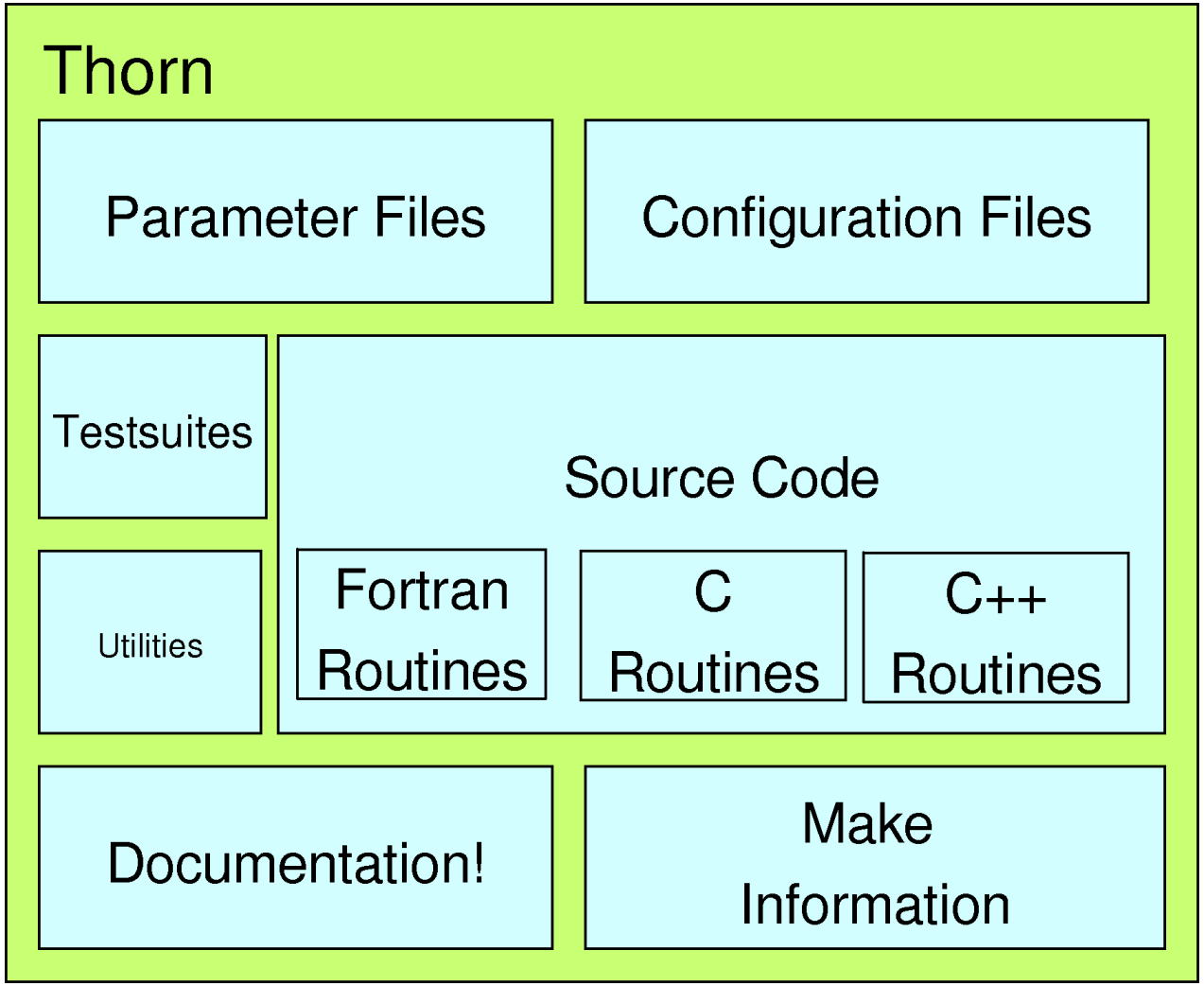
Writing Cactus Thorns

Plan:

- Thorn structure
- HelloWorld thorn
- Conway's Game of Life
  - standalone code
  - the `Life thorn`
- Solving the Wave Equation
  - standalone code
  - the BadWave thorn
  - advanced thorns: PUGH, MoL, AMR

Thorn Structure

Inside view of a plug-in module, or thorn for Cactus



Thorn Specification

Thorn configuration files:

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when

Thorn Specification

Thorn configuration files:

- interface.ccl declares:
  - an 'implementation' name
  - inheritance relationships between thorns
  - Thorn variables
  - Global functions, both provided and used
- schedule.ccl declares:
  - When the flesh should schedule which functions
  - When which variables should be allocated/freed
  - Which variables should be syncronized when
- param.ccl declares:
  - Runtime parameters for the thorn
  - Use/extension of parameters of other thorns

Writing a Hello World thorn

We now demonstrate the process of writing a thorn using a simple Hello World example.

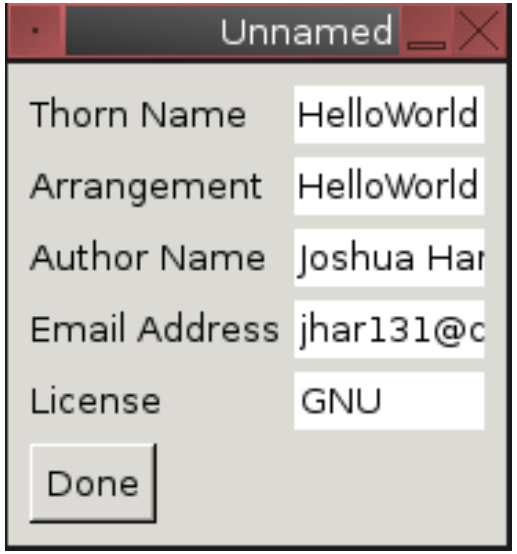
Here is the standalone C code for a Hello World program:

```
#include <stdio.h>
int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

Creating a New Thorn

To Create A New Thorn:

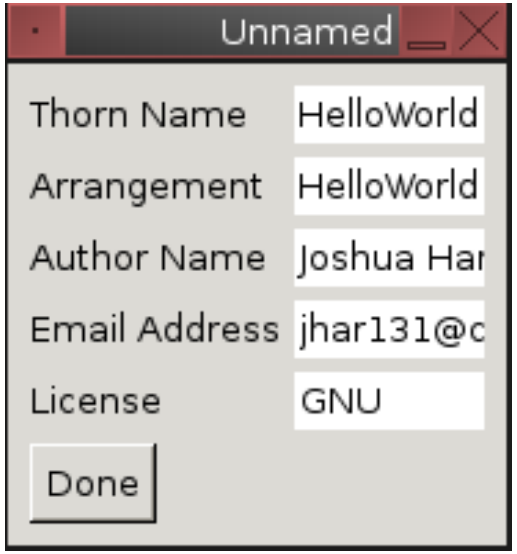
- Select Mojave – NewThorn from the drop down tab



Creating a New Thorn

To Create A New Thorn:

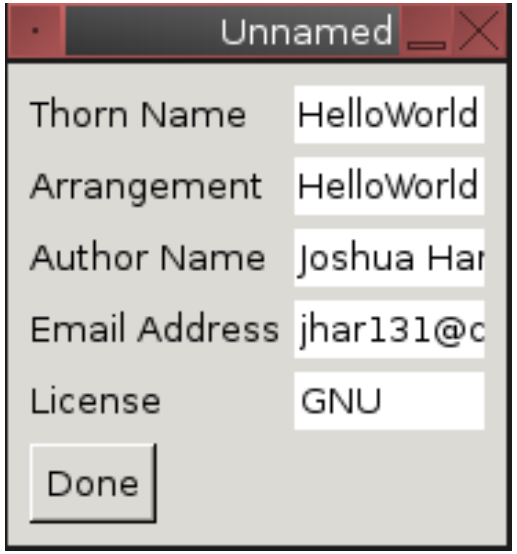
- Select Mojave – NewThorn from the drop down tab
- Title NewThorn and Arrangement as HelloWorld



Creating a New Thorn

To Create A New Thorn:

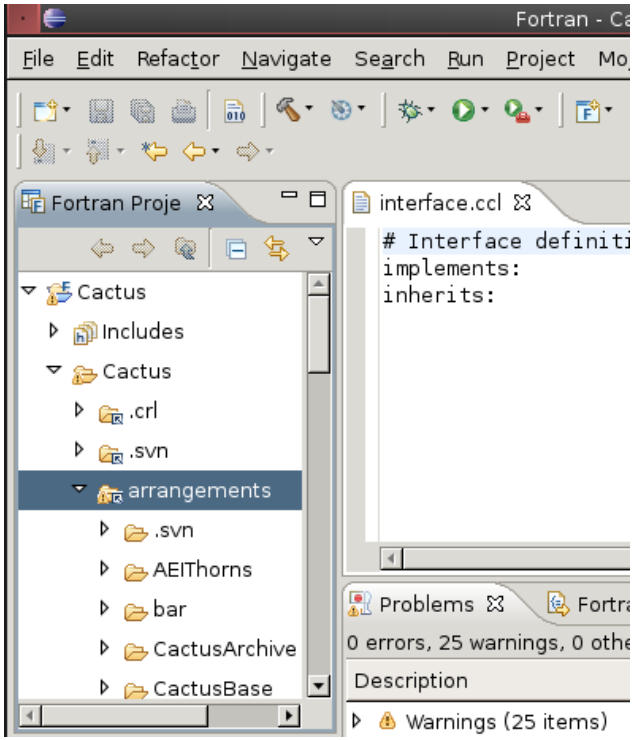
- Select Mojave – NewThorn from the drop down tab
- Title NewThorn and Arrangement as HelloWorld
- Fill the other fields as you see fit



Creating a New Thorn

From the Fortran Projects column:

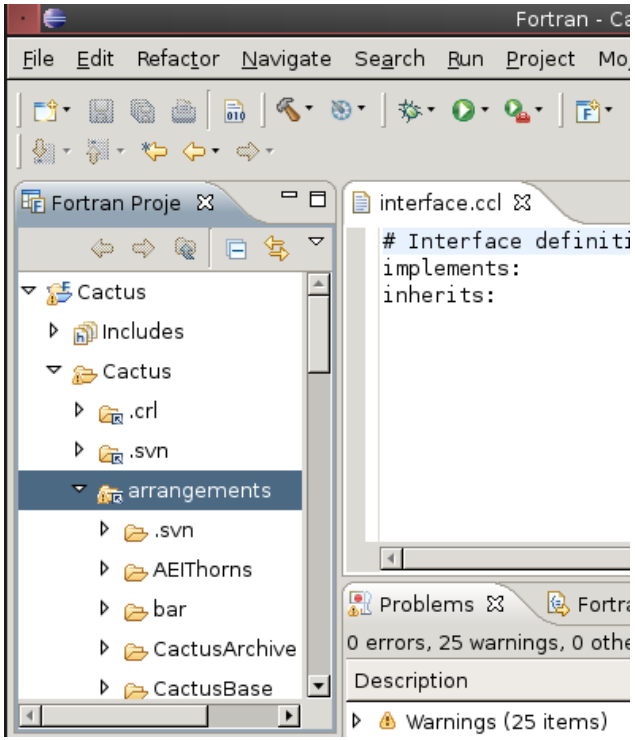
- Expand arrangements



Creating a New Thorn

From the Fortran Projects column:

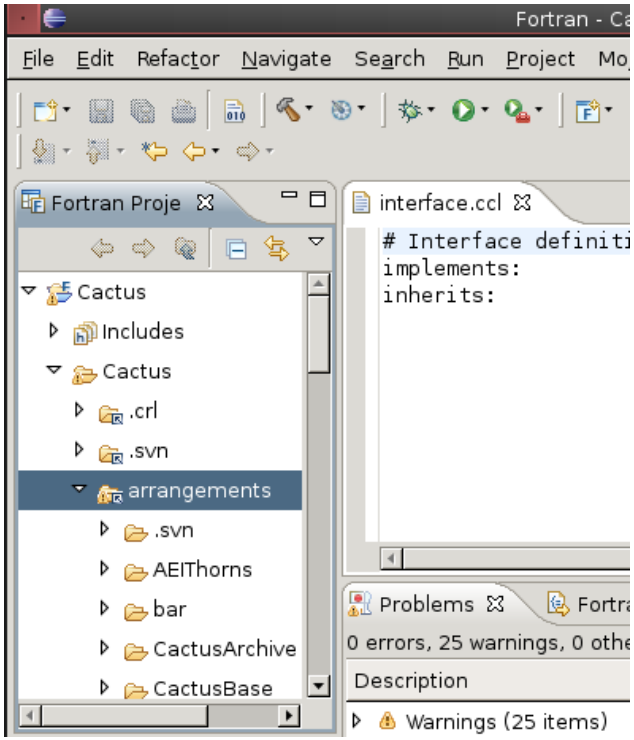
- Expand arrangements
- Expand the HelloWorld arrangement and thorn



Creating a New Thorn

From the Fortran Projects column:

- Expand arrangements
- Expand the HelloWorld arrangement and thorn
- Open interface.ccl and schedule.ccl



Hello World Thorn

Copy the following into each CCL file:

- `interface.ccl`:

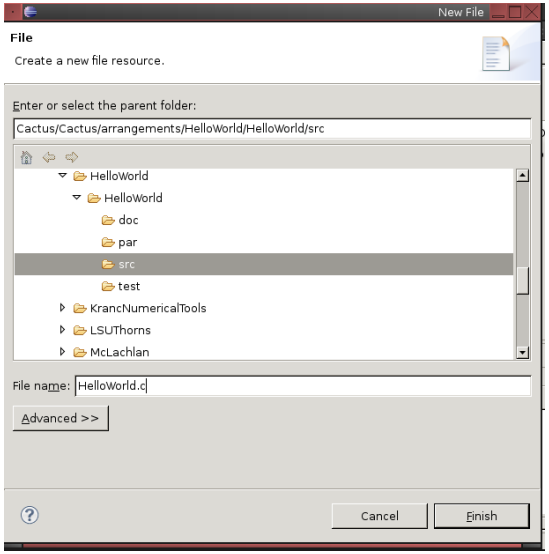
```
implements: HelloWorld
```

- `schedule.ccl`:

