SELF-ORGANIZATION OF SCROLL WAVES IN EXCITABLE MEDIA: PARALLEL SIMULATIONS ON A 64-OPTERON LINUX CLUSTER

A THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science

by

IGOR KAPLUN

Montclair State University

Montclair, NJ

November 2007

Abstract

SELF-ORGANIZATION OF SCROLL WAVES IN EXCITABLE MEDIA:
PARALLEL SIMULATIONS ON A 64-OPTERON LINUX CLUSTER

By: Igor Kaplun
Thesis Advisor: Roman M. Zaritski, Ph.D.
Thesis Committee Member: Dajin Wang, Ph.D.
Thesis Committee Member: Carl Bredlau, Ph.D.

Spiral waves have been observed and studied in a variety of biological, physical and chemical systems, known as excitable media. The most famous examples of excitable media include cardiac tissue, the Belousov-Zhabotinsky chemical reaction, and aggregation of starving slime mold amoeba.

It had been shown previously that spiral waves could self-organize into multi-armed spirals. A 3D analog of a 2D spiral wave is called a scroll wave. It rotates around a 1D imaginary tube known as a filament. A later study based on the so-called Puschino model has reported formation of multi-armed scroll waves in 3D. But a question of whether multi-armed scroll waves are a common property of excitable medium or just an artifact of that particular model remained unanswered.

To answer this question we investigated the spontaneous formation and stability of multi-armed scroll waves based on four alternative mathematical models. Most often we used a simple Decoupled Parallel Programming (DPP) approach, starting multiple non-communicating concurrent jobs on multiple available CPUs, corresponding to different parameter values. Isolated scroll ring, evolving into much more complex turbulent patterns, was used as the initial condition.

We observed self-organization of spiral turbulence into multi-armed scroll waves in all four considered models. The presented results may be relevant to the cardiac defibrillation research.

This study has been made possible by the exclusive use of the on-site high-performance 64-CPU “Parallel Monster” cluster: with a close to 64-fold speed-up, it has helped reduce the simulation time for a single set of experiments from over one year to under one week.

Acknowledgements

The 64-processor cluster “Parallel Monster” has been sponsored by the National Science Foundation (grant CTS-0319555).

The entire investigation relies wholly on the free open-source Linux/GNU operating system and numerous other free open-source software packages, such as GCC compiler suite, LAM and MPICH MPI implementations, MESA-3D Open-GL implementation, MJPEG tools (MPEG encoder), and many others.

I express my gratitude to my thesis advisor, Dr. Roman Zaritski, for the support and advice he provided throughout the course of this project, and to the thesis committee members, Dr. Dajin Wang and Dr. Carl Bredlau.

Table of Contents

List of Figures and Tables

1. Introduction

Sudden cardiac death kills more than 250,000 people each year in the US alone. In a healthy heart electrical impulses regularly propagate through the cardiac tissue and cause the heart's muscle fibers to contract. However, if a straight wave is obstructed, a whirlpool-like spiral wave of electrical activity can form and persist in the heart [1].

Spiral waves (see Fig. 1) are not just a property of a human hart, but any medium that can sustain an excitation. An excitable medium is nonlinear dynamical system that supports propagating waves, and does not support a propagation of another wave until some time has passed. The forest is an example of excitable medium. If a fire burns through the forest, it can't return to a burnt spot until the trees are re-grown.

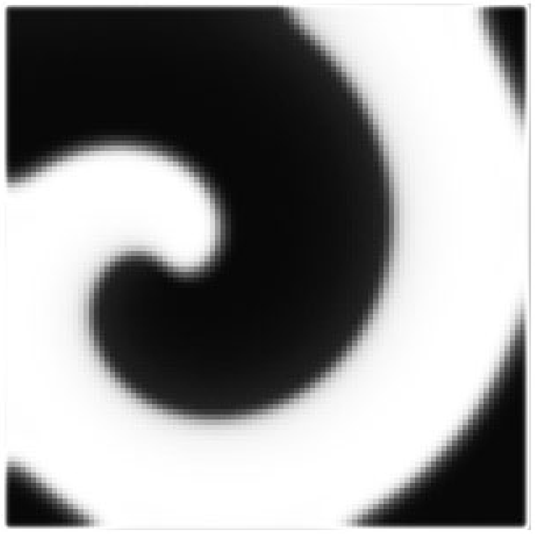

Figure 1. Spiral wave. (A snapshot of a simulation.)

Numerous biological, chemical, and physical systems are characterized as excitable media. The most famous examples include waves of electrical activity in a cardiac tissue, the Belousov-Zhabotinsky (BZ) chemical reaction, and aggregation of starving slime mold amoeba. In the case of the cardiac tissue, spiral waves correspond to cardiac arrhythmias, the leading cause of death in industrialized nations.

Previously, researchers have shown that multiple spiral waves (see Fig. 2) in 2D can attract to form multi-armed spiral waves [2] (see Fig. 3). Multiple interacting spiral waves are believed to be the primary mechanism of cardiac fibrillation, the most dangerous type of cardiac arrhythmia.

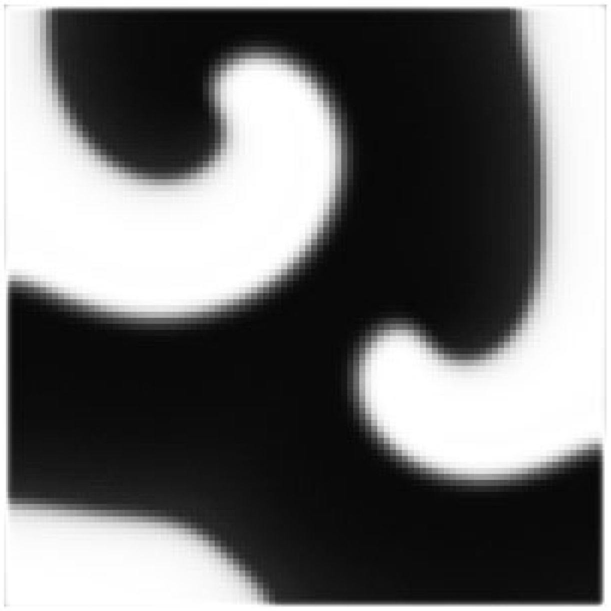

Figure 2. Interacting spiral waves.

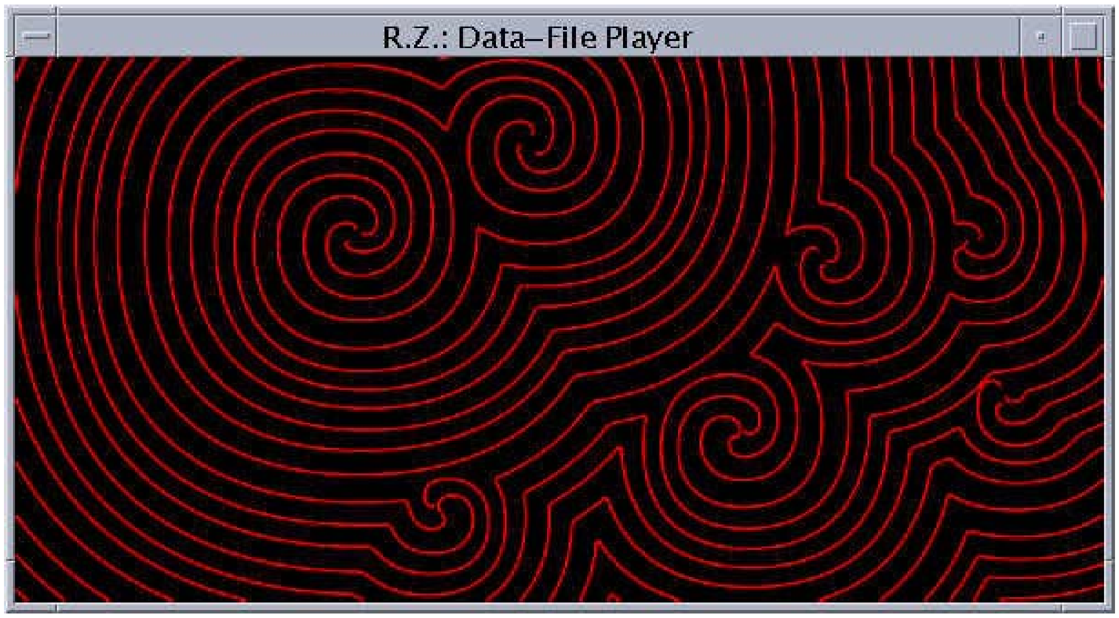

Figure 3. Long-term development spiral triplets in 2D.

A 3D analog of a 2D spiral wave is called a scroll wave. It rotates around a 1D imaginary tube known as a filament (see Fig. 4). A study based on the so-called Pushchino model extended the research of multi-armed spiral waves from 2D to 3D, and intermittent spontaneous formation of multi-armed scroll waves was reported [3] (see Fig. 5).

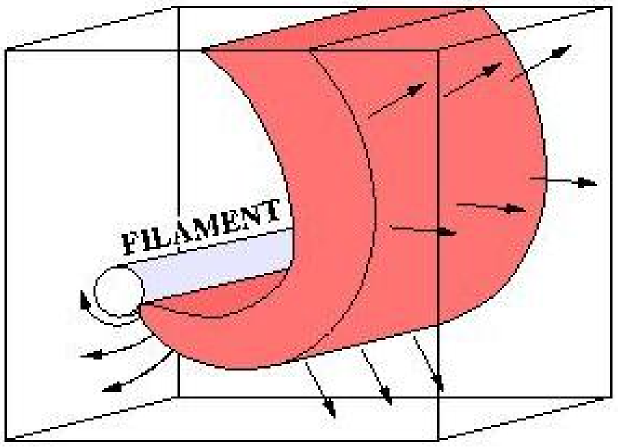

Figure 4. A scroll wave in 3D.

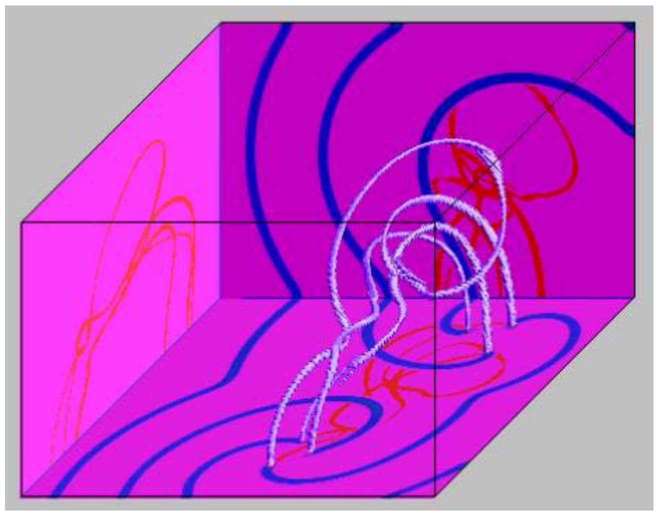

Figure 5. Formation of a three-armed scroll wave in 3D Pushchino model. For clarity, inside the block only the filaments of the scroll waves are shown [3].

However, there was a concern that the above findings are an artifact of that particular (Pushchino) model and not the common property of a generic excitable medium. In 2D, this concern was resolved with a study of a variety of alternative models [4]. But, for the 3D case, the question of whether a formation of spontaneous stable multi-armed scroll waves (MAS) is a robust generic effect remained unanswered. The presented investigation attempts to answer this question.

2. Models

Let us first review the four models we used in our study. Each of the four models is based on a system of Partial Differential Equations. We used a finite difference scheme with simple Euler integration and no-flux boundary conditions to solve the PDEs numerically on a computer.

2.1 Pushchino Model

This model was developed in 1980s at the Institute of Biological Physics, in Pushchino, Russia. Since then, it has been used for efficient computer simulations of both the 2D and 3D excitable media in a multitude of investigations [2-4]. Using Δ to denote the Laplacian operator, the system is

$$\frac{\partial u}{\partial t} = \Delta u + f(u) - v, \quad \frac{\partial v}{\partial t} = \frac{u - v}{\tau(u)},$$

where $u(x,y,z,t)$ is excitation variable, $v(x,y,z,t)$ characterizes the recovery process, and where

$$f(u) = \begin{cases} -C_1 u & \text{if } u < E_1 \\ C_2(u-a) & \text{if } E_1 \leq u \leq E_2 \\ -C_3(u-1) & \text{if } u > E_2 \end{cases}$$

$$\tau(u) = \begin{cases} \tau_1 & \text{if } u < B_1 \\ \tau_2 & \text{if } B_1 \leq u \leq B_2 \\ \tau_3 & \text{if } u > B_2 \end{cases}$$

with $C_1 = 4$, $C_3 = 15$, $E_1 = 0.018$, $\tau_1 = \tau_3 = 0.5$, $\tau_2 = 16.66$, $B_1 = 0.01$, $B_2 = 0.95$, $\alpha = E_1(C_1 + C_2)/C_2$, and $E_2 = [(C_1 + C_2) E_1 + C_3]/ (C_3 + C_2)$

The unexcited steady-state solution of this system is $(u, v) = (0, 0)$. At fully excited points $u \approx 1$, at refractory points (i.e., that have recently been excited but are not excited now) $v \approx 1$.

