COMPUTATIONAL ASTEROSEISMOLOGY

by

TRAVIS SCOTT METCALFE, B.S., M.A.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2001

Appendix C

Computer Codes

This appendix contains an archive of the source code for the software that I have used for the calculations presented in this dissertation.

EVOLVE.F is a streamlined version of the White Dwarf Evolution Code (WDEC) described in §4.3.2 with references to its origins and to the sources of data for the input physics. WDEC takes as input a hot starter model with a specific mass, which can come from detailed evolutionary calculations in the case of DOV stars, or from a simple polytropic approximation in the case of DBV and DAV stars. Using this starter model and other parameters specified in the header, WDEC adds an envelope with the specified composition and fractional mass and evolves the model quasi-statically until it reaches the specified temperature.

PULSATE.F uses the final output model produced by WDEC and calculates the $m = 0$ adiabatic non-radial oscillation periods of a specified spherical degree (ℓ) within a specified period range. The periods resulting from the adiabatic approximation typically differ from the non-adiabatic results by only a few thousandths of a second, which is well below the present level of observational noise.

PVM_FITNESS.F is the code that uses the message-passing routines of the Parallel Virtual Machine (PVM) software to allow the public-domain genetic algorithm PIKAIA to evaluate the fitnesses of trials in parallel rather than sequentially. This code automatically determines the number of processors available for the calculation, balances the load when machines with differing speeds are used, and works around crashed jobs in a sensible way.

FF_SLAVE.F is an interface between the parallel genetic algorithm and the streamlined version of WDEC. This code runs on each machine that is used to calculate white dwarf models. It uses the message-passing routines of PVM to receive sets of parameters from the master process, evaluates the white dwarf model specified by those parameters, compares the model periods to the observations, and returns a measure of fitness to PIKAIA.

The practical aspects of running the evolution and pulsation codes are addressed in the documentation archive at the end of this appendix.

C.1 EVOLVE.F

In digital dissertation: Hypertext version of evolution code.

C.2 PULSATE.F

In digital dissertation: Hypertext version of pulsation code.