```
schedule HelloWorld at CCTK_EVOL
{
  LANG: C
} "Print Hello World message"
```

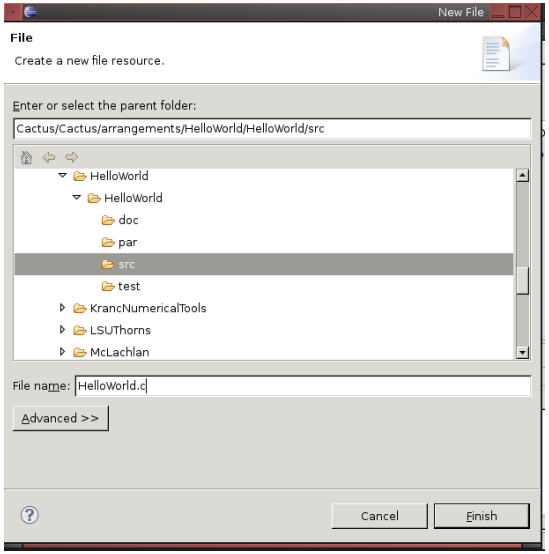
- `param.ccl`: empty

Running Cactus



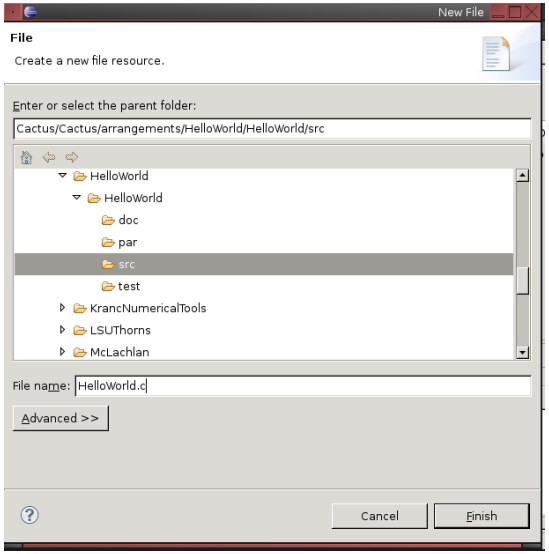
- Expand the src folder

Running Cactus



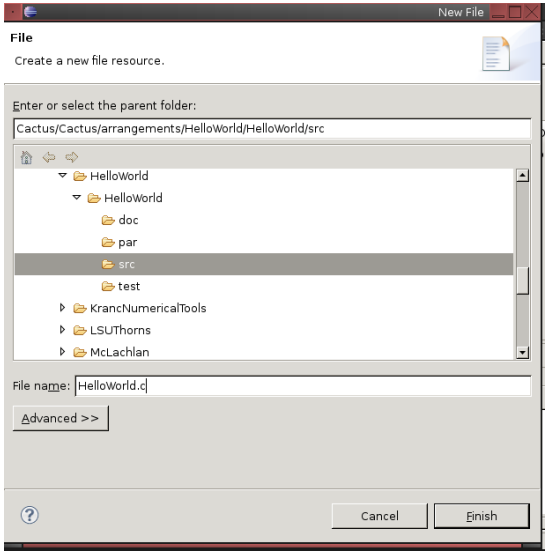
- Expand the src folder
- Right-Click src and select New – File

Running Cactus



- Expand the src folder
- Right-Click src and select New – File
- Make sure the parent folder is correct

Running Cactus



- Expand the src folder
- Right-Click src and select New – File
- Make sure the parent folder is correct
- Title the file: HelloWorld.c

Hello World Thorn cont.

- src/HelloWorld.c:

```
#include "cctk.h"
#include "cctk_Arguments.h"

void HelloWorld(CCTK_ARGUMENTS)
{
  DECLARE_CCTK_ARGUMENTS;
  CCTK_INFO("Hello World!");
  return;
}
```

- make.code.defn:

```
SRCS = HelloWorld.c
```

Running Cactus

- Move to the Cactus/Cactus/par subdirectory
- Create a new file: hello.par
- `hello.par`:

```
ActiveThorns = "HelloWorld"
Cactus::cctk_itlast = 10
```

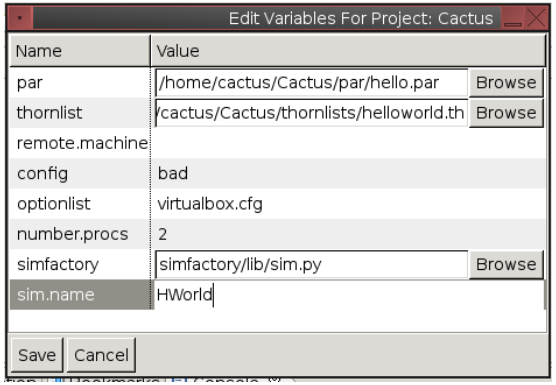
Running Cactus

- Move to the Cactus/Cactus/thornlists subdirectory
- Create a new file: hello.th
- `hello.th`:

