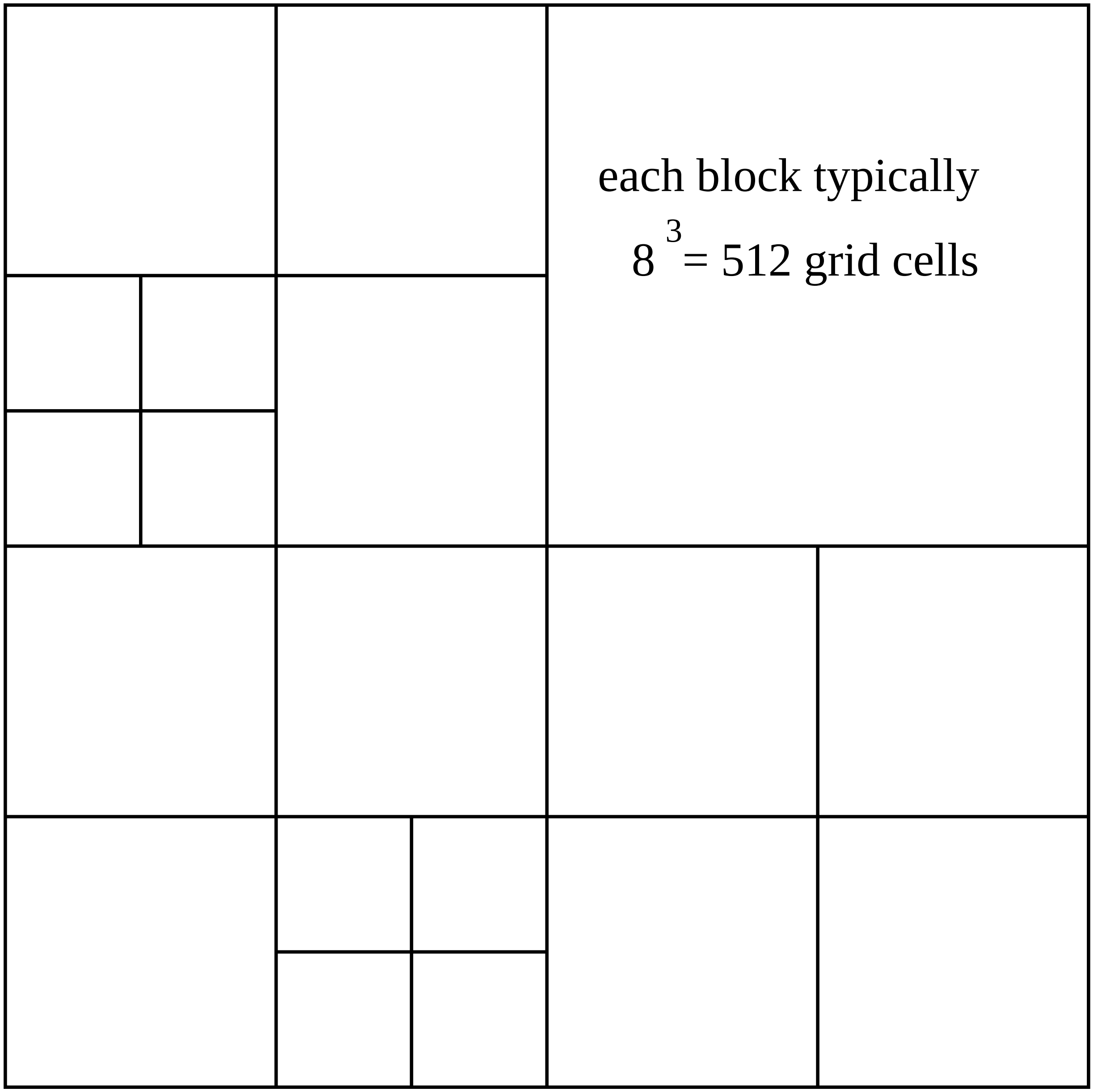
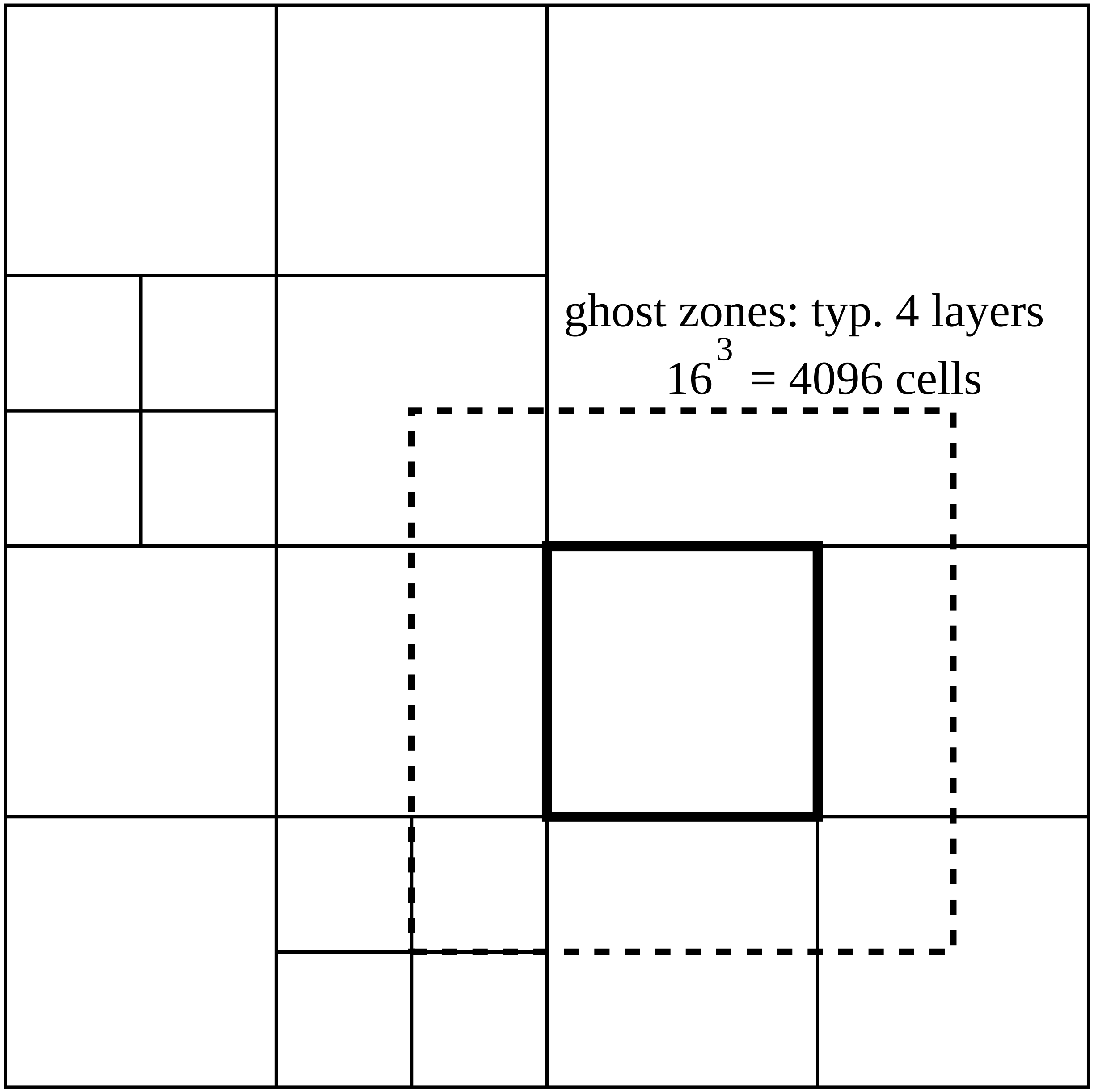
The Flash code

- framework of many different modules
  - *grid-types (AMR, uniform)*
