Using a generalized MPI Interface for Python

MYMPI

Timothy H. Kaiser, Ph.D.
tkaiser@sdsc.edu
Sarah Healy Leesa Brieger

http://nbcr.sdsc.edu/

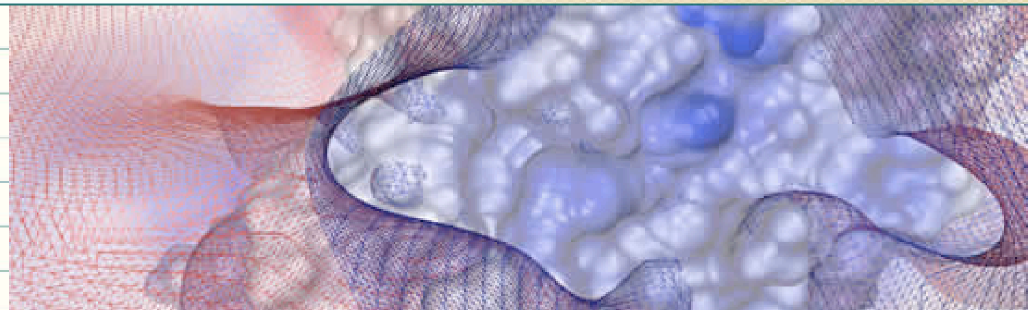
NBCR

NATIONAL BIOMEDICAL COMPUTATION RESOURCE

Conduct, catalyze and enable multiscale biomedical research

Events | Publications | Directory | Forum | Partners | Mission | Intranet | Home

- News
- Projects
- Related Activities
- Tools/Downloads
- Funding Opps
- User Services
- Search



HEADLINES

3-D Rabbit Heart Simulation Studies Offers New Insight to Arrhymia Induction

iAPBS Enables APBS Functions In CHARMM

Biological Networks provides new interactive environment for graph queries.

FEATURED CONFERENCES

Feb 18-22, 2006: Biophysical Society Meeting, 2006, Salt Lake City, Utah

Nov 12-18, 2005: Supercomputing 2005, Seattle, Washington.

Jul 20-22, 2005: DILS 2005, San Diego, CA.

Jun 25-June 29, 2005: ISMB 2005, Detroit, Michigan.

RELATED LINKS

Request an account

Join our mailing lists

Attend our training classes

View cluster status

Ask us a question

This Resource is supported by the National Institutes of Health (NIH) through a National Center for Research Resources program grant (P 41 RR08605) to researchers at the University of California, San Diego, including the San Diego Supercomputer Center (SDSC), the California Institute of Telecommunications and Information Technology (Calit2), The Center for Research in Biological Structure (CRBS), The Scripps Research Institute (TSRI), and Washington University in St. Louis.

Nation Center For Research Resources

National Biomedical Computation Resource
University of California, San Diego
9500 Gilman Drive, MC 0505 - La Jolla, CA 92093-0505
Phone: (858) 822-1079 - Fax: (858) 822-5407
Email Us

National Institutes of Health

Talk Topics

- Discuss MYMPI - MPI Python module
  - What?
  - Why?
- Applications
  - Continuity: cardiac mechanics and electrocardiology
  - Montage: astronomical mosaicking

MYMPI Overview

- A Python module written in C
- Does not require a special version of Python
- Contains calls to many important MPI routines
- Enables writing parallel programs based on the MPI standard
- Enables mixing of Python, C, and Fortran MPI programs

Hasn't this been done?

- Yes
  - pyMPI : http://sourceforge.net/projects/pympi
  - A special Python interpreter
- No
  - MYMPI is a module that works with a standard Python interpreter
  - Many significant differences

Interpreter vs. Module

- pyMPI is actually a custom version of the Python interpreter along with a module
- MYMPI, is a module only that can be used with a normal Python interpreter
  - Can use the same interpreter for serial and parallel applications
  - Don't need to maintain a second interpreter

Interpreter vs. Module

- pyMPI the Python interpreter is the parallel application
    - MYMPI the code executed by the interpreter is the parallel application
    - This can be seen in the start-up procedure

Startup

- Every “normal” MPI application must call MPI_Init
- pyMPI programs don’t not call MPI_Init because it is called when the interpreter starts
- MYMPI, programs explicitly call MPI_Init
- Having better control over when/how MPI_Init is called was one motivation for MYMPI

Syntax and Semantics

- MYMPI follows the syntax and semantics of C and Fortran MPI
- pyMPI has a more OOP flavor
- Knew from the beginning we would be using to write heterogeneous Python and Fortran applications
- Can use our existing training materials. Rewrote examples in Python

Size

- pyMPI is about 1.7 Mb and compiles in minutes
- MYMPI is a single file, 33 Kb, and compiles in < 3 seconds
- pyMPI contains most of MPI-1 (120+ routines)
- MYMPI contains about 30 important routines
- Another motivation was this makes extending and debugging easier, memory leak.

Routines

mpi_alltoall
Sends data from all to all processes
mpi_alltoallv
Sends data from all to all processes, with a displacement
mpi_barrier
Blocks until all process have reached this routine.
mpi_bcast
Broadcasts a message from the process with rank "root" to all other processes of the group.
mpi_comm_create
Creates a new communicator
mpi_comm_dup
Duplicates an existing communicator with all its cached information
mpi_comm_group
Accesses the group associated with given communicator
mpi_comm_rank
Determines the rank of the calling process in the communicator
mpi_comm_size
Determines the size of the group associated with a communictor
mpi_comm_split
Creates new communicators based on colors and keys
mpi_error
Checks for errors (not part of regular MPI)
mpi_finalize
Terminates MPI execution environment
mpi_gather
Gathers together values from a group of processes

Routines

mpi_gatherv
    Gathers into specified locations from all processes in a group
mpi_get_count
    Gets the number of "top level" elements
mpi_group_incl
    Produces a group by reordering an existing group and taking only listed members
mpi_group_rank
    Returns the rank of this process in the given group
mpi_init
    Initialize the MPI execution environment
mpi_iprobe
    Nonblocking test for a message
mpi_probe
    Blocking test for a message
mpi_recv
    Basic receive
mpi_reduce
    Reduces values on all processes to a single value
mpi_scatter
    Sends data from one task to all other tasks in a group
mpi_scatterv
    Scatters a buffer in parts to all tasks in a group
mpi_send
    Performs a basic send
mpi_status
    Return status information (not part of standard MPI)

Data types

- MPI_INT
  - Integer
- MPI_DOUBLE
  - 8 byte floating point number
- We have support for scalars and multiple dimension arrays using the Numeric package.
- Need to add character

Example Call

- Input argument lists closely match the list for C and Fortran MPI routines
- In C we would do a message receive into buffer of count integers from process source using the syntax:

```
MPI_Recv(&buffer,count,     MPI_INT, source, tag,  MPI_COMM_WORLD,&status);
```

Python:

```
buffer=mpi.mpi_recv(        count, mpi.MPI_INT, source, tag,  mpi.MPI_COMM_WORLD    )
status=mpi_status()
```

Where it works

- OSX
    - LAM , MPICH
- IBM - AIX
    - Native
- Redhat
    - LAM, MPICH ethernet
- SuSie
    - MPICH ethernet & MPICH Myrinet

Other uses

- Might be used in place of some F2PY calls
    - Subroutine call becomes a Send and Receive to another process
    - Opinion: Easier to write
- Communication between GUI and worker routines
- Facilitate use of parallel libraries such as SuperLU_DIST

Links

- MYMPI: nbcr.sdsc.edu/tools.php
- MPI: www-unix.mcs.anl.gov/mpi/
  - MPICH: www-unix.mcs.anl.gov/mpi/mpich/
  - LAM: www.lam-mpi.org/
- MPI on OSX (historical): www.sdsc.edu/~tkaiser/mac_stuff/mpi_osx.html
- Lecture notes: peloton.sdsc.edu/~tkaiser/mpi_stuff/workshop/index.html