Parameter C_2 is responsible for the excitability of the medium and determines the size of the spiral core. Higher values of C_2 correspond to a higher excitability and a smaller core size. The base value for this parameter is 0.75, which corresponds to the core diameter of approximately 10 distance units, or 15% of the spiral wavelength. In our simulations we varied this parameter and observed its effect on multi-armed scroll wave formation.

The space and time steps used in the difference scheme were $\Delta x = 0.6$ and $\Delta t = 0.05$.

2.2 Winfree Model

Originally introduced by Winfree in 1991, this model was later used by other researchers, for example, to study 3D scroll wave turbulence by Biktashev in 1998. The equations are

$$\frac{\partial u}{\partial t} = \Delta u + \epsilon^{-1}\left(u - \frac{u^3}{3} - v\right), \quad \frac{\partial v}{\partial t} = \epsilon(u + \gamma - \beta v),$$

with the choice of base parameter point $(\in, \beta, \gamma) = (0.39, 0.75, 0.5)$.

Note that, unlike in the previous model, $(u, v) = (0, 0)$ is not the unexcited steady-state solution. The steady-state values of u and v depend on the choice of parameters β and γ and are obtained by solving a cubic equation. For $(\beta, \gamma) = (0.75, 0.5)$, the unexcited steady state is given by $(u, v) = (-1, -2/3)$.

To avoid multiple recomputation of these steady-state values, in this model we fixed β and γ, and performed a thorough numerical bifurcation analysis for the parameter $\in$. Its reciprocal, i.e. $1/\in$, represents the excitability of the medium.

The space and time steps used in the difference scheme were $\Delta x = 0.5$ and $\Delta t = 0.02$.

2.3 Barkley Model

This model originally developed by Barkley in 1991 for the fast computer simulations of 2D excitable media. Later, this model and its minor variants were used by many other investigators both for the 2D case and 3D case. The system is

$$\frac{\partial u}{\partial t} = \Delta u + \epsilon^{-1} u(1-u)\left[u - \frac{(v+b)}{a}\right], \quad \frac{\partial v}{\partial t} = u - v,$$

where, in this study, the base parameter values are $\in = 0.02$, $a = 1.1$, and $b = 0.24$. For the numerical bifurcation analysis we varied the values of a and b.

The steady-state solution of this system is $(u, v) = (0, 0)$. The discrete space and time steps were $\Delta x = 0.5$ and $\Delta t = 0.02$.

2.4 Oregonator Model

This model was originally derived by Field and Noyes in 1974 to describe the chemical reactions in a Belousov-Zhabotinski medium. Although the original system has three dependent variables, many studies use its simplified version with two unknown functions, called the two-component Oregonator model:

$$\frac{\partial u}{\partial t} = \Delta u + \epsilon^{-1}\left[u(1-u) - (fv + \varphi)\frac{u-q}{u+q}\right],$$
$$\frac{\partial v}{\partial t} = u - v,$$

where, for the base parameter values we fix $\in = 0.1$, $\varphi = 0.03$, $f = 1.4$ and $q = 0.002$.

Generally, Oregonator model supports both excitable and oscillatory regimes. Our parameter choice is well within the excitable region.

Similarly to the Winfree model, $(0, 0)$ is not the steady-state solution of this system. Simulations started with $(u, v) = (0, 0)$ over the entire domain asymptotically relax to the steady state, which for the base value parameters is approximately $(0.0023, 0.0023)$.

The discretization steps were $\Delta x = 0.5$ and $\Delta t = 0.002$.

The above system is often used to describe a photosensitive BZ reaction, with the parameter φ corresponding (approximately proportional) to illumination intensity, which is one of the factors that determines the excitability of the medium.

3. Parallelization Approaches

The problem of determining the long-term evolution of scroll waves in large 3D domains is computationally demanding. To make the solution feasible, it requires parallelization and use of supercomputing resources. We developed two alternative approaches that take advantage of a parallel architecture: Message Passing Interface (MPI) algorithm and Decoupled Parallel Programming (DPP).

3.1 MPI Approach

To parallelize the computation in the main loop, a simple 1D domain decomposition can be used. The finite difference grid for the entire domain is evenly divided among the total number of CPUs available (see Fig. 6). Each mesh point of the domain is updated using the values of the four neighboring points, or, in the case of 3D, using the six neighbors (including also the front and the back points). For the grid points, whose neighbors belong to the same processor, there is no problem updating their value to the next time step. However, the points that are on the CPU's left or right "boundary" have to request neighbors from other CPUs.

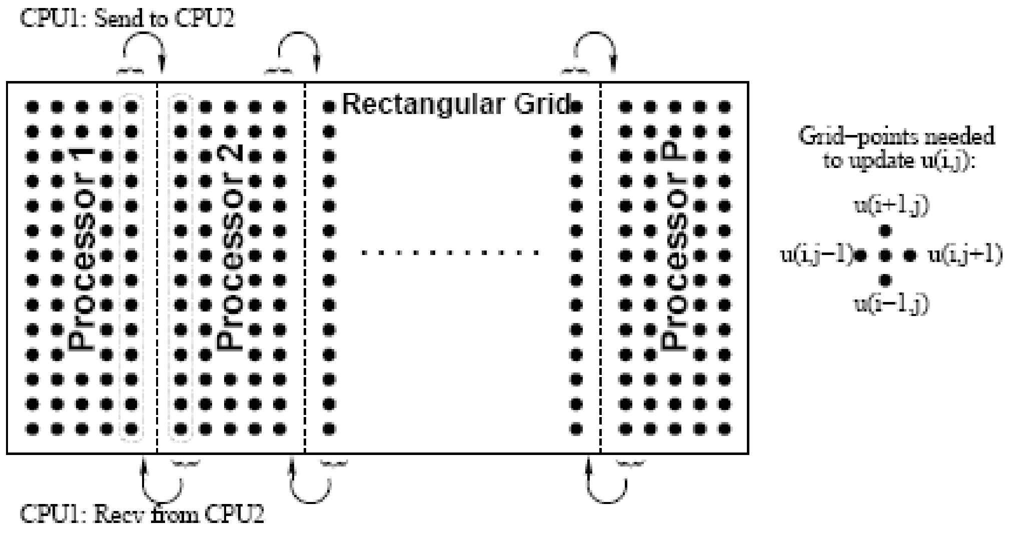

Figure 6. Dividing the grid evenly among all CPUs and using inter-processor message passing (2D version).

The simplicity of this parallelization approach allows for much flexibility in simulations, such as modifications of the mathematical model, experimenting with various boundary conditions and geometries (e.g., torus), introduction of simulated sensing probes and excitation sources, etc.

Figure 7 shows the scalability of this message-passing approach on 2D and 3D domains of various sizes. We see that in 2D the algorithm performance scales well throughout the large number of processors, whereas in 3D the speed-up begins to decrease beyond 5 - 15 processors.

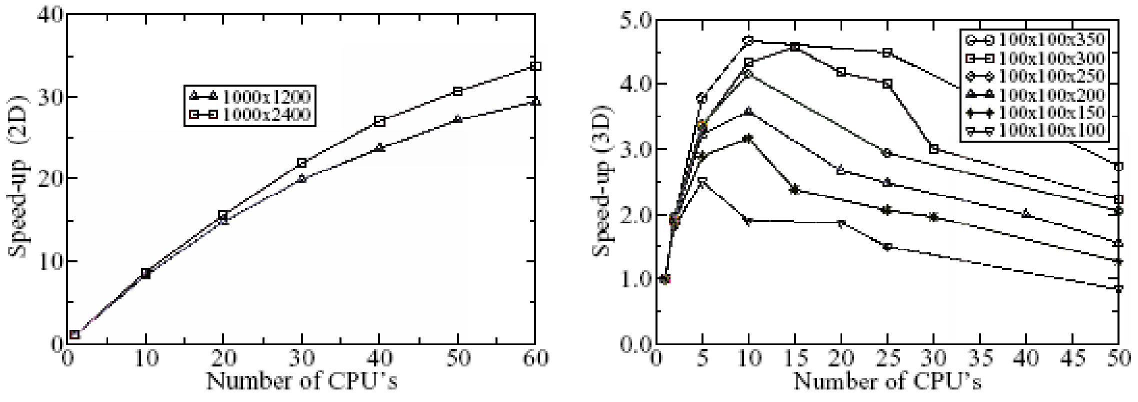

Figure 7. 2D (left) and 3D (right) speed-up curves for various domain sizes [5].

The dimension varied in these simulations was the one along which the slicing was done. More extended domains increase the number of grid points per slice, with the size of the slice boundary being constant. This corresponds to a higher computation-to-communication ratio and, as we can see, the speed-up curves for more extended domains lie above the ones that represent a square or a cube. (Note that due to the nature of the algorithm, the slicing dimension has to be exactly divisible by the number of computing processors.)

Although we attain higher computational gains on domains that deviate more from a perfect square or cube, the ones that are closer to a square or a cube are actually more interesting for the underlying scientific problem.

3.2 DPP Approach

The Decoupled Parallel Programming approach is useful when there is a need to run the same program many times with different parameter values. Independent non-communicating (non-MPI) jobs are started simultaneously on multiple processors with different parameter values. Starting such a parallel set of DPP jobs is achieved by using a script that automates the distribution of parameters. The main advantage of this approach is that there is no communication overhead, and an almost perfect speed-up is achieved, i.e. the speed-up equals the number of CPUs.

4. The "Parallel Monster" Linux Cluster

To run our computationally demanding 3D simulations, we make use of the on-site 64-processor Linux cluster "Parallel Monster", or PM (see Figs. 8 and 9) [6]. The PM cluster is located in the MSU Parallel Computing Laboratory [7] at the Department of Computer Science. It was assembled in the year 2003 by Dr. R. M. Zaritski using the National Science Foundation funds.

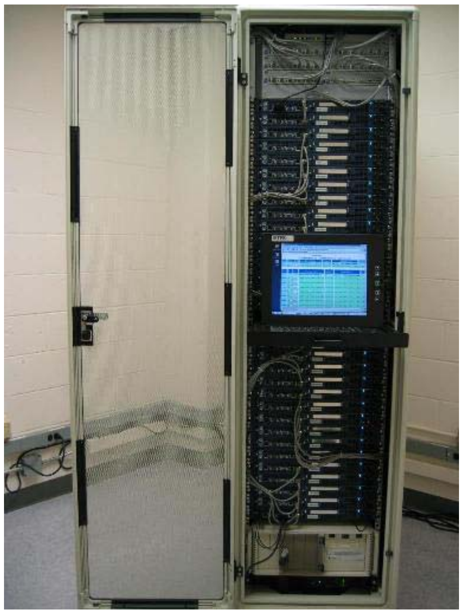

Figure 8. Parallel Monster cluster

Figure 9. Connections on the back of Master node.

The cluster has the total of 32 nodes (1 Master and 31 Slaves). Each node has two AMD Opteron 240 (1.4GHz) processors and 2 GB of main shared memory per Slave node, and 4 GB shared memory on the Master node. The nodes are interconnected via a Gigabit Ethernet switch over Cat 5e/6 copper cables (1000BaseT). The switch is 36-port HP Procurve 4108GL (see Fig. 10 and Table 1). The nodes and the cabinet for this cluster had been provided by Microway, Inc. [8], a cluster integration company. The cluster management is mediated through the specialized "NodeWatch™" boards built by Microway. The cluster is housed in a Microway 44U rackmount cabinet, with three door fans. It is powered by six 120V 15A power strips and has APC Smart-UPS power backup (750VA, 1U cabinet space). The total hardware cost of the cluster was approximately $100K.

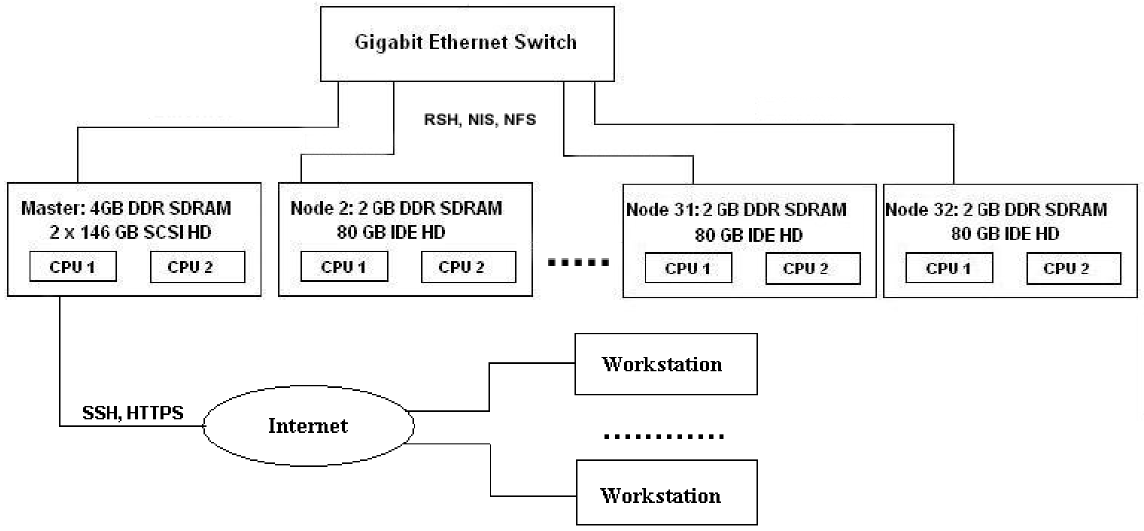

Figure 10. The structural diagram of the Parallel Monster cluster with interconnections.

