UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA

HIGH PERFORMANCE COMPUTING

EVALUANDO RENDIMIENTO COMPUTACIONAL USANDO OPENMP

AUTOR
Joaquín Ignacio Villagra Pacheco
Ingeniería Civil Informática
joaquin.Villagra@usach.cl

Santiago, Chile
Septiembre 2017

Índice de contenidos

1 INTRODUCCIÓN

El presente reporte ilustra el trabajo realizado con OpenMP para abordar el problema computacional de operación de matrices de forma paralela, particularmente de la ecuación de Schroedinger utilizada para modelar la difusión de una onda.

2 ECUACIÓN DE SCHRODINGER

La ecuación de Schrodinger, desarrollada por el físico austríaco Erwin Schrodinger en 1925, describe la evolución temporal de una partícula subatómica masiva de naturaleza ondulatoria y no relativista. Es de importancia central en la teoría de la mecánica cuántica, donde representa para las partículas microscópicas un papel análogo a la segunda ley de Newton en la mecánica clásica. Las partículas microscópicas incluyen a las partículas elementales, tales como electrones, así como sistemas de partículas, tales como núcleos atómicos.[1]

Para efectos de este estudio, se ocupa la versión dependiente del tiempo de la ecuación de Schrodinger para simular el comportamiento de difusión de una onda en medio acuoso.

$$H_{i,j}^{t} = 2H_{i,j}^{t-1} - H_{i,j}^{t-2} + c^2(\frac{dt}{dd})^2(H_{i+1,j}^{t-1} + H_{i-1,j}^{t-1}) + H_{i,j-1}^{t-1} + H_{i,j+1}^{t-1} - 4H_{i,j}^{t-1} \qquad (1)$$

3 DESARROLLO

3.1 ESTRATEGIA DE TRABAJO

Para desarrollar la simulación solicitada, se trabaja una versión más alto nivel y no paralelizada en Matlab y posteriormente la implementación paralela requerida en C.

3.2 DESARROLLO EN MATLAB

Se trabaja en Matlab con la finalidad generar un acercamiento al comportamiento de la ecuación de Schrodinger. Esta implementación permite visualizar el comportamiento de la onda simulada a través de las opciones gráficas del software.

En cuanto al código generado, puede revisarse en detalle en el Anexo correspondiente.

3.3 DESARROLLO EN C USANDO OPENMP

OPENMP como herramienta de paralelización ofrece una serie de métodos útiles para dividir cargas de trabajo, las cuales resultan ser aplicables en una variedad de problemas logrando mejoras notables en los tiempos de computo requeridos.

En este problema en particular no es posible generar una paralelización de todo el procesamiento producto del problema en si. El problema para paralelizar efectivamente radica en la dependencia del computo en función de un valor, el cual corresponde al estado de la matriz en un tiempo $t-1$ y $t-2$, lo que es imposible de asegurar a nivel de concurrencia o paralelismo. Por lo anterior es que la estrategia de paralización radica en aplicar el pragma correspondiente para paralelilzar el for intrínseco de la operatoria matricial.

4 RESULTADOS

Para efectos de evaluar rendimiento computacional de la implementación efectuada, se efectúa el calculo del tiempo requerido para ejecutar 2000, 4000 y 8000 iteraciones para una matriz de tamaño $128x128$ y para una de $256x256$.