C.3 PVM_FITNESS.F

```
      subroutine pvm_fitness (slave, num_jobs, npar, oldph, fitness)
c ---------------------------------------------
c     parallel fitness evaluation using PVM
c ---------------------------------------------
      implicit none
c
      include '../include/fpvm3.h'
c
      integer job, info, nhost, msgtype, iwhich, i
      integer mytid, dtid, tids(0:128), flag, ntask
      integer ttids(64), ptids(64), htids(64), flags(64)
      integer speed, narch, numt, npar, nspawn, last, wait
      integer num_jobs, ndone, length, par, trial, listen
      integer finished(1024),resubmitted(1024)
c
      double precision result, data(64)
      real fitness(1024), oldph(64,1024)
c
      character*40 hostname
      character*18 host
      character*8 slave, arch
      character*8 aout(64)

c ---------------------------------------------
c     initialize book-keeping variables
c ---------------------------------------------
      listen = 0
      wait = 0
      ndone = 0
      do job=1,num_jobs
         finished(job) = 0
         resubmitted(job) = 0
      enddo
```

```
c --------------------------------------------
c     enroll this program in PVM
c --------------------------------------------
      call pvmfmytid( mytid )
      call pvmfconfig( nhost, narch, dtid, host, arch, speed, info )
c --------------------------------------------
c     run jobs on slave nodes only
c --------------------------------------------
      arch = '.'
      flag = PvmTaskHost+PvmHostCompl
      nspawn = nhost-1
      call pvmfspawn( slave, flag, arch, nspawn, tids, numt )
c --------------------------------------------
c     check for problems spawning slaves
c --------------------------------------------
      if( numt .lt. nspawn ) then
         write(*,*) 'trouble spawning ',slave
         write(*,*) ' Check tids for error code'
         call shutdown( numt, tids )
      endif
c
      write(*,*)
c --------------------------------------------
c     send an initial job to each node
c --------------------------------------------
      do job=0,nspawn-1
c
         trial = job + 1
         do par=1,npar
            data(par) = INT((100*oldph(par,trial))+0.5)/100.
         enddo
c
         call pvmfinitsend( PVMDEFAULT, info )
         call pvmfpack( INTEGER4, trial, 1, 1, info )
         call pvmfpack( INTEGER4, npar, 1, 1, info )
```

```
        call pvmfpack( REAL8, data, npar, 1, info )
        msgtype  = 1
        call pvmfsend( tids(job), msgtype, info )
c
 11     format("job ",i3,3(2x,f4.2))
        write(*,11) trial,data(1),data(2),data(3)
c
      enddo
c
      write(*,*)
c
      do job=1,num_jobs
c ---------------------------------------------
c     listen for responses
c ---------------------------------------------
 25     msgtype  = 2
        call pvmfnrecv( -1, msgtype, info )
        listen = listen + 1
c
        if (info .GT. 0) then
           write(*,*) "<-- job ",job
           listen = 0
           wait = 0
c ---------------------------------------------
c     get data from responding node
c ---------------------------------------------
           call pvmfunpack( INTEGER4, trial, 1, 1, info )
           call pvmfunpack( REAL8, result, 1, 1, info )
           call pvmfunpack( INTEGER4, length, 1, 1, info )
           call pvmfunpack( STRING, hostname, length, 1, info )
c ---------------------------------------------
c     re-send jobs that return crash signal
c ---------------------------------------------
           if ((result .eq. 0.0).and.(resubmitted(trial).ne.1)) then
              write(*,*) "detected fitness=0 job: trial ",trial
```

```
               call sendjob
     &         (trial,hostname,'ffrslave',npar,resubmitted,oldph)
               goto 25
            endif
c
            fitness(trial) = result
            finished(trial) = 1
            ndone = ndone + 1
c
 33         format(i4,2x,i4,2x,a8,2x,3(f4.2,2x),f12.8)
            write(*,33) ndone,trial,hostname,oldph(1,trial),
     &                  oldph(2,trial),oldph(3,trial),result
c ---------------------------------------------
c     send new job to responding node
c ---------------------------------------------
 140        if (ndone .LE. (num_jobs-nspawn)) then
               trial = job + nspawn
               call sendjob
     &         (trial,hostname,slave,npar,resubmitted,oldph)
            endif
            goto 100
         endif
c ---------------------------------------------
c     re-submit crashed jobs to free nodes
c ---------------------------------------------
         if (ndone .GT.(num_jobs-nspawn)) then
            last = ndone-nspawn
            if (ndone .GE.(num_jobs-5)) last=ndone
            do trial=1,last
               if ((finished(trial).NE.1).AND.
     &         (resubmitted(trial).NE.1).AND.(wait.NE.1)) then
                  write(*,*) "detected crashed job: trial ",trial
                  call sendjob
     &            (trial,hostname,'ffrslave',npar,resubmitted,oldph)
                  wait = 1
```

```
                 goto 25
              endif
           enddo
        endif
c ---------------------------------------------
c     return to listen again or move on
c ---------------------------------------------
        if ((info .EQ. 0).AND.(listen .LT. 10000000)) goto 25
c
        write(*,*) "detected unstable jobs: setting fitness=0"
        do trial=1,num_jobs
           if ((finished(trial) .NE. 1).AND.
     &          (resubmitted(trial) .EQ. 1)) then
              fitness(trial) = 0.0
              finished(trial) = 1
              ndone = ndone + 1
              write(*,33) ndone,trial,hostname,oldph(1,trial),
     &          oldph(2,trial),oldph(3,trial),fitness(trial)
           endif
        enddo
        goto 199
 100    continue
     enddo
c ---------------------------------------------
c     kill any remaining jobs
c ---------------------------------------------
 199 iwhich = PVMDEFAULT
     call pvmftasks( iwhich, ntask, ttids(1), ptids(1),
     &               htids(1), flags(1), aout(1), info )
     do i=2,ntask
        call pvmftasks( iwhich, ntask, ttids(i), ptids(i),
     &                  htids(i), flags(i), aout(i), info )
        if ((aout(i) .EQ. 'ff_slave').OR.
     &      (aout(i) .EQ. 'ffrslave')) then
           call pvmfkill (ttids(i), info)
```

```
         endif
      enddo
c
      call pvmfexit(info)
c
      return
      end
c*****************************************************************
      subroutine sendjob(trial,hostname,slave,npar,resubmitted,oldph)
c
      implicit none
c
      include '../include/fpvm3.h'
c
      integer tids(0:128), numt, msgtype, par, npar, trial, info, flag
      integer resubmitted(1024)
c
      double precision data(64)
      real oldph(64,1024)
c
      character*40 hostname
      character*8 slave
c
      call pvmfspawn( slave, 1, hostname, 1, tids, numt )
c
      if ( numt .lt. 1 ) then
         write(*,*) 'trouble spawning',slave
         write(*,*) ' Check tids for error code'
         call shutdown( numt, tids )
      endif
c
      do par=1,npar
         data(par) = INT((100*oldph(par,trial))+0.5)/100.
      enddo
c
```

```
      call pvmfinitsend( PVMDEFAULT, info )
      call pvmfpack( INTEGER4, trial, 1, 1, info )
      call pvmfpack( INTEGER4, npar, 1, 1, info )
      call pvmfpack( REAL8, data, npar, 1, info )
      msgtype  = 1
      call pvmfsend( tids(0), msgtype, info )
c
 55   format("job --> ",a8,3(2x,f4.2))
      write(*,55) hostname,data(1),data(2),data(3)
c
      if (slave .EQ. 'ffrslave') resubmitted(trial) = 1
c
      return
      end
c***********************************************************************
      subroutine shutdown( nproc, tids )
c
      implicit none
c
      integer nproc, i, info, tids(*)
c
      do i=0, nproc
         call pvmfkill( tids(i), info )
      enddo
c
      call pvmfexit( info )
c
      return
      end
c***********************************************************************
```

