Module 9.3 – **Parallel Models for Simulation of Acentric Complex Phenomena**

Empedocles Research Group

Prof Salvatore Di Gregorio – Department of Mathematics - UNICAL
Prof Gino Mirocle Crisci – Department of Earth Sciences – UNICAL
Dr Giulio Iovine – CNR - IRPI
Dr Rocco Rongo– Department of Earth Sciences – UNICAL
Dr William Spataro – Department of Mathematics - UNICAL
Dr Donato D'Ambrosio – Department of Mathematics - UNICAL
Dr Maria Vittoria Avolio – Department of Mathematics - UNICAL
Dr Valeria Lupiano – CNR - IRPI

Prof Domenico Talia – DEIS - UNICAL

- Cellular Automata
- An empirical method for modelling macroscopic complex
- Cellular Automata Environments (CAMEL, CAMELot)
- Load Balancing
- Performance
- Geological Applications
  - SCIARA**:** Etnean Crisis of 2001 e 2002
  - SCIDDICA: Sarno (Italy) - 1998
  - PYR: Mt. Pinatubo (The Philippines) 1991

Cellular Automata VS Differential Equations

Everything should be made as simple as possible, but not simpler !
(Albert Einstein)

Cellular Automata is an "alternative", rather than an "approximation", to Differential Equations
(Tommaso Toffoli)

Complex Systems and Cellular Automata

- The main characteristics that are associated with Complex Systems regard the presence of an elevated of interacting elements, interaction non-linearity and appearance of *emergent* behaviors, with no corresponding microscopic analogous. Least but not last, auto-organization capacities
- Cellular Automata, together with Neural Nets and Genetic Algorithms, represent valid instruments for the description of complex phenomena

Cellular Automata

- Cellular Automata (CA) can be thought as abstract dynamical systems which play a role in discrete mathematics, such as that of partial differential equations in continuous math
- Studied and conceived by J. Von Neumann in the 1950s to study auto-reproducing phenomena
- The Cellular Automata approach involves *locality* (interactions between states) and *uniformity* (same evolution for each cell)
- Cellular Automata = Time and Space are discrete (usually square or hexagonal tessellations in case of 2D CA)

AC Applications

- **Environmental**: Lava flows, landslides, pollution, bio-remediation, earthquakes, forrest fires, soil erosion, etc
- **Bio-medicine**: Immune system, cancer cell growth
- **Industrial Applications:** Coffee percolation (ILLY), Tire mixture (Pirelli)
- **Fluid-dynamics**, **Traffic**, **ETC…**

Cellular Automata and Parallel Computing

- Traditional sequential computers do not offer a practical support for the implementation of CA, since the transition function should be applied to each cell, one after the other.
- The CA model represents a implicitly parallel computational model which can be easily implemented on parallel architectures due to the inherent parallelism of the model
- CA exploit data parallelism by partitioning cells among processing elements (PE) of a parallel computer.

UNIVERSITÀ DELLA CALABRIA

Cellular Automata and Parallel Computing

- *Data parallelism* can be exploited on SIMD machines, but implementations on MIMD machines result more efficient
- In fact, SIMD machines are suitable for CA that have **all** active cells during simulation
- However, in many natural phenomena, an elevated number of cells are usually in a passive or inert state, making the SIMD approach not efficient (many PEs are not effectively utilized!)

UNIVERSITÀ DELLA CALABRIA

Cellular Automata and Parallel Computing

- On MIMD machines, a CA can be implemented by mapping on each PE a process which updates a portion of cells.
- Multiprocessor machines result appropriate and efficient: each PE can individuate the stationary region and don't execute calculations for these cells
- When dealing with shared memory, even if no communications exists between borders of each region of each PE, speedup results in being at most 10-12 (=BOTTLENECK!)