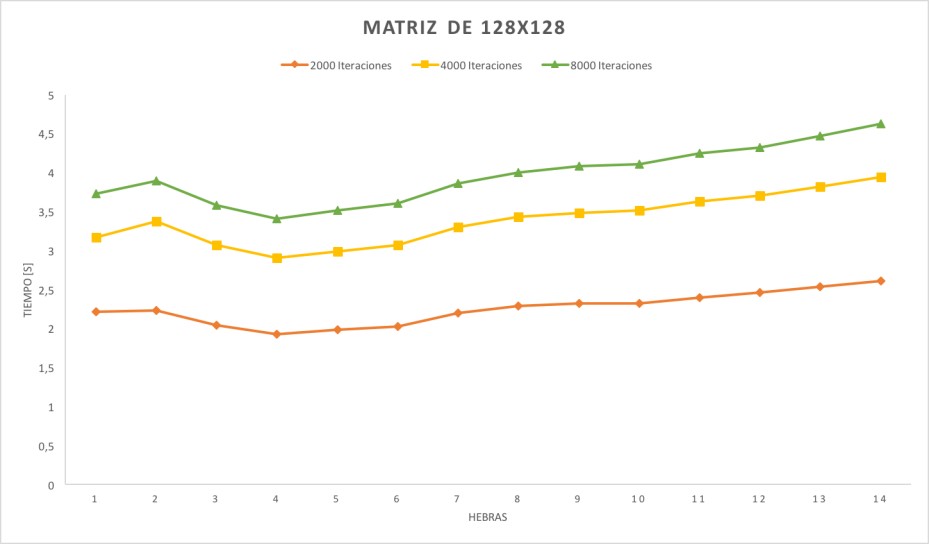

Figura 1: Tiempos en función de cantidad de hebras utilizadas para operar matriz de 128x128

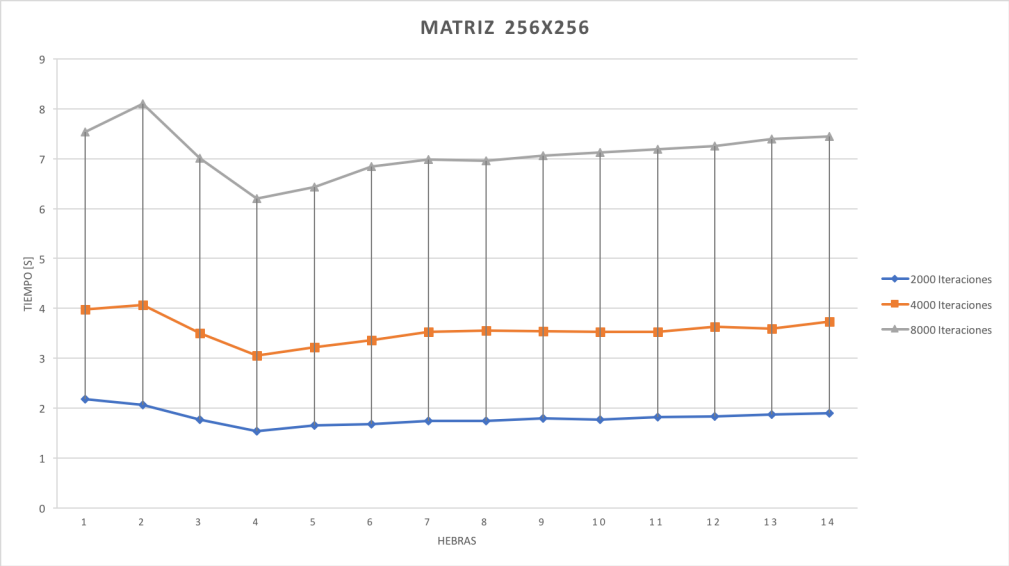

Figura 2: Tiempos en función de cantidad de hebras utilizadas para operar matriz de 256x256

5 CONCLUSIÓN

Es interesante notar que la bajada más significativa de tiempo se da con 4 hebras, lo cual se relaciona con el ordenador utilizado (quad-core). A su vez, llama la atención la subida existente en la ejecución con dos hebras, de donde se infiere que este número genera mayor overhead. Como desafío se podría implementar una solución híbrida con SIMD + OpenMP, la cual, en teoría, debería mejorar el rendimiento computacional. Por otro lado, para evaluar de mejor manera las capacidades de OpenMP se espera trabajar un problema con mayor opción de paralelización.

6 REFERENCIAS

[1] Schrödinger, Erwin (1933). Mémoires sur la mécanique ondulatoire. París: Félix-Alcan. ISBN 2-87647-048-9.. Reedición Jacques Gabay (1988).