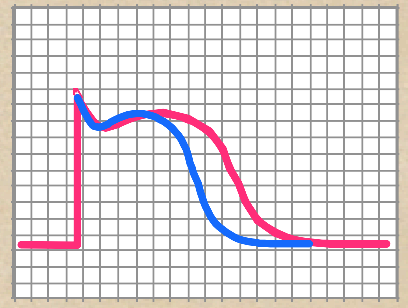
Demo

http://peloton.sdsc.edu/~tkaiser/mympi/Terminal.mov

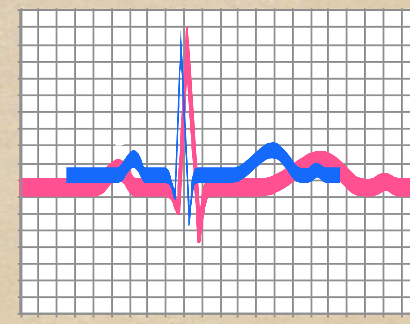
Application: Continuity 6.0

- A problem solving environment for multiscale physiology applications
- Most common applications are cardiac electrophysiogy and biomechanics
- Makes use of Python for:
  - High level object oriented design
  - Scripting and component integration
  - And now, for parallel processing
- Can be downloaded from www.continuity.ucsd.edu

Functional Interactions



Electrophysiology

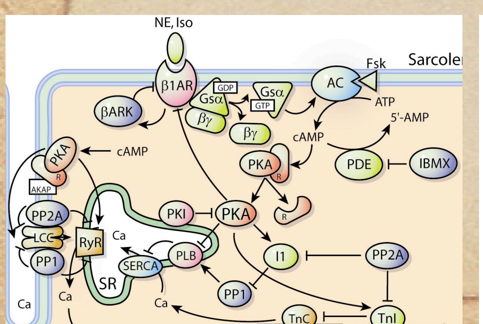


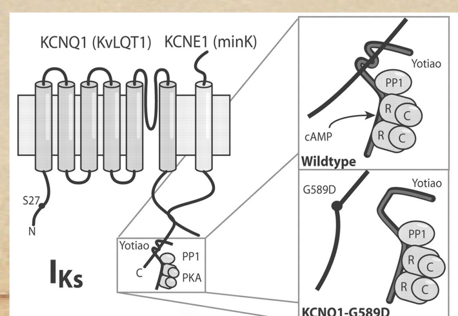
Neurohormonal Control

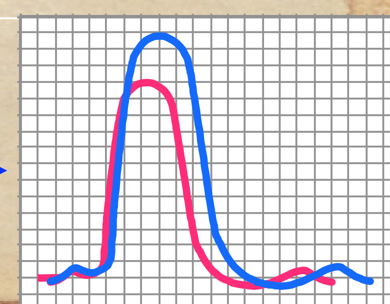
Genetic Mutations

Mechanics

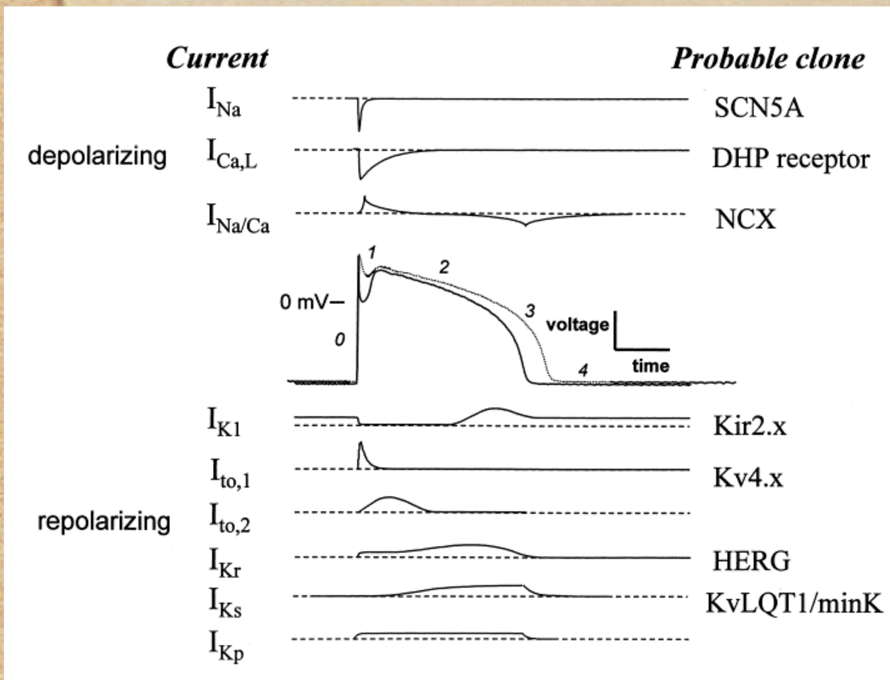
Ca²⁺ Handling

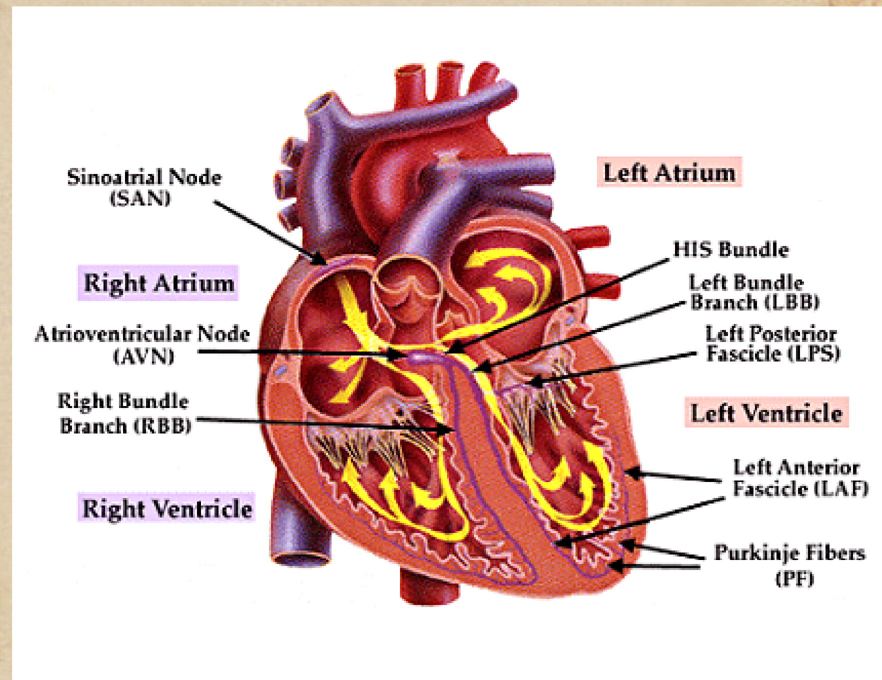






Cardiac Electrophysiology

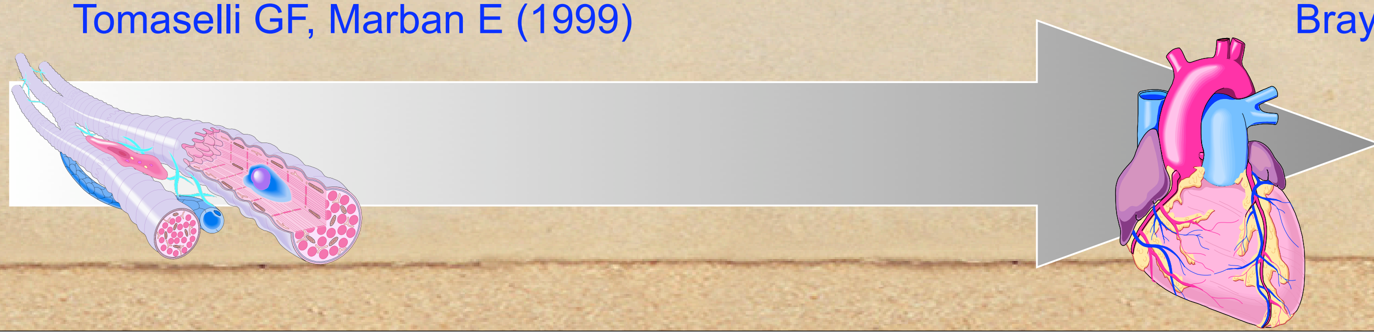




Tomaselli GF, Marban E (1999)

Bray (1994)

Cell



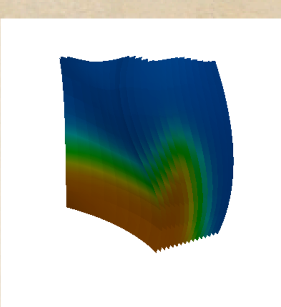
Organ

Some Results

- What we are going to see:
  - A preview from Thinkwell
  - See also: http://butler.cc.tut.fi/~malmivuo/bem/bembook/06/06.htm
- Normal
- Arrhythmia: When things go wrong

Modeling Cardiac Electrophysiology

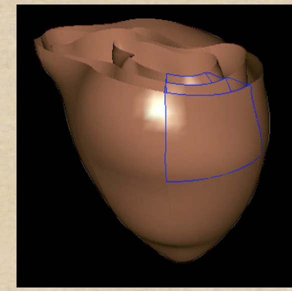
- Solution domain:
  - Wedge of tissue
  - Hopefully soon whole heart!