UNIVERSITÀ DELLA CALABRIA

Solution

- In order to avoid memory bus bottlenecks, a good solution is devising a system formed by an adequate number of cooperating PEs by means of Message Passing (i.e. MPI)
- This makes the system scalable, both if the number of PEs are increased, or if the CA dimensions are augmented (speed-up)
- *Data-parallelism*, an intrinsic property of CA, can be exploited by adopting a SPMD approach (Single Program, Multiple Data), where *N* processes are mapped on *N* PEs, which operates on a different set of data

UNIVERSITÀ DELLA CALABRIA

CA programming environments

- Many CA environments have been implemented on sequential computers (PCs, Workstations)
- Two approaches are individuated: Hardware and Software
- CAM (Cellular Automata Machine) (MIT, Boston, 1987) represents the most famous example of dedicated **hardware** architecture for studying CA (efficient, but few states ...)
- **Software** environments: P-CAM, PECANS, StarLogo, CAPE, CAMEL, CAMELot

UNIVERSITÀ DELLA CALABRIA

Caratteristics of Parallel CA Systems

- Main characteristics of a parallel software environment should be:
  - High level Programming layer for the design of computational models, independent of the underlying parallel architecture (e.g. CARPET language)
  - Graphic User Interface *(GUI)* which permits a complete visualzaion of the evolution of the phenomenon and the diplay of numerical values connected with the simulation
  - Tuning and control instruments which permit *steering* (global control) of the evolution of the phenomenon
  - Scalability to permit an efficient execution of the phenomenon on parallel computers

UNIVERSITÀ DELLA CALABRIA

CAMEL - CAMELot

- Camel is a CA environment for parallel architectures which exploit message passing
- First version in 1991 on a Transputer Network (Inmos Item-4000)
- CAMELot is a portable (MPI) version of CAMEL, developed under the COLOMBO Esprit project by ISI-CNR, EPCC and ENEA
- Parallelism is invisible to the user
- The user has only to specify the transition function of a single cell by means of a high level language, such as C or CARPET)

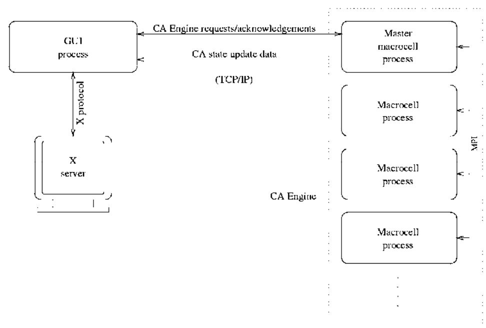
CAMELot Applications

- CAMELot has been applied with success to numerous scientific fields:
  - Modeling of macroscopic complex phenomena, such as lava flows, landslides, soil erorion, et
  - Fish reproduciton
  - “Microscopic” Highway Traffic Modelling
  - Image recognition
  - Genetic Algoritms

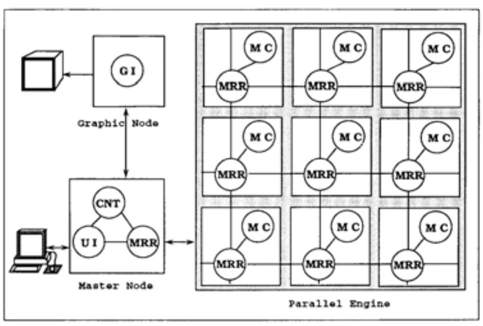
CAMELot - Architecture

- Camelot exploits the CA approach by means of the SPMD model (Single Program Multiple Data)
- The run time system is composed of a set of *macrocell* processes. Each *macrocell* implements a partition of cells on a single processor of the parallel computer (the user does not specify data addressing
- All *macrocell* processes are executed in parallel in order to update the state of cells that form the CA
- The Graphical Interface permits the dynamic visualization and on-line steering