7 ANEXO 1: CÓDIGO DESARROLLADO EN MATLAB

```
%valores de entrada para ecuacion multipaso
N = 256;
c = 1.0;
dt = 0.1;
dd = 2.0;
iteraciones = 500000;
%Definiendo matriz H
H_t2 = zeros(N, N);
H_t1 = zeros(N, N);
H = zeros(N, N);

%Asignando valores
for i=2:N
    for j=2:N
        if i>0.4*N && i<0.6*N && j>0.4*N && j<0.6*N
            H_t1(i,j) = 20.0;
        end
    end
end
%Procesamiento de iteracion para generar solucion pedida
for index = 1:iteraciones
    for i = 2:N-1
        for j=2:N-1
            if index == 1
                H(i,j) = 2*H_t1(i,j) + (c^2)/2*(dt/dd)^2 * (H_t1(i
                    +1,j) + H_t1(i-1,j) + H_t1(i,j-1) + H_t1(i,j+1)
                    - 4*H_t1(i,j));
            else
                if index == 2
                    H_t2 = H;
```

```
29                          H_t1(i,j) = 2*H_t2(i,j) + (c^2)/2*(dt/dd)^2 * (
                                H_t2(i+1,j) + H_t2(i-1,j) + H_t2(i,j-1) +
                                H_t2(i,j+1) - 4*H_t2(i,j));
30                      else
31                          H(i,j) = 2*H_t1(i,j) - H_t2(i,j) + (c^2)*(dt/dd
                                )^2 * (H_t1(i+1,j) + H_t1(i-1,j) + H_t1(i,j
                                -1) + H_t1(i,j+1) - 4*H_t1(i,j));
32                      end
33                  end
34              end
35          end
36          H_t2 = H_t1;
37          H_t1 = H;
38      end
39      imagesc(H);axis('off');axis('square');
```