  - *Rieman solvers (PPM, van Leer, . . .)*
  - *time-integrators (euler1, rk3, strang)*
  - *physical modules (self-gravity, MHD, radiation, turbulence stirring, nuclear burning, . . .)*
  - *particle modules (N-body, SPH)*
  - *multiple fluids*
- modules quite independent, well defined interface
- AMR - block-type, based on PARAMESH library
  - *relatively easy to code*
  - *can completely automatic (different refinement criteria)*
  - *less efficient*
- self-gravity in Flash
  - *multipole solver (bad scaling with #CPU, must be close to spherical symmetry)*
  - *multigrid solver (very slow - lot of communication, artifacts at different AMR levels)*
  - *Why not tree code?*

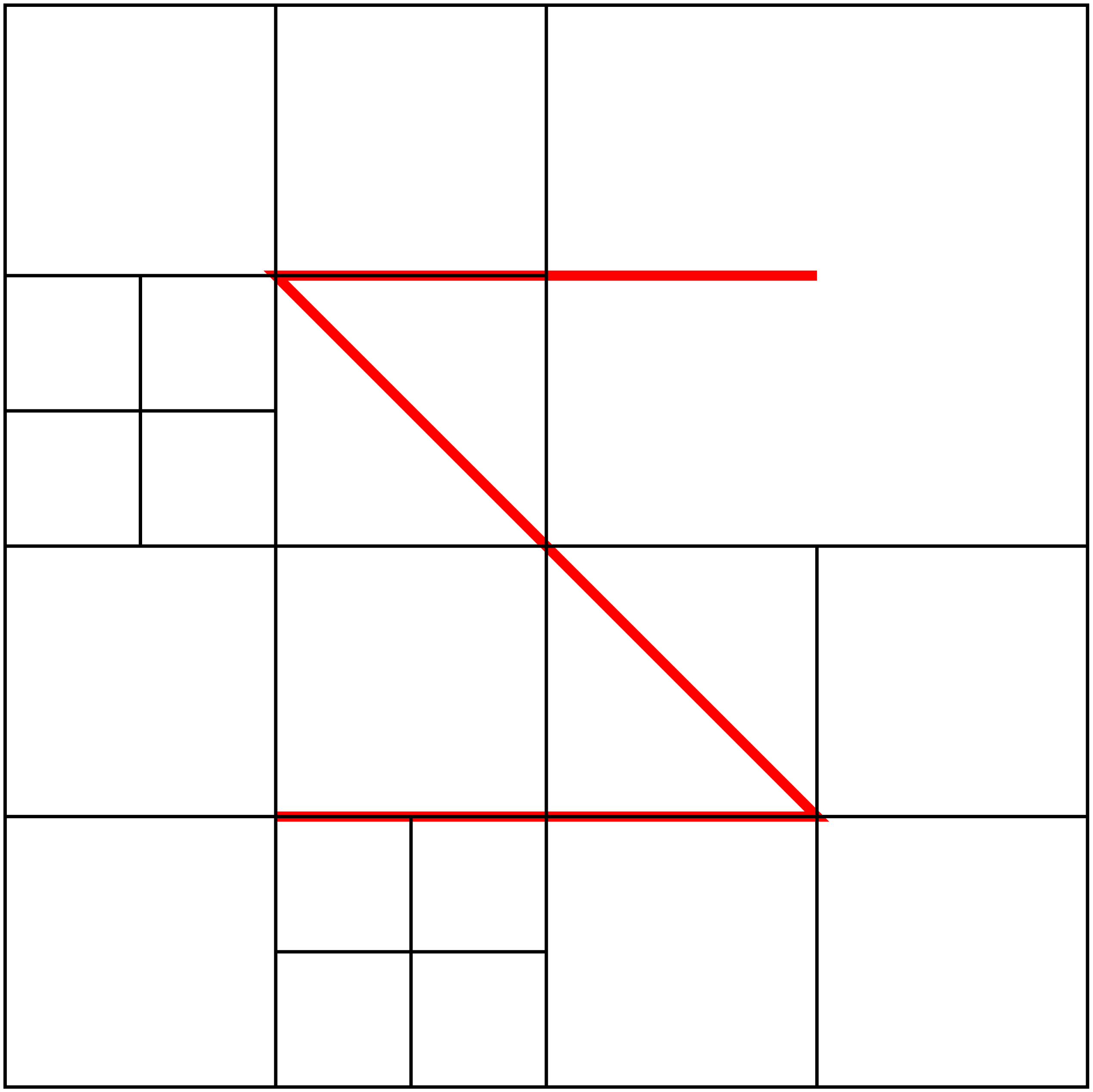