```
HelloWorld/HelloWorld
```

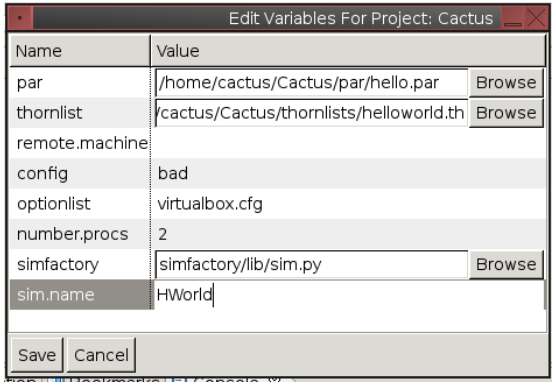
Running Cactus

- Select Mojave from the drop-down tab.



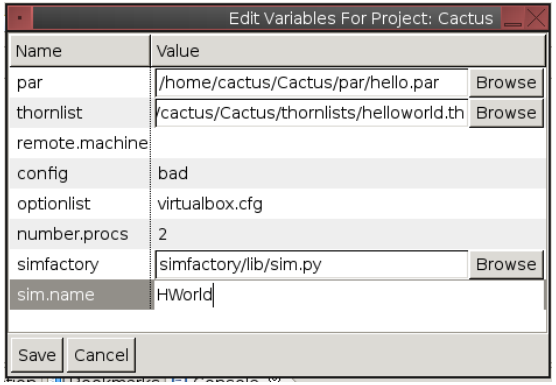
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables



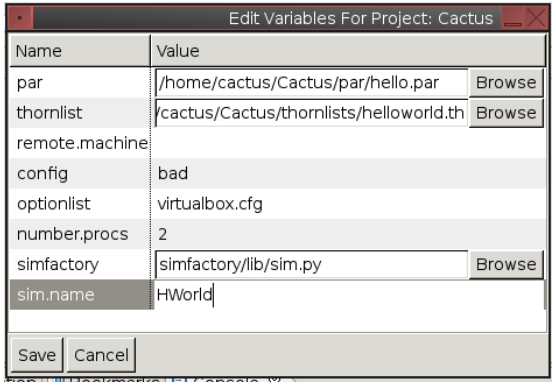
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:



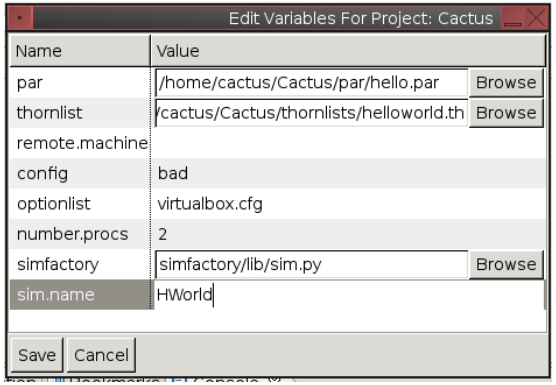
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field



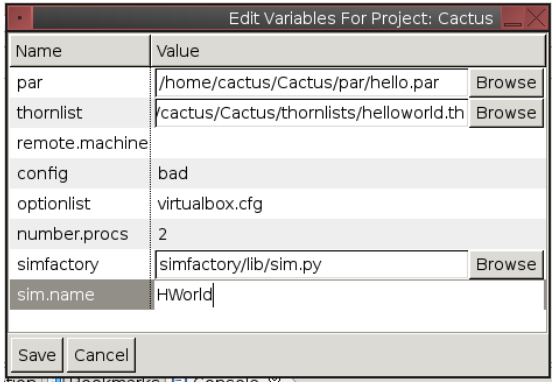
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field



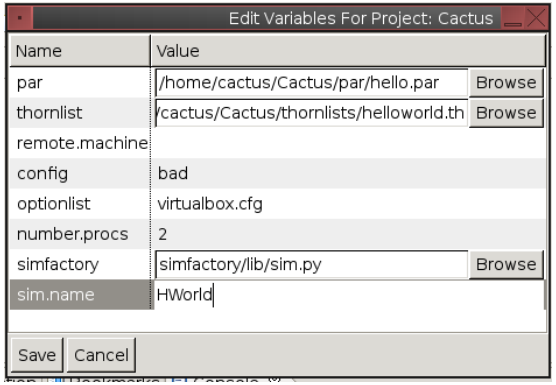
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field



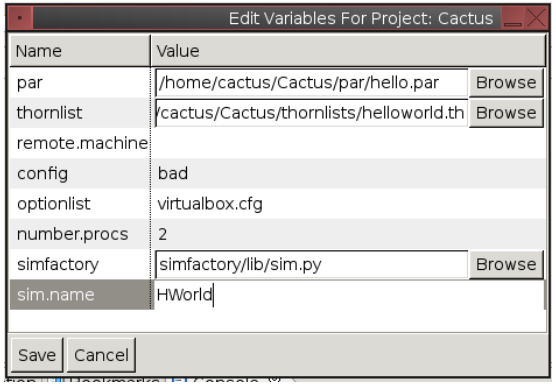
Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field
- Input HWorld in the sim.name field

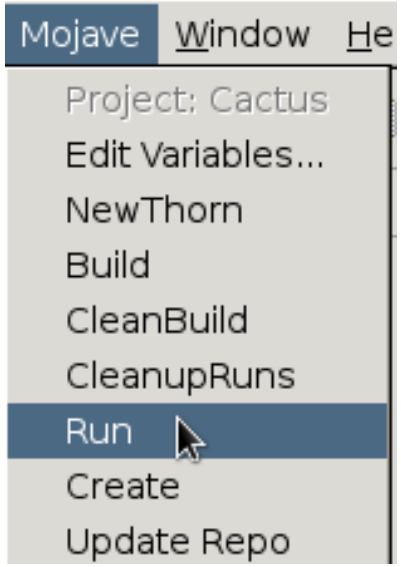


Running Cactus

- Select Mojave from the drop-down tab.
- Select Edit Variables
- Using the Browse button input:
  - hello.par in the par field
  - helloworld.th in the thornlist field
- Input bad2 in the config field
- Input HWorld in the sim.name field
- Select Save

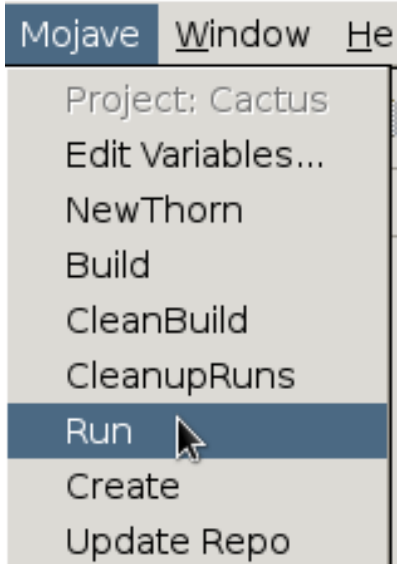


Running HelloWorld



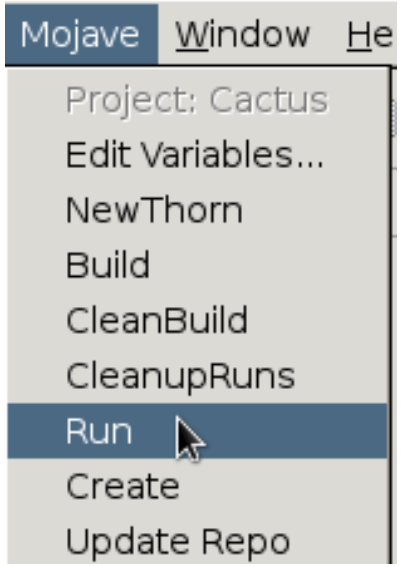
- To run HelloWorld:
  - From the Mojave drop down tab:

Running HelloWorld



- To run HelloWorld:
  - From the Mojave drop down tab:
  - Select Create

Running HelloWorld



- To run HelloWorld:
  - From the Mojave drop down tab:
  - Select Create
  - Select Run

Hello World Thorn

- Screen output:

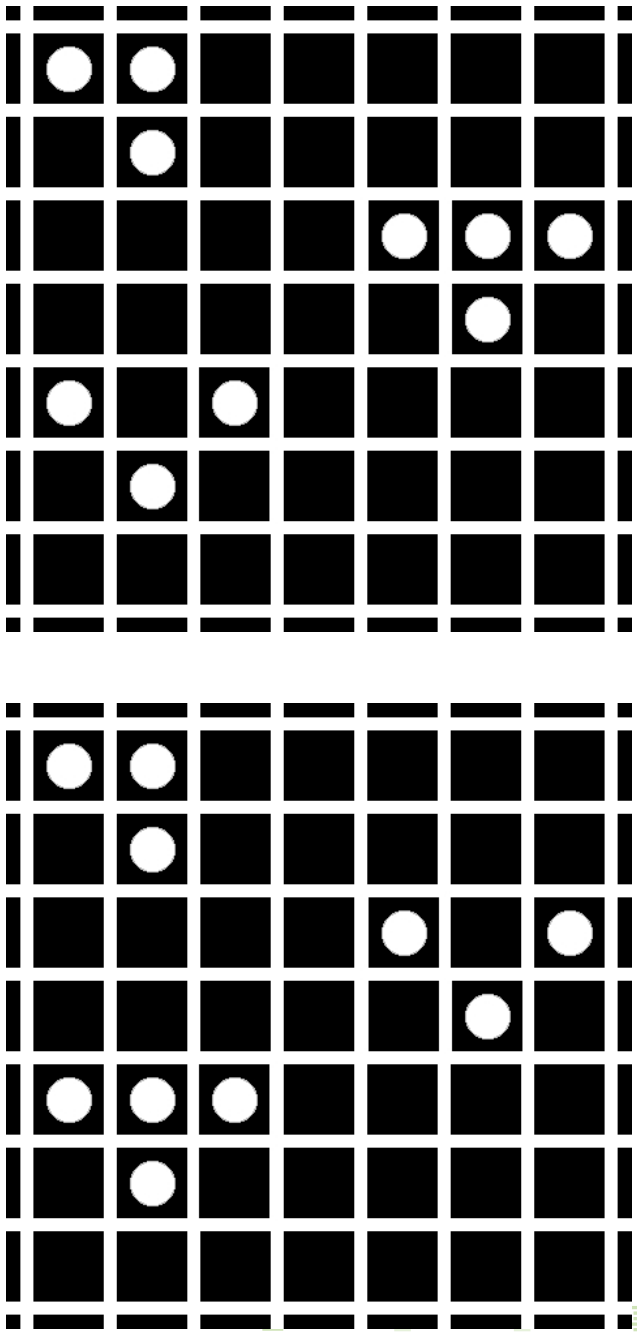
```
       10
  1   0101        ************************
  01  1010 10       The Cactus Code V4.0
 1010 1101 011       www.cactuscode.org
  1001 100101     ************************
    00010101
     100011      (c) Copyright The Authors
      0100       GNU Licensed. No Warranty
      0101

Cactus version:    4.0.b17
Compile date:      May 06 2009 (13:15:01)
Run date:          May 06 2009 (13:15:54-0500)
[...]