- Solution Method:
  - Collocation-Galerkin finite elements,
  - Crank-Nicholson time stepping
- Solver packages: RADAU (ODEs), SuperLU (PDEs)

Subcellular → Cellular → **Organ**

Operator Splitting Technique



Time Integration Loop – Crank Nicholson

t = 0

t = Δt/2

t = Δt

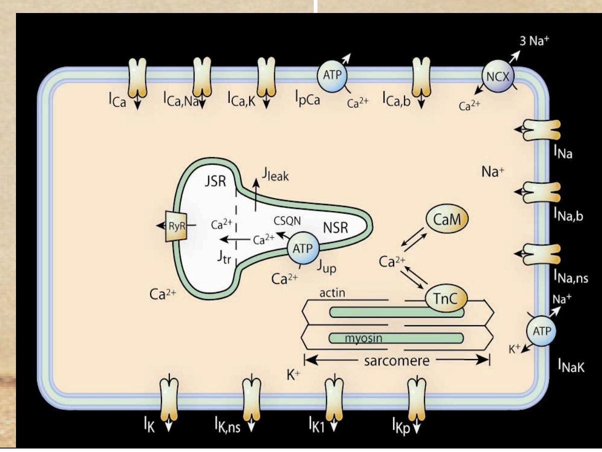
t = 3Δt/2

t = 2Δt

Reaction-diffusion equation (PDE)

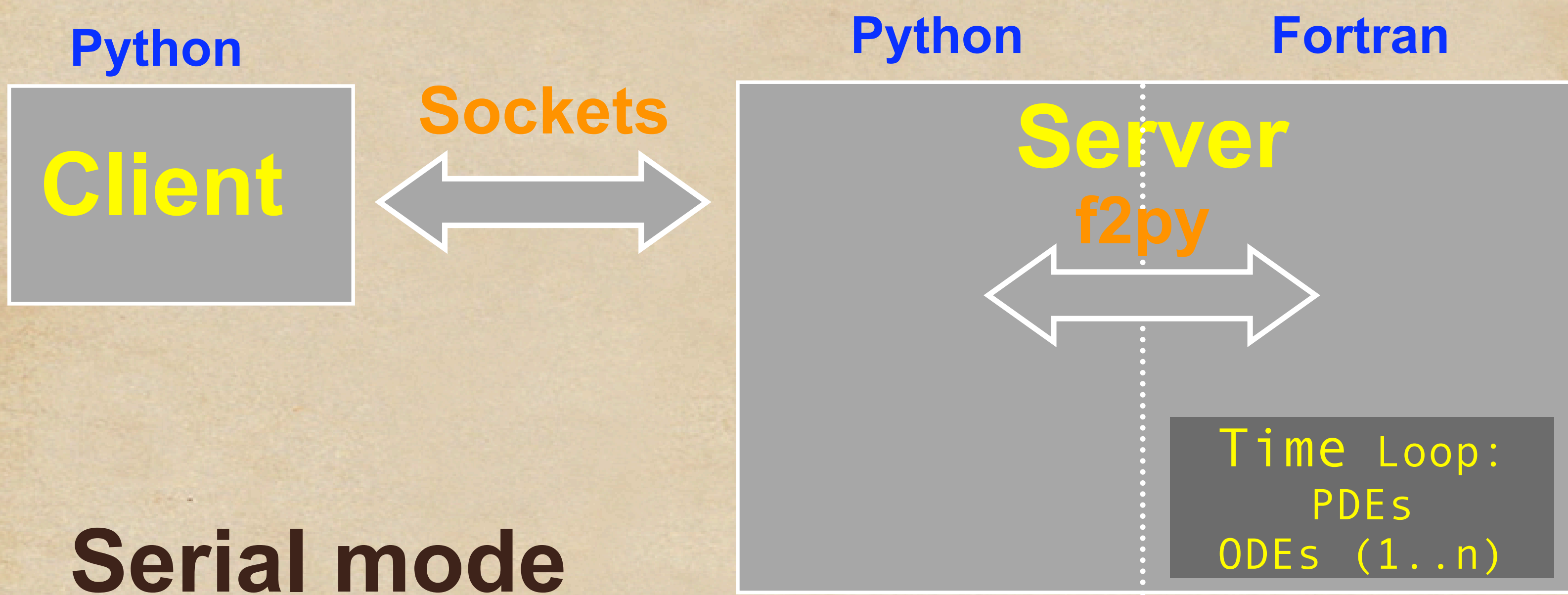
$$\frac{dV_m}{dt} = -\frac{1}{C_m} I_{ion} + \frac{1}{SC_m} \nabla \cdot \mathbf{D} \nabla V_m$$

Ionic model (ODEs)



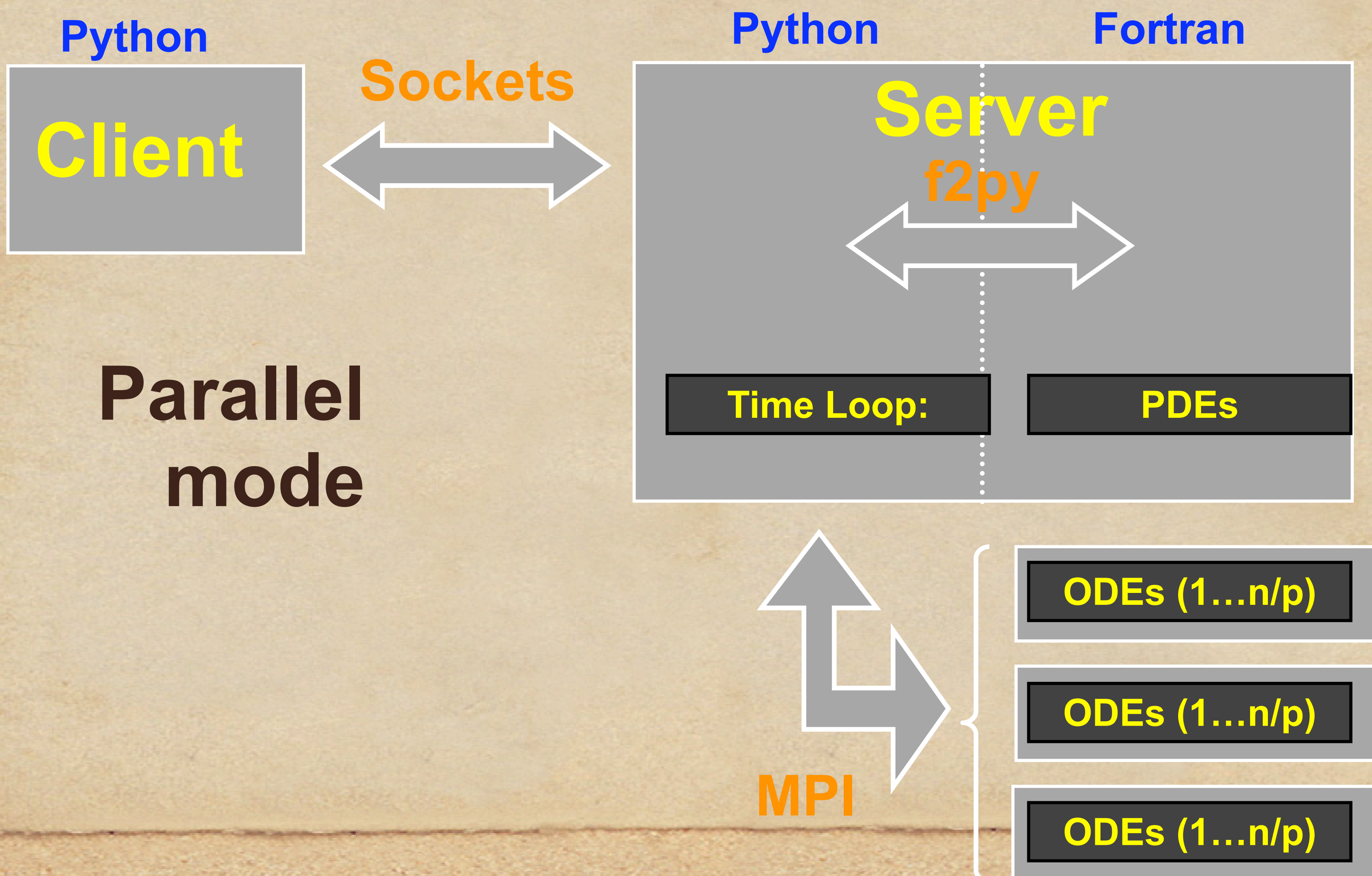
Qu Z, Garfinkel A (1999) *IEEE Trans Biomed Eng 46(9):1166-8*

