Caratteristiche di Sistemi Paralleli per AC

- Le principali caratteristiche di un ambiente software parallelo dovrebbero essere:
  - Un **layer di programmazione** ad alto livello per il design di modello computazionali, indipendente dall'architettura parallela sottostante (e.g. CARPET)
  - **Un'interfaccia GUI** (Graphical User Interface) che permette la completa visualizzazione dell'evoluzione del fenomeno ed il display dei valori numerici associati alla simulazione
  - Strumenti di ***tuning*** e controllo che permettono lo ***steering*** (controllo globale) dell'evoluzione del fenomeno
  - **Scalabilità** per permettere una esecuzione efficiente del fenomeno su calcolatori paralleli (non ottenibile su calcolatori sequenziali!)

Ambienti per Automi Cellulari (AC)

- Parecchi ambienti ad AC sono stati implementati su calcolatori sequenziali (PC, workstations)
- Due approcci possono essere individuati: via Hardware e via Software
- Il CAM (Cellular Automata Machine) rappresenta il più famoso esempio di architettura hardware dedicato per lo studio di AC (efficiente ma pochi stati...)
- Ambienti Software: P-CAM, PECANS, StarLogo, CAPE, CAMEL, CAMELot

L’Ambiente CAMELOT

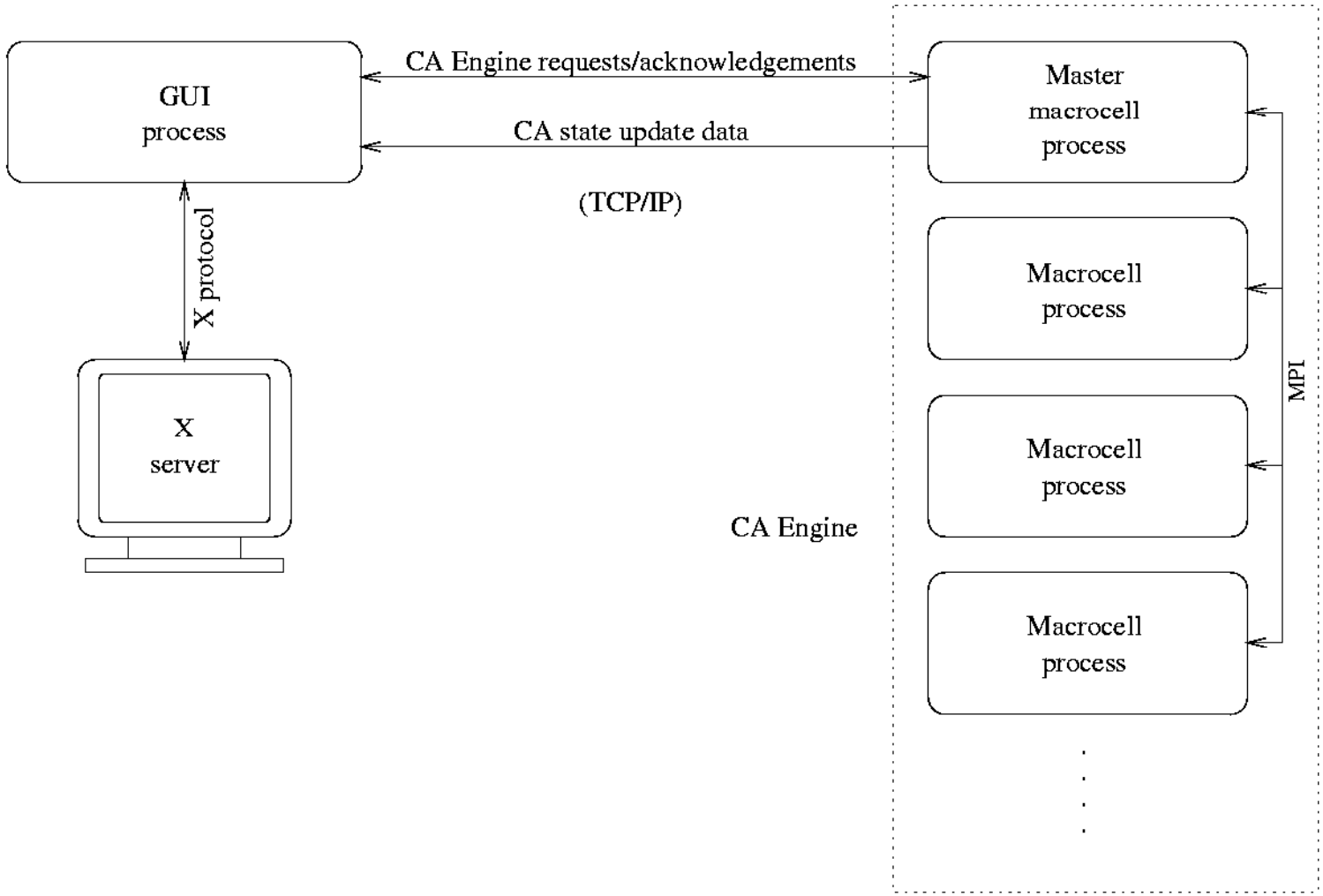
L’ambiente Camelot è un ambiente **software** per la programmazione ed esecuzione di Automi Cellulari

Camelot è composto da una interfaccia grafica, un linguaggio di supporto (CARPET) e di una finestra per il controllo della esecuzione dell’AC