Activating thorn Cactus...Success -> active implementation Cactus
Activation requested for
--->HelloWorld<---
Activating thorn HelloWorld...Success -> active implementation HelloWorld
--------------------------------------------------------------------------------
INFO (HelloWorld): Hello World!
INFO (HelloWorld): Hello World!
[...] 8x
--------------------------------------------------------------------------------
Done.
```

Conway's Game of Life

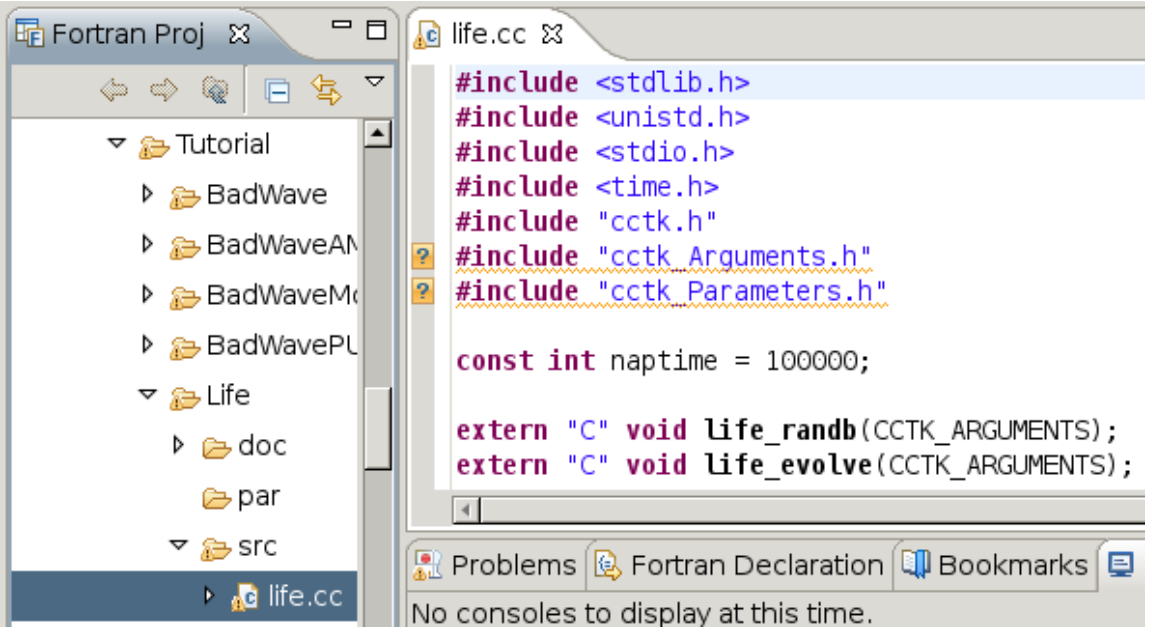
Conway's Game of Life

- Conway's Game of Life begins with a board of single-celled organisms. For each step of the game, a given cell lives if it is surrounded by two or three other cells. If a pixel on the board is surrounded by three cells, a cell is born on that pixel. If a living cell is surrounded by more than three cells, it dies.
- **Algorithm Conway**:
  1. Each element in an NxM matrix is initialized to 0 or 1.
  2. For each step of the game, generate a new board as follows: for each pixel on the current board, the value for the pixel for the new board is set as follows:
     1. If this pixel contains a 1 and if two or three 1's occupy any of its eight neighboring pixels, set the new value to 1.
     2. If this pixel contains a 0 and if three 1's occupy any of its eight neighboring pixels, set the new value to 1.
     3. If this pixel contains a 1 and the number of neighboring 1's is less than two or more than three, set the new value to 0.
     4. Otherwise set the new value to 0.

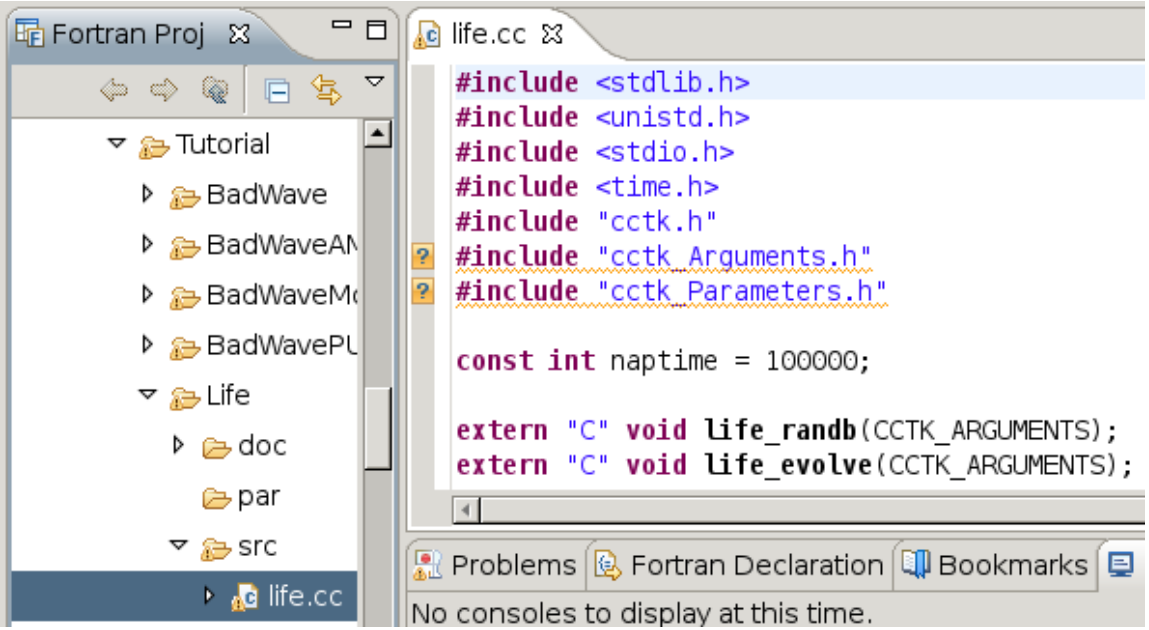
Consider the grid to the right. The cells in the upper left-hand corner each live because they each have two neighboring cells. In the upper-right hand corner, the cell surrounded by the three others dies from overcrowding on this iteration. In the lower left-hand corner, the space surrounded by three others bears a living cell.



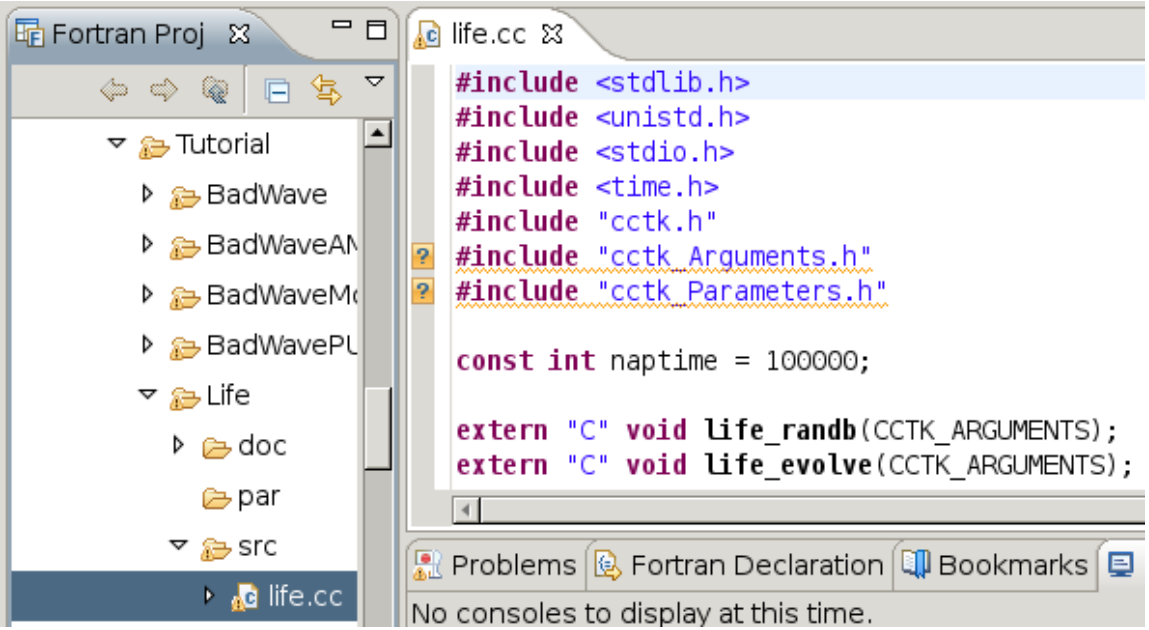
- The Cactus code for Conway's Game of Life can be found:



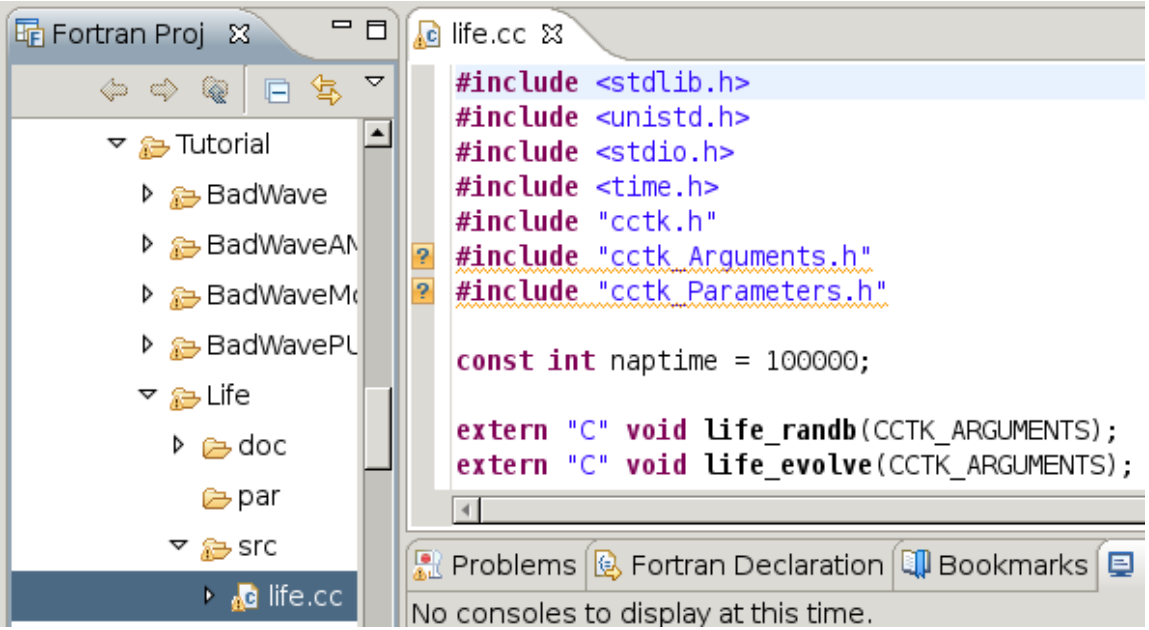
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc



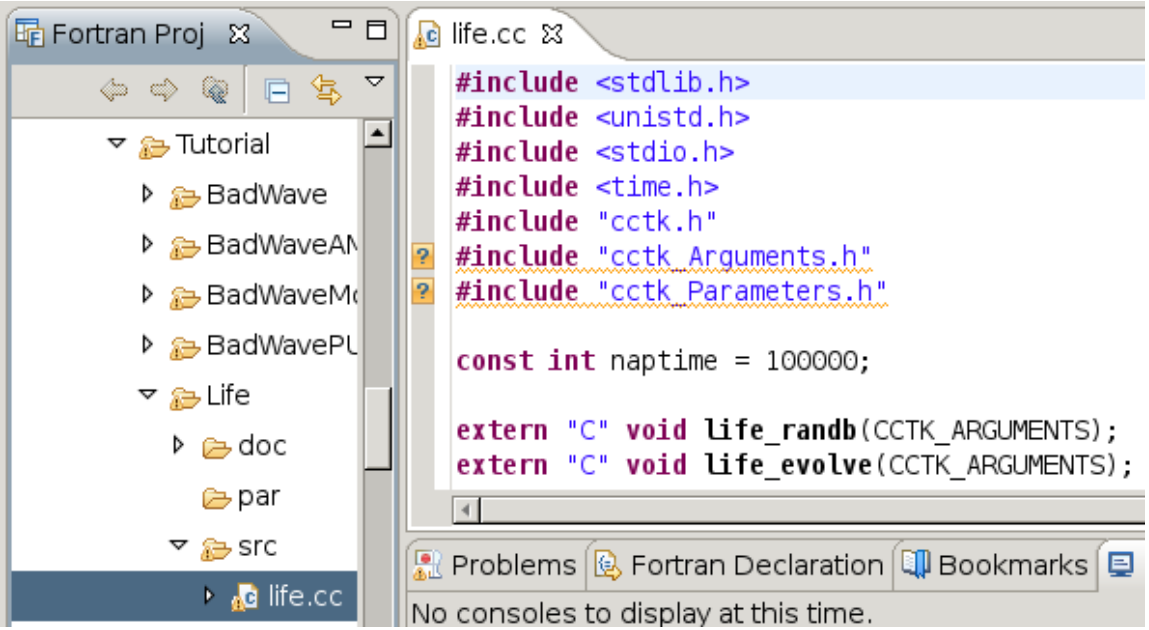
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:



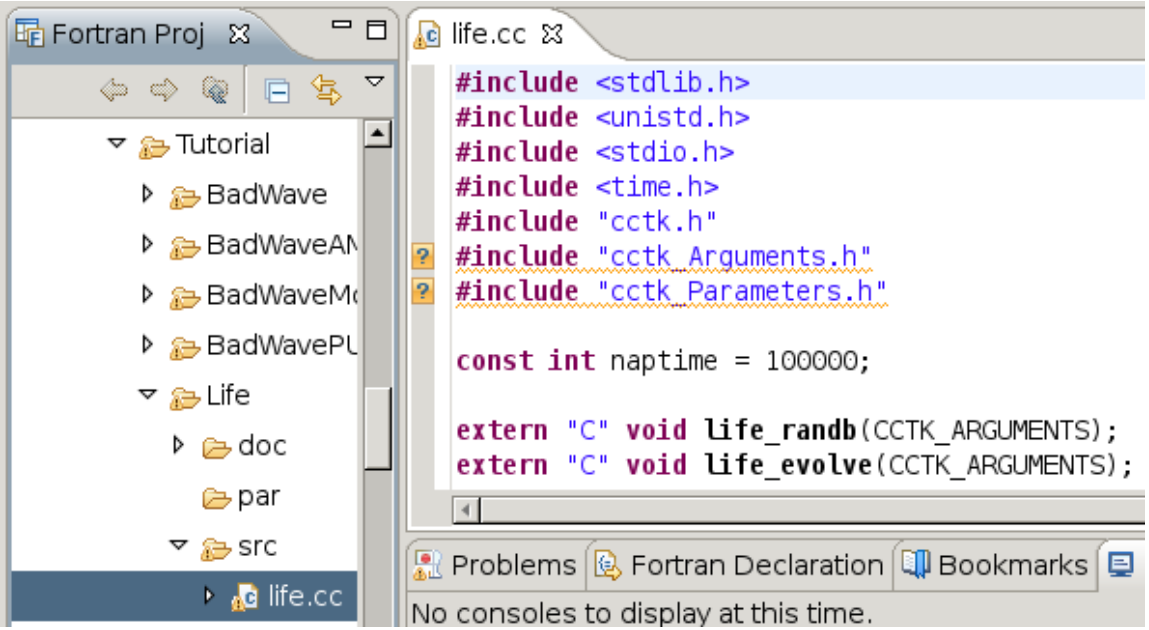
- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.



- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.
  - The removal of the print function. (void pboard)



- The Cactus code for Conway's Game of Life can be found:
- Cactus/arrangements/Tutorial/Life/src/life.cc
- Notable alterations:
  - The CCTK arguements and parameters declarations.
  - The removal of the print function. (void pboard)
  - The thorn IOJpeg will be used for output.



Notable Alterations:

Additional Header files for communication with Cactus:

- C code:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <time.h>
```