CAMEL – CAMELot - Architecture



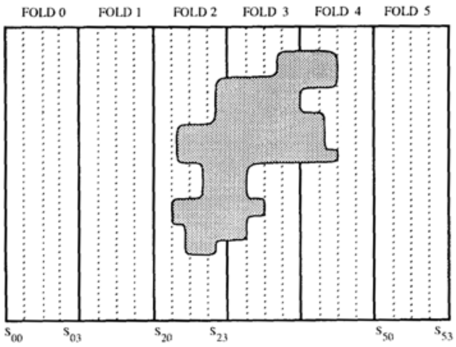
Camelot Architecture



Load Balancing

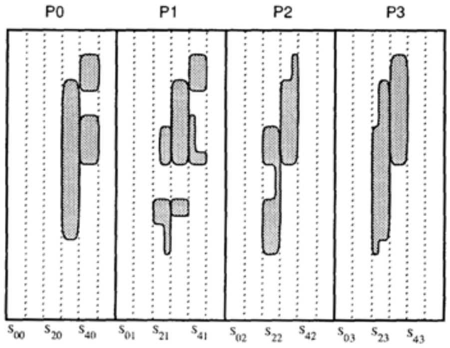
- In many phenomena (like lava flows, landslides, etc), the area corresponding to the active cells is restricted to one or few domains (active cells)
- Thus, it is not efficient to compute the new state of these cells, at least until they remain so
- CAMEL adopts a compromise between a static and dynamic load balancing (*scatter-decomposition*). The partitioning of cells is static, while the number of cells that are mapped on each partition is dynamic

Load Balancing (*Folding - Stripwise-scatter*)



Fold number =6 ; PE = 4

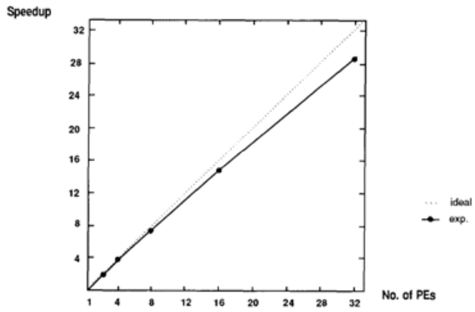
Load Balancing (*Folding - Stripwise-scatter*)



Performance

- CAMEL scalability is good
- Several tests have been executed in order to verify both *speed-up* measurements (increasing the number of processors in order to solve the problem in less time) and *scale-up* measurements (increasing the number of processors in order to solve bigger problems in the same time)
- For 32 PEs, speed-up is 28

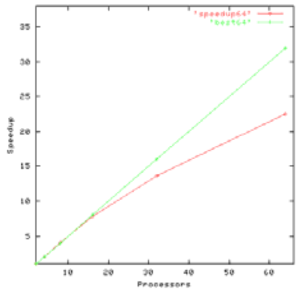
Performance di CAMEL - Transputers



CRAY T3E-900

Processors	*Sum (sec)*	*Total (sec)*	*Speedup*	*Optimum*
2	280.31	332.94	1	1
4	136.31	164.04	2.02	2
8	67.78	81.46	4.08	4
16	35.29	42.56	7.82	8
32	20.42	24.41	13.63	16
64	12.41	14.77	22.54	32

CRAY T3E-900



CAMELot : Again!

- CAMELot is an environment for the programming and seamlessly parallel execution of Cellular Automata.
- The system supports CARPET, a purpose-built language for CA programming. It offers a programming environment and a Graphical User Interface which enables the user to interact with the system while running a simulation and to view visualization of the simulated data.
- Since its simulator is very flexible with regard to cellular space sizes, cell structures, neighborhood structures and cellular automata rules, CAMELot can simulate almost all 1-, 2-D or 3-D cellular automata models.

Starting CAMELot

- Assuming the current working directory is the top directory of the CAMELot binary distribution, CAMELot is invoked from a UNIX shell using the command:

platform/camelot [*X options*] [*filename*]

Where
platform is the platform identifier (the supported platforms are sunos5, linux, irix6 and tru64);
filename is a CARPET source file;
X options are the standard X application command line flags (-display,-geometry,-iconic,-fn etc). These command line arguments are optional.