Hardware	**Specifications**
Cabinet & Nodes	Microway, Inc.
Nodes	32 (1 Master, 31 Slaves) Dual CPU
Processors	64 AMD Opterons 240 (64bit, 1.4GHz)
Memory	2GB Slave, 4GB Master (DDR SDRAM 333MHz ECC Reg)
Hard Drive	80GB IDE HD (2x146GB SCSI on Master)
Cluster Management	Microway NodeWatch™ cluster management hardware
Switch	HP Procurve 4108GL 36-port GigE switch
Network Cables	Cat 5e/6 Copper cables (1000BaseT)

Table 1. Parallel Monster Hardware

The cluster peak performance is 180 Gigaflops. All of the cluster software is free and open-source [9]. Most is licensed under the GNU Public License. The operating system is Red Hat Linux 9 and the parallel communication software includes MPICH (v1.2.5) [10] and LAM (v7.0.3) [11] – two most popular open source implementations of the Message Passing Interface (MPI) standard [12]. We use GCC compiler suite (v3.2). The cluster management software includes Microway NodeWatch™, Microway suite of custom RSH scripts, and Ganglia monitoring utility (see Table 2).

Software	**Specifications**
Operating System	Red Hat Linux 9
Compiler Suite	GCC (v3.2)
MPI Parallel Software	MPICH (v1.2.5), LAM (v7.0.3)
Cluster Management Tools	Microway NodeWatch™ and scripts, Ganglia

Table 2. Parallel Monster Software

Internally the cluster uses Remote Shell (RSH) for inter-node logins and command execution, Network Information Systems (NIS) for passwordless logins, and Network File System (NFS) for mounting user home directories from the Master. External remote logins to the Master node done via Secure Shell (SSH). The external web cluster management is done via HTTPS. See Table 3 for the summary of protocols.

Network	**Protocols**
Internal	TCP/IP, RSH, NIS, NFS
External	TCP/IP, SSH, HTTPS

Table 3. Network protocols used by Parallel Monster

The simulation programs were written in C++ and compiled using GCC compiler. There are two different versions of the program to accommodate MPI and DPP approaches. The source code for each version of the program is available in Appendix A.

5. Simulations, Data Visualization, and Bifurcation Analysis

We ran over two thousand numerical simulations. Each set of simulations was usually up to a week in duration. Most often we used the DPP approach, starting 64 non-communicating concurrent jobs on the 64 cluster CPUs, corresponding to different parameter values. An isolated scroll ring, evolving into much more complex turbulent patterns, was used as the initial condition. For each simulation we detected the scroll wave filaments and produced MPEG movies of those filaments (discarding the waves themselves) using the MPEG encoder from the freely available open-source "MJPEG Tools" package [13]. The use of MPEG compression results in a higher inter-platform compatibility and dramatically smaller movie files.

We confirmed the presence of multi-armed scroll waves for the Puschino model and observed the self-organization of spiral turbulence into multi-armed scroll waves in all other considered models [14,15] (see Figs. 11 and 12).

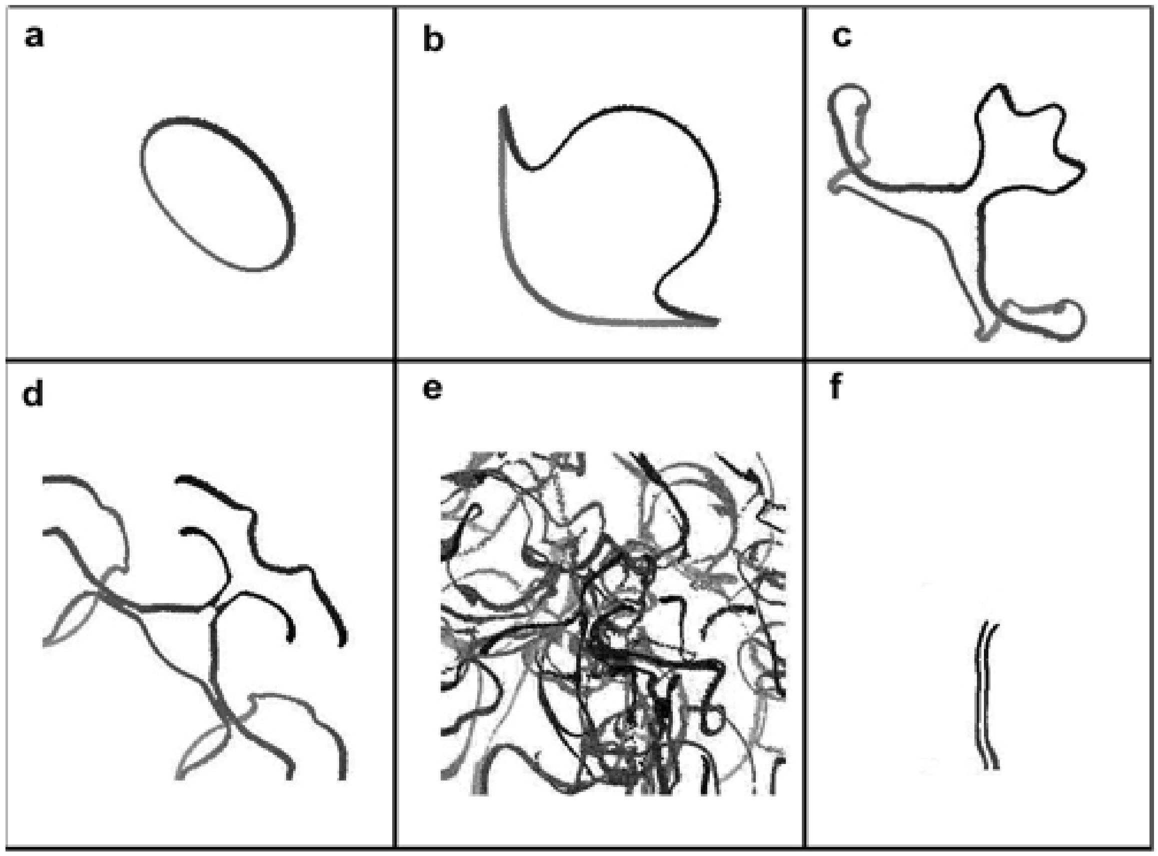

Figure 11. Long-term evolution of the initial scroll ring in the Winfree model.

Figure 11 shows a long-term development of scroll waves from the initial scroll ring (Panel (a)) for the Winfree model. Only filaments of the scroll waves are displayed. On Panels (b) and (c) the initial ring becomes distorted and eventually collides with the boundary, breaking into multiple scroll waves (Panel (d)). This leads to a chaotic pattern (Panel (e)). On Panel (f) we observe a spontaneous formation of double filament strand, corresponding to a two-armed scroll wave. It emits waves of higher frequency, which expel all other waves.

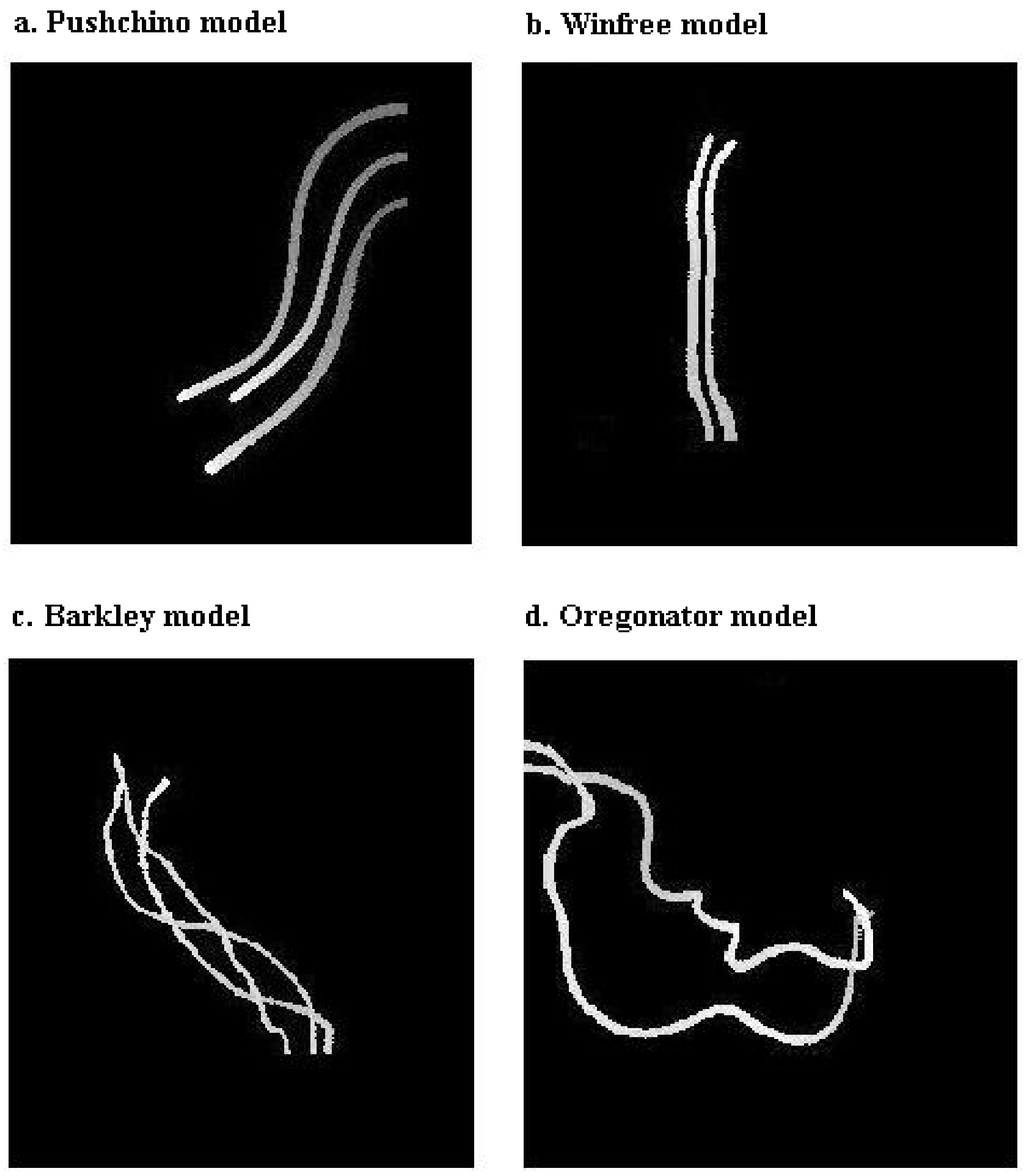

Figure 12. Spontaneous formation of multi-filament strands (corresponding to multi-armed scroll waves) was observed in all four models considered.

Figure 12 shows snapshots of long-term evolution results for each of the four considered models. As we can see, multi-filament strands are observed in all four models. Panels (a) and (c) show triple strands, which correspond to three-armed scroll waves in the Pushchino and Barkley models. Panels (b) and (d) show double strands, which correspond to two-armed scroll waves in the Winfree and Oregonator models.

However, not all low-excitability parameter points that led to multi-armed spiral waves in 2D, lead to multi-armed scroll waves in 3D. For some of those parameters, multi-armed scrolls have not been observed at all, and for some, multi-filament structures were much less stable: they appeared only transiently, for very brief periods of time, insufficient for "expelling" other lower-frequency vortex activity outside the simulated block.

The primary cause for such a change from the 2D case is the destabilizing effect of the "negative filament tension" – a trend of individual filaments to increase their length by bending, bulging, and looping, which is a purely 3D effect (also known as Winfree turbulence). The degree of such turbulence depends on the model and it cannot be predicted from 2D simulations.

The four 2D bifurcation diagrams (Figs. 13 - 16) taken from a prior 2D study [4] show where multi-armed spiral waves were observed in 2D. To summarize our findings for 3D simulations, on these diagrams we mark the parameter points where multi-armed spirals were observed by the "*" symbol. The "#" symbol marks the parameter points that were used to produce the corresponding snapshots on Fig. 12.

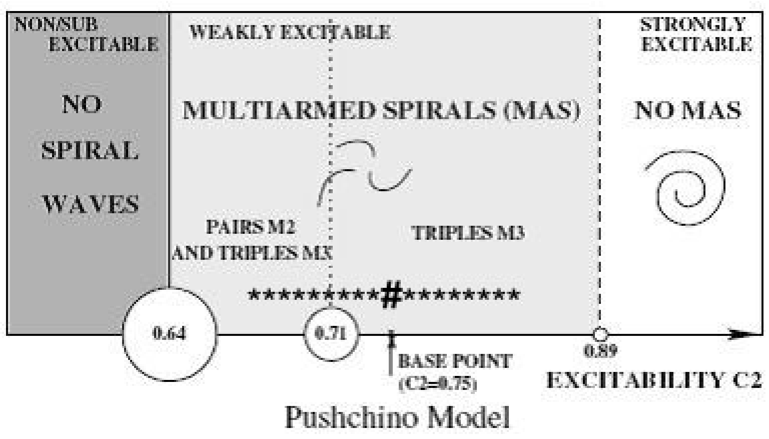

Figure 13. Bifurcation diagram for the Pushchino model. Decreasing circles on the C_2-axis symbolize a decrease in spiral core size (or filament diameter) as the excitability of the medium grows. The symbol "*" marks parameter points where multi-armed scroll waves were observed in 3D. The parameter point marked by "#" corresponds to the simulation shown on Fig. 12a.