AMR in Flash



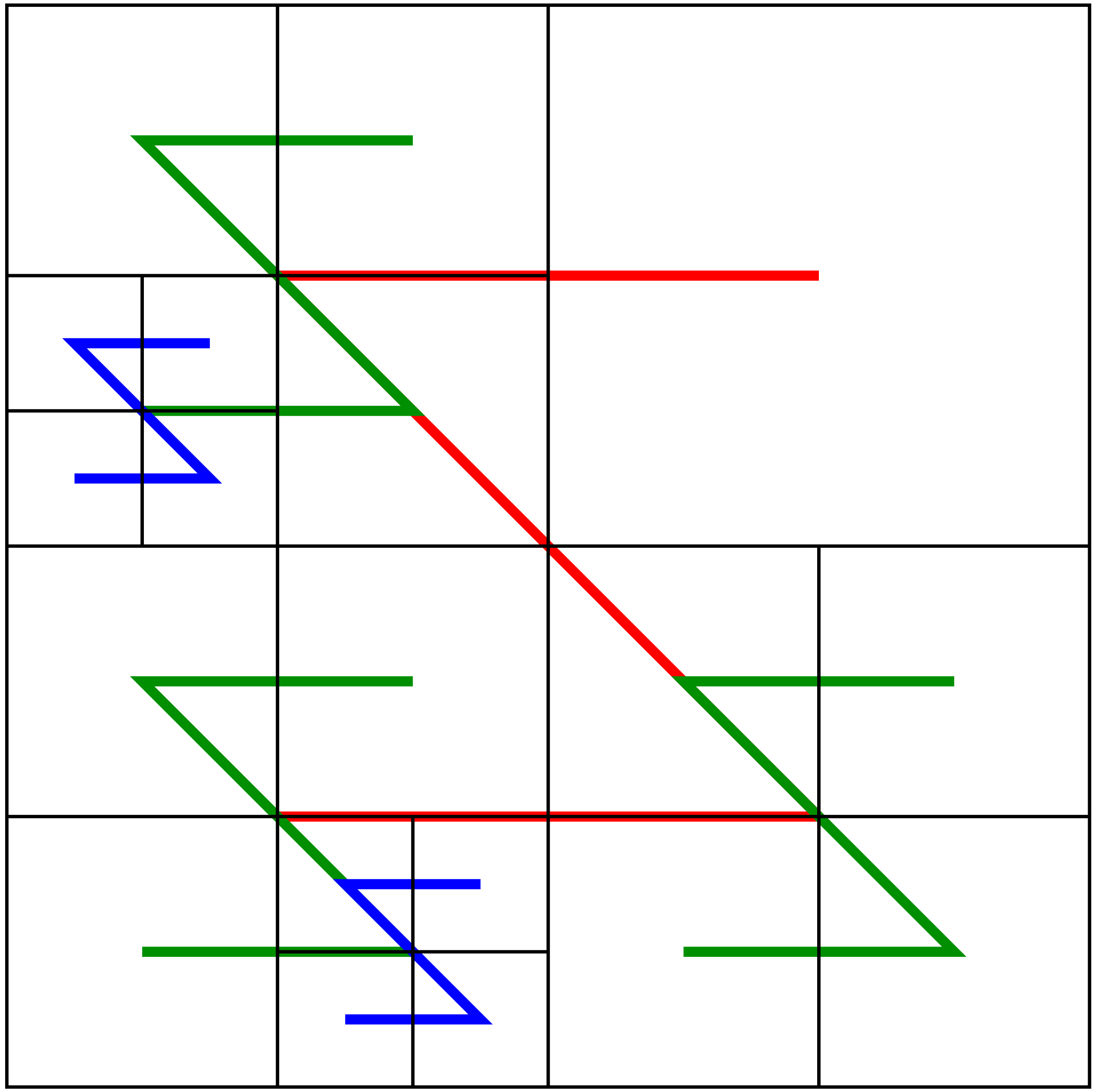
AMR in Flash



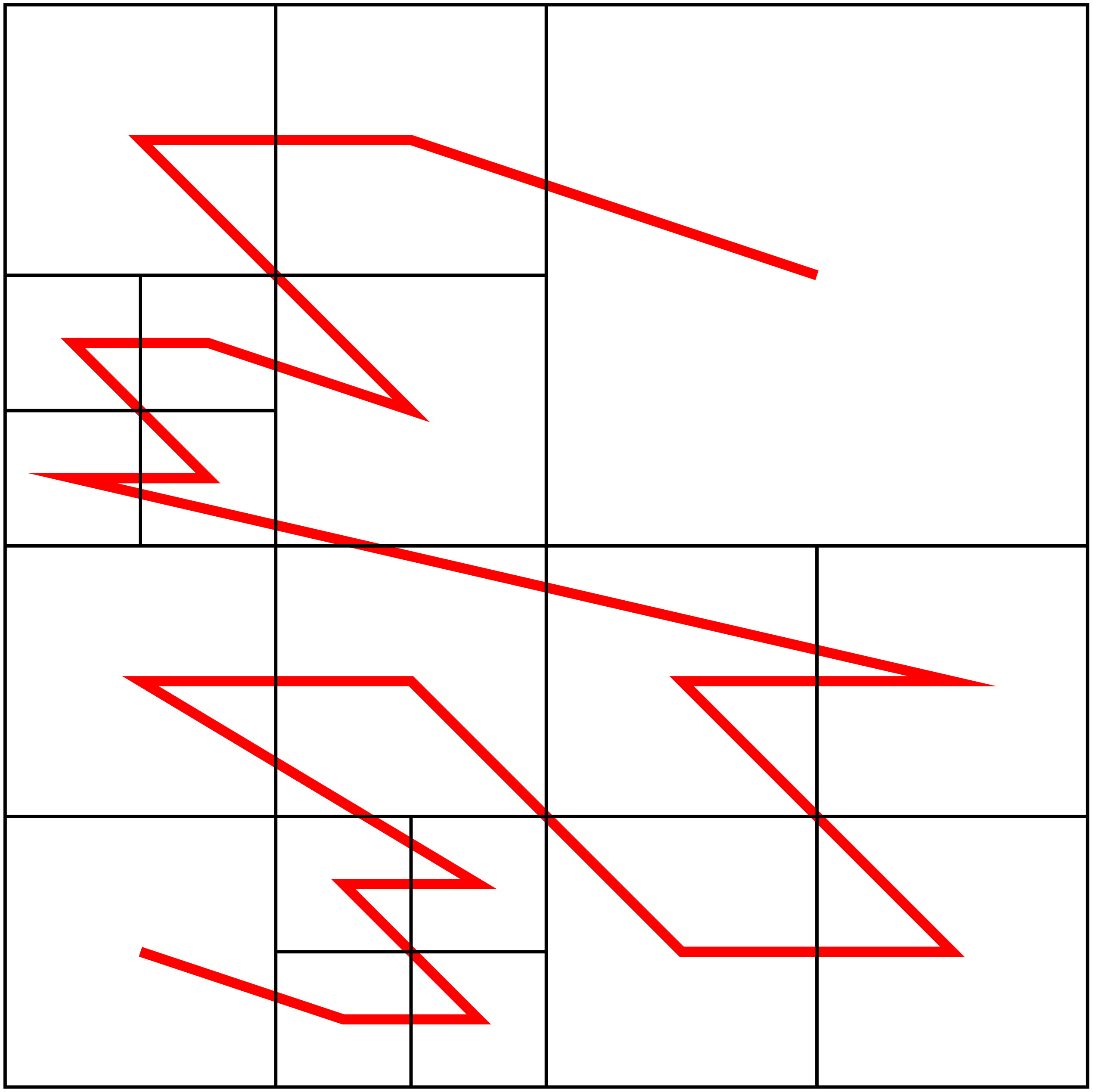
Load balancing - Morton curve