8 ANEXO 2: CÓDIGO DESARROLLADO EN C CON OPENMP

```
1  #include <math.h>
2  #include <ctype.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include <pmmintrin.h>
8
9  #ifdef _OPENMP
10 #include <omp.h>
11 #endif
12
13 #define K2 0.0025
14 #define K1 0.00125
15
16 void imprimir(float** matriz, int N){
17     int i,j;
18     for (i = 0; i < N; i++)
19     {
20             for (j = 0; j < N; j++)
21             {
22                     printf("%.01f ", matriz[i][j]);
23             }
24             printf("\n");
25     }
26     printf("--------------------------\n");
27 }
28
29 float** crear_Matriz(int N){
30     int i,j;
```

```
31      float** matriz = (float**) malloc(sizeof(float*)*N);
32      for (i = 0; i < N; i++)
33      {
34              matriz[i] = (float*) malloc(sizeof(float)*N);
35              for (j = 0; j < N; j++)
36              {
37                      matriz[i][j] = 0;
38              }
39      }
40      //imprimir(matriz, N);
41      return matriz;
42  }
43
44  int main(int argc, char * const argv[])
45  {
46      int i, j, c, dim, steps, hebras, iteration_exit, index;
47      char* file_exit = NULL;
48      opterr = 0;
49      while ((c = getopt (argc, argv, "N:T:H:f:t:")) != -1)
50      switch (c)
51      {
52              case 'N':
53                      dim = atoi(optarg);
54                      break;
55              case 'T':
56                      steps = atoi(optarg);
57                      break;
58              case 'H':
59                      hebras = atoi(optarg);
60                      break;
61              case 'f':
62                      file_exit = optarg;
63                      break;
64              case 't':
```

```
65                      iteration_exit = atoi(optarg);
66                      break;
67              case '?':
68                      if (optopt == 'N' ||optopt == 'T' ||optopt == '
                            H' ||optopt == 'f' ||optopt == 't')
69                              fprintf (stderr, "La opcion -%c  debe
                                    ir acompa ada de un argumento.\n",
                                    optopt);
70                      else if (isprint (optopt))
71                              fprintf (stderr, "No se reconoce opcion
                                    '-%c'.\n", optopt);
72                      else
73                              fprintf (stderr, "No se reconoce opcion
                                    '\\x%x'.\n",optopt);
74                      return 1;
75              default:
76                      abort ();
77      }
78      double timestart = omp_get_wtime();
79      //Definiendo matrices necesarias
80      float **H = crear_Matriz(dim);
81      float **H_t1 = crear_Matriz(dim);
82      float **H_t2 = crear_Matriz(dim);
83      float **H_aux;
84      #pragma omp parallel num_threads(hebras)
85      {
86              #pragma omp for schedule(dynamic, 4)
87              for (i = 1; i < dim; i++)
88              {
89                      for (j = 1; j < dim; j++)
90                      {
91                              if(i>0.4*dim && i<0.6*dim && j>0.4*dim
                                    && j<0.6*dim)
92                              H_t1[i][j] = 20.0;
```

```
93                      }
94              }
95      }
96      //Operando iteraciones de trabajo
97      for (index = 1; index < steps; index++)
98      {
99              #pragma omp parallel num_threads(hebras)
100             {
101                     #pragma omp for schedule(dynamic, 4)
102                     for (i = 1; i < dim-1; i++)
103                     {
104                             for (j = 1; j < dim-1; j++)
105                             {
106                                     if (index == 1)
107                                     {
108                                             H[i][j] = H_t1[i][j] +
                                                K1 * (H_t1[i+1][j] +
                                                 H_t1[i-1][j] + H_t1
                                                [i][j-1] + H_t1[i][j
                                                +1] - 4*H_t1[i][j]);
109                                     }
110                                     else
111                                     {
112                                             H[i][j] = 2*H_t1[i][j]
                                                - H_t2[i][j] + K2 *
                                                (H_t1[i+1][j] + H_t1
                                                [i-1][j] + H_t1[i][j
                                                -1] + H_t1[i][j+1] -
                                                 4*H_t1[i][j]);
113                                     }
114                             }
115                     }
116             }
117             H_aux = H_t2;
```

```
118                 H_t2 = H_t1;
119                 H_t1 = H;
120                 H = H_aux;
121                 //memcpy(H_t2,H_t1, dim*dim*sizeof(float))
122                 //Comprobando iteraci n de salida
123                 if(iteration_exit==index)
124                 {
125                         FILE *salida = fopen(file_exit,"w+b");
126                         fwrite(H, sizeof(float), dim*dim, salida);
127                         fclose(salida);
128                 }
129         }
130         double timefinish = omp_get_wtime();
131         printf("%f\n",timefinish-timestart);
132         return 0;
133     }
```