- Cactus code:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <time.h>
#include "cctk.h"                    //three additional headers
#include "cctk_Arguments.h"
#include "cctk_Parameters.h"
```

Notable Alterations:

Using Cactus' GFINDEX2D and the removal of INDEX2D:

- C code:

```
32    int cc = INDEX2D(i,j);

59    int cc = INDEX2D(i,j);
```

- Cactus code:

```
22   int cc = CCTK_GFINDEX2D(cctkGH, i, j);

36   int cc = CCTK_GFINDEX2D(cctkGH, i, j);
```

Definition Files

- `interface.ccl`:

```
# Interface definition for thorn Life
implements: life
inherits:

CCTK_INT boards TYPE=gf
{
        b, nb
} "boards"
```

- `param.ccl`:

```
# Parameter definitions for thorn Life

CCTK_INT random_seed "Initializer for the random number generator"
{
        *:* :: "Determines the starting position for the game of life"
} 0
```

Conways Game of Life: Cactus Implementation

- make.code.defn:

```
# Main make.code.defn file for thorn Life

# Source files in this directory
SRCS = life.cc

# Subdirectories containing source files
SUBDIRS =
```

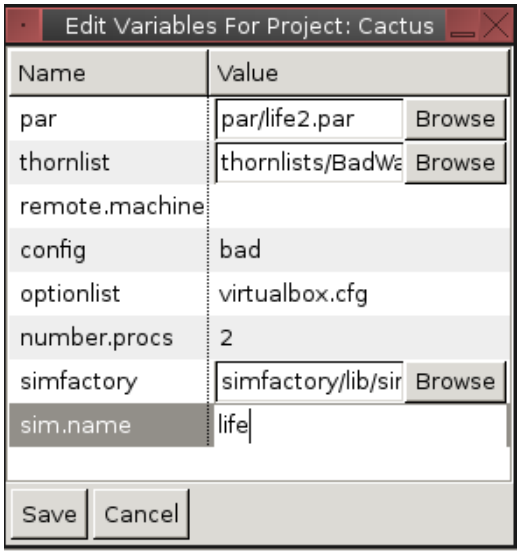
- schedule.ccl:

```
# Schedule definitions for thorn Life

storage: boards

schedule life_randb at INITIAL
{
        LANG: C
        SYNC: boards
} "setup board"