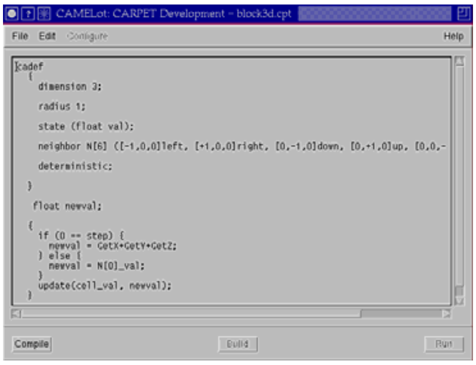
The CAMELot Development Window appears on the screen. It consists of three sections:

- A Menu Bar;
- An Editor subwindow with a scroll bar in each direction;
- A three-Button bar.

Editing a program

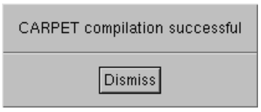
- A user may write a program using the editor window. Alternatively, they may open a previously saved program file using the Open option of the File menu. After any modifications the file must be saved using the Save or Save As option of the File menu; if a filename has been provided, this is done automatically when pressing the Compile button.
- Program editing is facilitated with the use of the Cut, Copy and Paste Options of the Edit menu. Shortcuts are available for all these functions.

Main Window - Editor



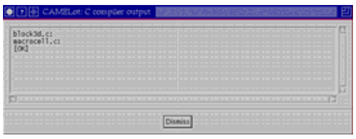
Program Compilation

- When the program (*.cpt) is ready, the user may compile it by clicking the Compile button. A successful compilation is followed by a pop-up window dismissible by clicking its Dismiss button. An erroneous compilation causes a beep and a pop-up window provides information about the error.
- This operation generates a standard *.c file that will be linked with other libraries (MPI, etc)



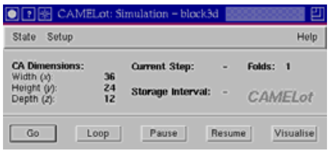
Building a Program

- The Build operation generates a Unix (Linux!) executable file for CA execution. In order to build a file the user must first set the configuration parameters by using the Configure menu. These define:
  1. The Dimensions of the CA Engine;
  2. The number of Processes to handle the task;
  3. The number of Folds into which the task is divided.
- The user can then build the executable by pressing the Build button. The output of the C compiler is shown to the user in a pop-up window.



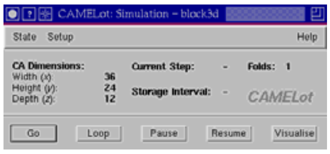
Running a Program

- The Configure menu of the Development Window includes a menu by which the user can initialise the collection of statistics for the basic functions of the CA Engine. This should be enabled before clicking the Run button. After successful compilation and building the program, the user can invoke the executable by clicking the Run button. This pops up the Simulation Window which consists of three parts, a Menu bar, a Display part and a Button bar.



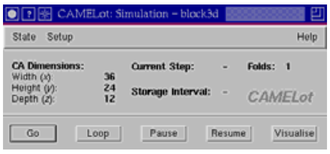
Running a Program

- The State menu contains an Initialise and a Save Option. The user may initialise a substate or the whole state of a CA using an existing file, or save the current status of the CA. The display part of the window contains information about the configuration of the CA and updates the current step when the CA is running.



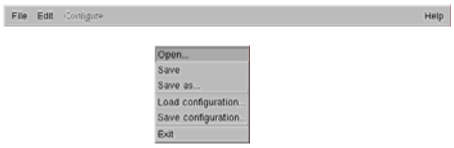
Running a Program

- The Go and Loop buttons initialise the CA execution, the former for a number of steps defined from the Setup menu, the latter indefinitely. The Pause button temporarily suspends CA execution and allows visualisation window examination, state saving or editing etc. The user may continue the CA execution by clicking on the Resume button or restart the execution by clicking Go or Loop. The Visualise button allows the visualisation of a substate in various formats. The statistics for the functions of the system are output periodically during the run or after stopping the CA Engine execution, according to the user's request.