9 ANEXO 3: IMÁGENES OBTENIDAS VARIANDO CANTIDAD DE ITERACIONES

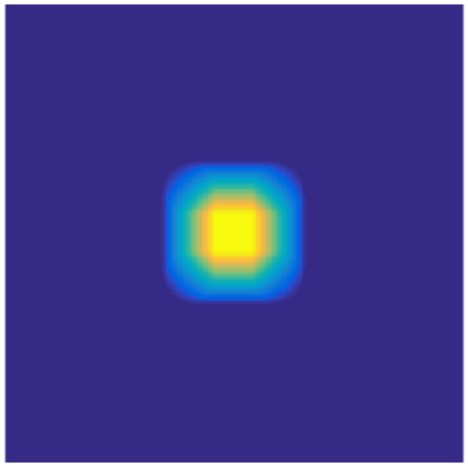

Figura 3: Matriz de 256x256 operada hasta las 300 iteraciones

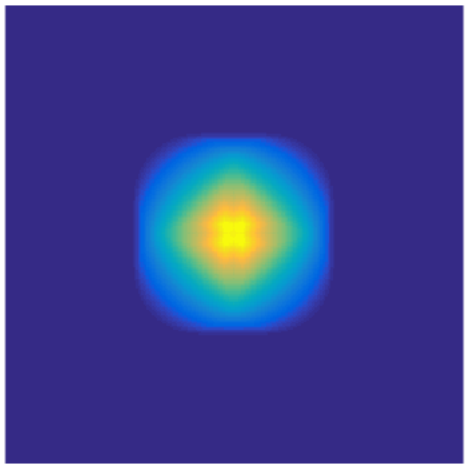

Figura 4: Matriz de 256x256 operada hasta las 625 iteraciones

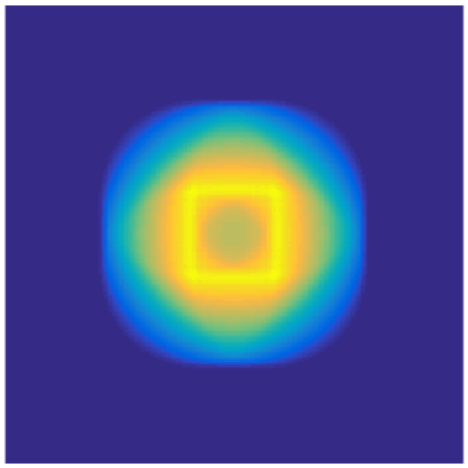

Figura 5: Matriz de 256x256 operada hasta las 1000 iteraciones

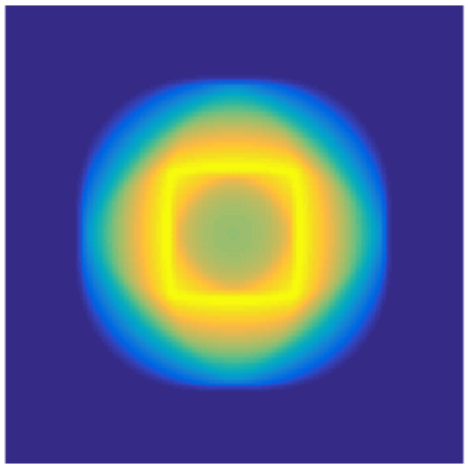

Figura 6: Matriz de 256x256 operada hasta las 1250 iteraciones

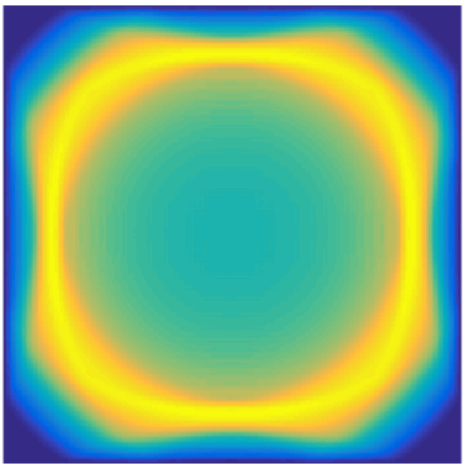

Figura 7: Matriz de 256x256 operada hasta las 2500 iteraciones