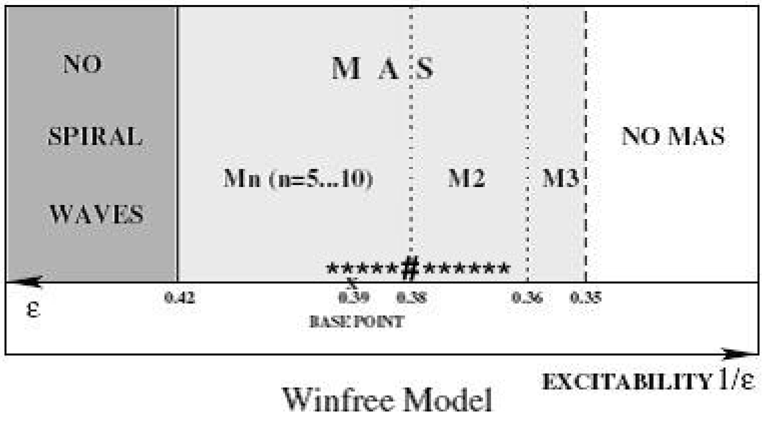

Figure 14. Bifurcation diagram for the Winfree model showing the principal 2D MAS bifurcation loci on the ∈-axis. The symbol “*” marks parameter points where multi-armed scroll waves were observed in 3D. The parameter point marked by “#” corresponds to the simulation shown on Fig. 12b.

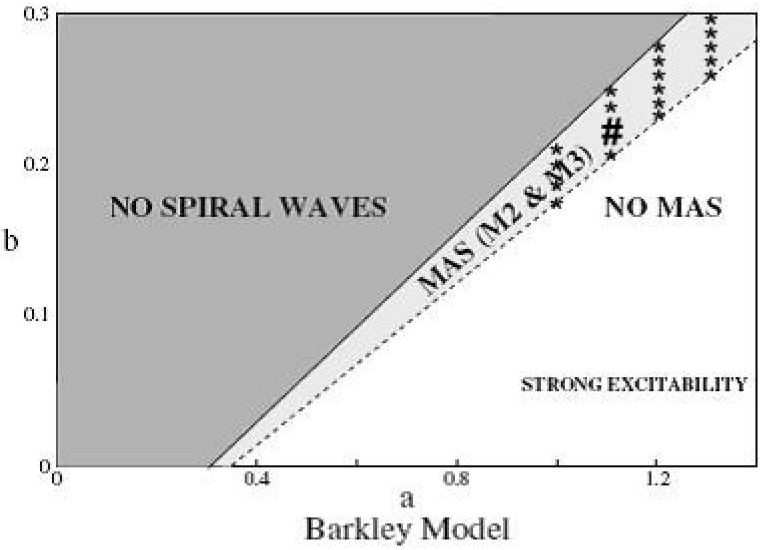

Figure 15. Bifurcation diagram for the Barkley model (for ∈ = 0.02) showing the approximate location of the 2D MAS region on the *a*-*b*-parameter plane. The symbol “*” marks parameter points where multi-armed scroll waves were observed in 3D. The parameter point marked by “#” corresponds to the simulation shown on Fig. 12c.

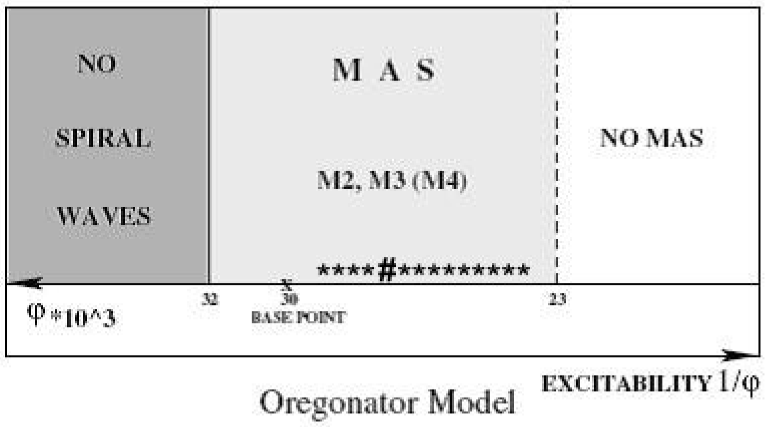

Figure 16. Bifurcation diagram for the parameter φ of the Oregonator model. The symbol "*" marks parameter points where multi-armed scroll waves were observed in 3D. The parameter point marked by "#" corresponds to the simulation shown on Fig. 12d.

6. Discussion

This study has confirmed the spontaneous formation of relatively stable multi-armed scroll waves in all four considered models. Thus, such a “crystallization” of multi-filament strands from a chaotic-looking pattern seems to be a common property of a generic excitable medium.

These findings may be important for the cardiac defibrillation research. Scientists believe that multiple interacting scroll waves are the primary mechanism of cardiac fibrillation. Thus, they have to consider the possibility of spontaneously formed higher-frequency multi-armed structures in arrhythmic hearts, which needs to be taken into account in the defibrillation and pacemaker research.

In the course of this computationally demanding study our 64-processor cluster has proven to be an essential tool to speed-up the simulations. Without the use of this cluster each simulation that took a week to complete would take over a year on a single-processor workstation. The Parallel Monster cluster is based on the so-called Beowulf approach [16] originally developed in NASA, which is using generic PC components, inter-connected by a generic network, and running free open-source software to build a high-performance parallel computing resource. Within the past decade this approach led to the dominance of clusters and to the virtual extinction of much more expensive traditional proprietary supercomputers [17]. The low cost of the Beowulf-type Parallel Monster hardware was complemented by the zero cost of the free open-source software used throughout the entire project.