The Old (Slow) Way

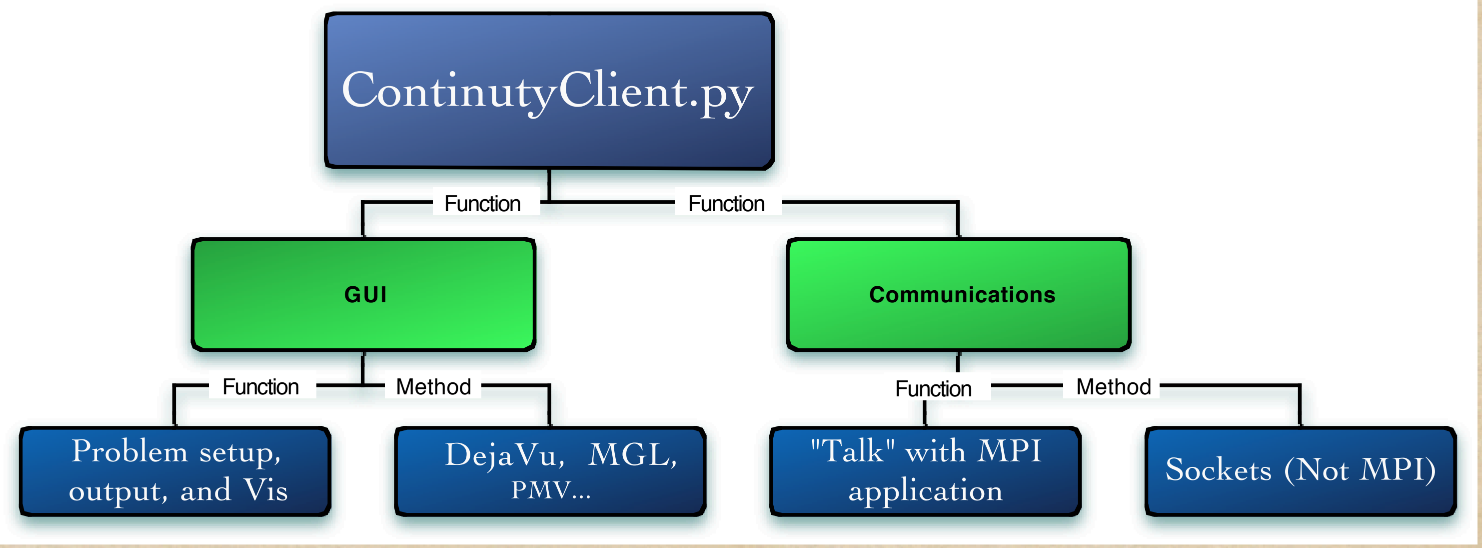


ODEs take 95-99% of
the execution time

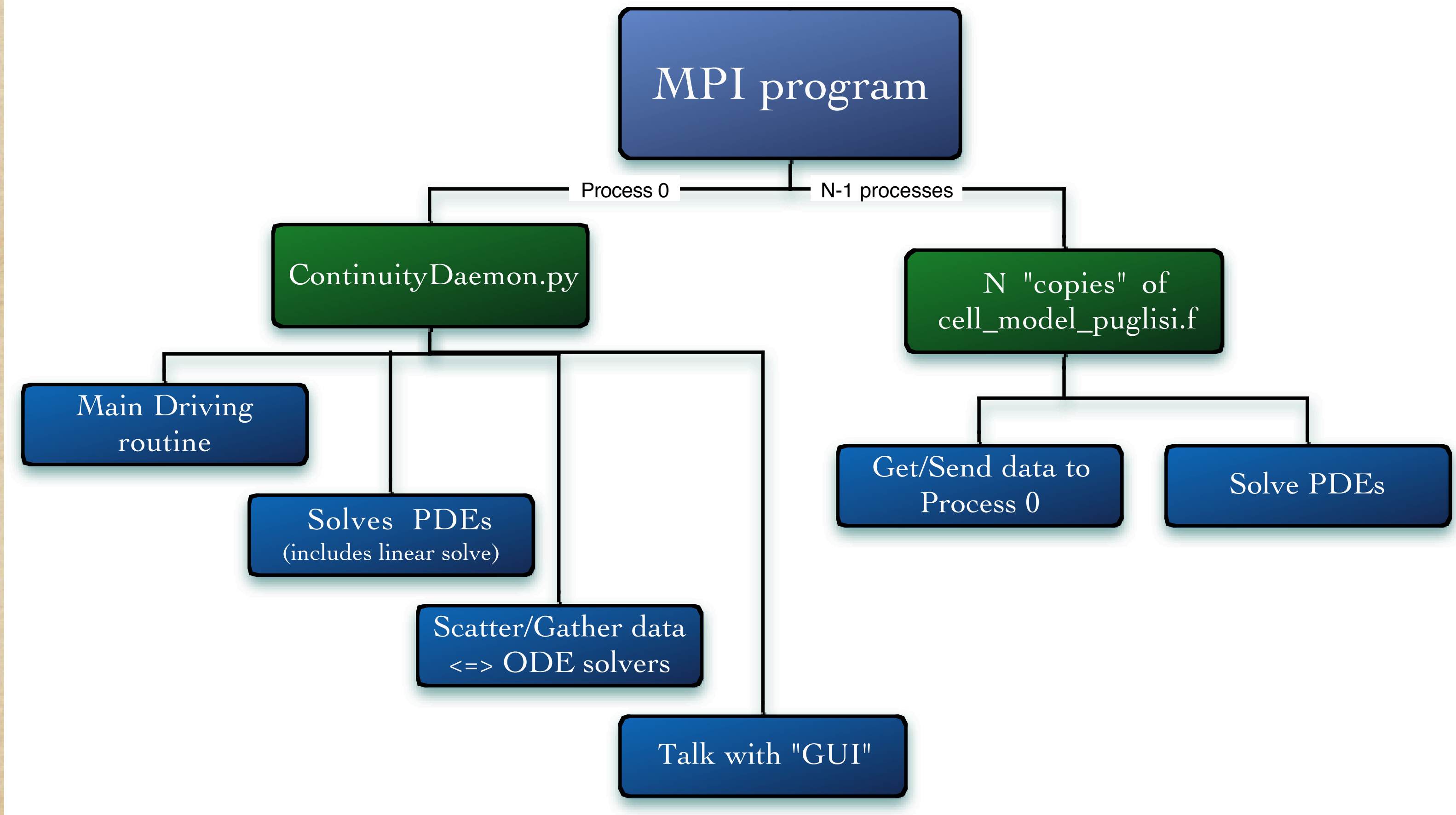
New Faster Way with Parallelization of ODEs