schedule life_evolve at EVOL
{
        LANG: C
        SYNC: boards
} "another generation lives or dies"
```

Running Cactus

To run Conway's Life through Mojave:

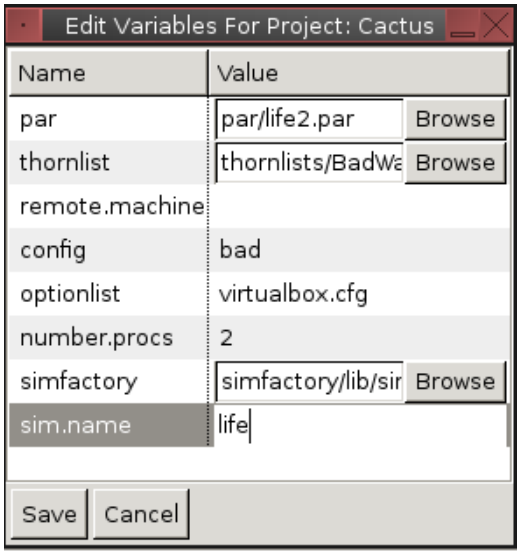
- Select Edit Variables from the Mojave tab



Running Cactus

To run Conway's Life through Mojave:

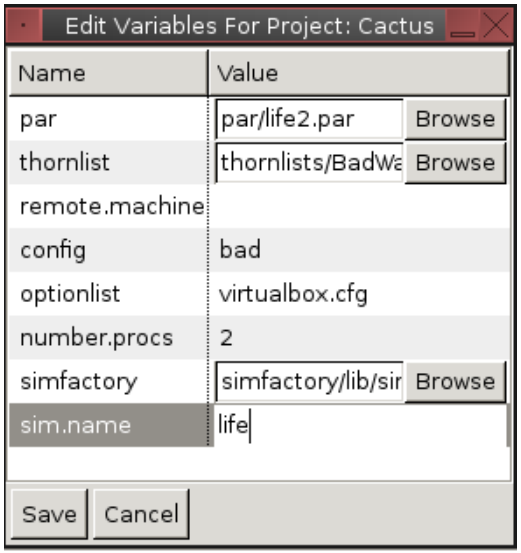
- Select Edit Variables from the Mojave tab
- For the par option:



Running Cactus

To run Conway's Life through Mojave:

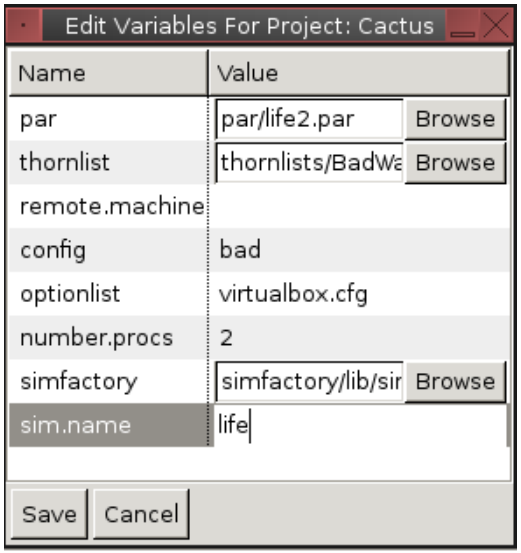
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par



Running Cactus

To run Conway's Life through Mojave:

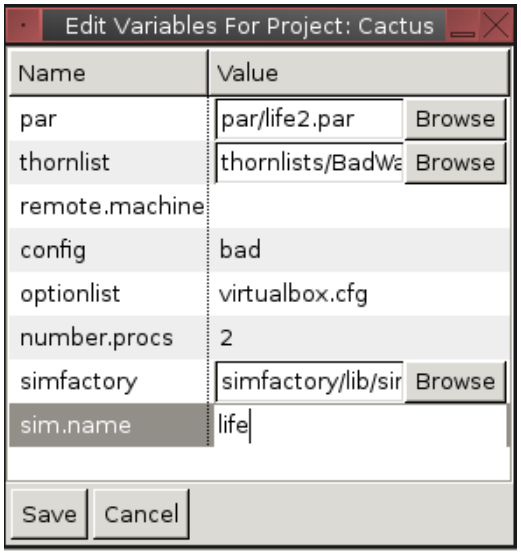
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:



Running Cactus

To run Conway's Life through Mojave:

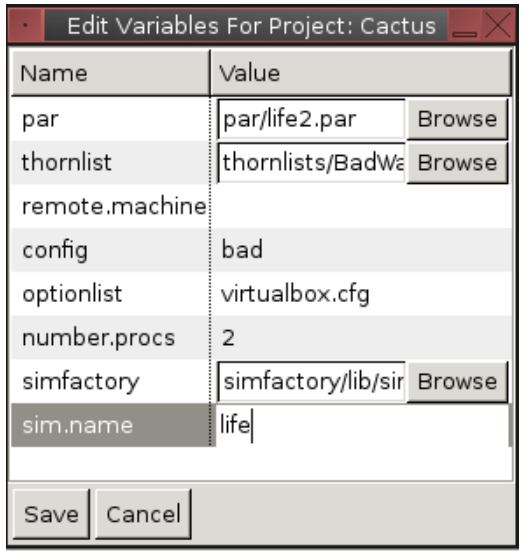
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad



Running Cactus

To run Conway's Life through Mojave:

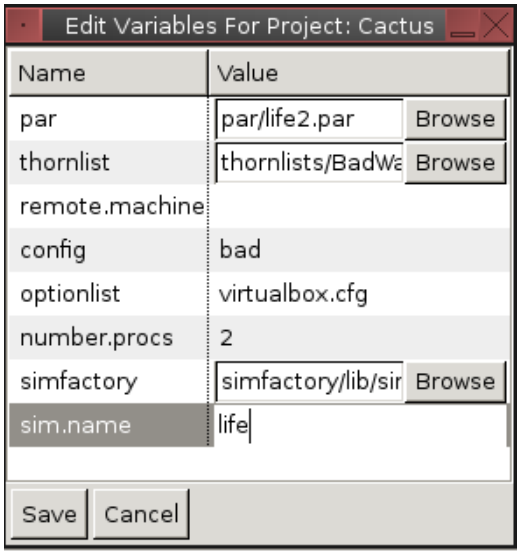
- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad
- Name the simulation life2



Running Cactus

To run Conway's Life through Mojave:

- Select Edit Variables from the Mojave tab
- For the par option:
  - Select Browse – Cactus – par – life2.par
- For the config option:
  - Input: bad
- Name the simulation life2
- Mojave-Create and Mojave-Run



Visualizing Conway's Life Results

- After running Conway's Life, you may navigate to the `simulations/life2/output-*` directory to see the JPEG image output.
- To animate the images, use the ImageMagick command `animate`. Since your VM uses minimal RAM, you may wish to animate only a fraction of the images:
- `animate "_[0-9][0-9]\."`
- This will animate images 11-99.

Parallelization Framework in Cactus

The Driver

- Special thorn
- Only one active for a run, choose at starttime
- The only code (almost) dealing with parallelism
- Other thorns access parallelism via API
- Underlying parallel layer transparent to application thorns
- Examples
  - PUGH: unigrid, part of the Cactus computational toolkit
  - Carpet: mesh refinement, `http://www.carpetcode.org`

Grid functions

Cactus provides methods to:

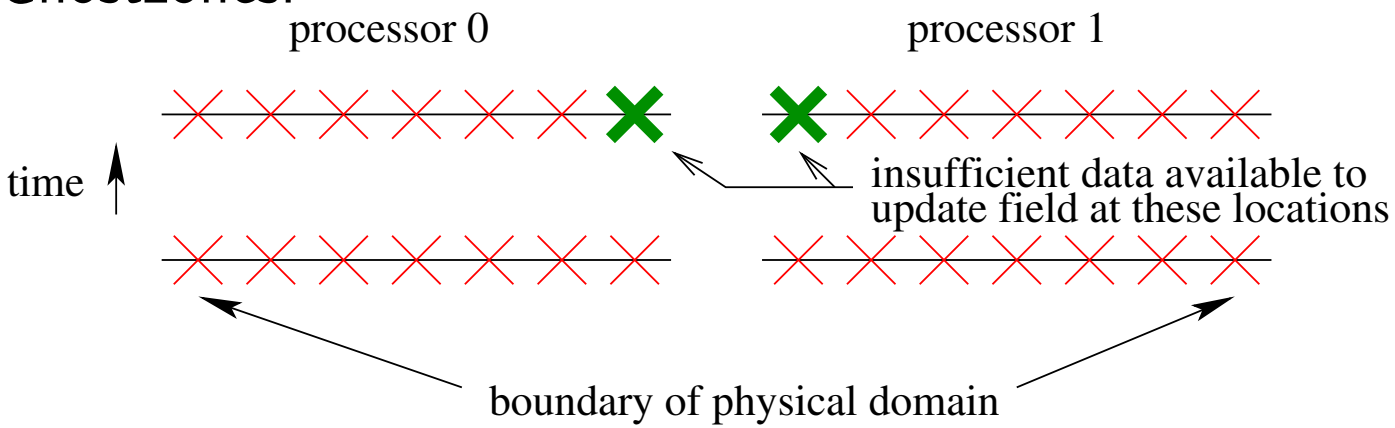
- Distribute variables across processes (grid function)
- Synchronize processor domain boundaries between processes
- Compute reductions across grid functions
- Actual implementation in driver thorn

Ghost Zones

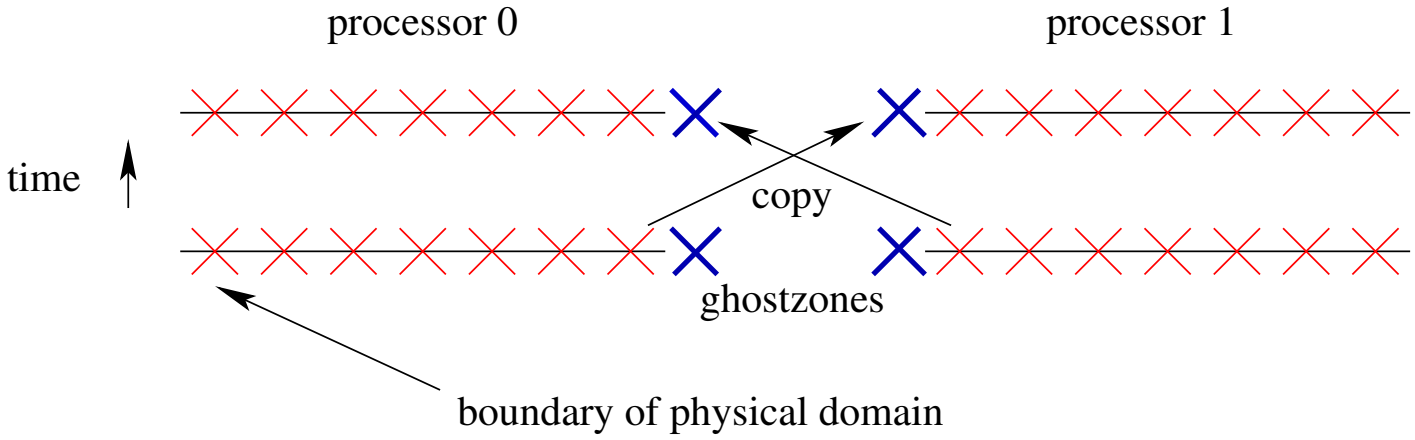
- Grid variables: distributed across processes
- Assumption: Most work done (quasi-) locally:
  True for hyperbolic and parabolic problems
- Split of computational domain into blocks
- Only communication at the borders (faces)
- At least stencil size many ghostzones

Ghost Zone Example

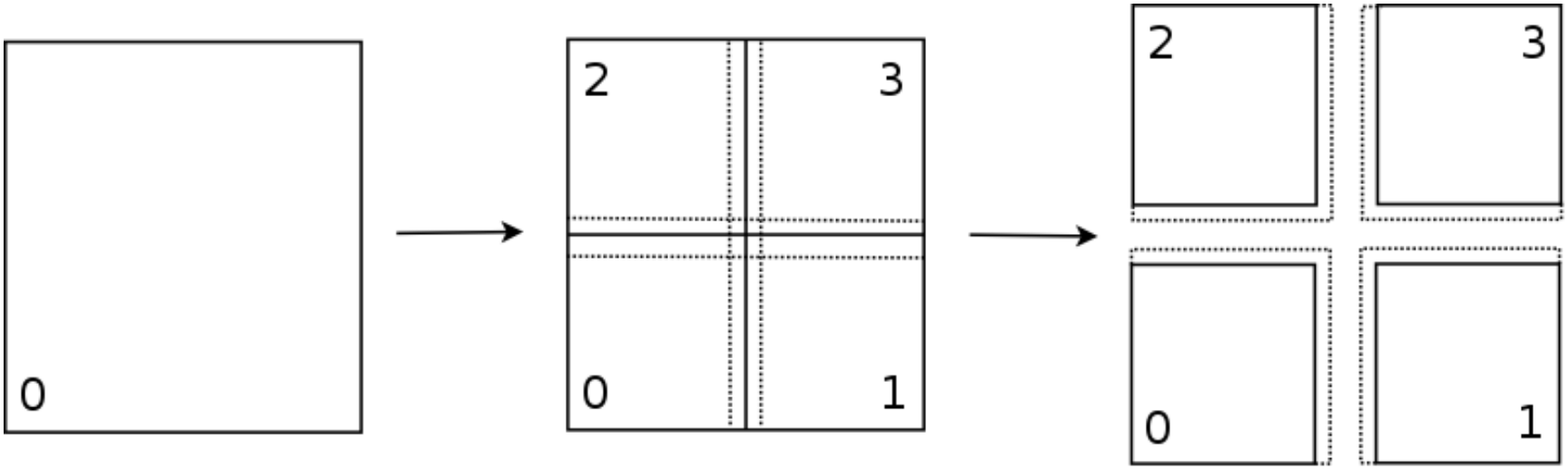
Without Ghostzones:



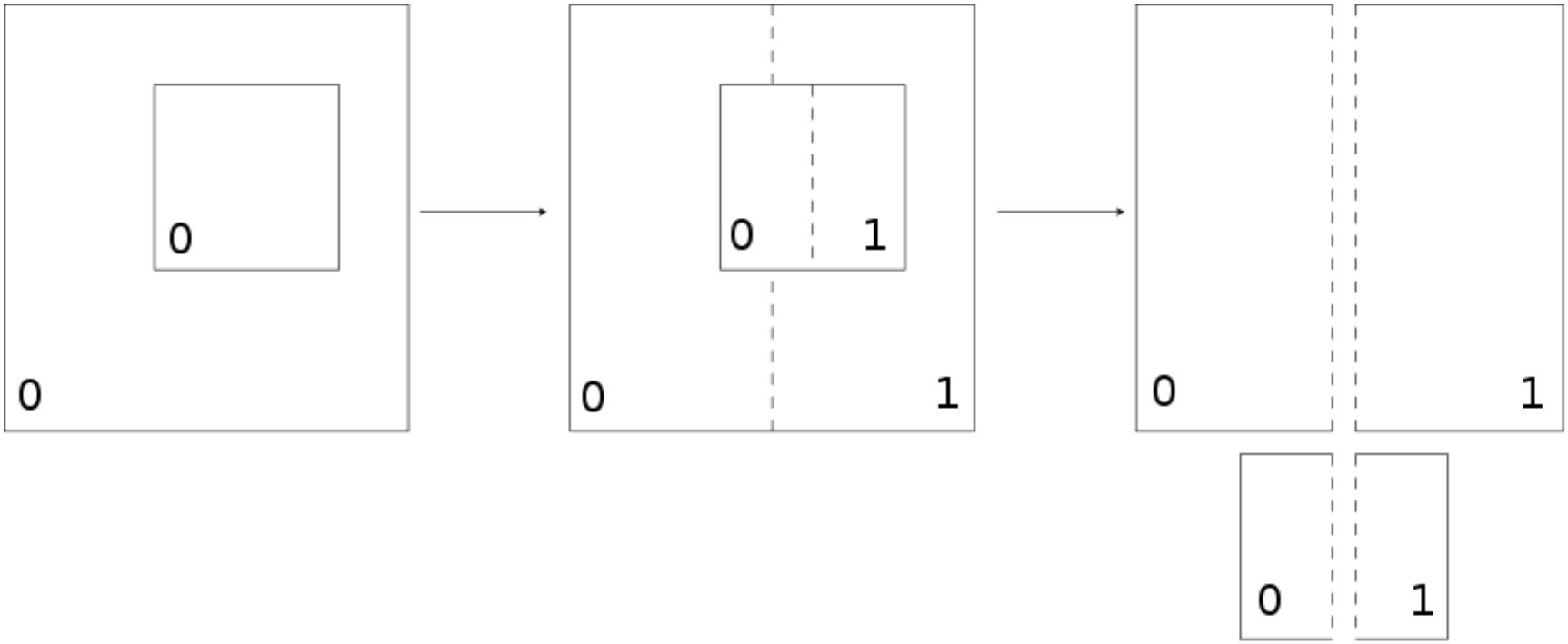
With Ghostzones:



Domain Decomposition



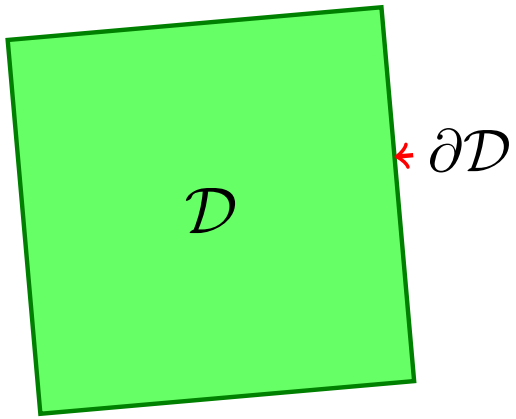
Mesh Refinement Decomposition



Solving the Wave Equation

The Wave Equation

- PDE which describes wave propagation in a medium
- one of the fundamental equations of mathematical physics
- *For a spatial domain $\mathcal{D}$, find a scalar field $\psi(x, y, z, t)$ which satisfies:*



$$\frac{\partial^2 \psi}{\partial t^2} = c^2 \nabla^2 \psi \qquad \textit{inside } \mathcal{D}\textit{, and}$$

$$B(\psi, \partial_i \psi)|_{\partial\mathcal{D}} = 0 \qquad \textit{at } \partial\mathcal{D}.$$

where c is the speed of the wave,
$B(\psi, \partial_i \psi)$ specifies the boundary conditions,
and ∇^2 is the Laplacian of ψ:

$$\nabla^2 \psi \equiv \left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right] \psi$$

First-order form

The wave equation is a second-order PDE for a single scalar function. To improve its numerical stability properties, we can rewrite it as a system of first-order PDEs:

$$\frac{\partial \psi}{\partial t} = c\vec{\nabla} \cdot \vec{p} \qquad \frac{\partial \vec{p}}{\partial t} = c\nabla\psi$$

with a new unknown vector variable $\vec{p}$. If $\vec{p} = \nabla\psi$, we recover the original scalar wave equation.

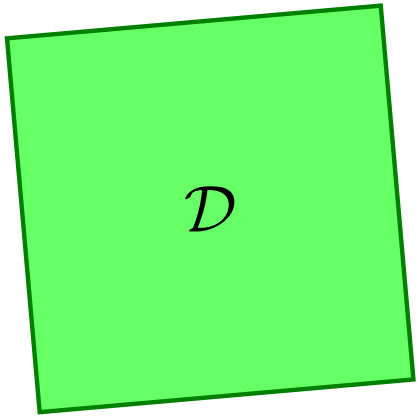
We are going to solve this system of PDEs inside a *cube* $\mathcal{D} = [-a, a]^3$ with the following boundary conditions:

$$\psi|_{\partial\mathcal{D}} = 0 \qquad \vec{p}_{\parallel}|_{\partial\mathcal{D}} = 0 \qquad \vec{p}_{\perp}|_{\partial\mathcal{D}_{i+}} = \vec{p}_{\perp}|_{\partial\mathcal{D}_{i-}}$$

where $\vec{p}_{\perp}$, $\vec{p}_{\parallel}$ are the (outward) normal and tangential components of $\vec{p}$, and $\partial\mathcal{D}_{i+}/\partial\mathcal{D}_{i-}$ are the opposite faces normal to the i-th axis.

Discretization

We will be solving the wave equation using finite difference (FD) methods on rectangular grids (adding mesh refinement later).



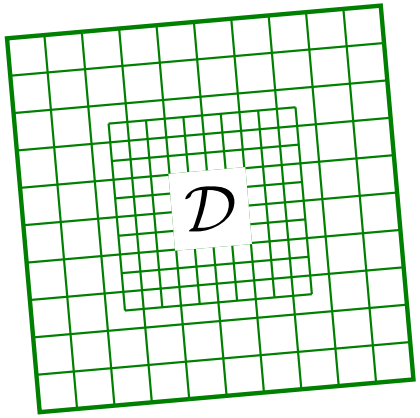
- spatial domain $\rightarrow$ rectangular grid
- continuous function $\rightarrow$ grid function
- partial derivative $\rightarrow$ FD operator
- PDE $\rightarrow$ system of algebraic equations

$$\left.\begin{array}{l} \frac{\partial \psi}{\partial t} = c\vec{\nabla} \cdot \vec{p} \\ \\ \frac{\partial \vec{p}}{\partial t} = c\nabla\psi \end{array}\right| \rightarrow \left|\begin{array}{l} D_t\psi = K_\psi := c(D_x p_x + D_y p_y + D_z p_z) \\ D_t p_x = K_x := cD_x\psi \\ D_t p_y = K_y := cD_y\psi \\ D_t p_z = K_z := cD_z\psi \end{array}\right.$$

where D_t, D_x, D_y and D_z are the *finite differencing* operators.

Discretization

We will be solving the wave equation using finite difference (FD) methods on rectangular grids (adding mesh refinement later).



- spatial domain $\longrightarrow$ rectangular grid
- continuous function $\longrightarrow$ grid function
- partial derivative $\longrightarrow$ FD operator
- PDE $\longrightarrow$ system of algebraic equations

$$\left.\begin{aligned} \frac{\partial \psi}{\partial t} &= c\vec{\nabla}\cdot\vec{p} \\ \frac{\partial \vec{p}}{\partial t} &= c\nabla\psi \end{aligned}\right| \rightarrow \left|\begin{aligned} D_t\psi &= K_\psi := c(D_x p_x + D_y p_y + D_z p_z) \\ D_t p_x &= K_x := cD_x\psi \\ D_t p_y &= K_y := cD_y\psi \\ D_t p_z &= K_z := cD_z\psi \end{aligned}\right.$$

where D_t, D_x, D_y and D_z are the *finite differencing* operators.

Spatial derivatives

We approximate spatial derivatives by finite differences of the grid function values at neighboring points. We will use a centered scheme, which is second-order accurate in grid spacing Δ:

$$(D_x\psi)_{i,j,k} = \frac{\psi_{i+1,j,k} - \psi_{i-1,j,k}}{2\Delta}$$

$$(D_y\psi)_{i,j,k} = \frac{\psi_{i,j+1,k} - \psi_{i,j-1,k}}{2\Delta}$$

$$(D_z\psi)_{i,j,k} = \frac{\psi_{i,j,k+1} - \psi_{i,j,k-1}}{2\Delta}$$

Runge-Kutta time integration

Simplest Runge-Kutta method: Newton integration (1st order accurate)

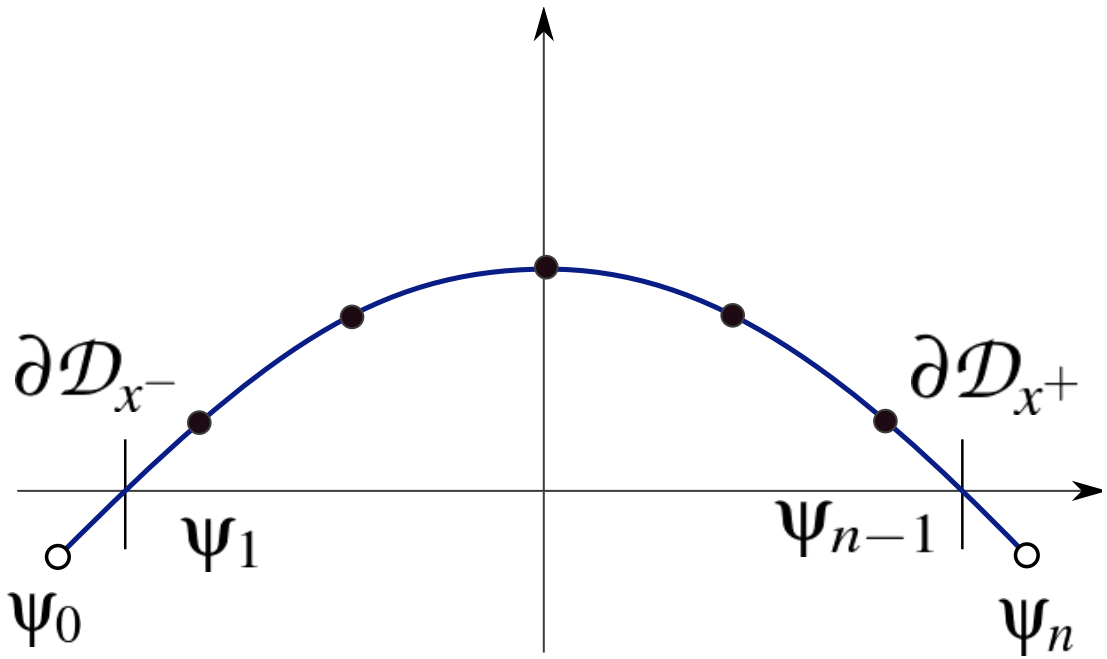
$$\frac{y_{n+1} - y_n}{\Delta t} = K_y(t_n, y_n) \qquad \rightarrow \qquad y_{n+1} = y_n + K_y(t_n, y_n)\Delta t$$

We will use a 2nd order accurate Runge-Kutta integration scheme (also known as a *Heun method*) with two intermediate steps:

$$\begin{aligned}
k_1 &= K_y(t_n, y_n) \\
\tilde{y}_n &= y_n + k_1 \Delta t, \\
k_2 &= K_y(t_n + \Delta t, \tilde{y}_n) \\
y_{n+1} &= y_n + \left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right)\Delta t
\end{aligned}$$

Discrete boundary conditions

We place the grid points in such a way that the boundary of the domain $\partial\mathcal{D}$ lies in between the two last grid points:

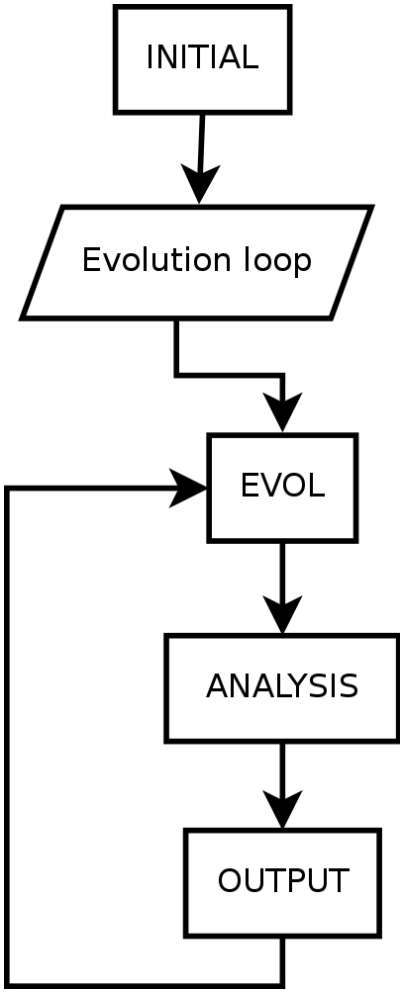


$$\psi|_{\partial\mathcal{D}} = 0$$
$$\vec{p}_{\parallel}|_{\partial\mathcal{D}} = 0$$
$$\vec{p}_{\perp}|_{\partial\mathcal{D}_{i+}} = \vec{p}_{\perp}|_{\partial\mathcal{D}_{i-}}$$
$$\Downarrow$$
$$\psi_{0,j,k} = -\psi_{1,j,k},$$
$$\vec{p}_{y,0,j,k} = -\vec{p}_{y,1,j,k},$$
$$\vec{p}_{z,0,j,k} = -\vec{p}_{z,1,j,k},$$
$$\vec{p}_{x,0,j,k} = \vec{p}_{x,1,j,k},$$

... etc.

Standalone code