Appendix A. SPINT (Spiral Interaction) Program Listing

A.1 MPI Version

```
/****************************************************
10-21-05, RZ/KP: 3D MPI code for 4 models:
 - 1. Pushchino Model (AP)
 - 2. Barkley Model (BK)
 - 3. Winfree Model (WF)
 - 4. Oregonator Model (OR) (2-component)
ported from Galaxy cluster to Parallel Monster cluster
(Red Hat 9, GCC 3.2, LAM/MPICH)
Multi-Model (MM) code: thanks to Jin Ju (JJ).
****************************************************/
using namespace std;

//#define GRAPHICS // show results graphically using OpenGL
#define TOFILE // save u into a file
#define PROBE // save probe results

//#define AP // PUSHCHINO model
#define BK // BARKLEY model
//#define WF // WINFREE model
//#define OR // (2-component) OREGONATOR model

#include "mpi++.h"

#include <iostream>
#include <fstream> //for file io
#include <cstdlib> //for exit(), system(), rand()
#include <cstring> //for file-name ops
//#include <cstdio> //for gets()
//#include <cmath> //for M_PI

//global stuff:
int NP=64;
int rank, lrank; // MPI and logical rank
int *rtol, *ltor; //conversion arrays between rank and logical rank
int size, csize; // csize = # of computing (slice) CPUs
int L; // number of layers in a slice
int P; // number of points in a layer
int N,M,K; //mesh size: hight, width, depth

int OPT = 0; // optimization on logical rank assignment

//****************************************

class Array3;

class DATA
{
```

```
public:
 DATA()
 {

  N=100; K=100; M=201; //default for all models
  DT=100000; // default

N=100; K=100; M=101;
DT = 50000;

N=100; K=100; M=101;
DT = 10000;


N=150; K=150; M=151;
DT=1000000;

//TMP:
N=180; K=190; M=201;
DT=300000; //5 hours
DT=1200000; //20 hours

N=180; K=190; M=201;

#ifdef AP
   name="AP";
  //dx = 1.2; dt = .1;
  dx = 0.6; dt = 0.05;
dx = 1.2; dt = .1;
  C1=4; C3=14; C2=0.75; //base parameter value
  E1=0.018; a=(C1+C2)*E1/C2;
  E2=(C2*a+C3)/(C2+C3);
  tau1=tau3=0.5; tau2=16.66;
  B1=0.01; B2=0.95;

  //DT=100000; //200x300x100: 6 hours
  //DT=1000000; //200x300x100: 60 hours
#endif //AP


#ifdef BK
  name = "BK";


//MIKHAILOV: e=0.02;
e = 0.02;
  //a=1.0; b=.2; //default parameters
  //a=1.1; b=.24; //base-point parameters (support S, M2, M3)
  //(Note: The largest dt with e=.02, a=1.0, b=.15, dx=1.0 is: dt = 0.0594!!!)
  //(Note: maximum dt is usually approx.= e)
  //dx = 1.0; dt = 0.05;
  dx = 0.5; dt = 0.02;

dx = 1.0; dt = 0.05;
a=1.1; b=.24;
```

```
dx = 1.0; dt = 0.05;
a=1.4; b=.23; //3D!!!!

dx = 1.0; dt = 0.05;
a=1.15; b=.19;

  //a=1.1; b=.2; //small core for benchmarking

//BARKLEY: e=0.005 (1/200) & 0.00666 (1/150); (Note: largest dt==e):
  //e = 0.005;
  //dx = 0.5; dt=0.005;   //too crude!!!
  //e= 0.005; dx = 0.25; dt = 0.0025;   a=1.0; b=.3;

#endif //BK


#ifdef WF
  name="WF";

  //Biktashev's IJBC98 paper:
  e = 0.3; b = 0.75; g = 0.5; // 1/e==excitability (e=0.4 -> very low)
  //dx = 0.5; dt = 0.03;
  dx = 0.5; dt = 0.02;
  uss=-1.0; vss=-2.0/3.0;
  e=0.38; //RZ, Summer2004: discovered stable pairs!!!
  e=0.39; //Base value: supports, M2, M3,..., M6,...


  //---------------------
  //e = 0.3; b = 0.5; g = 0.7; uss,vss=??? //Winfree(ch10)
  //e = 0.2; b = 0.5; g = 0.9; uss,vss=??? //Winfree(ch10) - more generic
  //---------------------

e=0.39;

 e=0.36; //FOR 3D!!!

#endif //WF


#ifdef OR
  name="OR";
  //From AlonsoPRL01/Beato paper:
  //e = 1/11.7, 1/1059; f = 1.4; fi = 0.005 -- 0.096; q = 0.002;

  //RZ, Summer 2004, Discovered MAS:
  //f = 1.4; q = 0.002; e = 1.0/11; fi = 0.03;
  //dx = 0.5; dt = 0.001; //needed for e < 0.08
  dx = 0.5; dt = 0.002; //OK for e > 0.08
  f=1.4; q=0.002; e=0.1; fi=0.03; //base point for IJBC!
  //Note1: steady-state is (a,a), where a is small(?)
  //Note2: for initial patches can use u=1, v=.4...
  //Note3: highest value of u is about 0.4!


  //TMP:
```

```
dx = 1.0; dt = 0.0022;
fi=0.0265;

#endif //OR

TStepuv = 1000000;
TStep = DT;
//TStepuv=TStep = 100; //for graphical debugging
TStepuv=TStep = 100;

//TMP:
TStep = 500;
TStepuv = TStep;

  NumFramesuv = DT/TStepuv + 1;
  NumFrames = DT/TStep + 1;
  XStepuv=XStep=1;

  RandomSeed=1;
  NumGen=10;

  WSize = 200; //for OpenGL graphical display
 }

//data:
 Array3 *u, *v;
 char *tip;
 int DT; //terminal discrete time (# of iterations)
 int WSize; //OpenGL graphical window size
 double dt, dx; //step sizes


#ifdef BK
 double e, a, b;
#endif

#ifdef AP
 double C1,C2,C3,a,E1,E2,E3,tau1,tau2,tau3,B1,B2;
#endif

#ifdef WF
 double e, b, g, uss, vss;
#endif

#ifdef OR
 double e, f, q, fi;
#endif

 int TStepuv; // how often write u/v to a file
 int TStep; //how often to compute/transmit tip curve
 int XStepuv, XStep; // how much to write to file

 char *name, *dname; char *uname, *vname, *tname, *sname;
 int NumFramesuv, NumFrames;
 int RandomSeed;
 int NumGen;
```

```
};//class DATA

DATA *dat; //global pointer to data storage

//*******************************************

class Array3
{
public:
 Array3() {n1=n2=n3=l=0; ap=NULL;}
 Array3(int n1_, int n2_, int n3_, double a)
 {
   n1=n1_; n2=n2_; n3=n3_;
  l = (n1+1)*(n2+1)*(n3+1);
  ap = new double [l];
  for(int i=0; i<l; i++) ap[i] = a;
 }

 ~Array3() { delete [] ap; }

 void Set(double a) { for(int i=0; i<l; i++) ap[i] = a; }

 inline double& operator()(int i, int j, int k)
 {
  //if ((i<0)||(j<0)||(k<0)||(i>n1)||(j>n2)||(k>n3))
  //{cout<<"\nArray referenced out of bounds!\n"; exit(5);}
  return ap[(n1+1)*(n3+1)*j + (n1+1)*k + i] ;
 }

 double* GetLayerPtr(int n) { return ap + n*(n1+1)*(n3+1); }
 double* GetPtr() { return ap; }

private:
  double *ap; //array pointer
  int n1, n2, n3, l;
}; //end of class Array3

//************************ - SLAVES  - ***********************

void Slave()
{
 int i,j,k,c;
 int DT = dat->DT;
 int TStepuv = dat->TStepuv;
 int TStep = dat->TStep;
 Array3 *tmpu;
 Array3 *u = new Array3(N,L-1,K,0);
 Array3 *uu = new Array3(N,L-1,K,0);
 Array3 *v = new Array3(N,L-1,K,0);
 char* tip = new char[P*L]; for(c=0;c<P*L;c++) tip[c]=0;


 double dt = dat->dt;
 double dx = dat->dx;
 double U,UU,V,DU; // temporary values for u,v,etc. to speed up
 double *up, *uup, *tmpup, *vp;

#ifdef AP
  double C1 = dat->C1, C2=dat->C2, C3 = dat->C3;
  double negC1=-C1, negC3=-C3;
  double f,tau;
  double a = dat->a;
  double E1 = dat->E1, E2 = dat->E2, E3 = dat->E3;
  double tau1 = dat->tau1, tau2 = dat->tau2, tau3 = dat->tau3;
  double B1 = dat->B1, B2 = dat->B2;
#endif //AP

#ifdef BK
  double a=dat->a, b=dat->b, e=dat->e;
  double ee = 1/e;
#endif //BK

#ifdef WF
  double e=dat->e, b=dat->b, g=dat->g;
  double ee = 1/e;
  double uss=dat->uss, vss=dat->vss;
#endif //WF

#ifdef OR
  double e=dat->e, f=dat->f, q=dat->q, fi=dat->fi;
  double ee = 1/e;
#endif //OR


 int left, right;
  if(lrank!=1) { left=ltor[lrank-1]; } else { left=ltor[csize]; }
  if(lrank!=csize) { right=ltor[lrank+1]; } else { right=ltor[1]; }

 MPI_Status *status = new  MPI_Status; //for MPI_Recv
 MPI_Request *request = new MPI_Request; //for MPI_Isend

#ifdef PROBE
 //probe file name:
 char *pfname = new char[20];
 char *pnum = new char[3]; pnum[0]=pnum[1]=pnum[2]=0;
 //NOTE!!! This limits NumCPU to < 100 !!!
 pnum[0] = lrank/10 + 48; pnum[1]=lrank%10 + 48;
 strcpy(pfname,"/scratch/probe");
 strcat(pfname, pnum);

 ofstream *Pfile; Pfile = new ofstream(pfname, ios::out);

 int ProbeJ = L/2; int ProbeI = N/2; int ProbeK = K/2; //probe coordinates
 //int ProbeJ = L/2; int ProbeI = N/2; int ProbeK = 10; //probe coordinates
 int ProbeFlag = 1; //waiting for wave front
 int ProbeLastFront = 0;
#endif //PROBE

 //stuff INITIAL CONDITIONS from Master:
  MPI_Recv((void*)u->GetPtr(), L*P, MPI_DOUBLE,
          0, 1111+lrank, MPI::COMM_WORLD, status);
 MPI_Recv((void*)v->GetPtr(), L*P, MPI_DOUBLE,
          0, 2222+lrank, MPI::COMM_WORLD, status);

 //optimizations:
 double dt_div_dxdx = dt/dx/dx;
 int N_plus_1 = N+1;
 int L_min_1 = L-1;
 int L_min_2 = L-2;


 //for FILAMENT DETECTION:
 double minU=.1, maxU=.3;
#ifdef AP
 double dtxdtxconst= dt*dt*.0009;
#endif //AP
#ifdef BK
 double dtxdtxconst= dt*dt*.1;
#endif //BK
#ifdef WF
 double dtxdtxconst= dt*dt*.1;
 minU=.2; maxU=.6;
#endif //WF
#ifdef OR
 double dtxdtxconst= dt*dt*.1;
#endif //OR



//----------------------MAIN (TIME) LOOP-------------------------------
 for(int DiscrTime=0;DiscrTime<=DT;DiscrTime++)
 {

  up=u->GetPtr();
  uup=uu->GetPtr();
  vp =v->GetPtr();

  //Periodically zeroing out small stuff:
  if (DiscrTime%100==0) for(j=0;j<=L_min_1;j++) for(k=0;k<=K;k++) for(i=0;i<=N;i++)
    { int cur=j*P+k*N_plus_1+i; U=up[cur]; V=vp[cur];
      if((U<.001)&&(U>-.001)) up[cur]=0; if((V<.001)&&(V>-.001)) vp[cur]=0;
    }

  //cout one pixel (for debugging):
  //if(lrank==1) cout<<up[5*P+5*N_plus_1+5]<<" ";

//!!!!!!!!!!!!!!!!!!!!!!!!!!---MAIN PART---!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  for(j=1;j<=L_min_2;j++) for(k=1;k<K;k++) for(i=1;i<N;i++)
  {

    int cur=j*P+k*N_plus_1+i;
    U=up[cur]; //U=(*u)(i,j,k);
    V=vp[cur]; //V=(*v)(i,j,k);

#ifdef AP
    if(U<=E1) { f = negC1*U; }
    else if(U<=E2) { f = C2*(U-a); }
    else { f = negC3*(U-1); }

    uup[cur] = U + dt*(f-V) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    if(U<B1) { tau = tau1; } else if(U<B2) { tau = tau2; } else { tau=tau3; }

    vp[cur] = V + dt*(U-V)/tau;
#endif //AP

#ifdef BK
    uup[cur] = U + dt*ee*U*(1-U)*(U-(V+b)/a) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    vp[cur] = V + dt*(U-V);
#endif //BK

#ifdef WF
    U=U+uss; V=V+vss;
    uup[cur] = U + dt*ee*(U - U*U*U/3.0 - V) + dt_div_dxdx*
     (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U+ 6*uss)
     - uss;

    vp[cur] = V + dt*e*(U + g - b*V) - vss;
#endif //WF

#ifdef OR
    uup[cur] = U + dt*ee*(U - U*U - (f*V + fi)*(U-q)/(U+q)) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    vp[cur] = V + dt*(U-V);
#endif //OR

  } //end of Inner Loop

#ifdef PROBE
  U=up[ProbeK+ProbeJ*N_plus_1+ProbeI]; UU=(*uu)(ProbeI,ProbeJ,ProbeK);
  if(ProbeFlag==1)//waiting for Front
    {
     if((UU>.7)&&(UU>U))
            { ProbeFlag=0;

        *Pfile<<(DiscrTime-ProbeLastFront)*dt<<endl;

        ProbeLastFront = DiscrTime; }
    }
  else //ProbeFlag=0, waiting for Tail
    {
     if((UU<.2)&&(UU<U)) { ProbeFlag =1; }
    }
#endif //PROBE
```

```
//BOUNDARY CONDITIONS:

//RIGHT SEND:
  if(lrank<csize)
  {
   MPI_Send((void*)uu->GetLayerPtr(L_min_2), P, MPI_DOUBLE, right,
            11+DiscrTime, MPI::COMM_WORLD);
   //MPI_Send((void*)v->GetLayerPtr(L_min_2), P, MPI_DOUBLE, right,
   //         22+DiscrTime, MPI::COMM_WORLD);
  }

//LEFT RECEIVE:
  if(lrank==1)
    {
     for(i=0;i<=N;i++) for(k=0;k<=K;k++)
     {
       (*uu)(i,0,k) = (*u)(i,1,k);
       (*v)(i,0,k)  = (*v)(i,1,k);
     }
    }
  else
    {
     MPI_Recv((void*)uu->GetLayerPtr(0), P, MPI_DOUBLE, left,
              11+DiscrTime, MPI::COMM_WORLD, status);