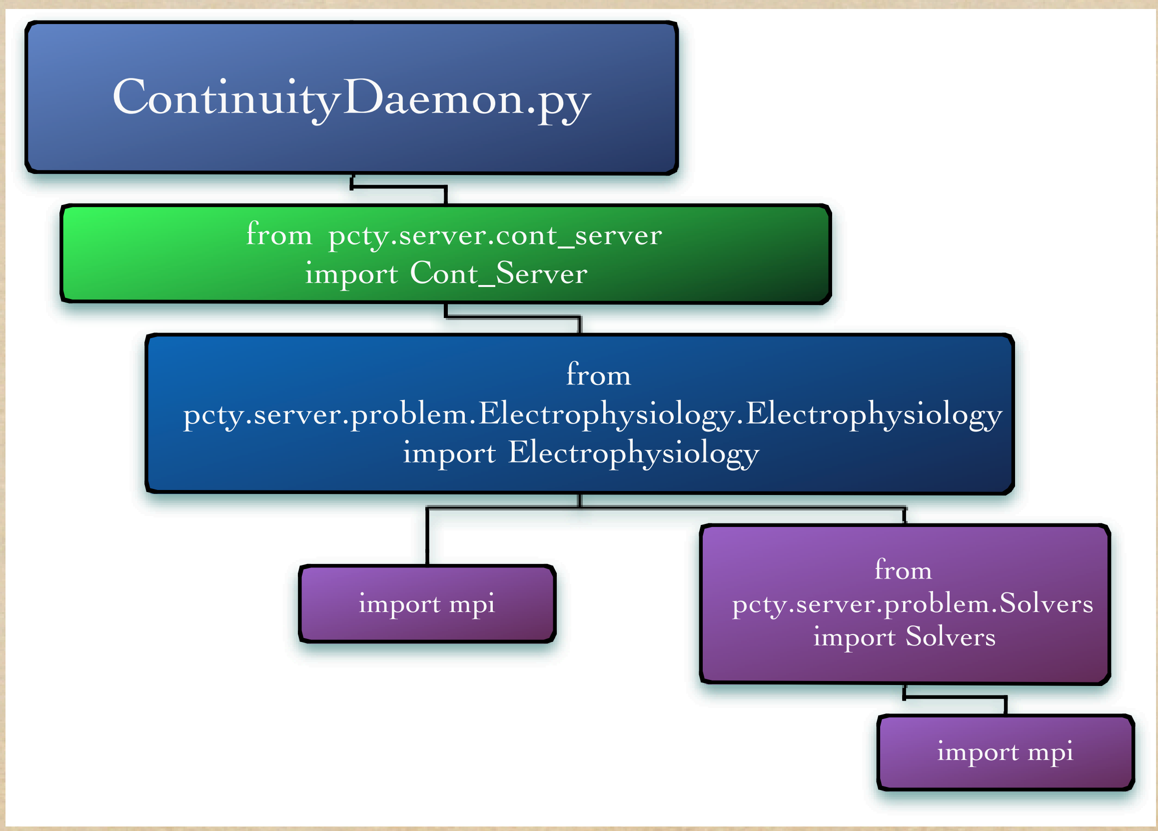
ContinuityClient



Continuity Server



Continuity Server (process 0)



Electrophysiology Setup routine

```
def _cgint_mpi_init (self):
  import mpi
  data = {}
  data['myid'] = mpi.mpi_comm_rank(mpi.MPI_COMM_WORLD)
  data['numprocs'] = mpi.mpi_comm_size(mpi.MPI_COMM_WORLD)
  # Data to broadcast: dt - timestep
  #                    stimon - stimulus time
  #                    stimlen - stimulus length
  data['icount'] = 1
  mpi.mpi_bcast(self.N_gauss_pts,data['icount'],mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
  data['dt'] = self.control.intprm_dtout
  mpi.mpi_bcast(data['dt'],data['icount'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  stimon = self.control.intprm_stimon
  mpi.mpi_bcast(stimon,data['icount'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  stimlen = self.control.intprm_stimlen
  mpi.mpi_bcast(stimlen,data['icount'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)

  # Gather local_gps to here.
  data['local_gps'] = zeros(data['numprocs'],"i")
  data['local_gps'] = mpi.mpi_gather(data['local_gps'][0],1,mpi.MPI_INT,1,mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
  # Data to scatter: iPar - k_rel
  #                  param_set_list - epi/M/endo
  #                  rPar - stim, trigger, t1_rel
  self.times = zeros(self.N_gauss_pts,'f')
  data['disps'] = zeros(data['numprocs'],"i")
  data['odesdisps'] = zeros(data['numprocs'],"i")
  data['newdisps'] = zeros(data['numprocs'],"i")
  data['odeslocal_gps']= zeros(data['numprocs'],"i")
  for i in range(1,len(data['local_gps'])):
    data['disps'][i] = data['disps'][i-1]+ data['local_gps'][i-1]
  result = mpi.mpi_scatterv(self.param_set_list.astype('i'),data['local_gps'],data['disps'],mpi.MPI_INT,0,mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
  result = mpi.mpi_scatterv(self.iPar.astype('i'),data['local_gps'],data['disps'],mpi.MPI_INT,0,mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
  mpi.mpi_bcast(self.num_fields,data['icount'],mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
  self.bcl = 1000.0  #SNH must fix to stim form data
  mpi.mpi_bcast(self.bcl,data['icount'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  result = mpi.mpi_scatterv(self.rPar,data['local_gps'],data['disps'],mpi.MPI_DOUBLE,0,mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  for i in range(data['numprocs']):
    data['newdisps'][i] = data['disps'][i] + self.N_gauss_pts
  result = mpi.mpi_scatterv(self.rPar,data['local_gps'],data['newdisps'],mpi.MPI_DOUBLE,0,mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  for j in range(self.num_fields):
    for i in range(data['numprocs']):
      data['newdisps'][i] = data['newdisps'][i] + self.N_gauss_pts
    result = mpi.mpi_scatterv(self.rPar,data['local_gps'],data['newdisps'],mpi.MPI_DOUBLE,0,mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  for i in range(data['numprocs']):
    data['odeslocal_gps'][i] = data['local_gps'][i] * self.Num_ODEs
    data['odesdisps'][i] = data['disps'][i] *  self.Num_ODEs

  return data
```

Electrophysiology
Time step loop

```
while (finished == 0):
  if (not fmod(round(t_now,2),1)):    # Print time every second
    print 'Tnow = ', t_now
# Part 1: Evaluate ODEs at collocation points via Fortran processes
  start = time()

  odeSolver(odedata, t_now)  #_cgint_mpi_odesolver

  if(t_now >= t_stop):
    finished = 1
  odeSolverFinish(finished, odedata)
  #ODE_time = ODE_time + t_odes


  self._cgint_pack_source(source)

  end = time()
  elapsed_time = end-start
  ODE_time = ODE_time + elapsed_time

# Part 2: Solution of PDEs through an Ax=b linear solve
  start = time()

  self.PDEs_now,elapsed_time = self.EP_module.p_cont2axb(self.global_equa,coefA,coefB,dt, \
                               source,self.PDEs_now,self.control.assemblers.int_assemble_callback,elapsed_time)

  end = time()
  elapsed_time = end - start
  PDE_time = PDE_time + elapsed_time

# Part 3: Evaluate PDE solution at gauss points to update ODEs for next step
  start = time()

  self._cgint_eval_soln()

  end = time()
  elapsed_time = end - start
  Eval_time = Eval_time + elapsed_time

  ren.rmout_callback(self.gbl_vars,self.ODEs_now,t_now, self.mesh.deriv_per_var,self.mesh.depen_var,self.mesh.global_nodes, \
                     self.Num_ODEs*self.mesh.gaus_per_elem*self.mesh.elements)
  t_now = t_now + dt

# end while loop
print 'Total Integration Time = ', ODE_time+ PDE_time +Eval_time
print 'ODE Time = ', ODE_time
print 'PDE_Time = ', PDE_time
print 'Evaluation Time', Eval_time

if (parallel):
  self._cgint_mpi_finalize(odedata)
```

Electrophysiology Other MPI routines called from main loop

```
def _cgint_mpi_odesolver(self, data, t_now):
  # Scatter current ODE solution
  import mpi
  result = mpi.mpi_scatterv(self.ODEs_now,data['odeslocal_gps'],data['odesdisps'],mpi.MPI_DOUBLE,0,mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  # Gather ODE solution and dvdt once fortran processes are done.
  self.ODEs_now = mpi.mpi_gatherv(result,0,mpi.MPI_DOUBLE,data['odeslocal_gps'],data['odesdisps'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)
  self.DVoltDt = mpi.mpi_gatherv(result,0,mpi.MPI_DOUBLE,data['local_gps'],data['disps'],mpi.MPI_DOUBLE,0,mpi.MPI_COMM_WORLD)

def _cgint_mpi_odesolver_finish(self, finished, data):
  import mpi
  mpi.mpi_bcast(finished,data['icount'],mpi.MPI_INT,0,mpi.MPI_COMM_WORLD)
```

cell_model_puglisi.f

Set up

```
 call mpi_bcast(num_gps,icount,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call mpi_bcast(dt,icount,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
 call mpi_bcast(stimon,icount,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
 call mpi_bcast(stimlen,icount,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
! Calculate how many gauss pts this process is getting
! Assume only  1 python process.
 fprocs = numprocs - 1
 fid = myid - 1
 local_num_gps = num_gps/fprocs
 rem = num_gps - (num_gps/fprocs)*fprocs
 if (fid < rem) then
  local_num_gps = local_num_gps + 1
 endif
! Gather local_num_gps to Python process for scattering
 write(*,*) 'Proc = ', myid, ' sent ', local_num_gps !Crashes here???
 call mpi_gather(local_num_gps,1,MPI_INTEGER,local_gps,numprocs,MPI_INTEGER,0,MPI_COMM_WORLD)
! Now receive scattered data relevant to local gauss points:
! param_set_list, rPar,iPar
 allocate(param_set_list(local_num_gps),iPar(local_num_gps))
 call mpi_scatterv(param_set_list,num_gps,num_gps,MPI_INTEGER,
&  param_set_list,local_num_gps,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call mpi_scatterv(iPar,num_gps,num_gps,MPI_INTEGER,iPar,
&    local_num_gps,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call mpi_bcast(num_fields,icount,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 call mpi_bcast(bcl,icount,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
 allocate(rPar((2+num_fields)*local_num_gps))
 call mpi_scatterv(rPar,num_gps,num_gps,MPI_DOUBLE_PRECISION,
& rPar(1),local_num_gps,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
 call mpi_scatterv(rPar,num_gps,num_gps,MPI_DOUBLE_PRECISION,
& rPar(local_num_gps + 1),local_num_gps,MPI_DOUBLE_PRECISION,
& 0,MPI_COMM_WORLD,ierr)
 do j = 1,num_fields
  call mpi_scatterv(rPar,num_gps,num_gps,MPI_DOUBLE_PRECISION,
& rPar((j+1)*local_num_gps+1),local_num_gps,MPI_DOUBLE_PRECISION,
& 0,MPI_COMM_WORLD,ierr)
 enddo
```