CAMELot Functionality Overview

- The CAMELot environment supports 3 different types of Windows. We will examine them in order of appearance when using the environment.
- The Development Window pops up when running CAMELot and, when it is closed, CAMELot exits



The **Open** and **Save As** options pop up a window which allows the user to navigate through the filesystem and select the desired filename. For a file to be visible by Open, its name must have the extension .cpt. The Save option is only available if a filename has been specified for a file being edited. The **Exit** button exits CAMELot; the Delete button usually available on X Window titlebars is disabled for this window.

CAMELot Functionality Overview

- The characteristics of the program which are set under the Configure menu of the Development Window are automatically saved in a file named `progname.cnf`, progname being the full pathname of the CARPET file, every time the users saves the CARPET file.
- They are automatically retrieved when the CARPET file is Opened. In addition to this automatic facility, the Save and Load Configuration options allow the user to explicitly save and retrieve the configuration of the model

CAMELot Functionality Overview

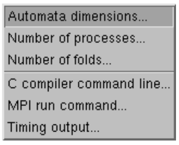
- Menu EDIT

Cut, Copy, Paste, Find, Find Next, Replace

CAMELot Functionality Overview

- The Configure menu is made available after a successful compilation. It allows the user to modify the following parameters:

1. The Dimensions of the CA (*x*-Length, *y*-Height, *z*-Width);
2. The number of Processes to handle the task;
3. The number of Folds to which the CA is divided in the Length axis.
4. The C compiler pathname and flags;
5. The MPI run command;
6. The Timing output.



CAMELot Functionality Overview

UNIVERSITÀ DELLA CALABRIA

- It is worth noting that:

1. For best performance the Length of the CA must be an exact multiple of the product of the number of Processes with the number of Folds;
2. A 1-D CA has only the *x* axis available and a 2-D CA has only the *x* and *y* axes available.

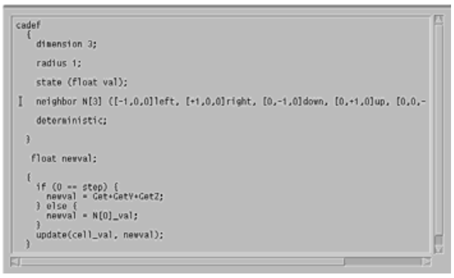
Controlling XDR Output

- Starting from release 1.2 of CAMELot, XDR is used for the file I/O, this allows CAMELot data files to be portable between different machine architectures. The user can control the use of XDR through the use of the C compiler command line option of the Configure menu

The Editor of the Development Window

UNIVERSITÀ DELLA CALABRIA

- The user may Open a file and use the Editor to view and modify it.



The Development Window

UNIVERSITÀ DELLA CALABRIA

- The available buttons are:

Compile
Build
Run

- (*Compile*) This button compiles the current program in the Editor. This compilation checks for CARPET syntactic errors and generates the C source and header files for the specified CA model. The compiler handles both C (/* */) and C++ (//) style comments. A failed compilation is accompanied by a beep; a window is popped up containing the error messages and the cursor in the Editor is positioned at the first line reported to contain an error. If there is a beep but no error message is displayed then the automatic Save has failed.

The Development Window

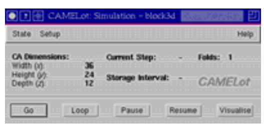
UNIVERSITÀ DELLA CALABRIA

- (*Build*) This button compiles and links the CA Engine code with the generated C source and header files for the CA. It invokes the C compiler specified in the Configure menu and redirects its output to the pop-up window generated.
- (*Run*) This button spawns the CA Engine processes specified in the Configuration menu using the MPI run command as it appears in the respective option of the same menu. It also spawns the Simulation Window discussed next and makes the Build and Run buttons unavailable.

Simulation Window

UNIVERSITÀ DELLA CALABRIA

- This allows the whole state or specific substates to be initialised or saved. The Close option closes the Simulation Window as well as all the Visualisation Windows and terminates the execution of the CA Engine.
- A Substate can be saved in a binary file using the State-Save-Substate sequence of options. In order for the file to be subsequently detected as a substate file, it must be saved with the extension .cmt. Saving the Configuration involves saving status-specific data in a file with the extension .cpj, as well as all the substates in files with filenames constructed as follows: if the Configuration filename is cfn.cpj the substates are saved in filenames named cfn000.cmt, cfn001.cmt, etc.