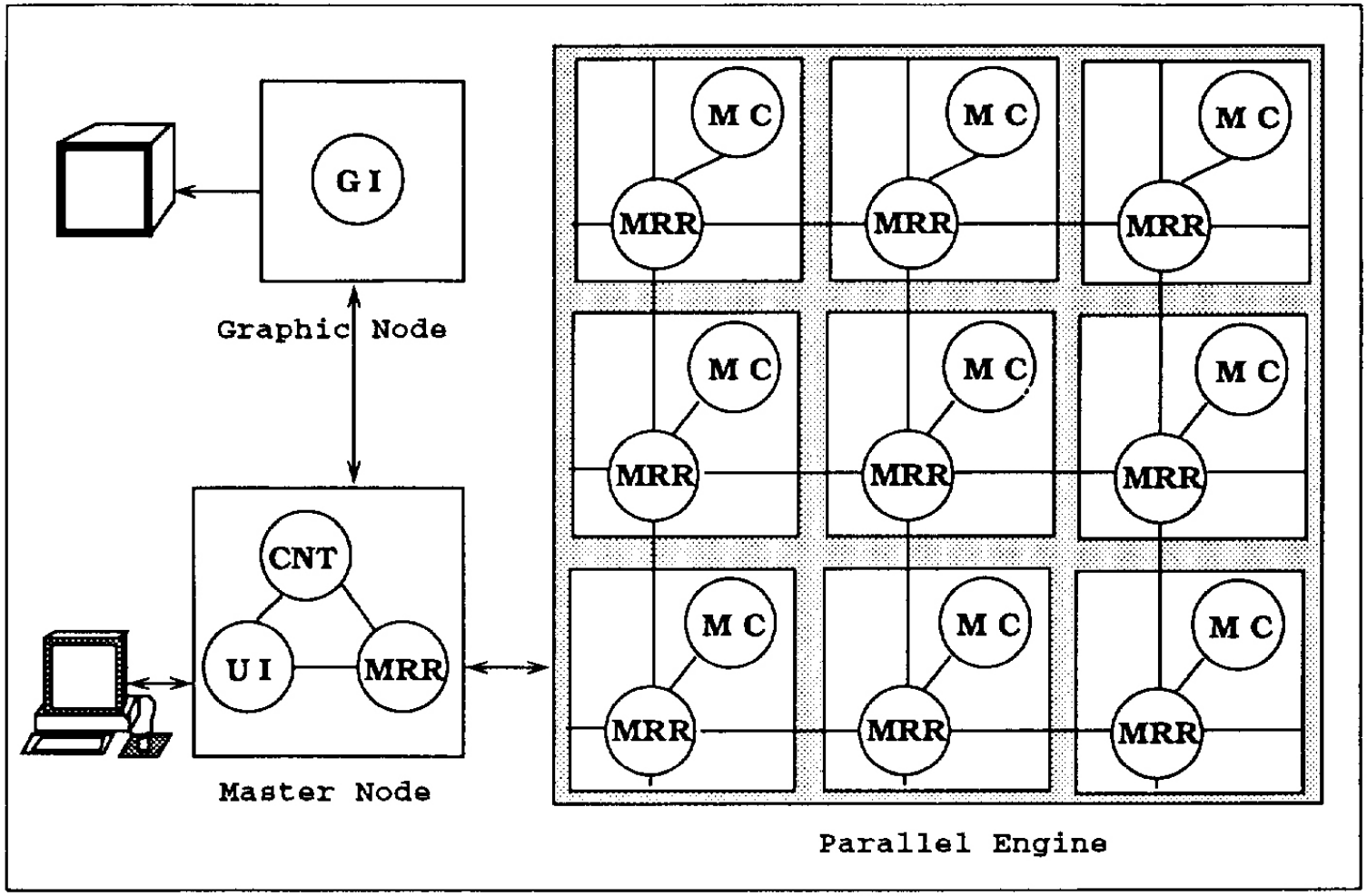
Il parallelismo risulta invisibile all’utente

L’utente deve solamente specificare la funzione di transizione di una singola cella tramite un linguaggio ad alto livello (e.g. C, CARPET)