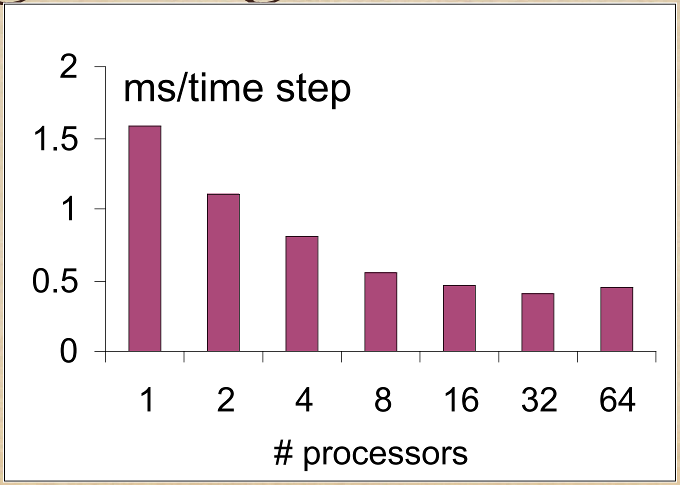
cell_model_puglisi.f

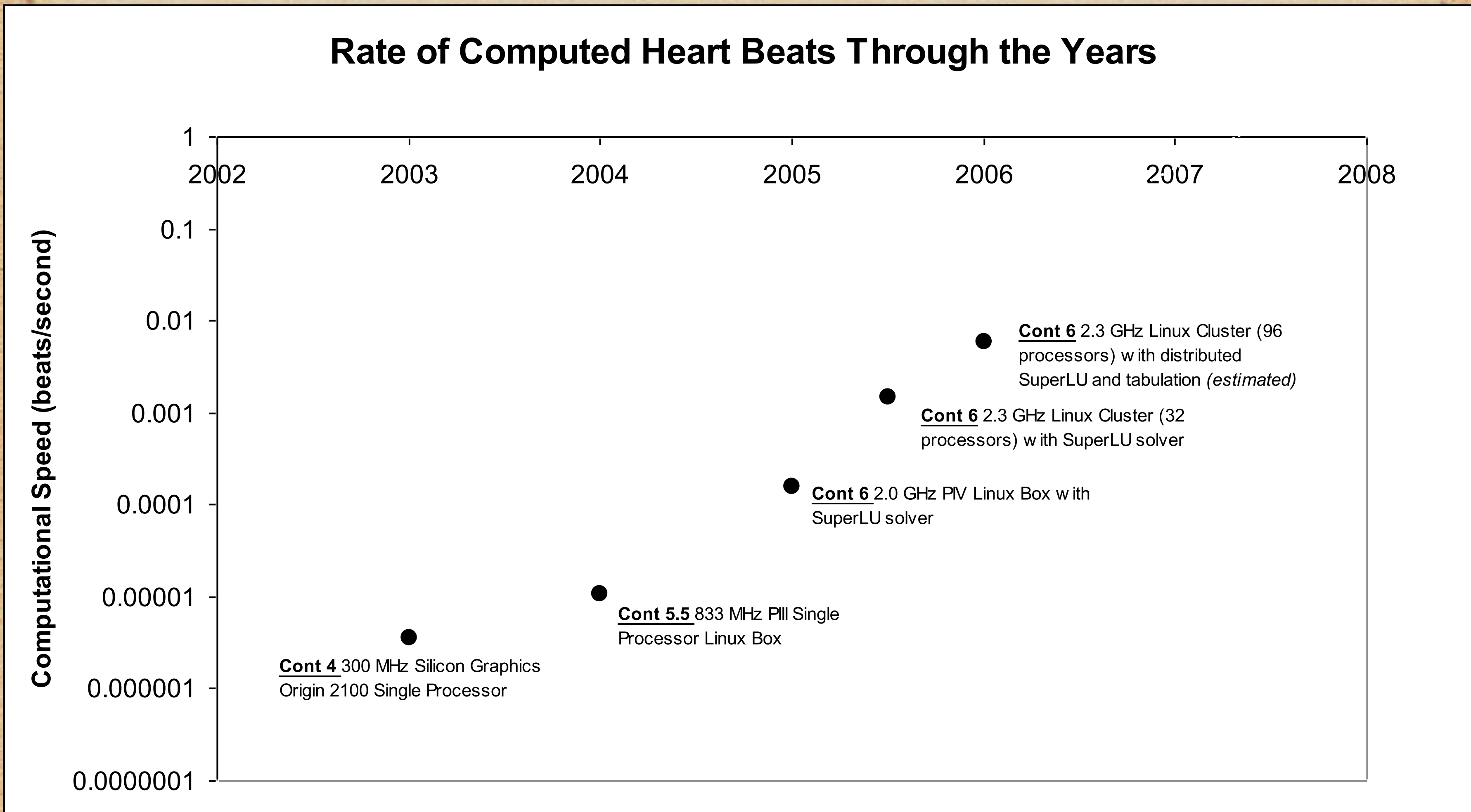
Time step loop

```
do while(finished .eq. 0)
  ! Recv local ode data from python via scatterv
  call mpi_scatterv(local_odes_now,num_gps,num_gps,
 &     MPI_DOUBLE_PRECISION, local_odes_now,local_num_gps*num_odes,
 &     MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
  ! Solve cell model
  !t1 = etime(ta)
  !write(*,*) 'odes = ', local_ODEs_now(1)
  !write(*,*) 'before cm2odesolver'
  !write(*,*) 'before after= ', local_ODEs_now(1)
  call cm2odesolver(local_ODEs_now,t_now,t_end,local_num_gps,
 & param_set_list,rPar,iPar,local_dvdt,stimon,stimlen,num_fields,
 & bcl,dads,eads,times)
  ! Send current ode and dvdt data back to python via gatherv
  !write(*,*) 'odes after= ', local_ODEs_now(1)
  call mpi_gatherv(local_odes_now,local_num_gps*num_odes,
 &     MPI_DOUBLE_PRECISION,local_odes_now,numprocs,numprocs,
 &     MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
  call mpi_gatherv(local_dvdt,local_num_gps,
 &      MPI_DOUBLE_PRECISION,local_dvdt,numprocs,numprocs,
 &     MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
  call mpi_bcast(finished,icount,MPI_INTEGER,0,MPI_COMM_WORLD,
 &   ierr)
  !write(*,*) 'after bcast'
  !t2 = etime(ta)
  !todes = t2-t1
  !write(*,*) 'after reduce'
  t_now = t_end
  t_end = t_end + dt
enddo
```

Parallel Performance

- Communication Limited
- Ethernet not myrinet
- MYMPI is slower than normal MPI but allows a heterogeneous programming environment




Rate of Computed Heart Beats Through the Years
Computational Speed (beats/second)
1
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
2002
2003
2004
2005
2006
2007
2008
Cont 6 2.3 GHz Linux Cluster (96 processors) w ith distributed SuperLU and tabulation (estimated)
Cont 6 2.3 GHz Linux Cluster (32 processors) w ith SuperLU solver
Cont 6 2.0 GHz PIV Linux Box w ith SuperLU solver
Cont 5.5 833 MHz PIII Single Processor Linux Box
Cont 4 300 MHz Silicon Graphics Origin 2100 Single Processor