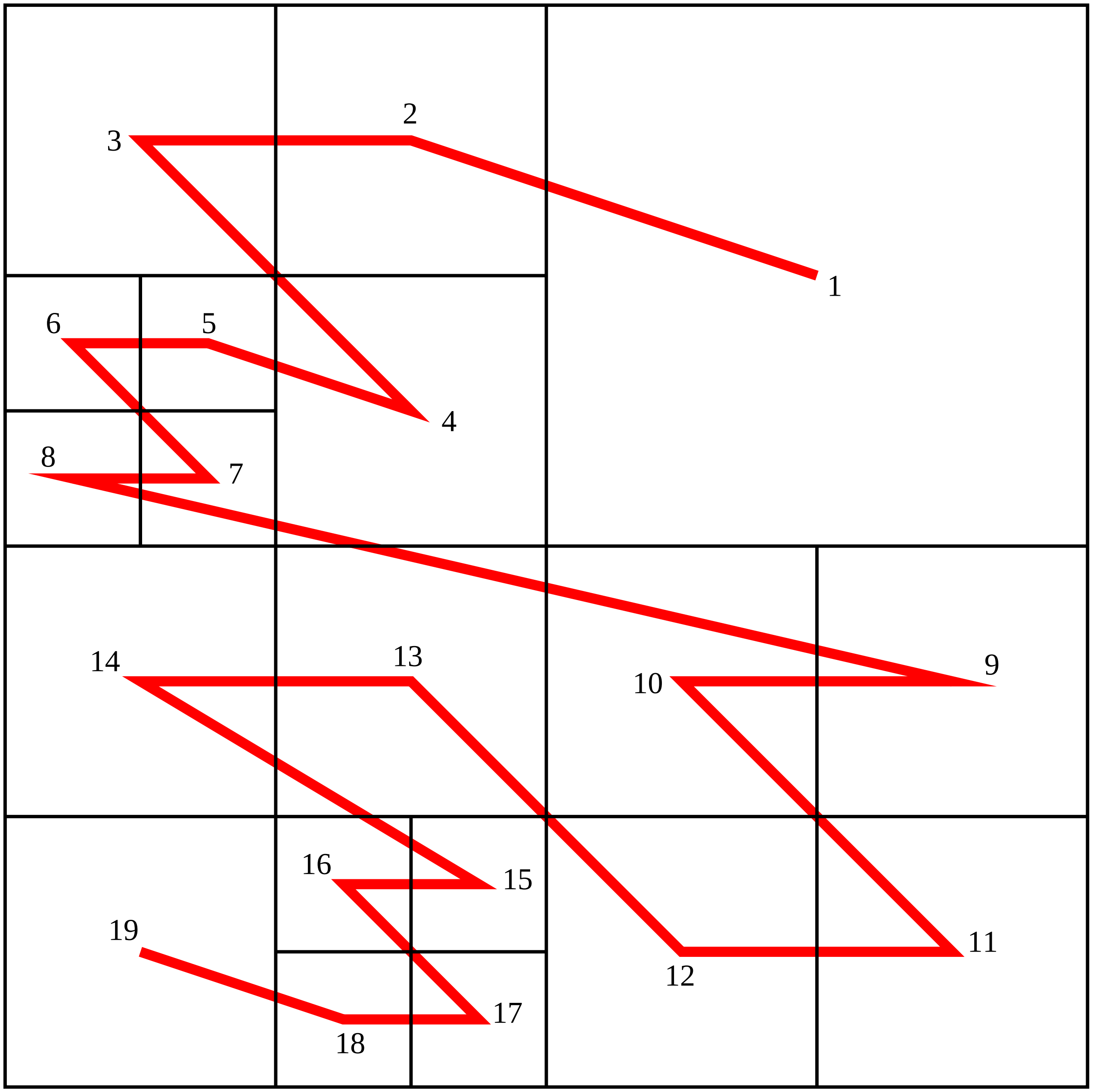
Load balancing - Morton curve



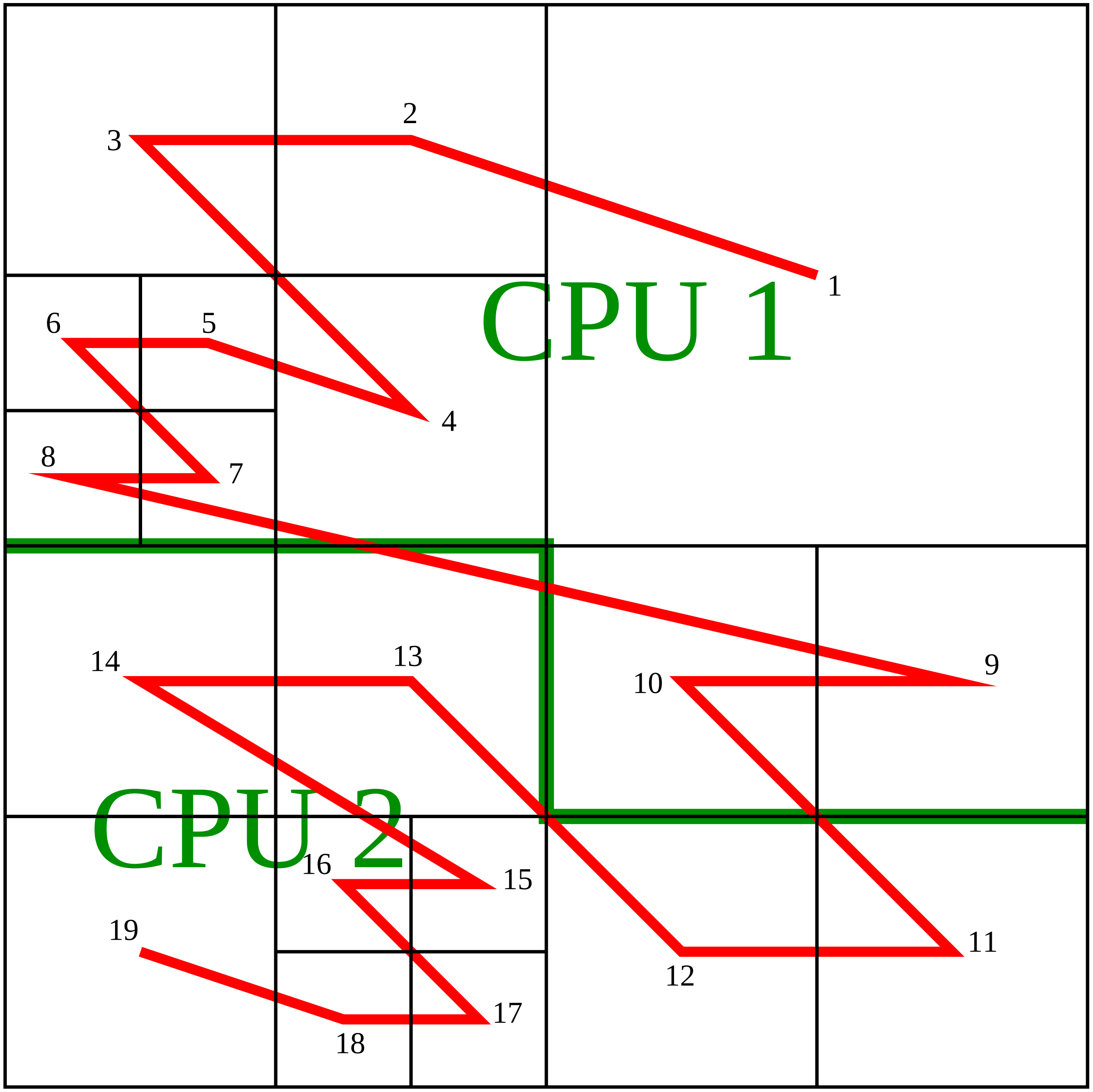
Load balancing - Morton curve



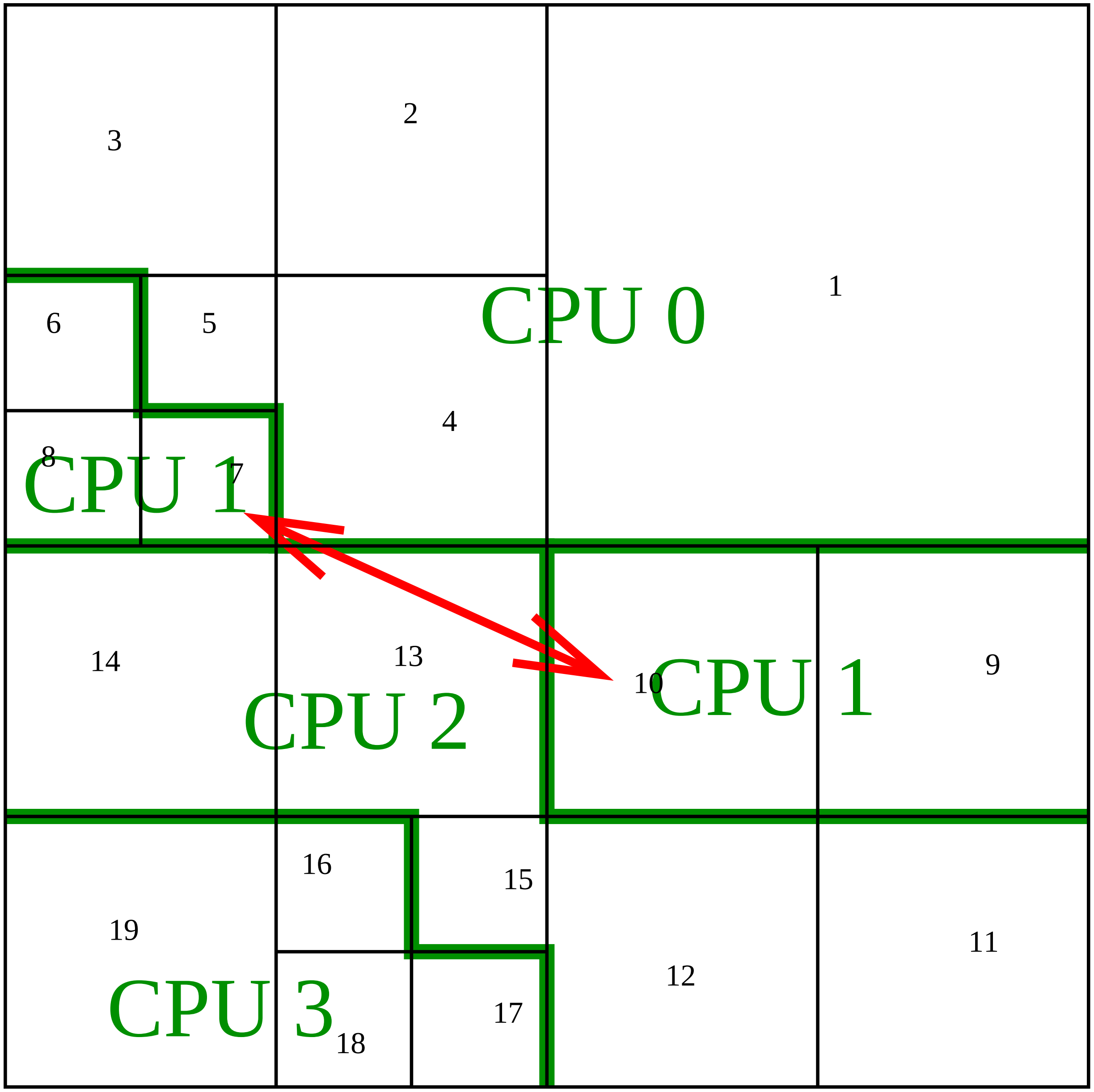