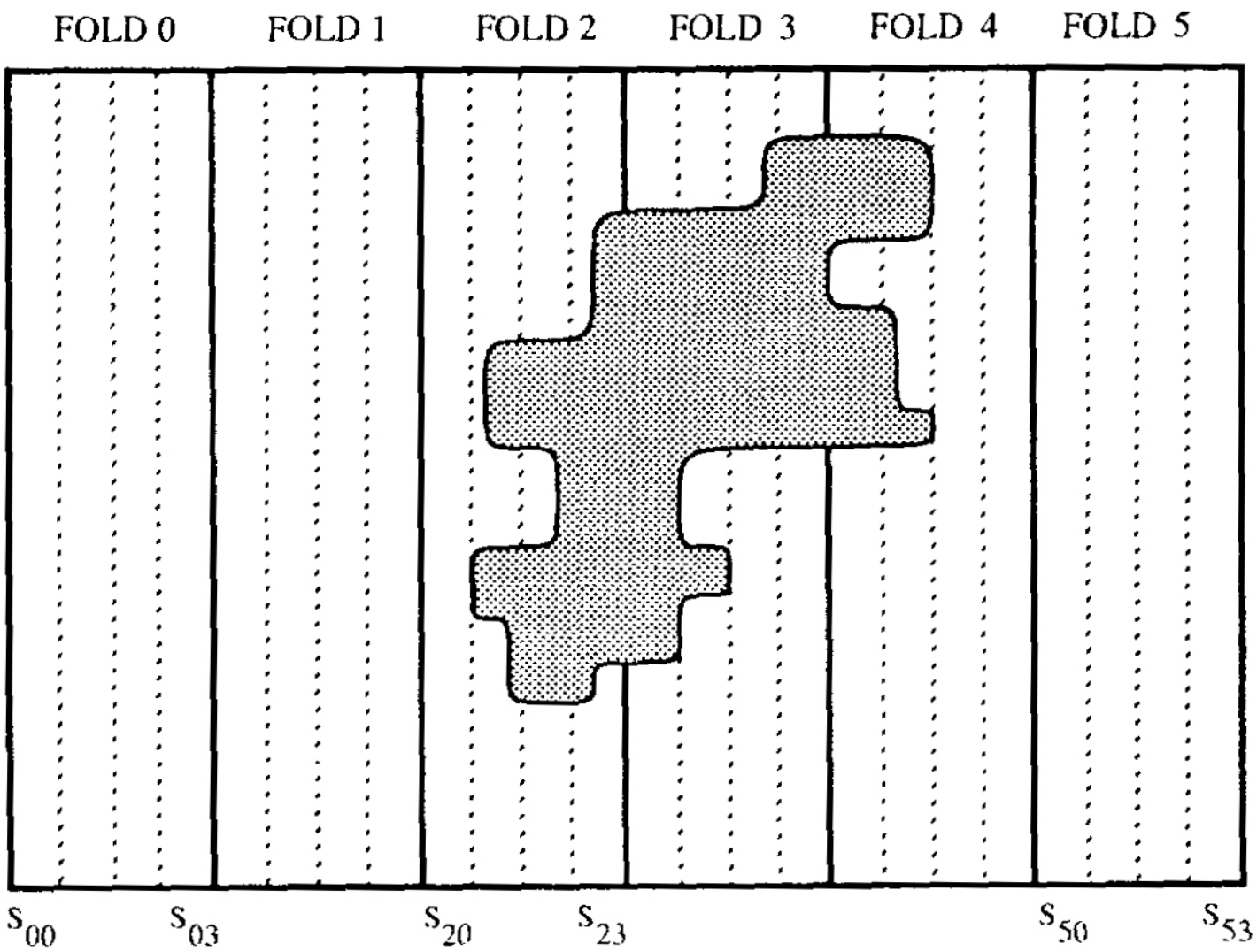
CAMEL – CAMELot - Architecture



Camelot Architecture

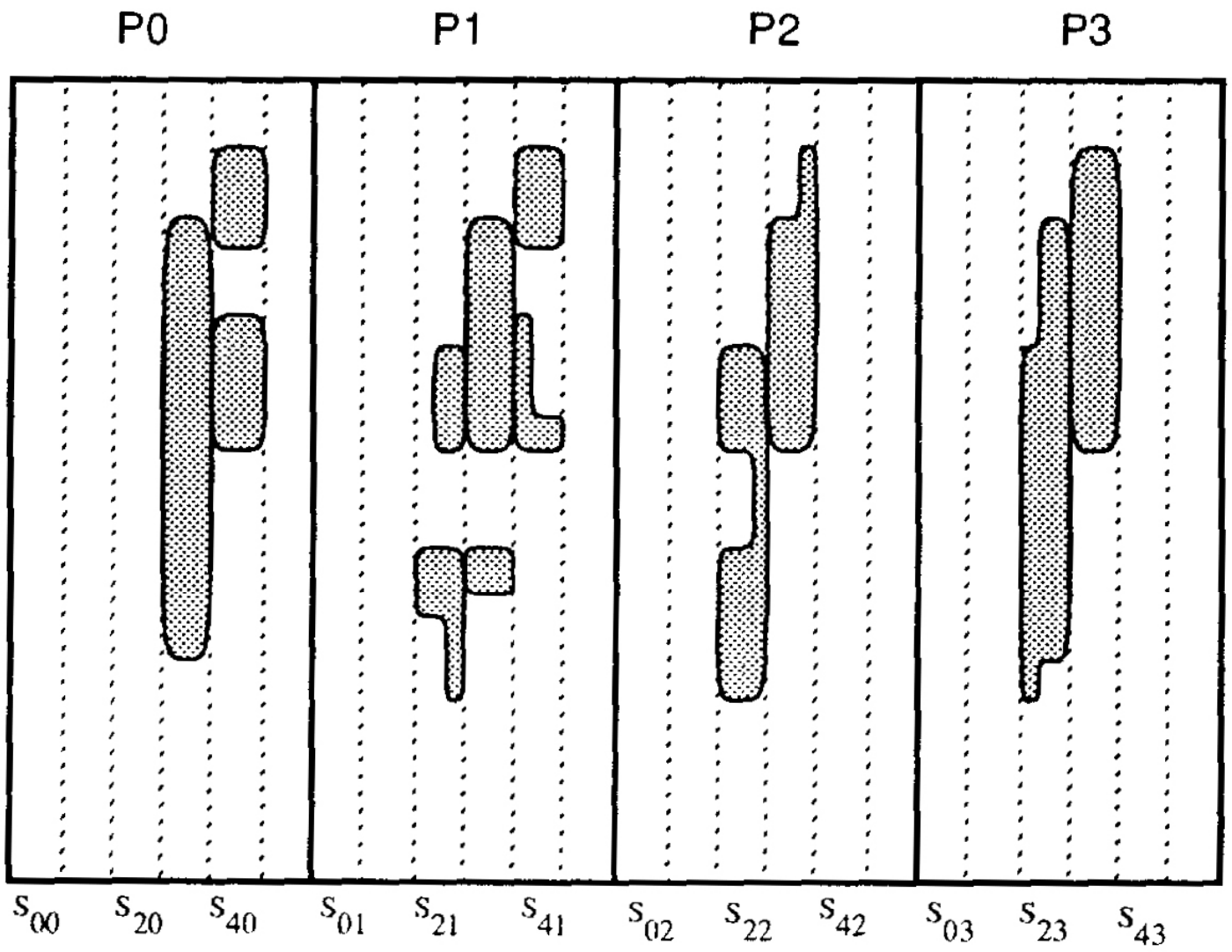


Load Balancing (*Folding - Stripwise-scatter*)