Future Work

- Extension to whole heart anatomical model -- parallelization of the PDEs
- Parallel SuperLU
- Electro-mechanical model

Montage Mosaicking

Leesa Brieger

International Virtual Observatory Alliance

Aus-VO

AVO
ASTROPHYSICAL VIRTUAL OBSERVATORY

China-VO

Astro Grid

vo-i

RVO

CVO

NVO
NATIONAL VIRTUAL OBSERVATORY

IVOA

SVO
Spanish Virtual Observatory

DRACO

JVO
JAPANESE VIRTUAL OBSERVATORY

France

HVO
Hungarian Virtual Observatory

GERMAN ASTROPHYSICAL
GAVO
VIRTUAL OBSERVATORY

URL's

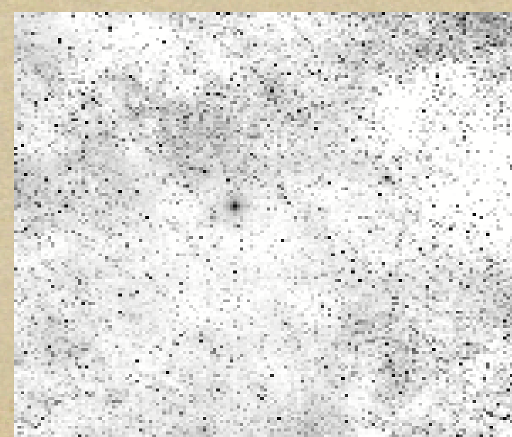
- Montage:
  - http://montage.ipac.caltech.edu/
- Montage gallery:
  - http://montage.ipac.caltech.edu/gallery.html
- NVO:
  - http://www.us-vo.org/
- IVOA:
  - http://www.ivoa.net/

Montage Mosaicking

2MASS mosaic near galactic center

Without background corrections:

With background corrections:



Montage/NVO

- Build an atlas of surveys
  - standardization
  - comparison and composition of information

- Provide on-demand services
  - science-grade images (cut-outs)
  - user-requested mosaics
  - large-scale mosaicking services

Montage: Caltech/SDSC Collaboration

- Mosaic the entire 2MASS archive
- Reformat the archive
- Create 2MASS addition to the Hyperatlas that is being constructed
- 1734 plates (pages) in the atlas, 3 bands in 2MASS: 5202 6-deg mosaics to produce

Montage: Caltech/SDSC Collaboration

- Create the infrastructure to do large-scale mosaics
- data management –

  Storage Resource Broker (SRB)
- workflow management -

  MYMPI
- computations take place on TeraGrid Itanium2 and IBM Power4 systems

Montage Data

- Atlas mosaics are 6-degree squares
- Input for one mosaic: 1300-4000 raw input files, each of 2MB
- Output for one mosaic: a single 4GB FITS file or subdivided tiles of that, eg. 144 tiles each of 27 MB
- Input archive: around 8 TB,
- Output archive: around 7 TB

Montage Data Flow

- Input data (2MASS archive) stored locally in a parallel, high-performance filesystem (gpfs)
- 2MASS archive replicated locally in SRB and at Caltech on spinning disk, served up from the IRSA server
- Output data archived locally in SRB, accessible from everywhere there is an SRB client

Montage Workflow

- Stand-alone executables compose a mosaic
- Independent steps on input data – currently task-oriented, could be data-oriented
- some serial steps, some parallel (MPI) steps
- Python MPI, Perl, shell scripts

Montage Workflow

Organize many mosaics into a single multi-CPU, MPI job:

- the serial steps done for all mosaics simultaneously, one on each CPU
- the parallel steps done for one mosaic at a time, using all the CPUs of the job

Python Workflow Script

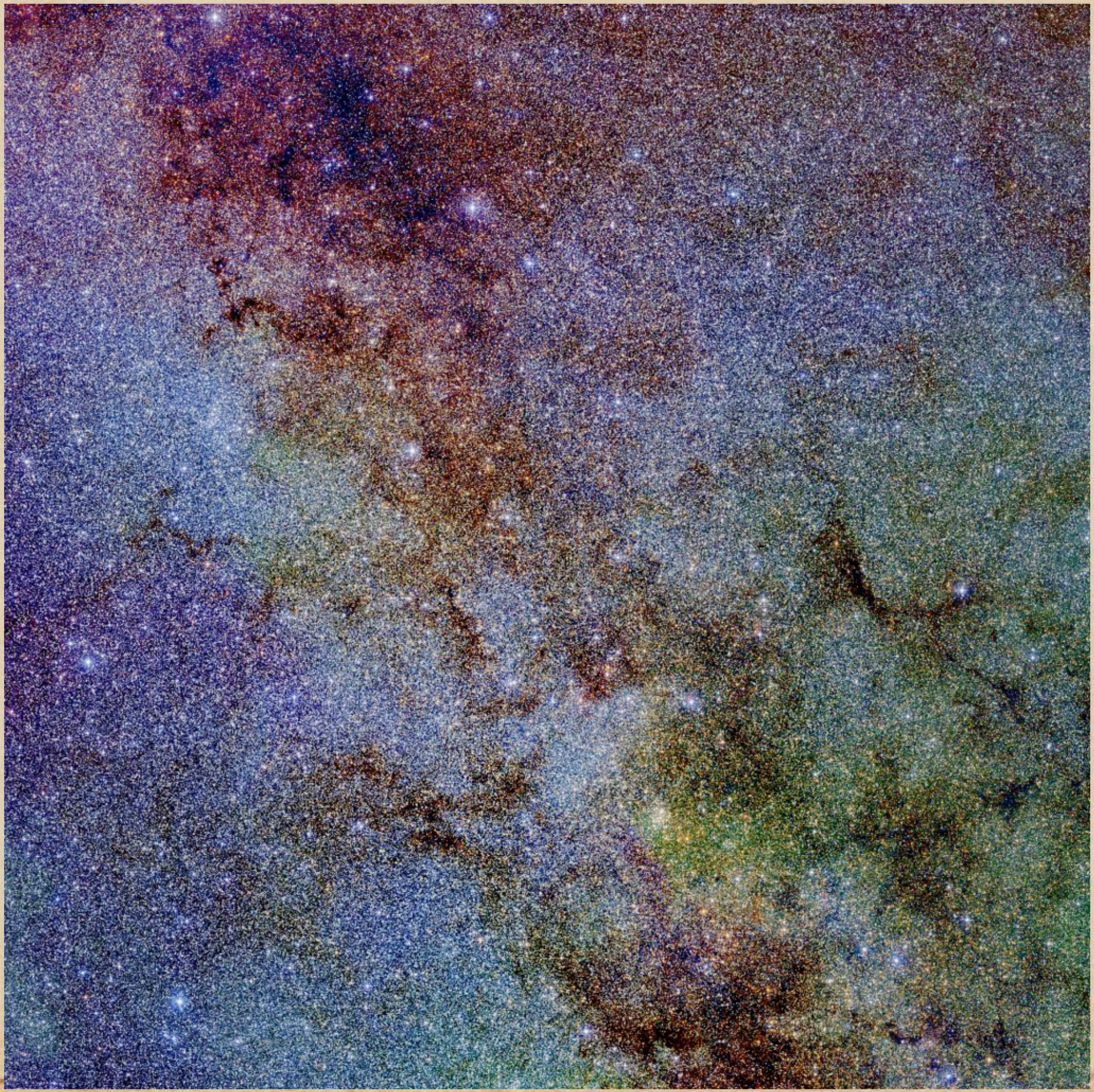
```
#!/usr/local/apps/python/Python2.4.1/bin/python
# Script to run the galactic plane example.
import os
import sys
import string
import subprocess
import Numeric
from Numeric import *
import mpi
import posix
bandlist = ['h','j','k']
cmd=""
for arg in sys.argv:
   cmd=cmd+ arg+" "
print "python command line arguments: " ,cmd
myid_numprocs =  mpi.mpi_start(len(sys.argv),cmd)
myid=mpi.mpi_comm_rank(mpi.MPI_COMM_WORLD)
numprocs=mpi.mpi_comm_size(mpi.MPI_COMM_WORLD)
```