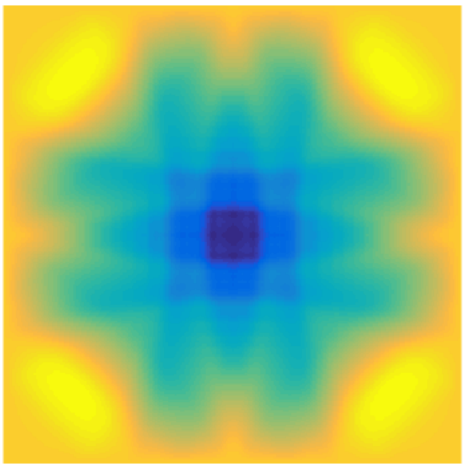

Figura 8: Matriz de 256x256 operada hasta las 5000 iteraciones

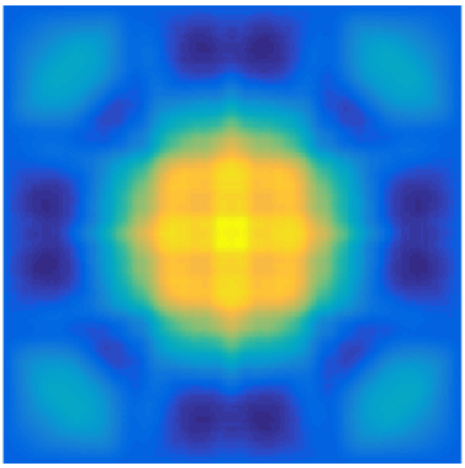

Figura 9: Matriz de 256x256 operada hasta las 10000 iteraciones

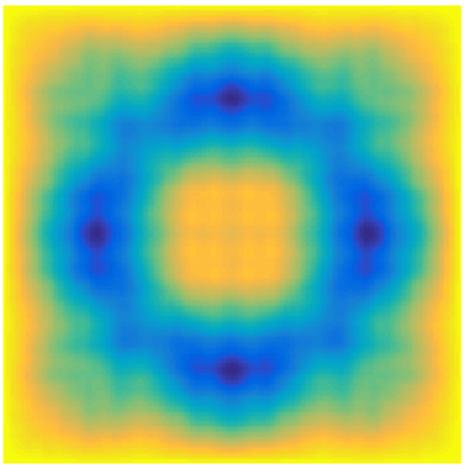

Figura 10: Matriz de 256x256 operada hasta las 20000 iteraciones

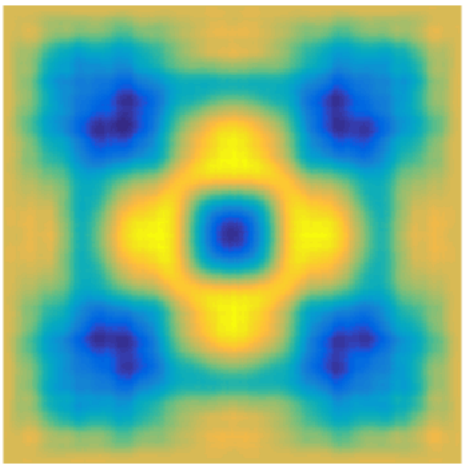

Figura 11: Matriz de 256x256 operada hasta las 40000 iteraciones

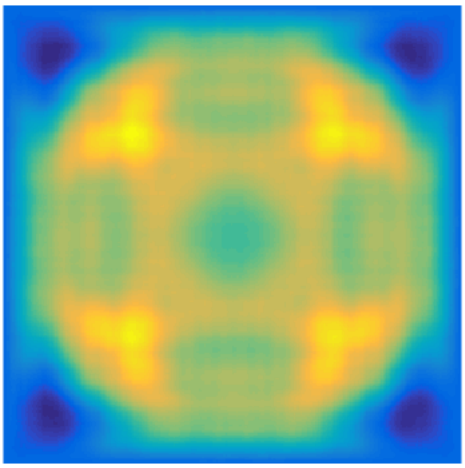

Figura 12: Matriz de 256x256 operada hasta las 60000 iteraciones

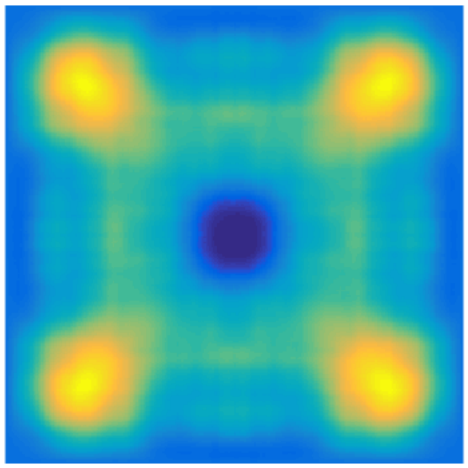