     //MPI_Recv((void*)v->GetLayerPtr(0), P, MPI_DOUBLE, left,
     //        22+DiscrTime, MPI::COMM_WORLD, status);
    }

//LEFT SEND:
  if(lrank>1)
  {
   MPI_Send((void*)uu->GetLayerPtr(1), P, MPI_DOUBLE, left,
            33+DiscrTime, MPI::COMM_WORLD);
   //MPI_Send((void*)v->GetLayerPtr(1), P, MPI_DOUBLE, left,
   //         44+DiscrTime, MPI::COMM_WORLD);
  }

//RIGHT RECEIVE:
  if(lrank==csize)
  {
    for(i=0;i<=N;i++) for(k=0;k<=K;k++)
     {
       (*uu)(i,L_min_1,k) = (*u)(i,L_min_2,k);
       (*v)(i,L_min_1,k)  = (*v)(i,L_min_2,k);
     }
  }
  else
  {
   MPI_Recv((void*)uu->GetLayerPtr(L_min_1), P, MPI_DOUBLE, right,
           33+DiscrTime, MPI::COMM_WORLD, status);
   //MPI_Recv((void*)v->GetLayerPtr(L_min_1), P, MPI_DOUBLE, right,
   //        44+DiscrTime, MPI::COMM_WORLD, status);
  }

//Vertical SIDES:
  for(j=0;j<=L_min_1;j++) for(i=0;i<=N;i++)
  {
  (*uu)(i,j,0) = (*u)(i,j,1);
  (*v)(i,j,0)  = (*v)(i,j,1);
  (*uu)(i,j,K) = (*u)(i,j,K-1);
  (*v)(i,j,K)  = (*v)(i,j,K-1);
  }
//Horizontal SIDES:
  for(j=0;j<=L_min_1;j++) for(k=0;k<=K;k++)
  {
  (*uu)(0,j,k) = (*u)(1,j,k);
  (*v)(0,j,k)  = (*v)(1,j,k);
  (*uu)(N,j,k) = (*u)(N-1,j,k);
  (*v)(N,j,k)  = (*v)(N-1,j,k);
  }

 tmpu = u; u = uu; uu = tmpu;
 tmpup = up; up = uup; uup = tmpup;

 if(DiscrTime%TStep==0)
  {
  if(DiscrTime%TStepuv==0)
   {
     MPI_Send((void*)u->GetPtr(), L*(N+1)*(K+1),
          MPI_DOUBLE, 0, lrank, MPI::COMM_WORLD);
     //MPI_Isend((void*)v->GetPtr(), L*(N+1)*(K+1),
     //        MPI_DOUBLE, 0, lrank+100, MPI::COMM_WORLD, request);

   }//TStepuv

   //TIP CURVES:
   for(c=0;c<P*L;c++) tip[c]=0;
   for(j=0;j<=L_min_2;j++) for(i=1;i<N;i++) for(k=1;k<K;k++)
     {
     U=uup[j*P+k*N_plus_1+i];
     DU=up[j*P+k*N_plus_1+i]-U; DU = DU*DU;
     if((U>minU)&&(U<maxU)&&(DU<dtxdtxconst))
                     { tip[j*P+k*N_plus_1+i]=1; }
     }//for
   MPI_Send((void*) tip, L*P, MPI_CHAR, 0, lrank+200, MPI::COMM_WORLD);
  }//Tstep


 }//end of main loop

#ifdef PROBE
 Pfile->close();
#endif //PROBE

 return;

} //end of Slave()

//************************ - MASTER: IC's - ***********************

void ClassicScrollK_IC()
{ int i,j,k;
```

```
  Array3 *u=dat->u, *v=dat->v;
  for(i=N/2;i<=N;i++) for(j=0;j<=2*M/3;j++) for(k=0;k<=K;k++) (*u)(i,j,k)=1.0;
  for(i=N/2-5;i<=N;i++) for(j=0;j<=M/3;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=1.0;
  for(i=N/2-50;i<=N/2;i++) for(j=2*M/3-50;j<=2*M/3;j++) for(k=K/2;k<=K/2+50;k++) (*u)(i,j,k)=1.0;
  for(i=N/2-50;i<=N/2;i++) for(j=2*M/3-50;j<=2*M/3;j++) for(k=K/2;k<=K/2+50;k++) (*v)(i,j,k)=0.0;
#ifdef OR
  for(i=N/2-5;i<=N;i++) for(j=0;j<=M/3;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=0.4;
#endif
}

void clusterIC()
{ int i,j,k,l,cluster=6;
  Array3 *u=dat->u, *v=dat->v;
 for(l=0;l<cluster;l++) {
  for(i=N/2;i<=N;i++) for(j=2*M/5+5+l*20;j<=2*M/5+15+l*20;j++) for(k=0;k<=K;k++) (*u)(i,j,k)=1.0;
  for(i=N/2-5;i<=N;i++) for(j=2*M/5+l*20;j<=2*M/5+10+l*20;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=1.0;
#ifdef OR
  for(i=N/2-5;i<=N;i++) for(j=2*M/5+l*20;j<=2*M/5+10+l*20;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=0.4;
#endif
 }
}

inline double sq(double a) {return a*a;}

void ScrollRing_IC(int II,int JJ,int KK,int shiftI,int shiftJ,int shiftK,int R)
{ int i,j,k;
  double tmpval;
  if(R==0) { II=N/2; JJ=M/2; KK=K/2;
           R=shiftI=shiftJ=shiftK=20*int(.1/dat->dt); }
  double RR=R*R;
  Array3 *u=dat->u, *v=dat->v;
  for(i=0;i<=N;i++) for(k=0;k<=K;k++) for(j=0;j<=M;j++)
   if(sq(i-II)+sq(j-JJ)+sq(k-KK)<=RR) (*u)(i,j,k)=1.0;
  II+=shiftI; JJ+=shiftJ; KK+=shiftK;
  for(i=0;i<=N;i++) for(k=0;k<=K;k++) for(j=0;j<=M;j++)
   if(sq(i-II)+sq(j-JJ)+sq(k-KK)<=RR)
     {
      (*v)(i,j,k)=1.0;
#ifdef OR
      (*v)(i,j,k)=0.4;
#endif
     }
}



//************************* - Master() - ***********************
#ifdef GRAPHICS
#include <pthread.h> //for pthread_create()
#include <unistd.h> //for sleep()
//#include <time.h> //for nanosleep()
//timespec rqtp; //for nanosleep()
#include "graphth.cc"
#endif //GRAPHICS

void Master()
```

```
{
  int i,j,k,sl,c;
  int DT = dat->DT;
  int NumFramesuv = dat->NumFramesuv;
  int NumFrames = dat->NumFrames;
  int XStep = dat->XStep;
  int XStepuv = dat->XStepuv;
  int TStepuv = dat->TStepuv;
  int TStep = dat->TStep;


#ifdef TOFILE
  //IMPORTANT: 1. Need existent directory ./Data/
  //           2. The program has to be called ./spint.cc
  dat->dname = new char[100]; strcpy(dat->dname,""); //directory
  dat->uname = new char[100]; //u fncn
  dat->vname = new char[100]; //v fncn
  dat->tname = new char[100]; //tips
  dat->sname = new char[100]; //statistics
  char *tmpstring = new char[100];
  ofstream *DATE1; DATE1 = new ofstream("TMPDIRNAME",ios::out);
  *DATE1<<"Data/"<<dat->name; DATE1->close();
  system("date '+_%m-%d-%y_%H:%M/' >> TMPDIRNAME");
  system("mkdir `cat TMPDIRNAME`");
  system("cp spint.cc `cat TMPDIRNAME`");
  system("echo '\n' >> TMPDIRNAME");
  ifstream *DATE2; DATE2 = new ifstream("TMPDIRNAME",ios::in);
  //DATE2->gets(&tmpstring,'\n');//<--doesN'T work on gcc3.2, use "getline" instead
  DATE2->getline(tmpstring,100);
  DATE2->close();
  strcat(dat->dname,tmpstring);
  strcpy(dat->uname,dat->dname); strcat(dat->uname,"u.sim");
  strcpy(dat->vname,dat->dname); strcat(dat->vname,"v.sim");
  strcpy(dat->tname,dat->dname); strcat(dat->tname,"tc-K.sim");
  strcpy(dat->sname,dat->dname); strcat(dat->sname,"stat.txt");

  ofstream *Uf,*Vf,*Tf,*Sf,*STATUSf;
  //Uf = new ofstream(dat->uname,ios::out);
  //Vf = new ofstream(dat->vname,ios::out);
  Tf = new ofstream(dat->tname,ios::out);
  Sf = new ofstream(dat->sname,ios::out);
  STATUSf = new ofstream("TMPSTATUS",ios::out);

  *Sf<<"# Model: "<<dat->name<<endl;
  *Sf<<"# Directory name: "<<dat->dname<<endl;
#ifdef AP
  *Sf<<"# C2 = "<<dat->C2<<endl;
#endif //AP
#ifdef BK
  *Sf<<"# a = "<<dat->a<<endl;
  *Sf<<"# b = "<<dat->b<<endl;
  *Sf<<"# e = "<<dat->e<<endl;
#endif //BK
  *Sf<<"# RandomSeed = "<<dat->RandomSeed<<endl;
  *Sf<<"# NumGen = "<<dat->NumGen<<endl;
  *Sf<<"# N = "<<N<<", M = "<<M<<", K = "<<K<<", DT = "<<DT<<endl;
  *Sf<<"# dt = "<<dat->dt<<", dx = "<<dat->dx<<endl;
  *Sf<<"# "<<csize<<" slices of thickness "<<L<<endl;
  *Sf<<"# NumFrames for tip curves = "<<NumFrames<<endl;
  *Sf<<"# NumFrames for u/v = "<<NumFramesuv<<endl;
  *Sf<<"# XStep for tip curves = "<<XStep<<endl;
  *Sf<<"# XStepuv for u/v = "<<XStepuv<<endl;
  *Sf<<"# TStep for tip curves = "<<TStep<<endl;
  *Sf<<"# TStepuv for u/v = "<<TStepuv<<endl;
  *Sf<<"# Number of points for tip curves (K-projection): "
    <<(N-1)/XStep<<" x "<<(M-1)/XStep<<endl;
  *Sf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
  *Sf<<"# Number of points for u/v: "
    <<(N-1)/XStepuv<<" x "<<(M-1)/XStepuv<<" x "<<(K-1)/XStepuv<<endl;
  *Sf<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<NumFramesuv<<endl;

  *Sf<<"\n# Start & Finish times:\n";
  Sf->close();
  system("date >> `cat TMPDIRNAME`/stat.txt");

  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
  //*U<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<(K-1)/XStepuv<<" "<<NumFramesuv<<endl;
  //*V<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<(K-1)/XStepuv<<" "<<NumFramesuv<<endl;
#endif // TOFILE

  Array3* u = dat->u = new Array3(N,M,K,0);
  Array3* v = dat->v = new Array3(N,M,K,0);
  char* tip = dat->tip = new char[(N+1)*(M+1)*(K+1)];
    for(c=0;c<(N+1)*(M+1)*(K+1);c++) tip[c]=0;

//INITIAL CONDITIONS:
  //single classic spiral:
    //ClassicScrollK_IC();
    //ScrollRing_IC(0,0,0,0,0,0,0);

//ScrollRing_IC(N/2,M/2,K/2,20,20,20,40);

//TMP:
//ScrollRing_IC(N/2,M/2,K/2,16,18,20,35);
ClassicScrollK_IC();
//clusterIC();

  //propagate I.C.'s to slaves:
  //MPI_Request *request = new MPI_Request; //for MPI_Isend
  for(sl=1;sl<=csize;sl++)
    {
    MPI_Send((void*)u->GetLayerPtr((sl-1)*(L-2)), L*P, MPI_DOUBLE,
            ltor[sl], 1111+sl, MPI::COMM_WORLD);
    MPI_Send((void*)v->GetLayerPtr((sl-1)*(L-2)), L*P, MPI_DOUBLE,
            ltor[sl], 2222+sl, MPI::COMM_WORLD);
    }

#ifdef GRAPHICS //create graphics thread:
  pthread_t graphthread;
  pthread_create(&graphthread, NULL, graphthfn, NULL);
#endif //GRAPHICS
```

```
//MAIN LOOP (monitor slaves):
  MPI_Status *status = new  MPI_Status; //for MPI_Recv
  for(int DiscrTime=0;DiscrTime<=DT;DiscrTime=DiscrTime+TStep)
  {
//COLLECT info from slaves:
      for(sl=1;sl<=csize;sl++)
        {
         if(DiscrTime%TStepuv==0)
           {
             MPI_Recv((void*)u->GetLayerPtr((sl-1)*(L-2)), L*P,
                     MPI_DOUBLE, ltor[sl], sl, MPI::COMM_WORLD, status);
            //MPI_Recv((void*)v->GetLayerPtr((k-1)*(L-2)), L*P,
            //        MPI_DOUBLE, ltor[sl], sl+100, MPI::COMM_WORLD, status);
           }
         MPI_Recv((void*)(tip+(sl-1)*(L-2)*P), L*P, MPI_CHAR,
                 ltor[sl], sl+200, MPI::COMM_WORLD, status);
        }

#ifdef TOFILE
      /*
         for(i=1;i<=N-1;i=i+XStepuv)
         for(j=1;j<=M-1;j=j+XStepuv)
         for(k=1;k<=K-1;k=k+XStepuv)
          {
           Uf->put( (char) (100*((*u)(i,j,k))) );
            //cout<<((*u)(i,j,k))<<"  "<<endl;
          }
      */

      char tmpchar1,tmpchar2;
      int N_plus_1=N+1;
      for(int i=1;i<=N-1;i=i+XStep) for(int j=1;j<=M-1;j=j+XStep)
       {
         tmpchar1=tmpchar2=0;
         for(int k=K-1;k>=1;k=k-XStep)
           {
            tmpchar1 = tip[j*P+k*N_plus_1+i];
            if(tmpchar1) tmpchar2 = 100 + 155*(K-k)/K;
           }
         Tf->put(tmpchar2);
       }

       *STATUSf<<"TipFrame "<<DiscrTime/TStep+1<<" of "<<NumFrames<<endl;
#endif //TOFILE

  }//for(DiscrTime=...) //end of MAIN LOOP

#ifdef TOFILE
  //Uf->close();
  //Vf->close();
  Tf->close();
  STATUSf->close();
  system("date >> `cat TMPDIRNAME`/stat.txt");
#endif //TOFILE
  return;
}
```

```
//************************* - MAIN - ***************************

int main(int argc, char *argv[])
{
 MPI::Init(argc, argv);
 size = MPI::COMM_WORLD.Get_size(); csize = size-1;
 rank = MPI::COMM_WORLD.Get_rank();
 char processor_name[MPI_MAX_PROCESSOR_NAME]; int namelen;
 MPI_Get_processor_name(processor_name,&namelen);

 rtol = new int[size]; ltor = new int[size];
 for(int i=0;i<size;i++) rtol[i]=ltor[i]=i;
 if(OPT&&(size==NP))
 {
  //rtol[0]=0; rtol[1]=3; rtol[2]=5; //....
  //for(int i=0;i<size;i++) ltor[rtol[i]]=i;
 }
 lrank = rtol[rank];

 //cout<<"    Rank = "<<rank<<"/"<<size
 //<<" Logical Rank = "<<lrank<<" on Node = "<<processor_name<<endl;
 //cout.flush();

 dat = new DATA;

 M = (M - 1)/csize*csize + 1;
 //N = (N - 1)/(dat->XStep)*(dat->XStep) + 1;
 N = (N - 1)/(dat->XStepuv)*(dat->XStepuv) + 1;
 //K = (K - 1)/(dat->XStep)*(dat->XStep) + 1;
 K = (K - 1)/(dat->XStepuv)*(dat->XStepuv) + 1;

 L = 2 + (M - 1)/csize;
 P = (N + 1)*(K + 1);

 if(rank==0) { Master(); } else { Slave(); }

 MPI::Finalize();
 return 0;
}
```