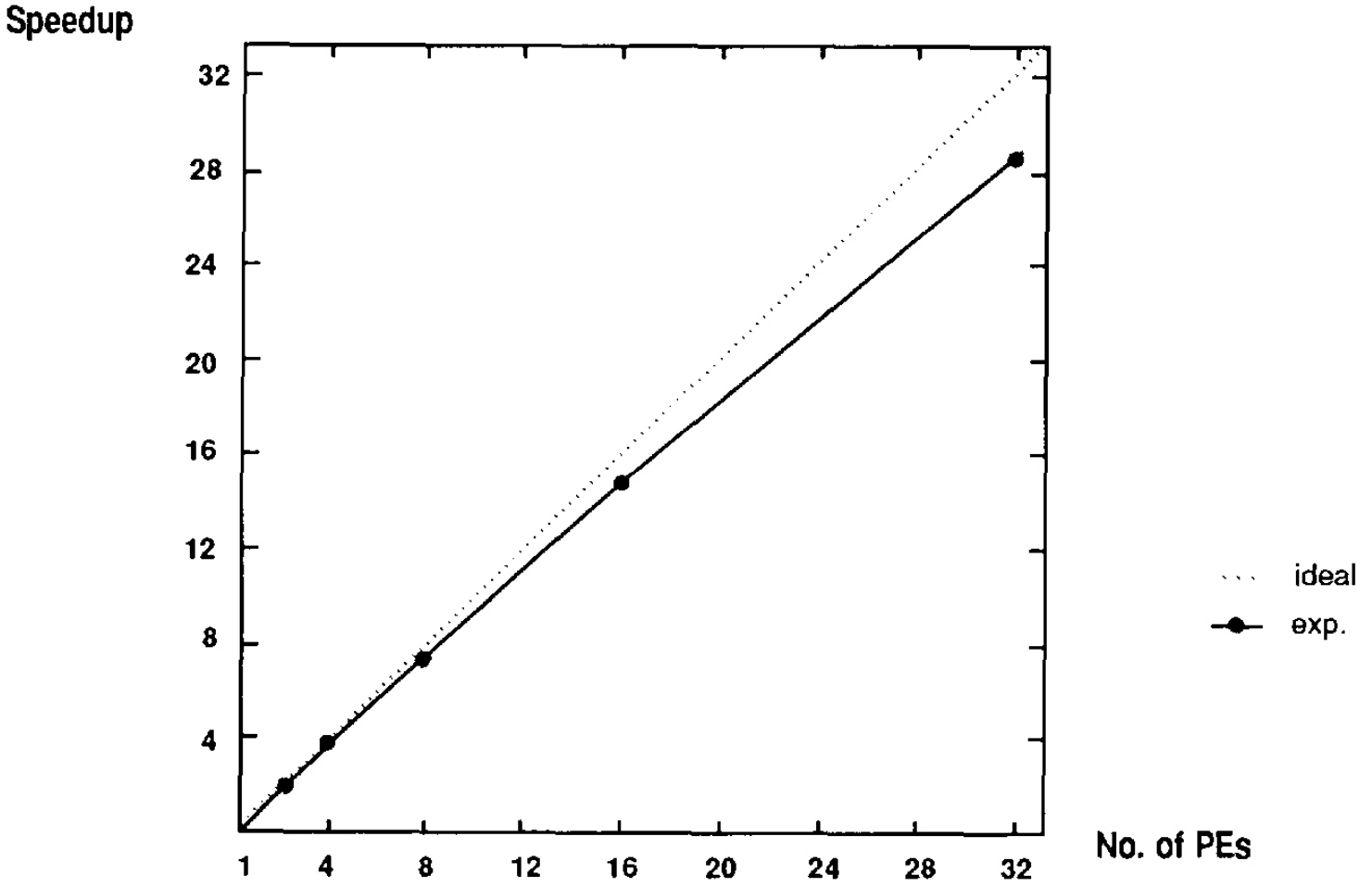


Fold number =6 ; PE = 4

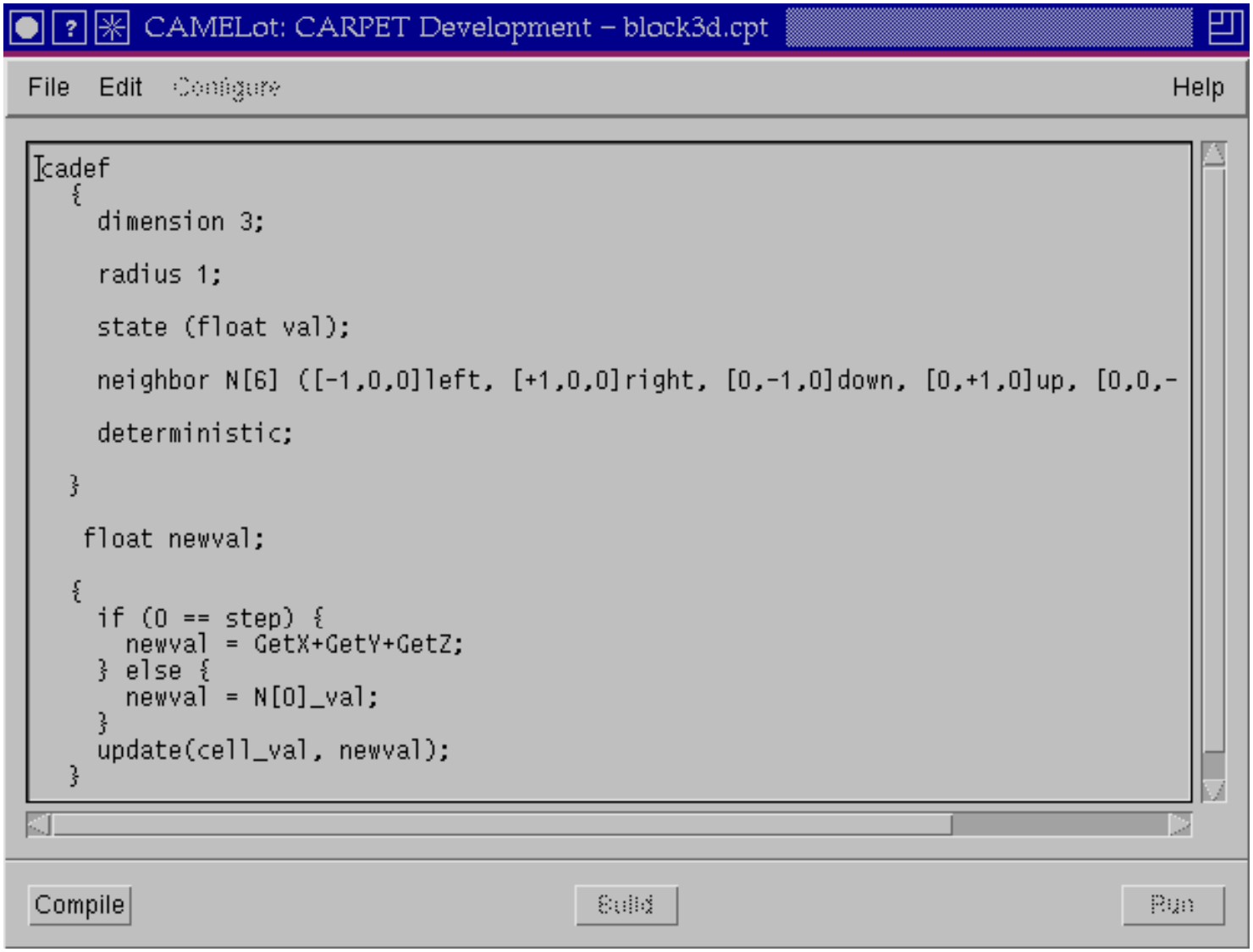
Load Balancing (*Folding - Stripwise-scatter*)