Configuration File *.cpj

- The data contained in a configuration file are:

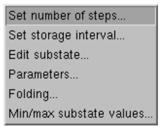
1. The number of Dimensions;
2. The per-dimension Sizes;
3. The current Generation of the CA Engine;
4. The number of States;
5. The number of Folds;
6. The number of Global Parameters;
7. The values of the Global Parameters.

- Information stored in configuration or substate files can be loaded into the CA Engine using the State-Initialise options.

Setup Menu

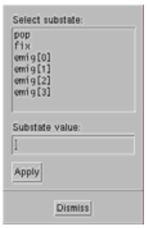
The Setup Menu allows the following to be adjusted:

1. Steps to run the Engine;
2. Storage interval;
3. Substate editing (one cell);
4. Parameter editing;
5. Active Fold setting;
6. Manual setting of the per-substate minimum and maximum values for colour mapping.



State and Parameter Editing

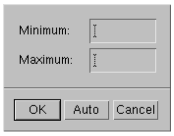
- *State Setting*: Allows the user to view and set the values of the substates of one cell manually. The possible substate names are made available through a menu.
- *Parameter Setting:* Allows the adjustment of a global parameter. Parameters can be adjusted using the names they have in the program. The possible names are made available through a menu similar to the one shown in the figure



Color Range settings

- *Manual setting of the per-substate minimum and maximum values for colour mapping:*

This option allows the user to override the automatic per-substate minimum and maximum calculation executed as part of the colour mapping strategy. Specifying the minimum and maximum enables visualising parts of the data with greater detail. This does not affect the evolution of the model, although it speeds up the visualisation process. The system reverts to the automatic mechanism if the users clicks on the Auto button of the menu.



Display Screen

- Displays the current values of the following:
- The Dimensions;
- The Current Step of the CA Engine;
- The Periodic Storage Interval;
- The number of Folds. ***N.B.:*** "Folds: 1" indicates that no partitioning into multiple folds was done at compile time; i.e.. the CA is considered to consist of one single fold.

Button Bar

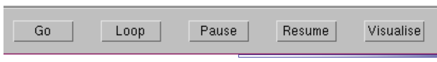
Go: Starts the CA Engine until the generation counter reaches the number of iterations specified in the corresponding Setup menu option.

Loop: Same as Go except that it starts an infinite (INT_MAX iterations) CA evolution.

Pause: Temporarily suspends CA Engine execution. This can be restarted with any of three buttons.

1. Go will restart the Engine until it reaches the specified number of iterations;
2. Loop will restart the Engine for infinite iterations;
3. Resume will continue the operation of the Engine from the step where it stopped. It will Loop if Loop was selected before Pause was pressed, or continue until the specified (possibly revised) finishing point is reached otherwise.

Visualise: Allows the initialisation of a Visualisation window. The user is prompted to set the visualisation period and select the substate to be visualised.



Visualization Modes

- The possible types of visualisation depend on the number of dimensions of the model:
- *1-D Models:* The visualisation is drawn in horizontal lines from left to right. The vertical dimension of the window corresponds to time. The user can therefore see how the model changes with time. When the vertical dimension of the screen is exhausted, the visualisation restarts from the first line overwriting the first visualisation.
- *2-D Models:* They are represented in an orthogonal manner, *x* running horizontally and *y* running vertically, the origin being the bottom left corner of the window.
- *3-D Models: x-y, x-z* or *y-z* planes of a 3-D model can be displayed either as orthographic (as above) or isometric projections. The coordinate of the plane (i.e.. *z* value for an *x-y* plane, *y* value for a *x-z* plane etc) is specified by the user via a dialog box with scale widgets.