```
int main(int argc,char **argv) {
    FILE *fp = fopen("psi.d.asc","w+");
    BadWave_Init();
    BadWave_ApplyBounds();
    BadWave_print(fp);
    // evolve 100 steps
    for(int i=0;i<100;i++) {
        iter = i;
        BadWave_Evolve();
        BadWave_ApplyBounds();
        BadWave_Evolve2();
        BadWave_TmpApplyBounds();
        BadWave_print(fp);
    }
    fclose(fp);
    return 0;
}
```



Standalone code: parameters and definitions

```
// Parameters:
bool verbose = true;      // enable verbose output
double wave_speed = 1.0; // wave speed factor
const int isiz = 32, jsiz = 32, ksiz = 32; //grid size

// 1D array to hold the grid and a function to find 3D indices
typedef double gridfunc[isiz*jsiz*ksiz];
inline int INDEX3D(int i,int j,int k) { return (i+j*isiz)*ksiz+k; }

// Grid functions:
gridfunc psi,px,py,pz;            // psi and p
gridfunc tpsi,tpx,tpy,tpz;        // temporary variables
gridfunc kpsi,kpx,kpy,kpz;        // the kernels
gridfunc k2psi,k2px,k2py,k2pz;

// Other global variables: iteration, grid spacing and time step
int iter=0;    double dx, dy, dz, dt;
```

Writing thorn BadWave: param.ccl

```
# Parameter definitions for thorn BadWave

private:
CCTK_REAL wave_speed "characteristic speed at boundary"
{
  "0.1:*" :: "wave speed"
} 1.

private:
CCTK_BOOLEAN verbose "whether to print stuff"
{
    :: "verbose flag"
} 0
```

Writing thorn BadWave: interface.ccl

```
# Interface definition for thorn BadWave
implements: BadWaveTutorial
inherits: grid

CCTK_REAL wave_vars TYPE=gf
{
    psi
    px, py, pz
} "Basic components of wave equation code"

CCTK_REAL tmp_wave_vars TYPE=gf { tpsi, tpx, tpy, tpz } <...>
CCTK_REAL kernels TYPE=gf {
    kpsi kpx, kpy, kpz, k2psi, k2px, k2py, k2pz
} <...>
```

Writing thorn BadWave: schedule.ccl