Performance di CAMEL - Transputers

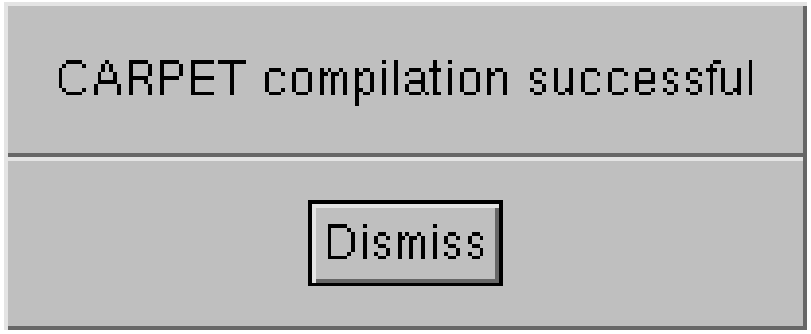


Finestra principale - Editor



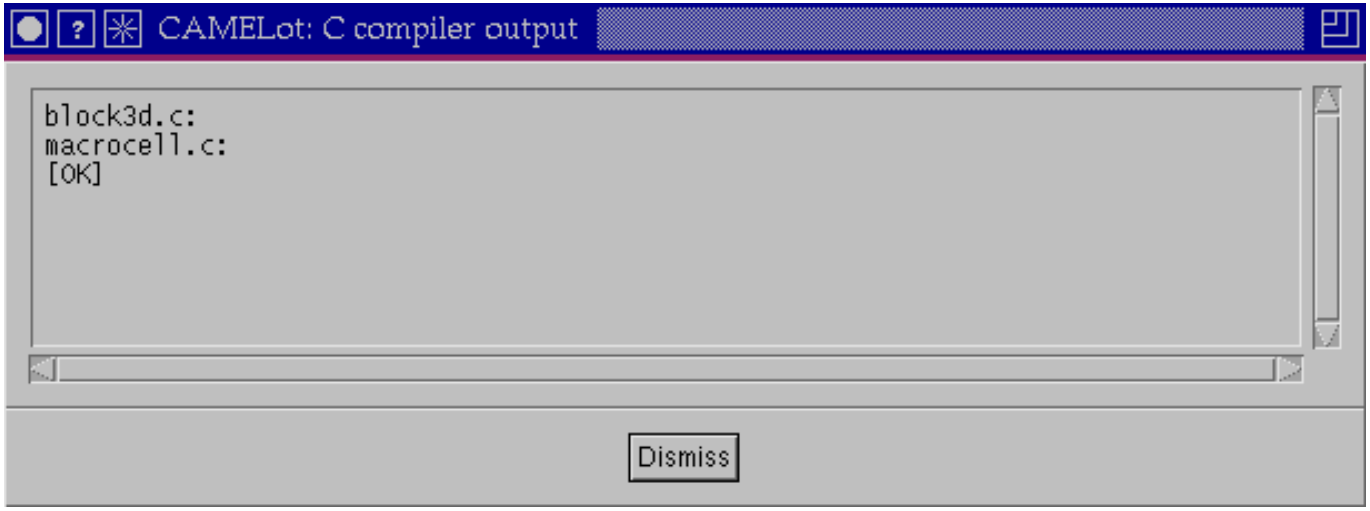
Compilazione

- Dopo aver scritto il programma in **Carpet** (*.cpt) bisogna compilarlo. **Camelot** genera un file *.c per essere successivamente linkato con varie libreria (MPI, etc) per ottenere un eseguibile.



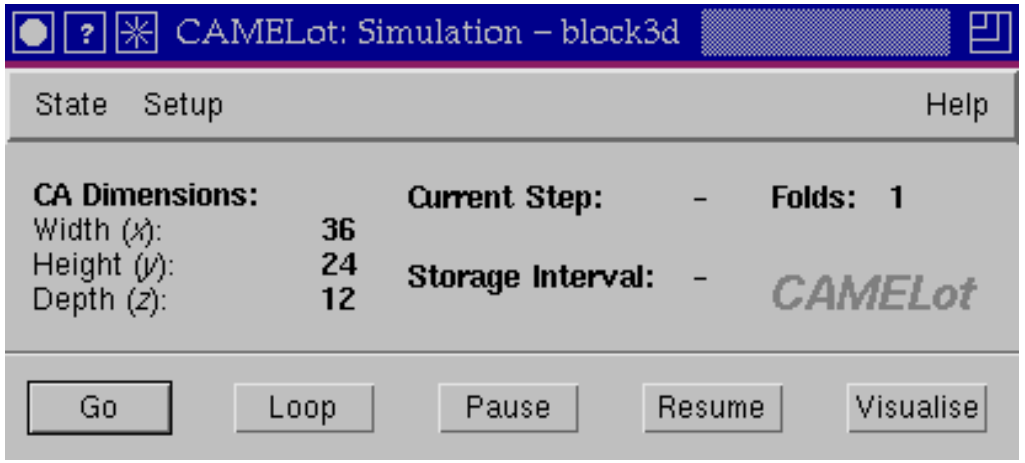
Build

- Per genera l'eseguibile Linux (Unix) bisogna specificare le **dimensioni** dell'Automa Cellulare (es. 100x100), i **numeri di processori** (1!) ed il **numero di Folds** (1!)