Figura 13: Matriz de 256x256 operada hasta las 80000 iteraciones

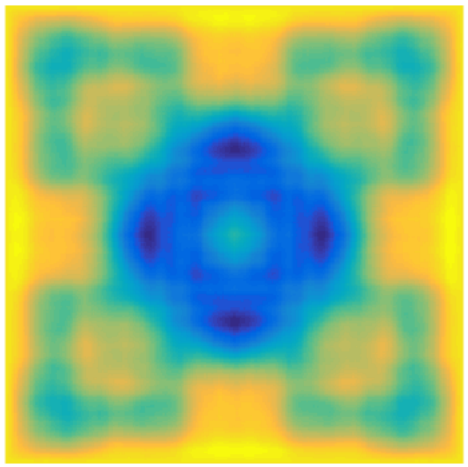

Figura 14: Matriz de 256x256 operada hasta las 100000 iteraciones

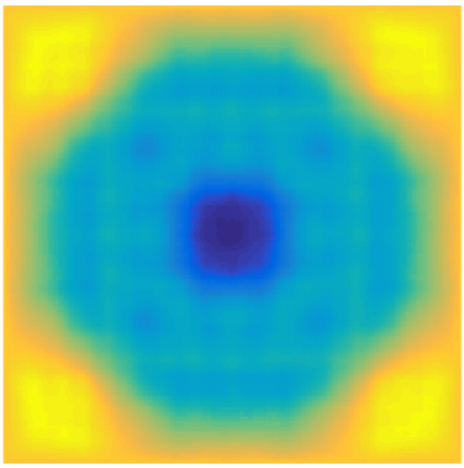

Figura 15: Matriz de 256x256 operada hasta las 150000 iteraciones

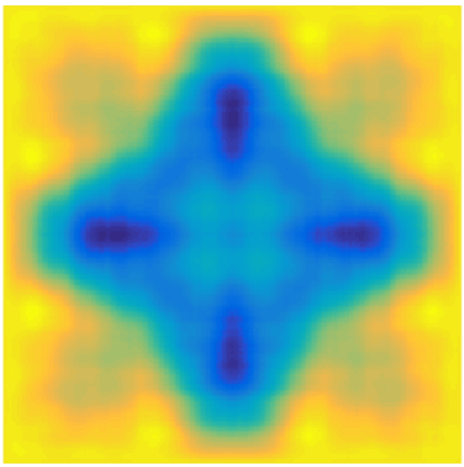

Figura 16: Matriz de 256x256 operada hasta las 200000 iteraciones

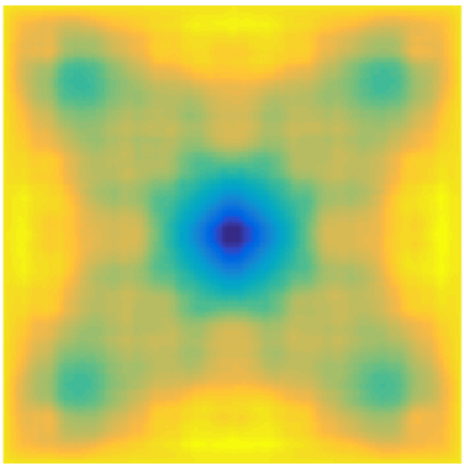

Figura 17: Matriz de 256x256 operada hasta las 300000 iteraciones

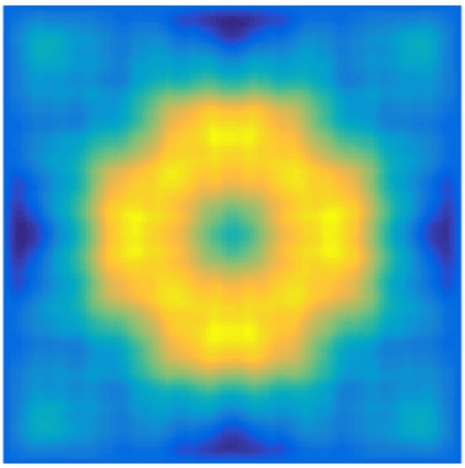

Figura 18: Matriz de 256x256 operada hasta las 500000 iteraciones