C.4 FF_SLAVE.F

```
      program ff_slave
c -----------------------------------------------
c fitness function slave program
c -----------------------------------------------
      implicit none
c
      include '../include/fpvm3.h'
c
      integer info, mytid, mtid, msgtype, speed, length, i
      integer n, nhost, narch, dtid, hostid, trial
c
      double precision ff, data(32), result
c
      character*40 hostname,machine,arch
c -----------------------------------------------
c enroll this program in PVM
c -----------------------------------------------
      call pvmfmytid( mytid )
c -----------------------------------------------
c get the master's task id
c -----------------------------------------------
      call pvmfparent( mtid )
c -----------------------------------------------
c receive data from master host
c -----------------------------------------------
      msgtype = 1
      call pvmfrecv( mtid, msgtype, info )
      call pvmfunpack( INTEGER4, trial, 1, 1, info )
      call pvmfunpack( INTEGER4, n, 1, 1, info )
      call pvmfunpack( REAL8, data, n, 1, info )
c -----------------------------------------------
c perform calculations with data
c -----------------------------------------------
```

```
      result = ff( n, data )
c --------------------------------------------
c send result to master host
c --------------------------------------------
      call pvmftidtohost( mytid, hostid )
 100  call pvmfconfig( nhost, narch, dtid, hostname, arch, speed, info )
      if (dtid .ne. hostid) goto 100
      length = len(hostname)
      machine = hostname(1:length)
c
      call pvmfinitsend( PVMDEFAULT, info )
      call pvmfpack( INTEGER4, trial, 1, 1, info )
      call pvmfpack( REAL8, result, 1, 1, info )
      call pvmfpack( INTEGER4, length, 1, 1, info )
      call pvmfpack( STRING, machine, length, 1, info )
      msgtype  = 2
      call pvmfsend( mtid, msgtype, info )
c --------------------------------------------
c leave PVM before exiting
c --------------------------------------------
      call pvmfexit(info)
c
      stop
      end
c*******************************************************************
```

C.5 Documentation

In digital dissertation: Documentation archive.

- evolution code: (PS/PDF)
- prep code: (PS/PDF)
- pulsation code: (PS/PDF)