Esecuzione

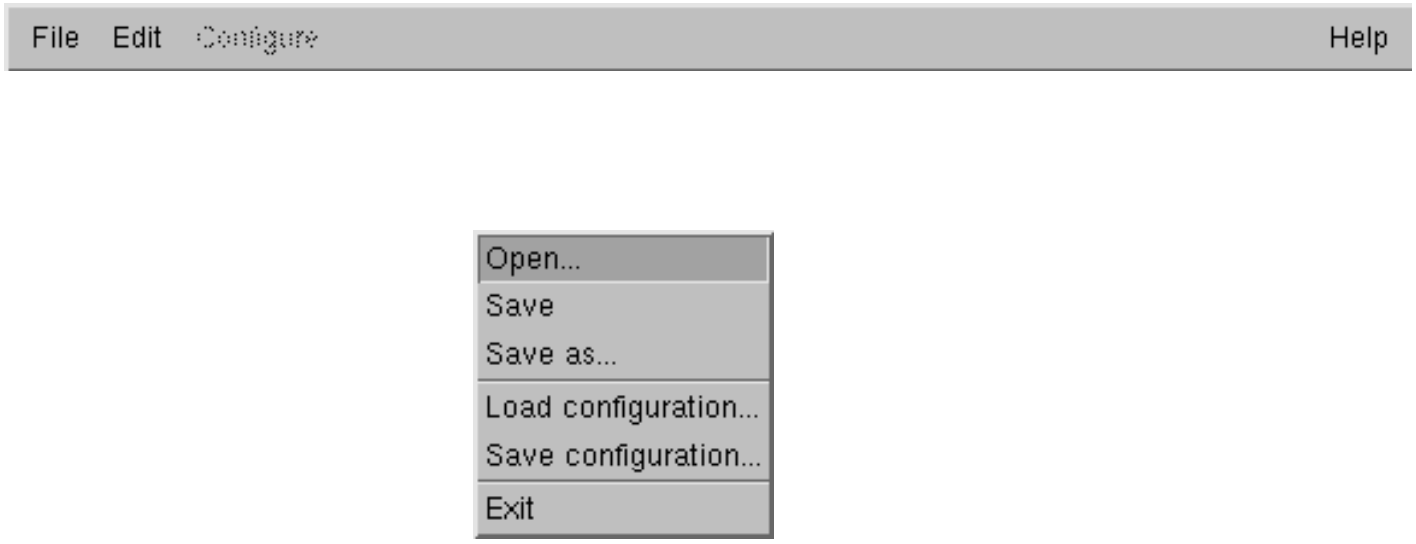
- Se tutto è andato a buon fine, si fa girare l'eseguibile con RUN
- GO e LOOP eseguono l'AC per un certo numero di passi; PAUSE per pausa; RESUME per riprendere
- VISUALISE visualizza uno stato dell'AC



L’ambiente di sviluppo

Menu FILE

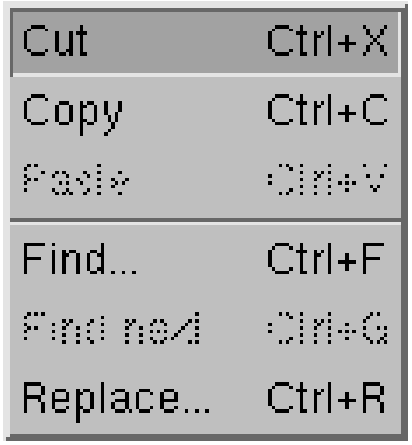
Aprire, salvare, salva come, carica configurazione, salva configurazione, esci



L’ambiente di sviluppo

Menu EDIT

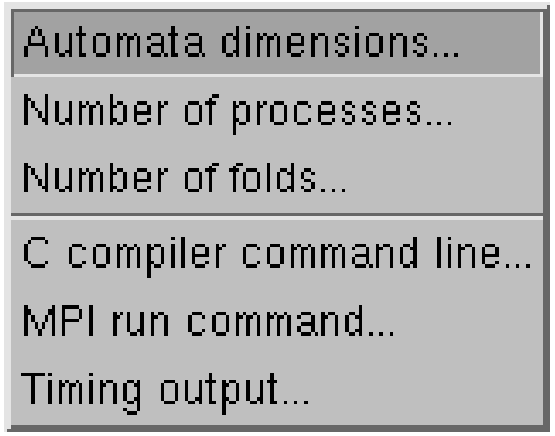
Cut, Copy, Paste, Find, Find Next, Replace



