861364	7.1	8	0.9
16	0.874367	7.0	16	0.4
Serial	5.756345	1.066637597	1	1.066637597

Tabela 1.

O desempenho do programa paralelo está exemplificado no gráfico abaixo, onde percebe-se que até 8 threads a eficiência é crescente, porém a partir desse número ela passa a ficar estagnada. Neste trabalho optou-se por não utilizar a estratégia de escalonamento dinâmica visto que, pelo mapa inicial, a maioria das atividades teriam o mesmo tempo. Sendo assim, o overhead para implementar o dinamismo poderia prejudicar o desempenho do algoritmo.



[1] https://github.com/ahme5760/Game_of_Life_MPI
[2]https://github.com/giovannicaprio/t1_paralela_jogo_da_vida_openmp

Código Paralelo

```c
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#define k 40
#define                                   position(row,col)
//printf("%c[%d;%dH",k,row,col)
#define clear() //printf("%c[2J",k)
#define ROWS 175

char  *TOP[ROWS];
char  *BOTTOM[ROWS];
char  *FULL[ROWS] = {
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "....O...................................",
  ".....O..................................",
  "...OOO..................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
```

```
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
"........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "........................................",
  "..........O.............................",
  "...........O............................",
  ".........OOO............................",
};

void game( char**, char**);
void initialize( );
void display( char ** );


// ------------------------------------------------------------------------
// initialize
// initialize top and bottom of grid
void initialize( ) {
  int x;
   //allocate memory and copy input grid to top and bottom for processing
  for (x = 0; x< ROWS; x++ )  {
     TOP[x] = (char *) malloc( (strlen( FULL[0] ) + 1 ) * sizeof( char ) );
    strcpy( TOP[x], FULL[x] );
     BOTTOM[x] = (char *) malloc( (strlen( TOP[0] )+1) * sizeof( char )  );
    strcpy( BOTTOM[x], TOP[x] );
  }
}

// Display current iteration
void display( char* dish[] ) {
  int j;
  for (j=0; j<ROWS; j++ ) {
    position( j, 0 );
    printf( "%s \n", dish[j] );
  }

}

void game( char** part, char** nextStep) {
  /*
    * When given a portion of the grid, this function will calculate the
   * next step and return the new portion of the grid
   */
  int m, n, row;
  int rowSize = strlen( part[0] );
  int partSize = ROWS;

  int th_id, nthreads;

      #pragma omp parallel for private(row, m) schedule(static)
  for (row = 0; row < ROWS; row++) {              // loop through the rows in grid
```

```
                for ( m = 0; m < rowSize; m++) {              // loop
through each character in the row
                int o, n, side = 0;
                char curr = part[row][m];
                     // search a 3 by 3 area in the grid for any live
cells 'O'
                 for ( o = row - 1; o <= row + 1; o++) {
                   int realo = o;// traverse from bottom to top
                   if (o == -1)
                    realo = partSize - 1;
                   if (o == partSize)
                    realo = 0;
                   for ( n = m - 1; n <= m + 1; n++) {
                    int realn = n; // traverse left to right
                    if (n == -1)
                     realn = rowSize - 1;
                    if (n == rowSize)
                     realn = 0;
                    if (o == row && n == m)
                     continue;
                    if (part[realo][realn] == 'O')
                     side++;
                   }
                  }
                  //implementacao da regra do jogo de acordo com
o que encontrou em volta
                if (curr == 'O') {
                  if (side < 2 || side > 3)
                    nextStep[row][m] =  '.';
                  else
                    nextStep[row][m] = 'O';
                }
                if (curr == '.') {
                  if (side == 3)
                    nextStep[row][m] = 'O';
                  else
                    nextStep[row][m] = '.';
                }
               }
              }
             }

             //function to get wall time
             double get_wall_time(){
               struct timeval time;
               if (gettimeofday(&time,NULL)){
                 //  Handle error
                 return 0;
               }
                return (double)time.tv_sec + (double)time.tv_usec *
.000001;
             }


             int main( int argc, char* argv[] ) {

               double start,end;
               start = get_wall_time();
               int steps = 10000;      // # of steps in game
               int u;
               char **part, **nextPart, **tmp;
               //empty screen for next step
                clear();
                initialize();
                part   = TOP;
                nextPart = BOTTOM;
                //display( part );       // show initial step
                for (u = 0; u < steps; u++) { // show rest of the steps
                   // follow rules in game of life to current step and
display next step
                 game(part, nextPart);
                 //display( part );
                 //printf("%s\n"," --- NOVA JOGADA ---" );
                  // This is just for user to be able to see whats going
on. This must be removed for timing calculations
                 //sleep(1);
                 tmp = part; // obtain next step
                 part = nextPart;
                 nextPart = tmp;
                }
                //display(part); // show result ie. final step


                end = get_wall_time();
                printf("Total time elasped is %lf \n",(end-start));
               }
```