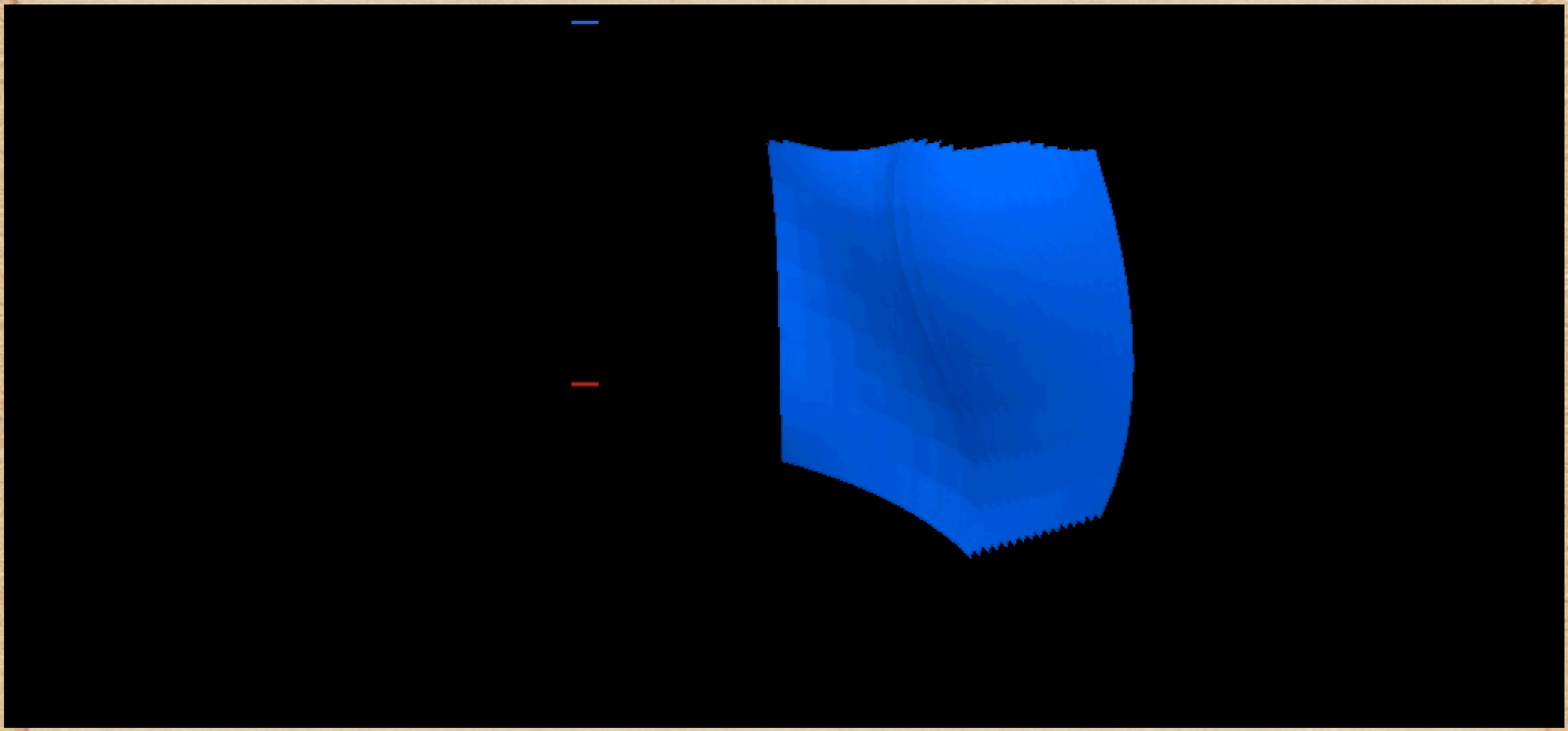
```
# List of plates for the production run
platefile = open('production-pages', 'r')
plates = platefile.readlines()
numplates = len(plates)
platefile.close()

parent_dir = os.getcwd()
#print "top dir: ", parent_dir

myindex = myid

while myindex<numplates:
 myplates = plates[myindex].split()
 myplate = myplates[0]
 plate_dir = "plate."+myplate

 try:
  os.mkdir(plate_dir)
 except OSError:
  pass
```

```
for band in bandlist:

  os.chdir(parent_dir)
  os.chdir(plate_dir)
  try:
   os.mkdir(band+"-tiles")
  except OSError:
   pass
  os.chdir(band+"-tiles")

# Image table for raw data
  res = os.system('mImgtbl -r raw images.tbl')
  if not (res == 0):
   print "Error: Problem with images.tbl for plate "+myplate+", band "+band+"\n"
  else:
   print "OK: images.tbl for plate "+myplate+", band "+band+" \n“

# Make header
  res = os.system('mMakePlateHdr images.tbl plate.hdr')
  if not (res == 0):
   print "Error: Problem with mMakePlateHdr for plate "+myplate+", band "+band+"\n
  else:
   print "OK: mMakePlateHdr for plate "+myplate+", band "+band+"\n"
```

```
# Synchronize before the parallel step
   mpi.mpi_barrier(mpi.MPI_COMM_WORLD)
# Parallel Projections - one at a time
   for i in range(0,numprocs):
     if( myid == i):
       print "Starting projections for plate", myplate," band ",band
       res = os.system("time mpirun -machinefile prod_pbs_nodefile -np "+str
       (numprocs)+"  /usr/local/apps/Montage_v3.0_beta23/bin/mProjPara -p raw -f
       images.tbl plate.hdr projdir stats.tbl")
       if  not (res == 0):
         print "Error: Problem with projections for plate "+myplate+"\n"
       else:
         print "OK: Projections done for plate ", myplate," band ",band
# Synchronize processes as they go through this loop
      mpi.mpi_barrier(mpi.MPI_COMM_WORLD)

# Image table for reprojected data
   res = os.system('mImgtbl -r projdir proj.tbl')
   . . .
# errorfile.close()
mpi.mpi_finalize()
```

Montage: 3 infrared bands => a color image





Back

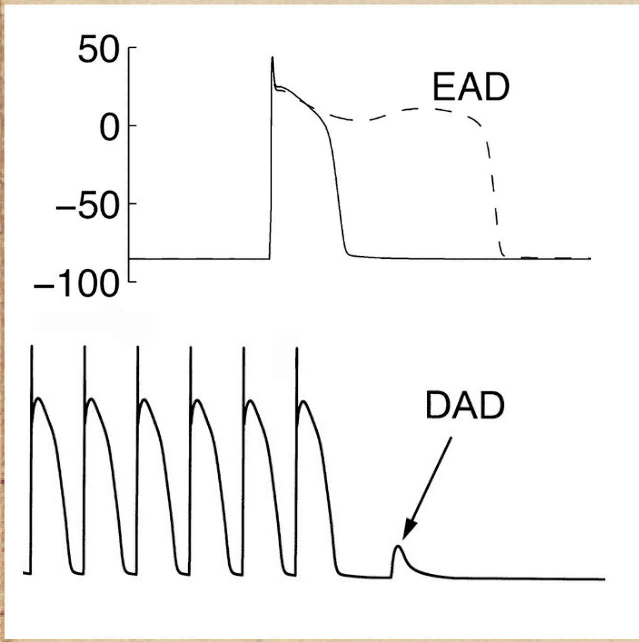
Back

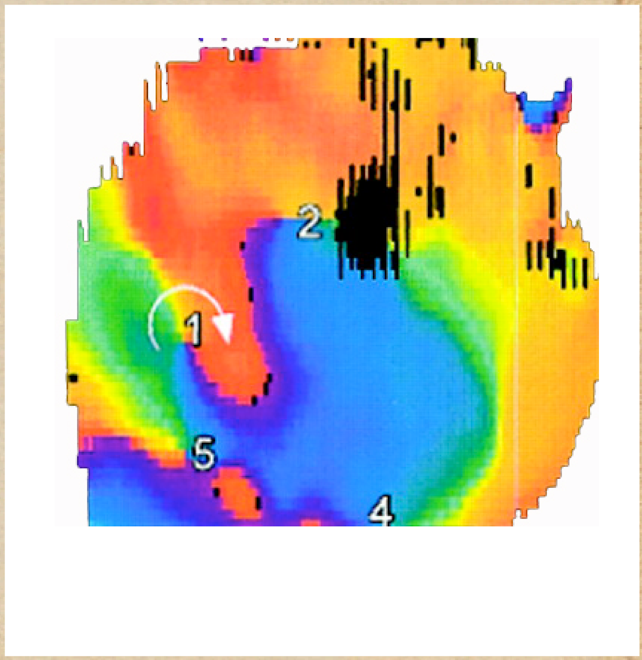
Ventricular Arrhythmia

Triggered Activity

Reentry

↑ Dispersion of repolarization





Saucerman (2004)
Beuckelmann (1998)

Jalife (2000)