L’ambiente di sviluppo

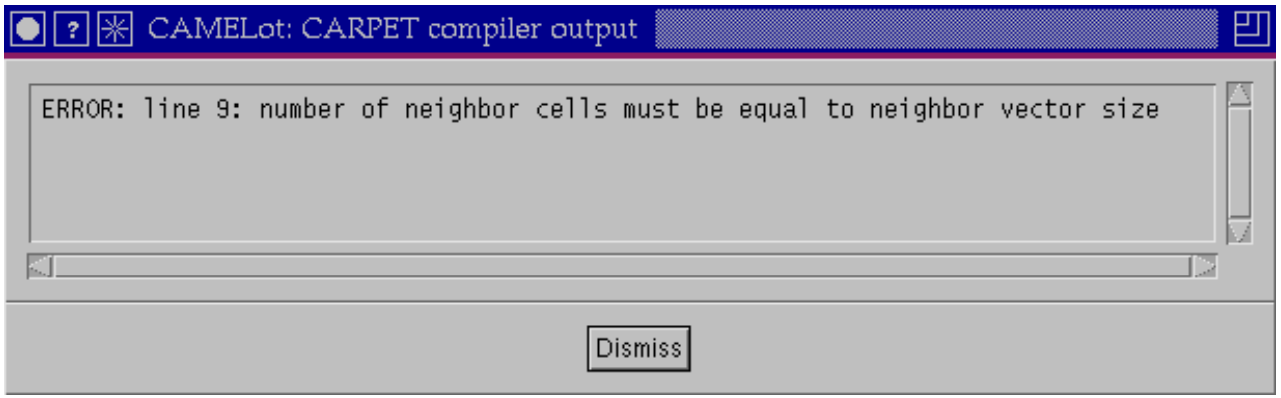
Menu CONFIGURE

Serve per “configurare” l’AC



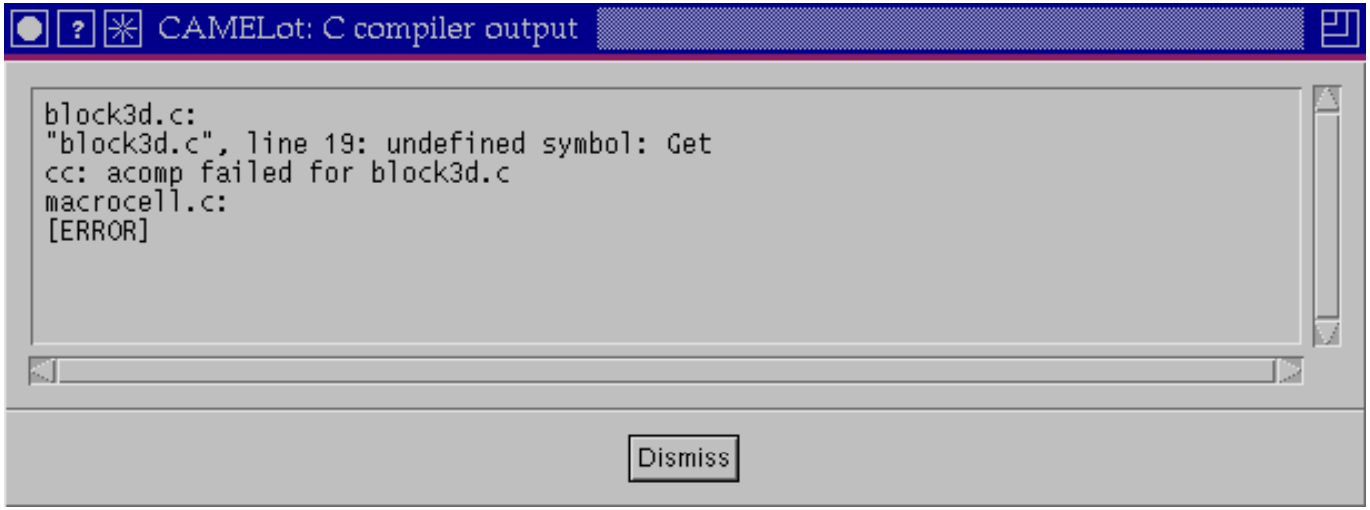
Errori di compilazione

- Vengono individuati se l'utente ha commesso errori di sintassi del programma CARPET



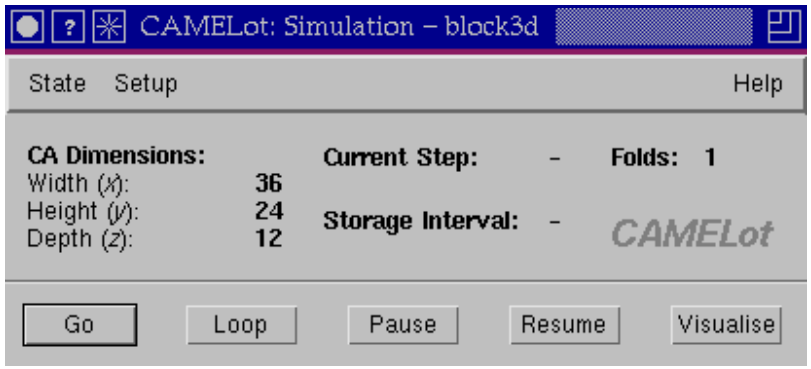
Errori di BUILD

- Il Build compila e fa il link del programma *.c (generato dalla compilazione) con il codice vero e proprio del motore CA



Finestra di Simulazione

- Facendo il Run viene visualizzata la finestra di simulazione



- Esso è composto da un menu, una parte descrittiva dell'AC e da vari bottoni.

Finestra di Simulazione

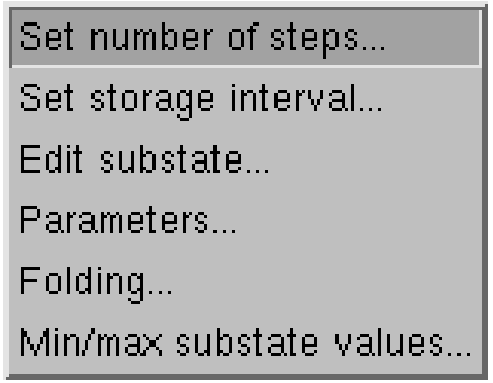
- Menu STATE: permette di caricare o salvare, singoli sottostati o configurazioni dell'AC in esame.
- Gli stati hanno estensione *.cmt (binari!)
- Le configurazioni hanno estensione *.cpj
- Una tipica configurazione contiene informazioni quali, numero di dimensioni, numero di generazioni, numero di stati, etc

Finestra di Simulazione

Menu SETUP

Tra l'altro, permette di settare:

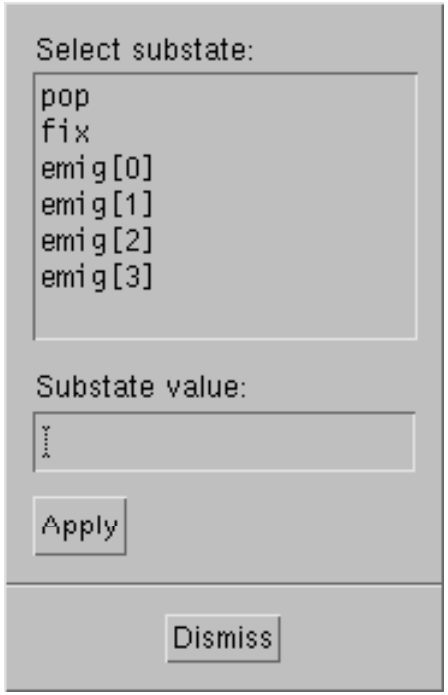
Il numero di passi da eseguire, numero di passi di salvataggio, editare uno stato, etc.