UNIVERSITÀ DELLA CALABRIA

The CARPET programming language

- CARPET is a programming language for the definition of cellular automata-based models and their transition functions, designed as an extension to ANSI C. A CARPET program consists of the following sections: a global declaration section, known as the *cadef* (CA DEFinition) section; a transition function; and an optional *steering* function.
- The user has only to specify the transition function of one generic cell

UNIVERSITÀ DELLA CALABRIA

Steering

- The steering function is an optional feature of a CARPET program by which the user can affect the flow of the program as a result of global reductions on regions of the model
- The steering function is defined in a separate section of the CARPET program, similarly to the update function. *The main difference is that the update function is applied separately in each cell, whereas the steering function is global for the model.* Any code inside the steering statement is copied verbatim to the generated file, with the exception of the `region_<op>()` statements which are translated to a global reduction function.

UNIVERSITÀ DELLA CALABRIA

Modifying the program flow

- The user can modify the flow of the program inside the steering section in either of the following two ways:

1. call the function cpt_set_param (float *old_p, float new_p), which sets the global parameter pointed by old_p to the value of new_p;
2. call the function cpt_abort(), which terminates the execution of the program without exiting the CA Engine.

- Inside the steering code, the user has access *only* to the following CARPET defined variables:

1. `DimX, DimY, DimZ;`
2. `step;`
3. global parameter values.

UNIVERSITÀ DELLA CALABRIA

Region definition

Syntax

```
region <region-name> (<min_x>:<max_x>,<min_y>:<max_y>,
                      <min_z>:<max_z>);
```

Remarks

The user specifies a region as part of the cadef block of the program, using a declaration of the above form. This is used to allow global reduction operations within the steering block of the CARPET program.

UNIVERSITÀ DELLA CALABRIA

Global reduction

Syntax

```
region_<op> (<region-name>, <state>);
```

Remarks

The region_<op>() function is available inside the steering function. It returns a value of the same type as its state argument. It applies the reduction operation op to state state all the cells in region region-name. The supported operations are as follows:

1. `max`
2. `min`
3. `sum`
4. `prod`
5. `land (logical and)`
6. `band (binary and)`
7. `lor (logical or)`
8. `bor (binary or)`
9. `lxor (logical exclusive or)`
10. `bxor (binary exclusive or)`

UNIVERSITÀ DELLA CALABRIA

Steering

Example

```
cadef {
  dimension 3;
  region Inside (start+2:end-2,:,:);
  ...
  state (float val);
  parameter (pi 3.141);
  }
...
  steering {
     float min = region_min (Inside, val);
     if (min < 4.0) {
           cpt_set_param (&pi, 3.14159);
     } else if (min > 100.0) {
           cpt_save (“aborted”);
           cpt_abort ();
           }
  }   // steering
```

General Layout

```
cadef
{
      declarations
}
[transition function local variable
declarations and subroutine prototypes]
{
      transition function code
}
[transition function subroutines]

[
steering
{
      steering function code
}
]
```

where items in [...] brackets are optional

Transition Function

- The transition function (and its subroutine functions, if any) may contain the following CARPET statements, in addition to C code :

```
cell_substate
DimX, DimY, DimZ
GetX, GetY, GetZ
NFolds
NProcs
random()
randomise()
srandom()
step
update()
```

parameter references

CA Definition - Example

```
cadef
{
    declaration;
    declaration;
        ...
    declaration;
}
```

where *declaration* can be:

```
deterministic       // Deterministic CA!
dimension           // AC Dimension
neighbour           // neighborhood
parameter           // parameter list
radius              // neighborhood radius
region              // …steering
state               // states (char, int, float,
                    // array…)

Threshold            // …steering
```

CA Definition - Example

```
cadef
{
      dimension 3;
      radius 1;
      region Inside (start+1:end-2, :,:);
      state (float val; int val2);
      neighbour N[6] ([-1,0,0]left,[1,0,0]right,
       [0,-1,0]down,[0,1,0]up,[0,0,-1]in, [0,0,1]out);
      parameter (pi 3.14159);
      deterministic;
      threshold (cell_val == 3);
}
```