A.2 DPP Version

```
/*
-----------------------------------------------------------------------------
  spint.cc: Uniprocessor/DPP 3D FHN Simulation C++ Code.
-----------------------------------------------------------------------------
2-23-06, RZ/KP: 3D DPP code for 4 models:
 - 1. Pushchino Model (AP)
 - 2. Barkley Model (BK)
 - 3. Winfree Model (WF)
 - 4. Oregonator Model (OR) (2-component)
for Parallel Monster cluster
(based on 3D-MPI-MM and 2D-DPP-MM)
*/
using namespace std;

//#define GRAPHICS // show results graphically using OpenGL
//#define TOFILE // save u into a file
#define PROBE // save probe results

//#define AP // PUSHCHINO model
//#define AP_DPP

//#define BK // BARKLEY model
//#define BK_DPP

#define WF // WINFREE model
#define WF_DPP

//#define OR // (2-component) OREGONATOR model
//#define OR_DPP


#include <iostream>
#include <fstream> //for file io
#include <cstdlib> //for exit(), system(), rand()
#include <cstring> //for file-name ops
//#include <cstdio> //for gets()
//#include <cmath> //for M_PI

//global stuff:
int N,M,K; //mesh size: hight, width, depth
int P; // number of points in a layer

//*******************************************

class Array3;

class DATA
{
public:
 DATA()
 {
  N=100; K=100; M=201; //default for all models
  DT=100000; // default

N=100; K=100; M=101;
DT = 50000;

N=100; K=100; M=101;
DT = 10000;


N=150; K=150; M=151;
DT=1000000;

N=300; K=300; M=301;
DT=65000; //300x300x300: 20 hrs WF,OR
DT=100000; //300x300x300: 30 hrs WF,OR
DT=150000; //300x300x300: 45 hrs WF,OR

//TMP:
N=400; K=400; M=400;
//DT=1200000;


#ifdef AP
   name="AP";
  //dx = 1.2; dt = .1;
  dx = 0.6; dt = 0.05;
dx = 1.2; dt = .1;
  C1=4; C3=14; C2=0.75; //base parameter value
  E1=0.018; a=(C1+C2)*E1/C2;
  E2=(C2*a+C3)/(C2+C3);
  tau1=tau3=0.5; tau2=16.66;
  B1=0.01; B2=0.95;

  //DT=100000; //200x300x100: 6 hours
  //DT=1000000; //200x300x100: 60 hours
#endif //AP


#ifdef BK
  name = "BK";

//MIKHAILOV: e=0.02;
e = 0.02;
  //a=1.0; b=.2; //default parameters
  //a=1.1; b=.24; //base-point parameters (support S, M2, M3)
  //(Note: The largest dt with e=.02, a=1.0, b=.15, dx=1.0 is: dt = 0.0594!!!)
  //(Note: maximum dt is usually approx.= e)
  //dx = 1.0; dt = 0.05;
  dx = 0.5; dt = 0.02;

dx = 1.0; dt = 0.05;
a=1.1; b=.24;

dx = 1.0; dt = 0.05;
a=1.4; b=.23; //3D!!!!
```

```
//TMP:
dx = 0.4; dt = 0.01;

  //a=1.1; b=.2; //small core for benchmarking

//BARKLEY: e=0.005 (1/200) & 0.00666 (1/150); (Note: largest dt==e):
  //e = 0.005;
  //dx = 0.5; dt=0.005;   //too crude!!!
  //e= 0.005; dx = 0.25; dt = 0.0025;   a=1.0; b=.3;

#endif //BK


#ifdef WF
  name="WF";

  //Biktashev's IJBC98 paper:
  e = 0.3; b = 0.75; g = 0.5; // 1/e==excitability (e=0.4 -> very low)
  //dx = 0.5; dt = 0.03;
  dx = 0.5; dt = 0.02;
  uss=-1.0; vss=-2.0/3.0;
  e=0.38; //RZ, Summer2004: discovered stable pairs!!!
  e=0.39; //Base value: supports, M2, M3,..., M6,...

  //---------------------
  //e = 0.3; b = 0.5; g = 0.7; uss,vss=??? //Winfree(ch10)
  //e = 0.2; b = 0.5; g = 0.9; uss,vss=??? //Winfree(ch10) - more generic
  //---------------------

e=0.39;

 e=0.36; //FOR 3D!!!

#endif //WF


#ifdef OR
  name="OR";
  //From AlonsoPRL01/Beato paper:
  //e = 1/11.7, 1/1059; f = 1.4; fi = 0.005 -- 0.096; q = 0.002;

  //RZ, Summer 2004, Discovered MAS:
  //f = 1.4; q = 0.002; e = 1.0/11; fi = 0.03;
  //dx = 0.5; dt = 0.001; //needed for e < 0.08
  dx = 0.5; dt = 0.002; //OK for e > 0.08
  f=1.4; q=0.002; e=0.1; fi=0.03; //base point for IJBC!
  //Note1: steady-state is (a,a), where a is small(?)
  //Note2: for initial patches can use u=1, v=.4...
  //Note3: highest value of u is about 0.4!

#endif //OR

TStepuv = 1000000;
TStep = DT;
//TStepuv=TStep = 100; //for graphical debugging
```

```
TStepuv=TStep = 100;


//TMP:
//TStepuv=TStep = 500;


  NumFramesuv = DT/TStepuv + 1;
  NumFrames = DT/TStep + 1;
  XStepuv=XStep=1;

  RandomSeed=1;
  NumGen=10;

  WSize = 200; //for OpenGL graphical display
 }

//data:
 Array3 *u, *v;
 char *tip;
 int DT; //terminal discrete time (# of iterations)
 int WSize; //OpenGL graphical window size
 double dt, dx; //step sizes


#ifdef BK
 double e, a, b;
#endif

#ifdef AP
 double C1,C2,C3,a,E1,E2,E3,tau1,tau2,tau3,B1,B2;
#endif

#ifdef WF
 double e, b, g, uss, vss;
#endif

#ifdef OR
 double e, f, q, fi;
#endif

 int TStepuv; // how often write u/v to a file
 int TStep; //how often to compute/transmit tip curve
 int XStepuv, XStep; // how much to write to file

 char *name, *dname; char *uname, *vname, *tname, *sname;
 int NumFramesuv, NumFrames;
 int RandomSeed;
 int NumGen;

};//class DATA

DATA *dat; //global pointer to data storage

//*******************************************
```

```
class Array3
{
public:
 Array3() {n1=n2=n3=l=0; ap=NULL;}
 Array3(int n1_, int n2_, int n3_, double a)
 {
   n1=n1_; n2=n2_; n3=n3_;
  l = (n1+1)*(n2+1)*(n3+1);
  ap = new double [l];
  for(int i=0; i<l; i++) ap[i] = a;
 }

 ~Array3() { delete [] ap; }

 void Set(double a) { for(int i=0; i<l; i++) ap[i] = a; }

 inline double& operator()(int i, int j, int k)
 {
  //if ((i<0)||(j<0)||(k<0)||(i>n1)||(j>n2)||(k>n3))
  //{cout<<"\nArray referenced out of bounds!\n"; exit(5);}
  return ap[(n1+1)*(n3+1)*j + (n1+1)*k + i] ;
 }

 double* GetLayerPtr(int n) { return ap + n*(n1+1)*(n3+1); }
 double* GetPtr() { return ap; }

private:
  double *ap; //array pointer
  int n1, n2, n3, l;
}; //end of class Array3


//************************* - IC's - ************************

void ClassicScrollK_IC()
{ int i,j,k;
  Array3 *u=dat->u, *v=dat->v;
  for(i=N/2;i<=N;i++) for(j=0;j<=2*M/3;j++) for(k=0;k<=K;k++) (*u)(i,j,k)=1.0;
  for(i=N/2-5;i<=N;i++) for(j=0;j<=M/3;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=1.0;
#ifdef OR
  for(i=N/2-5;i<=N;i++) for(j=0;j<=M/3;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=0.4;
#endif
}

void cluster_IC()
{ int i,j,k,l,cluster=1;
  Array3 *u=dat->u, *v=dat->v;
 for(l=0;l<cluster;l++) {
  for(i=N/2;i<=N;i++) for(j=2*M/5+5+l*20;j<=2*M/5+15+l*20;j++) for(k=0;k<=K;k++) (*u)(i,j,k)=1.0;
  for(i=N/2-5;i<=N;i++) for(j=2*M/5+l*20;j<=2*M/5+10+l*20;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=1.0;
#ifdef OR
  for(i=N/2-5;i<=N;i++) for(j=2*M/5+l*20;j<=2*M/5+10+l*20;j++) for(k=0;k<=K;k++) (*v)(i,j,k)=0.4;
#endif
 }
}
```

```
inline double sq(double a) {return a*a;}

void ScrollRing_IC(int II,int JJ,int KK,int shiftI,int shiftJ,int shiftK,int R)
{ int i,j,k;
  double tmpval;
  if(R==0) { II=N/2; JJ=M/2; KK=K/2;
             R=shiftI=shiftJ=shiftK=20*int(.1/dat->dt); }
  double RR=R*R;
  Array3 *u=dat->u, *v=dat->v;
  for(i=0;i<=N;i++) for(k=0;k<=K;k++) for(j=0;j<=M;j++)
   if(sq(i-II)+sq(j-JJ)+sq(k-KK)<=RR) (*u)(i,j,k)=1.0;
  II+=shiftI; JJ+=shiftJ; KK+=shiftK;
  for(i=0;i<=N;i++) for(k=0;k<=K;k++) for(j=0;j<=M;j++)
   if(sq(i-II)+sq(j-JJ)+sq(k-KK)<=RR)
     {
      (*v)(i,j,k)=1.0;
#ifdef OR
      (*v)(i,j,k)=0.4;
#endif
     }
}


#ifdef GRAPHICS
#include <pthread.h> //for pthread_create()
#include <unistd.h> //for sleep()
//#include <time.h> //for nanosleep()
//timespec rqtp; //for nanosleep()
#include "graphth.cc"
#endif //GRAPHICS


//==================================MAIN()==================================

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

 dat = new DATA;

#ifdef AP
  if(argc==2) {dat->C2=atof(argv[1]);}
#endif //AP


#ifdef BK
  if(argc==3) {dat->a=atof(argv[1]); dat->b=atof(argv[2]);}
#endif //BK

#ifdef WF
  if(argc==2) {dat->e=atof(argv[1]);}
#endif //WF

#ifdef OR
  //if(argc==2) {dat->e=atof(argv[1]);}
  if(argc==2) {dat->fi=atof(argv[1]);}
#endif //OR
```

```
  N = (N - 1)/(dat->XStep)*(dat->XStep) + 1;
  M = (M - 1)/(dat->XStep)*(dat->XStep) + 1;
  K = (K - 1)/(dat->XStep)*(dat->XStep) + 1;
  P = (N + 1)*(K + 1);

  int i,j,k,c;
  double dt = dat->dt, dx = dat->dx;
  int DT = dat->DT;
  int NumFramesuv = dat->NumFramesuv;
  int NumFrames = dat->NumFrames;
  int XStep = dat->XStep;
  int XStepuv = dat->XStepuv;
  int TStepuv = dat->TStepuv;
  int TStep = dat->TStep;


  Array3* tmpu;
  Array3* u = dat->u = new Array3(N,M,K,0);
  Array3* uu = new Array3(N,M,K,0);
  Array3* v = dat->v = new Array3(N,M,K,0);
  char* tip = dat->tip = new char[(N+1)*(M+1)*(K+1)];
    for(c=0;c<(N+1)*(M+1)*(K+1);c++) tip[c]=0;

#ifdef AP
  double C1 = dat->C1, C2=dat->C2, C3 = dat->C3;
  double negC1=-C1, negC3=-C3;
  double f,tau;
  double a = dat->a;
  double E1 = dat->E1, E2 = dat->E2, E3 = dat->E3;
  double tau1 = dat->tau1, tau2 = dat->tau2, tau3 = dat->tau3;
  double B1 = dat->B1, B2 = dat->B2;
#endif //AP

#ifdef BK
  double a=dat->a, b=dat->b, e=dat->e;
  double ee = 1/e;
#endif //BK

#ifdef WF
  double e=dat->e, b=dat->b, g=dat->g;
  double ee = 1/e;
  double uss=dat->uss, vss=dat->vss;
#endif //WF

#ifdef OR
  double e=dat->e, f=dat->f, q=dat->q, fi=dat->fi;
  double ee = 1/e;
#endif //OR


 //for FILAMENT DETECTION:
 double minU=.1, maxU=.3;
#ifdef AP
 double dtxdtxconst= dt*dt*.0009;
```

```
#endif //AP
#ifdef BK
 double dtxdtxconst= dt*dt*.1;
#endif //BK
#ifdef WF
 double dtxdtxconst= dt*dt*.1;
 minU=.2; maxU=.6;
#endif //WF
#ifdef OR
 double dtxdtxconst= dt*dt*.1;
#endif //OR


 double U,UU,UUU,V,DU; // temporary values for u,v,etc. to speed up
 double *up, *uup, *tmpup, *vp;
 double dt_div_dxdx = dt/dx/dx;
 int N_plus_1 = N+1;

#ifdef TOFILE
  //IMPORTANT: 1. Need existent directory ./Data/
  //           2. The program has to be called ./spint.cc
  dat->dname = new char[100]; //directory
  dat->uname = new char[100]; //u fncn
  dat->vname = new char[100]; //v fncn
  dat->tname = new char[100]; //tips
  dat->sname = new char[100]; //statistics
  char *tmpstring = new char[100];
  ofstream *DATE1; DATE1 = new ofstream("TMPDIRNAME",ios::out);
  *DATE1<<"Data/"<<dat->name; DATE1->close();
  system("date '+_%m-%d-%y_%H:%M/' >> TMPDIRNAME");
  system("mkdir `cat TMPDIRNAME`");
  system("cp spint.cc `cat TMPDIRNAME`");
  system("echo '\n' >> TMPDIRNAME");
  ifstream *DATE2; DATE2 = new ifstream("TMPDIRNAME",ios::in);
  //DATE2->gets(&tmpstring,'\n');//<--doesN'T work on gcc3.2, use "getline" instead
  DATE2->getline(tmpstring,100);
  DATE2->close();
  strcat(dat->dname, tmpstring);
  strcpy(dat->uname,dat->dname); strcat(dat->uname,"u.sim");
  strcpy(dat->vname,dat->dname); strcat(dat->vname,"v.sim");
  strcpy(dat->tname,dat->dname); strcat(dat->tname,"t.sim");
  strcpy(dat->sname,dat->dname); strcat(dat->sname,"stat.txt");

  ofstream *Uf,*Vf,*Tf,*Sf,*STATUSf;
  //Uf = new ofstream(dat->uname,ios::out);
  //Vf = new ofstream(dat->vname,ios::out);
  Tf = new ofstream(dat->tname,ios::out);
  Sf = new ofstream(dat->sname,ios::out);
  STATUSf = new ofstream("TMPSTATUS",ios::out);

  *Sf<<"# Model: "<<dat->name<<endl;
  *Sf<<"# Directory name: "<<dat->dname<<endl;
#ifdef AP
  *Sf<<"# C2 = "<<dat->C2<<endl;
#endif //AP
#ifdef BK
```

```
  *Sf<<"# a = "<<dat->a<<endl;
  *Sf<<"# b = "<<dat->b<<endl;
  *Sf<<"# e = "<<dat->e<<endl;
#endif //BK
  *Sf<<"# RandomSeed = "<<dat->RandomSeed<<endl;
  *Sf<<"# NumSpir = "<<dat->NumSpir<<endl;
  *Sf<<"# N = "<<N<<", M = "<<M<<", DT = "<<DT<<endl;
  *Sf<<"# dt = "<<dat->dt<<", dx = "<<dat->dx<<endl;
  //*Sf<<"# NumFrames for tips = "<<NumFrames<<endl;
  *Sf<<"# NumFrames for u/v = "<<NumFramesuv<<endl;
  //*Sf<<"# XStep for tips = "<<XStep<<endl;
  *Sf<<"# XStepuv for u/v = "<<XStepuv<<endl;
  //*Sf<<"# TStep for tips = "<<TStep<<endl;
  *Sf<<"# TStepuv for u/v = "<<TStepuv<<endl;
  //*Sf<<"# Number of points for tips: "
  //<<(N-1)/XStep<<" x "<<(M-1)/XStep<<endl;
  //*Sf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
  *Sf<<"# Number of points for u/v: "
    <<(N-1)/XStepuv<<" x "<<(M-1)/XStepuv<<endl;
  *Sf<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<NumFramesuv<<endl;
  *Sf<<"\n# Start & Finish times:\n";
  Sf->close();
  system("date >> `cat TMPDIRNAME`/stat.txt");
  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
  //*Uf<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<NumFramesuv<<endl;
  //*Vf<<(N-1)/XStepuv<<" "<<(M-1)/XStepuv<<" "<<NumFramesuv<<endl;
#endif //TOFILE

#ifdef AP_DPP
  dat->tname = new char[100]; //tip curve fncn
  strcat(dat->tname,"/scratch/t");
  //Now, we have only one parameter, C2
  if(argc>1) {strcat(dat->tname,"_c2_"); strcat(dat->tname,argv[1]);}
  strcat(dat->tname,".sim");
  ofstream *Tf;
  Tf = new ofstream(dat->tname,ios::out);
  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
#endif //AP_DPP

#ifdef BK_DPP
  dat->tname = new char[100]; //tip curve fncn
  strcat(dat->tname,"/scratch/t");
  if(argc>1) {strcat(dat->tname,"_a"); strcat(dat->tname,argv[1]);}
  if(argc>2) {strcat(dat->tname,"_b"); strcat(dat->tname,argv[2]);}
  strcat(dat->tname,".sim");
  ofstream *Tf;
  Tf = new ofstream(dat->tname,ios::out);
  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
#endif //BK_DPP


#ifdef WF_DPP
  dat->tname = new char[100]; //tip curve fncn
  strcat(dat->tname,"/scratch/t");
  //only one parameter, e
  if(argc>1) {strcat(dat->tname,"_e"); strcat(dat->tname,argv[1]);}
  strcat(dat->tname,".sim");
  ofstream *Tf;
  Tf = new ofstream(dat->tname,ios::out);
  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
#endif //WF_DPP

#ifdef OR_DPP
  dat->tname = new char[100]; //tip curve fncn
  strcat(dat->tname,"/scratch/t");
  //only one parameter, e or fi
  //if(argc>1) {strcat(dat->tname,"_e"); strcat(dat->tname,argv[1]);}
  if(argc>1) {strcat(dat->tname,"_fi"); strcat(dat->tname,argv[1]);}
  strcat(dat->tname,".sim");
  ofstream *Tf;
  Tf = new ofstream(dat->tname,ios::out);
  *Tf<<(N-1)/XStep<<" "<<(M-1)/XStep<<" "<<NumFrames<<endl;
#endif //OR_DPP

#ifdef PROBE
 //probe file name:
 char *pfname = new char[100];
 strcpy(pfname,"/scratch/probe");
 strcat(pfname, argv[1]);
#ifdef BK_DPP
 strcat(pfname, argv[2]);
#endif //BK_DPP

 ofstream *Pfile; Pfile = new ofstream(pfname, ios::out);

 int ProbeJ = M/2; int ProbeI = N/2; int ProbeK = K/2; //probe coordinates
 int ProbeFlag = 1; //waiting for wave front
 int ProbeLastFront = 0;
#endif //PROBE

//INITIAL CONDITIONS:
  //single classic spiral:
    //ClassicScrollK_IC();
    //ScrollRing_IC(0,0,0,0,0,0,0);

//ScrollRing_IC(N/2,M/2,K/2,20,20,20,40);

//TMP:
ScrollRing_IC(N/2,M/2,K/2,16,18,20,35);
//cluster_IC();
//ClassicScrollK_IC();

#ifdef GRAPHICS //create graphics thread:
  pthread_t graphthread;
  pthread_create(&graphthread, NULL, graphthfn, NULL);
#endif //GRAPHICS

//----------------------MAIN (TIME) LOOP-------------------------------

 for(int DiscrTime=0;DiscrTime<=DT;DiscrTime++)
 {
  up=u->GetPtr();
  uup=uu->GetPtr();
  vp =v->GetPtr();

  //Periodically zeroing out small stuff:
  if (DiscrTime%100==0) for(j=0;j<=M;j++) for(k=0;k<=K;k++) for(i=0;i<=N;i++)
    { int cur=j*P+k*N_plus_1+i; U=up[cur]; V=vp[cur];
      if((U<.001)&&(U>-.001)) up[cur]=0; if((V<.001)&&(V>-.001)) vp[cur]=0;
    }

  //cout one pixel (for debugging):
  //if(lrank==1) cout<<up[5*P+5*N_plus_1+5]<<" ";

//!!!!!!!!!!!!!!!!!!!!!!!!!!---MAIN PART---!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  for(j=1;j<M;j++) for(k=1;k<K;k++) for(i=1;i<N;i++)
  {

    int cur=j*P+k*N_plus_1+i;
    U=up[cur]; //U=(*u)(i,j,k);
    V=vp[cur]; //V=(*v)(i,j,k);

#ifdef AP
    if(U<=E1) { f = negC1*U; }
    else if(U<=E2) { f = C2*(U-a); }
    else { f = negC3*(U-1); }

    uup[cur] = U + dt*(f-V) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    if(U<B1) { tau = tau1; } else if(U<B2) { tau = tau2; } else { tau=tau3; }

    vp[cur] = V + dt*(U-V)/tau;
#endif //AP

#ifdef BK
    uup[cur] = U + dt*ee*U*(1-U)*(U-(V+b)/a) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    vp[cur] = V + dt*(U-V);
#endif //BK

#ifdef WF
    U=U+uss; V=V+vss;
    uup[cur] = U + dt*ee*(U - U*U*U/3.0 - V) + dt_div_dxdx*
     (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
      + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U+ 6*uss)
     - uss;

    vp[cur] = V + dt*e*(U + g - b*V) - vss;
#endif //WF

#ifdef OR
    uup[cur] = U + dt*ee*(U - U*U - (f*V + fi)*(U-q)/(U+q)) + dt_div_dxdx*
    (up[cur+1] + up[cur-1] + up[cur+P] + up[cur-P]
       + up[cur+N_plus_1] + up[cur-N_plus_1] - 6*U);

    vp[cur] = V + dt*(U-V);
#endif //OR

  } //end of Inner Loop

#ifdef PROBE
  //UU=up[ProbeJ*P+ProbeK*N_plus_1+ProbeI]; UUU=(*uu)(ProbeJ,ProbeK,ProbeI);
  UU=(*u)(ProbeJ,ProbeK,ProbeI); UUU=(*uu)(ProbeJ,ProbeK,ProbeI);
  //*Pfile<<UU<<" "<<UUU<<endl; //debuging
  if(ProbeFlag==1)//waiting for Front
    {
     if((UUU>.7)&&(UUU>UU))
            { ProbeFlag=0;

         *Pfile<<(DiscrTime-ProbeLastFront)*dt<<endl;

         ProbeLastFront = DiscrTime; }
    }
  else //ProbeFlag=0, waiting for Tail
    {
     if((UUU<.2)&&(UUU<UU)) { ProbeFlag =1; }
    }
#endif //PROBE

//BOUNDARY CONDITIONS:
//LEFT & RIGHT:
  for(i=0;i<=N;i++) for(k=0;k<=K;k++)
   {
   (*uu)(i,0,k) = (*u)(i,1,k);
   (*v)(i,0,k)  = (*v)(i,1,k);
   (*uu)(i,M,k) = (*u)(i,M-1,k);
   (*v)(i,M,k)  = (*v)(i,M-1,k);
   }
//FRONT & BACK:
  for(j=0;j<=M;j++) for(i=0;i<=N;i++)
   {
   (*uu)(i,j,0) = (*u)(i,j,1);
   (*v)(i,j,0)  = (*v)(i,j,1);
   (*uu)(i,j,K) = (*u)(i,j,K-1);
   (*v)(i,j,K)  = (*v)(i,j,K-1);
   }
//TOP & BOTTOM:
  for(j=0;j<=M;j++) for(k=0;k<=K;k++)
   {
   (*uu)(0,j,k) = (*u)(1,j,k);
   (*v)(0,j,k)  = (*v)(1,j,k);
   (*uu)(N,j,k) = (*u)(N-1,j,k);
   (*v)(N,j,k)  = (*v)(N-1,j,k);
   }

 tmpu = u; u = uu; uu = tmpu;
 tmpup = up; up = uup; uup = tmpup;
 //TIP CURVES:
if(DiscrTime%TStep==0)
  {
   for(c=0;c<P*M;c++) tip[c]=0;
   for(j=1;j<M;j++) for(i=1;i<N;i++) for(k=1;k<K;k++)
     {
     U=uup[j*P+k*N_plus_1+i];
     DU=up[j*P+k*N_plus_1+i]-U; DU = DU*DU;
     if((U>minU)&&(U<maxU)&&(DU<dtxdtxconst))
                        { tip[j*P+k*N_plus_1+i]=1; }
     }//for
  }

#if defined(TOFILE)||defined(AP_DPP)||defined(BK_DPP)||defined(WF_DPP)||defined(OR_DPP)
   //save tip curve data into a file
  if(DiscrTime % TStep == 0)
    {
     char tmpchar1,tmpchar2;
     for(int i=1;i<=N-1;i=i+XStep) for(int j=1;j<=M-1;j=j+XStep)
      {
       tmpchar1=tmpchar2=0;
       for(int k=K-1;k>=1;k=k-XStep)
         {
          tmpchar1 = tip[j*P+k*N_plus_1+i];
          if(tmpchar1) tmpchar2 = 100 + 155*(K-k)/K;
         }
       Tf->put(tmpchar2);
      }
#ifdef TOFILE
      *STATUSf<<"TipFrame "<<DiscrTime/TStep+1<<" of "<<NumFrames<<endl;
#endif //TOFILE
    }
#endif //FILE||DPP

 } //****************************** End of Main Loop ********************************

#ifdef PROBE
 Pfile->close();
#endif //PROBE

#ifdef TOFILE
  Uf->close();
  //Vf->close();
  //Tf->close();
  STATUSf->close();
  system("sleep 2");
  //system("bringprobes.csh");
  system("date >> `cat TMPDIRNAME`/stat.txt");
#endif //TOFILE

#if defined(AP_DPP)||defined(BK_DPP)||defined(WF_DPP)||defined(OR_DPP)
  Tf->close();
#endif

 return 0;
}
```