Finestra di Simulazione

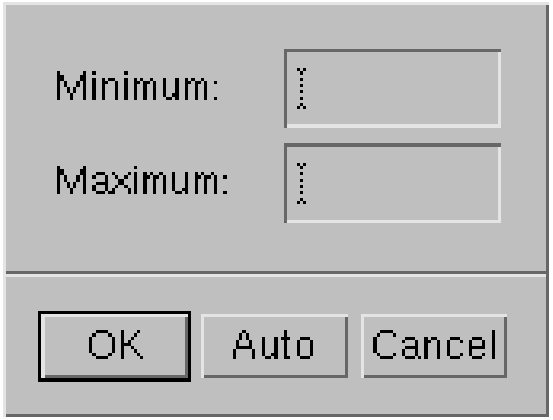
- Editare uno stato

E’ possibile editare (modificare) il valore dei singoli stati che compongono un’AC



Range di Colori

Questa opzione permettere di settare manualmente il range di visualizzazione dei colori per ogni sottostato, dal colore blu per il valore minimo al rosso per il valore massimo

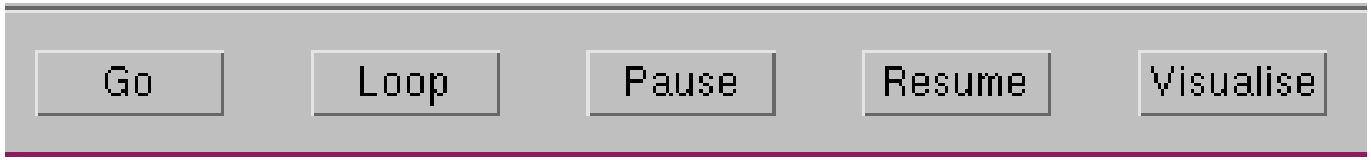


Finestra di Esecuzione

- In questa finestra vengono visualizzati le dimensioni dell'AC, il passo corrente, il passo di salvataggio dati e i folds (bilanciamento del carico)

Finestra di Esecuzione

- GO: VAI! ...fino al passo specificato nel menu Setup
- LOOP: VAI! ...per sempre (!)
- PAUSE: Pausa!
- RESUME: Ripristino!
- VISUALISE: Visualizza AC (1D, 2D, 3D)



Visualizzazione

- 1D CA: La visualizzazione contiene linee orizzontali che corrispondono all'AC. In orizzontale viene visualizzato il "trascorrere del tempo"
- 2D CA: Visualizzazione ortogonale semplice
- 3D CA: Possibilità di visualizzare gli stati in modo ortografico, oppure isometrico (l'utente fissa il piano x,y o z)

Linguaggio CARPET

- CARPET e' un linguaggio per la definizione di modelli AC e le loro funzioni di transizioni. E' una estensione di ANSI C.
- L'utente deve specificare la funzione di transizione di **una** cella generica
- Le parti che compongono un programma Carpet sono:

1. Sezione dichiarazione AC
2. Funzione di Transizione
3. (opzionale) Funzione di ottimizzazione Steering)

Layout generale

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

La funzione di transizione

- E’ composto da istruzioni C (tranquilli, NON C++ !!!!)
- In aggiunta, le seguenti istruzioni CA:

  - `cell_`*`substate`*
  - `DimX, DimY, DimZ`
  - `GetX, GetY, GetZ`
  - `NFolds`
  - `NProcs`
  - `random()`
  - `randomise()`
  - `srandom()`
  - `step`
  - `update()`
  - dichiarazione parameter

Definizione di AC - Esempio

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

dove *declaration* può essere:

```
deterministic  //(AC “deterministici”)

dimension  //(numero di dimensioni dell’AC

neighbour  // vicinato

parameter  // lista parametri globali

radius     // raggio vicinato

region     // serve nell’ottimizzazione

state      // stati (char, int, float, double,
                  (array di…)

threshold // // serve nell’ottimizzazione
```

Definizione di AC - Esempio

```
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);
}
```

Funzione di transizione

Si scrive in C, ma per *accedere* ai valori degli stati (sottostati) dell'AC si usa l'struzione `cell_sottostato`

Es:

```
cadef
{
 state (float temp);
}

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

Funzione di transizione

- Per *aggiornare* il valore di un sottostato si usa:

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

Es:

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

Gioco della Vita di Conway (1970)

Le regole

- Il Gioco della Vita è stato inventato da John Conway. Brevemente, il gioco è basato su una “griglia” di celle, ognuna delle quali ha un vicinato di otto celle adiacenti. Una cella è o **occupato** da un organismo o meno. Le regole per la generazione della generazione successiva, basata sulla precedente, sono:
- **Morte:** Se una cella viva ha 0, 1, 4, 5, 6, 7, or 8 vicine vive, l’organismo muore (0, 1: per solitudine; da 4 a 8: per sovrappopolazione).
- **Sopravvivenza:** Se una cella viva ha 2 o 3 celle vive nel vicinato, l’organismo sopravvive alla prossima generazione.
- **Nascita:** Se una cella morta ha esattamente 3 celle vive nell’intorno, esso diventa viva.

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);
}
```

Configurazioni iniziali

- Una volta compilato il programma, si può eseguire una configurazione iniziale dell'AC in due modi:

1. Tramite settaggio esplicito dei parametri e dei sottostati (edit substate)
2. Stati caricati da file binari precedentemente creati
3. Caricare una configurazione iniziale (valori iniziali di parametri + sottostati), generato tramite un programma C.

Esempio di generatore di configurazione (Life)

- Il programma deve generare due tipi di file, un *.cpj e tanti *.cmt per quanti sono gli stati dell'AC (Es. Life (1 stato!!!): un file `life1.cpj` e un file `life1.cmt`)
- Il file `life1.cpj` deve contenere 7+1 dati di tipo `int`, e cioè, in ordine:

1. Dimensioni AC
2. Dim x
3. Dim y
4. Dim z
5. Generazione (passo) AC
6. Numeri sottostati
7. Numero folds (1)
8. (Numero parametri globali + lista parametri)

Esempio life1.cpj

```
…
#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);
```

Esempio life1.cmt

```
…
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);
```