Transition Function

`cell_substate` instruction

```
Es:

cadef
{
 state (float temp);
}

float val;
val = cell_temp+3;
```

Transition Function

```
update(cell_substate, value)
```

This is the only way to set the value of a cell substate by means of the program. This is done in order to ensure that the state of all cells is set in lock step in the next generation *after* the update has been issued

```
Es:

val=10;
update(cell_life, val);
```

Conway's Game of Life (1970)

The Rules

- The Game of Life was invented by John Conway. The game is played on a field of cells, each of which has eight neighbors (adjacent cells). A cell is either occupied (by an organism) or not. The rules for deriving a generation from the previous one are these:
- **Death:** If an occupied cell has 0, 1, 4, 5, 6, 7, or 8 occupied neighbors, the organism dies (0, 1: of loneliness; 4 thru 8: of overcrowding).
- **Survival:** If an occupied cell has two or three neighbors, the organism survives to the next generation.
- **Birth:** If an unoccupied cell has three occupied neighbors, it becomes occupied.

Game of Life – CA Definition

```
/* Conway's Game of Life implementation
 *
 *
 */
#define alive 1
#define dead 0
cadef
  {
    dimension 2;
    radius 1;
    state ( int life );
    neighbor Moore[] ( N[ 0,-1], NW[-1,-1], W[-1, 0],
       SW[-1,1],S[ 0, 1], SE[ 1, 1], E[ 1, 0], NE[ 1, -1]  );
}
```

Game of Life – Transition Function

```
int i;
int sum;
{
  sum = 0;
  for( i=0 ; i<8; i++)
      sum = Moore[i]_life + sum;
  if ( sum == 3  || ( sum == 2 && cell_life == 1) )
      update (cell_life, alive);
  else
      update (cell_life, dead);
}
```

Initial Configurations

- Once that the program is compiled, an initial configuration can be executed in the following manners:

1. Explicit setting pf parameters and substates
2. States loaded from previously generated binary files
3. Load an initial configuration (parameter values and substates), generated by a typical C program

Example of Configurator generator for Conway's Game of Life

- The program has to generrate two types of files, a *.cpj and so many *.cmt for each CA state (Es. Life (1 state!!!), thus a file `life.cpj` and a file `life.cmt`)
- `life.cpj` has to contain 7+1 `int` type data in the **following** order:

1. AC Dimensions
2. Dim x
3. Dim y
4. Dim z
5. AC Generation (step)
6. Number of states
7. Number of folds (1)
8. (Number of global parameters + parameters list)

life1.cpj

UNIVERSITÀ DELLA CALABRIA

```
…
#define dimx 100
#define dimy 100
#define dimz 1
…
FILE *fid;
dimensions=2;
generation=0;
n_sottostati=1;
n_fold=1;
n_parametri=0;
…
fid=fopen("life1.cpj","wb");
fwrite(&dimensions,sizeof(int),1,fid);
fwrite(&d_dimx,sizeof(int),1,fid);
fwrite(&d_dimy,sizeof(int),1,fid);
fwrite(&d_dimz,sizeof(int),1,fid);
fwrite(&generation,sizeof(int),1,fid);
fwrite(&n_sottostati,sizeof(int),1,fid);
fwrite(&n_fold,sizeof(int),1,fid);
fwrite(&n_parametri,sizeof(int),1,fid);

fclose(fid);
```

life1.cmt

UNIVERSITÀ DELLA CALABRIA

```
…
int life[dimx][dimy];
…
FILE *fid;
…
for(i=0;i<dimx;i++)
   for(j=0;j<dimy;j++)
      life[i][j]=rand()%2  // inizializzazione random
…
fid=fopen("life1.cmt","wb");
for(j=0;j<dimy;j++){
  for(i=0;i<dimx;i++)
     fwrite(&life[i][j],sizeof(int),1,fid);  //stato di tipo int
  }
fclose(fid);
```