```
# Schedule definitions for thorn BadWave
storage: wave_vars, tmp_wave_vars, kernels

schedule BadWave_Init at INITIAL
{ LANG: C
} "Startup and initialize"

schedule BadWave_ApplyBounds at POSTSTEP
{   LANG: C
    SYNC: wave_vars
} "Apply boundary conditions"

schedule BadWave_Evolve at EVOL <...>
schedule BadWave_TmpApplyBounds at EVOL after BadWave_Evolve <...>
schedule BadWave_Evolve2 at EVOL after BadWave_TmpApplyBounds <...>
```

Adapting the source code for Cactus

Modifications to standalone code:

- include Cactus headers:

```
#include "cctk.h"
#include "cctk_Arguments.h"
#include "cctk_Parameters.h"
```

- declare all scheduled functions with external C linkage:

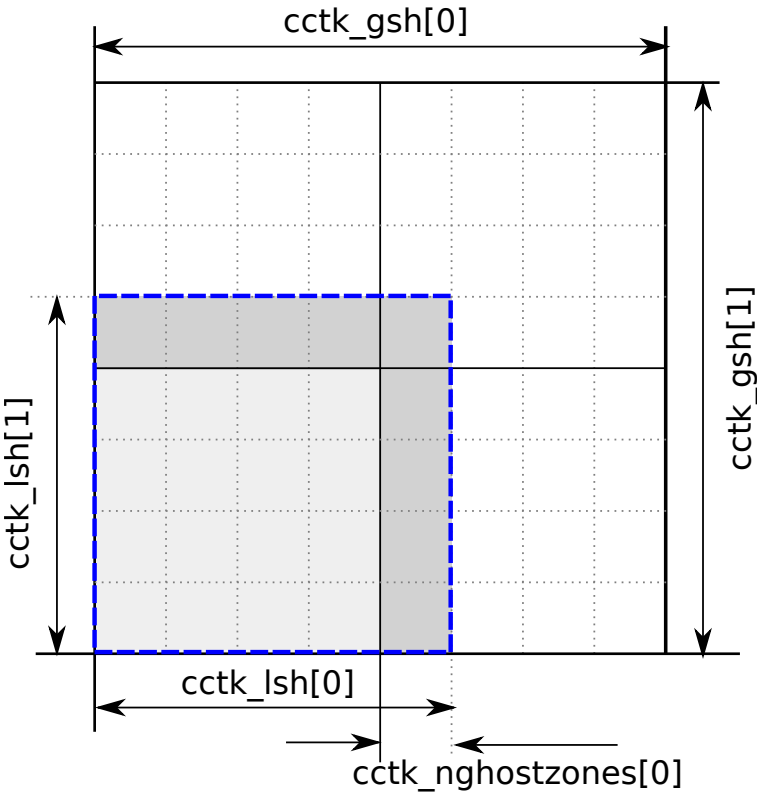
```
extern "C" void BadWave_Init(CCTK_ARGUMENTS);
extern "C" void BadWave_ApplyBounds(CCTK_ARGUMENTS);
extern "C" void BadWave_TmpApplyBounds(CCTK_ARGUMENTS);
extern "C" void BadWave_Evolve(CCTK_ARGUMENTS);
extern "C" void BadWave_Evolve2(CCTK_ARGUMENTS);
```

- add the CCTK_ARGUMENTS macro to scheduled functions:

```
void BadWave_Init(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    ...
}
```

this tells functions which part of the grid they will be working on.

Grid definitions in Cactus



- `cctk_gsh`: global grid dimensions
- `cctk_lsh`: local grid dimensions
- `cctk_delta_space`: grid spacing
- `cctk_nghostzones`: number of ghostzones
- `cctk_bbox`: whether a boundary is an outer one
- `CCTK_GFINDEX3D(i,j,k)`: computes local 1D grid index

Simple parameter file

Parfile: `par/BadWavePUGH.par`

- include the driver thorn (PUGH)
- specify grid size
- specify BadWave parameters

```
ActiveThorns = "BadWave PUGH"

cactus::cctk_itlast   = 100
pugh::global_nsize    = 32
pugh::ghost_size      = 1

BadWavePUGH::wave_speed = 2.0
```

More advanced parameter file

Parfile: `par/BadWavePUGHv2.par`
Includes new thorns:

- `CoordBase`: coordinate extents
- `CardGrid3D`: provides variety of grid configurations
- `SymBase`: basic thorn for specifying symmetries of the domain
- `IOBasic, IOUtil, IOScalar, IOASCII`: basic and advanced I/O
- `Time`: provides global time and iteration variables
- `PUGHSlab, PUGHReduce, LocalReduce`: required by I/O thorns

Method of Lines

- Method of lines is a general name for an approach of solving PDEs with time variable, in which the PDE is approximated by a system of interdependent ODEs for each grid point.
- Method of lines allows to extend time integration methods developed for ODEs to PDEs.
- Cactus thorn MoL implements several different time integration methods.
- We can use these methods if we modify our thorn: `BadWaveMoL`

Registering evolved variables with MoL

- in Wave.cc:

```
void BadWaveMoL_Register(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    MoLRegisterEvolved(CCTK_VarIndex("BadWaveMoL::psi"),
                       CCTK_VarIndex("BadWaveMoL::kpsi"));
    ...
}
```

- in schedule.ccl:

```
storage: wave_vars[3], kernels[3]
schedule BadWaveMoL_Register in MoL_Register
{
    LANG: C
} "Register for MoL"
```

Registering evolved variables with MoL

- in `interface.ccl`:

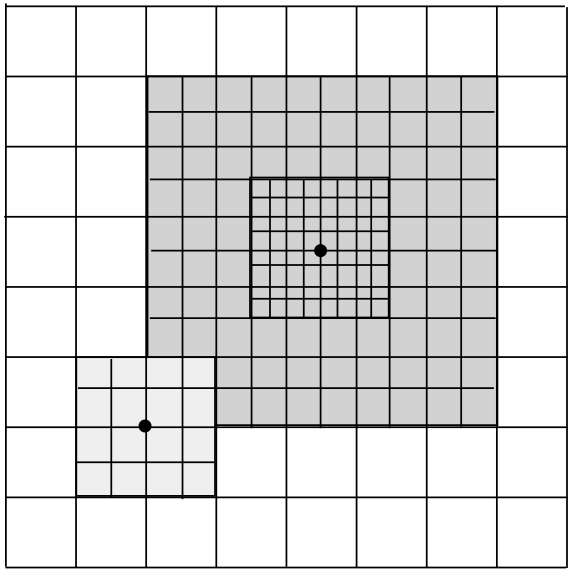
```
CCTK_INT FUNCTION MoLRegisterEvolved \
  (CCTK_INT IN EvolvedIndex, CCTK_INT IN RHSIndex)
USES FUNCTION MoLRegisterEvolved
```

- in the parfile (`par/BadWaveMoL.par`):

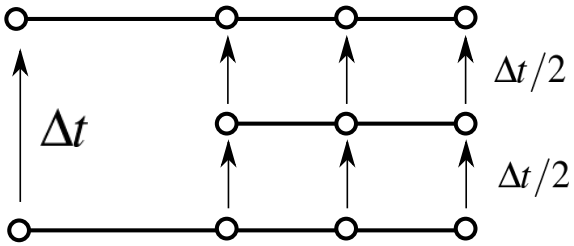
```
ActiveThorns = "BadWaveMoL carpet
CarpetIOBasic CarpetIOASCII CarpetIOScalar CarpetLib LoopC
IOUtil SymBase Time CarpetReduce CartGrid3D MoL"

MoL::ODE_Method = "RK4"
MoL::MoL_Intermediate_Steps = 4
MoL::MoL_Num_Scratch_Levels = 1
```

Berger-Oliger mesh refinement

- "Box-in-box" refinement:



- Time update:



STARTUP
PARAMCHECK

↑ BASEGRID
INITIAL, POSTINITIAL

↓ RESTRICT

↑ ANALYSIS

Main loop

↑ REGRID

↑ Advance time
EVOL (incl. MoL)

↓ RESTRICT

↑ CHECKPOINT
ANALYSIS

Fixed mesh refinement

Parfile: par/BadWaveFMR.par

```
ActiveThorns = "CarpetRegrid2 Dissipation
SpaceMask SphericalSurface"

CarpetRegrid2::num_levels_1 = 2
Carpet::max_refinement_levels = 2
CarpetRegrid2::num_centres = 1
CarpetRegrid2::Position_x_1 = 0.5
CarpetRegrid2::Position_y_1 = 0.5
CarpetRegrid2::Position_z_1 = 0.5
CarpetRegrid2::radius_1[1] = 0.1
#CarpetRegrid2::radius_1[2] = 0.1
CarpetRegrid2::verbose = "yes"
Carpet::init_fill_timelevels="yes"
Dissipation::vars = "BadWaveMoL::psi"
```

Adaptive mesh refinement

Modifications to the code:

```
void BadWaveAMR_BoxMover(CCTK_ARGUMENTS)
{
    DECLARE_CCTK_ARGUMENTS;
    int max_refinement_levels = 30;

    radius[max_refinement_levels*0+1]=.05;

    // Make the box wiggle
    position_x[0] = position_y[0] = position_z[0] = 0.7+0.05*sin(cctk_time)
    active[0]=1;
    num_levels[0]=2; // two levels: 0=base grid, 1=first refined grid

    // Turn on the next box
    active[1] = 1;
    radius[max_refinement_levels*1+1]=.05;
    position_x[1]=position_y[1]=position_z[1]=.3;
    num_levels[1]=2;
}
```

Adaptive mesh refinement

- Modifications to the `schedule.ccl`:

```
schedule BadWaveAMR_BoxMover at preregrid
{
    LANG: C
} "Jiggle the box"
```

- Modifications to the `interface.ccl`:

```
inherits: grid, CarpetRegrid2
```

Adaptive mesh refinement

Modifications in the parfile (pars/BadWaveAMR.par):

```
CarpetRegrid2::regrid_every = 2
Carpet::max_refinement_levels = 3
CarpetRegrid2::num_centres = 2

CarpetRegrid2::num_levels_1 = 3
CarpetRegrid2::Position_x_1 = 0.3
CarpetRegrid2::Position_y_1 = 0.3
CarpetRegrid2::Position_z_1 = 0.3
CarpetRegrid2::radius_1[1] = 0.2
CarpetRegrid2::radius_1[2] = 0.1

CarpetRegrid2::num_levels_2 = 2
CarpetRegrid2::Position_x_2 = 0.7
CarpetRegrid2::Position_y_2 = 0.7
CarpetRegrid2::Position_z_2 = 0.7
CarpetRegrid2::radius_2[1] = 0.05
```

Adaptive mesh refinement

Modifications in the parfile (pars/BadWaveAMR.par):

```
ActiveThorns = "IOJPeg CarpetSlab CarpetInterp
         CarpetInterp2 AEILocalInterp"
IOJpeg::out_every                = 1
IOJpeg::out_vars                 = "BadWaveAMR::psi"

IOJPeg::gridpoints = interpolate
IOJpeg::array2d_x0          = 0.0
IOJpeg::array2d_y0          = 0.0
IOJpeg::array2d_z0          =  0.5
IOJpeg::array2d_npoints_i   = 101
IOJpeg::array2d_dx_i        =  0.01
IOJpeg::array2d_dy_i        =  0
IOJpeg::array2d_dz_i        =  0
IOJpeg::array2d_npoints_j   = 101
IOJpeg::array2d_dx_j        =  0
IOJpeg::array2d_dy_j        =  0.01
IOJpeg::array2d_dz_j        =  0
```