References

1. Gil Bub, Alvin Shrier, and Leon Glass *“Spiral Waves Break Hearts: New Research Stresses the Importance of Communication Between Cardiac Cells” Inside Science News Services*, February, 2005
2. R. M. Zaritski and A. M. Pertsov *"Stable Spiral Structures and Their Interaction in Two-Dimensional Excitable Media." Physical Review E*, Vol. 66 (6), art. no. 066120, p. 1-6, 2002
3. R. M. Zaritski, S. F. Mironov, and A. M. Pertsov "*Intermittent Self-Organization of Scroll Wave Turbulence in Three-Dimensional Excitable Media.*" *Physical Review Letters*, Vol. 92 (16), art. no. 168302, p. 1-4, 2004
4. R. M. Zaritski, J. Ju and I. Ashkenazi “*Spontaneous Formation of Multi-Armed Spiral Waves in various Simple Models of Weakly Excitable Media.*“ International Journal of Bifurcation and Chaos, Vol. 15 (12), p. 4087-4094, 2005
5. R. M. Zaritski and K. Pal *“Optimization of Simple Reaction-Diffusion PDE Simulations on a 64-Opteron Linux Cluster.” Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, Vol. 2, edited by H. R. Arabnia et. al., p. 730-736, CSREA Press, 2006
6. “Parallel Monster” Cluster Home Page: *http://roman.montclair.edu/Research/Parallel/PM/*
7. MSU Parallel Processing Laboratory Home Page: *http://roman.montclair.edu/Research/Parallel/*
8. Microway Inc. Home Page: *http://www.microway.com/*
9. R. M. Zaritski *"Using Open Source Software for Scientific Simulations, Research, and Publishing." The Journal of Computing Sciences in Colleges*, Vol. 19 (2), p. 218-222, December 2003
10. MPICH Home Page: *http://www-unix.mcs.anl.gov/mpi/mpich/*
11. LAM Parallel Computing Home Page: *http://www.lam-mpi.org/*
12. Message Passing Interface Forum Home Page: *http://www.mpiforum.org/*
13. R. M. Zaritski "MPEG Compression of 2-D Data Files." *HOWTO, Linux Journal Web Publication, http://www.linuxjournal.com/article/6273*, September 2002
14. R. M. Zaritski and I. Kaplun *“Parallel Simulations of Spontaneous Multi-Armed Scroll Waves in Excitable Media” (Abstract), Proceedings of the Fifth IMACS International Conference on Nonlinear Evolution Equations and Wave Phenomena: Computation and Theory*, Athens GA, 2007
15. I. Kaplun *“Using 64-Opteron Linux Cluster for Numerical Simulations of Spontaneous Multi-Armed Scroll Waves in Excitable Media” (Abstract), Proceedings of Sigma Xi First Annual Student Research Symposium*, 2007
16. Beowulf Home Page: http://www.beowulf.org
17. Top 500 Supercomputers Home Page: *http://www.top500.org/*