Load balancing - Morton curve



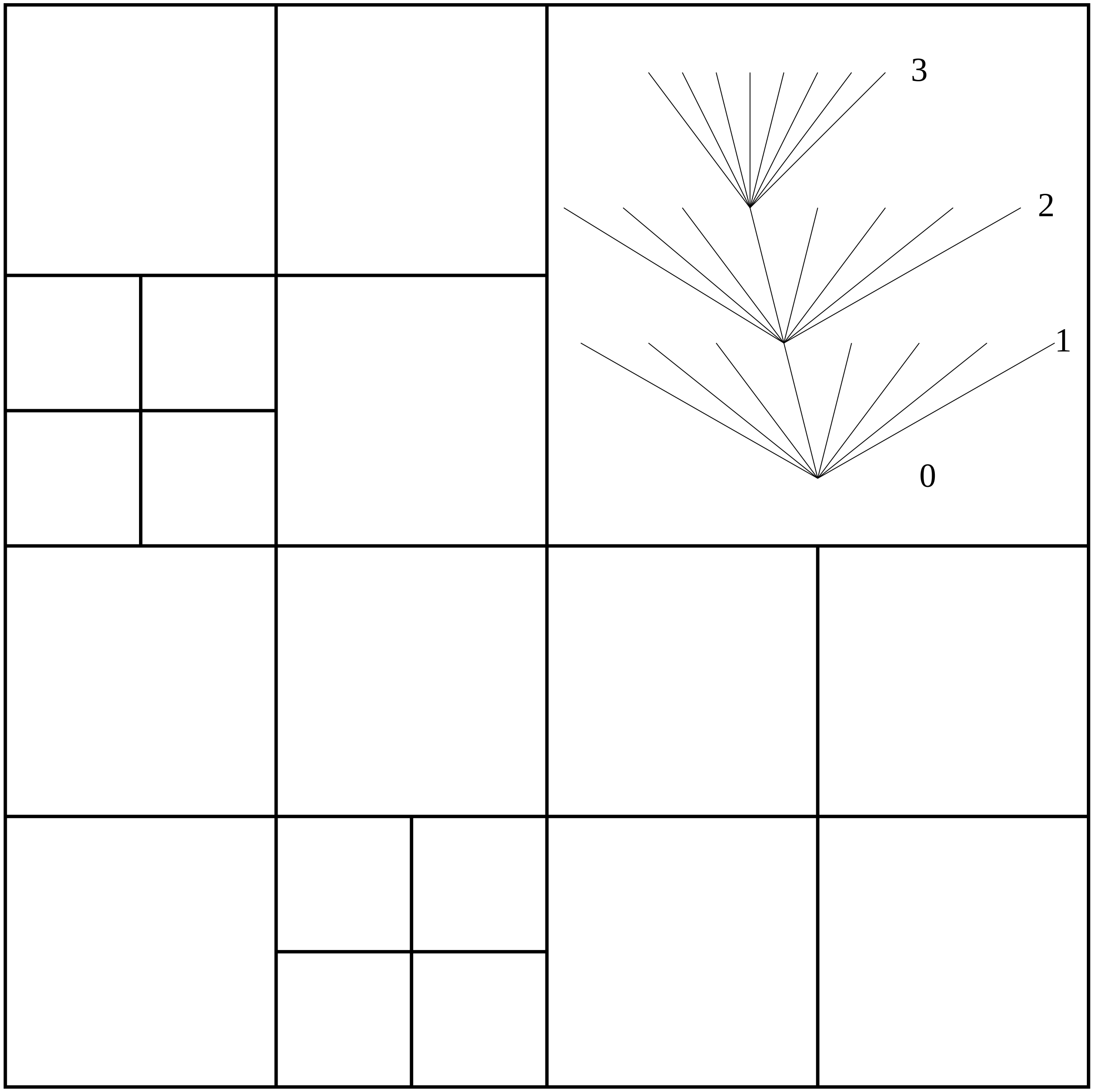
Load balancing - Morton curve



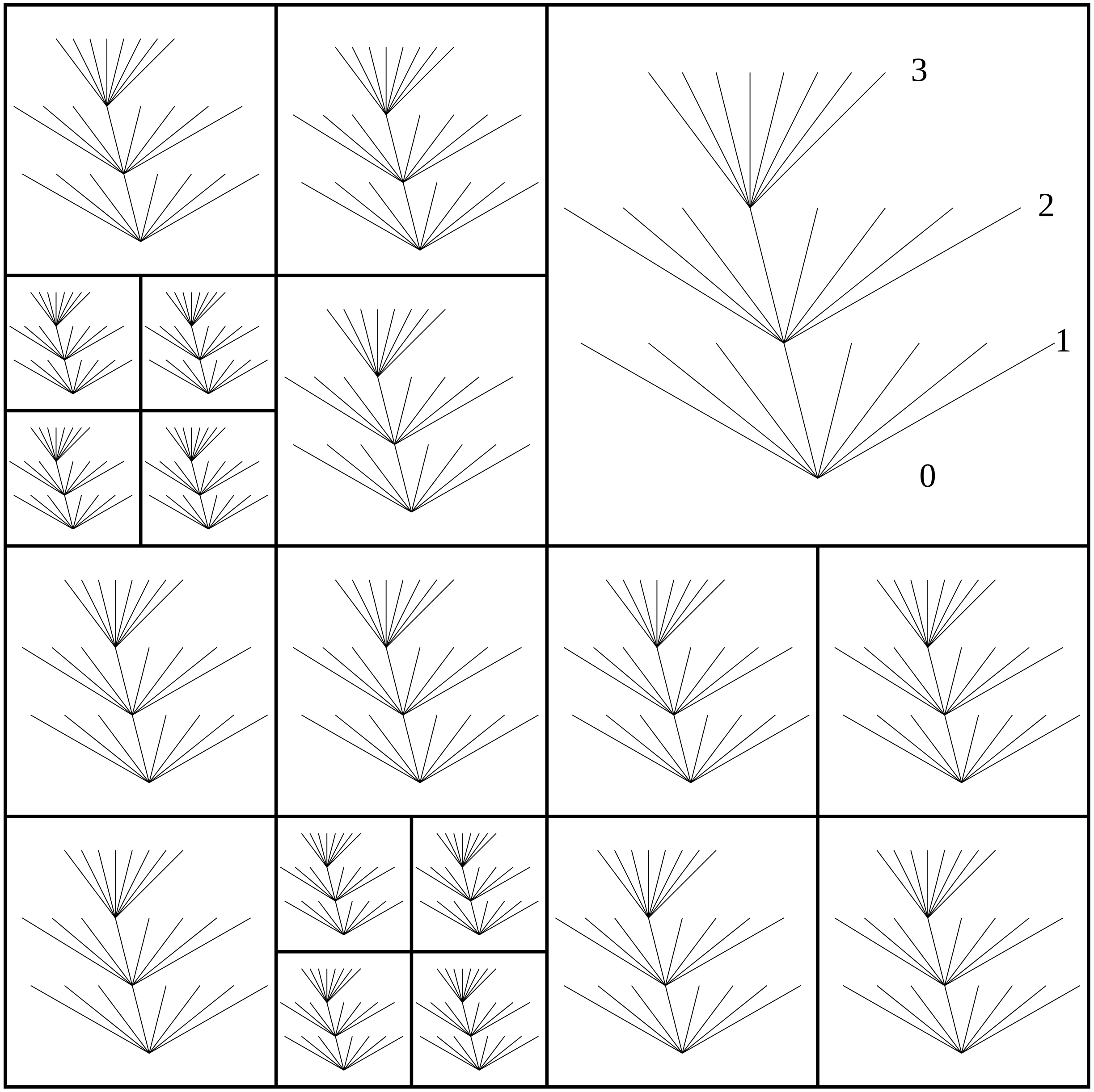
Load balancing - Morton curve



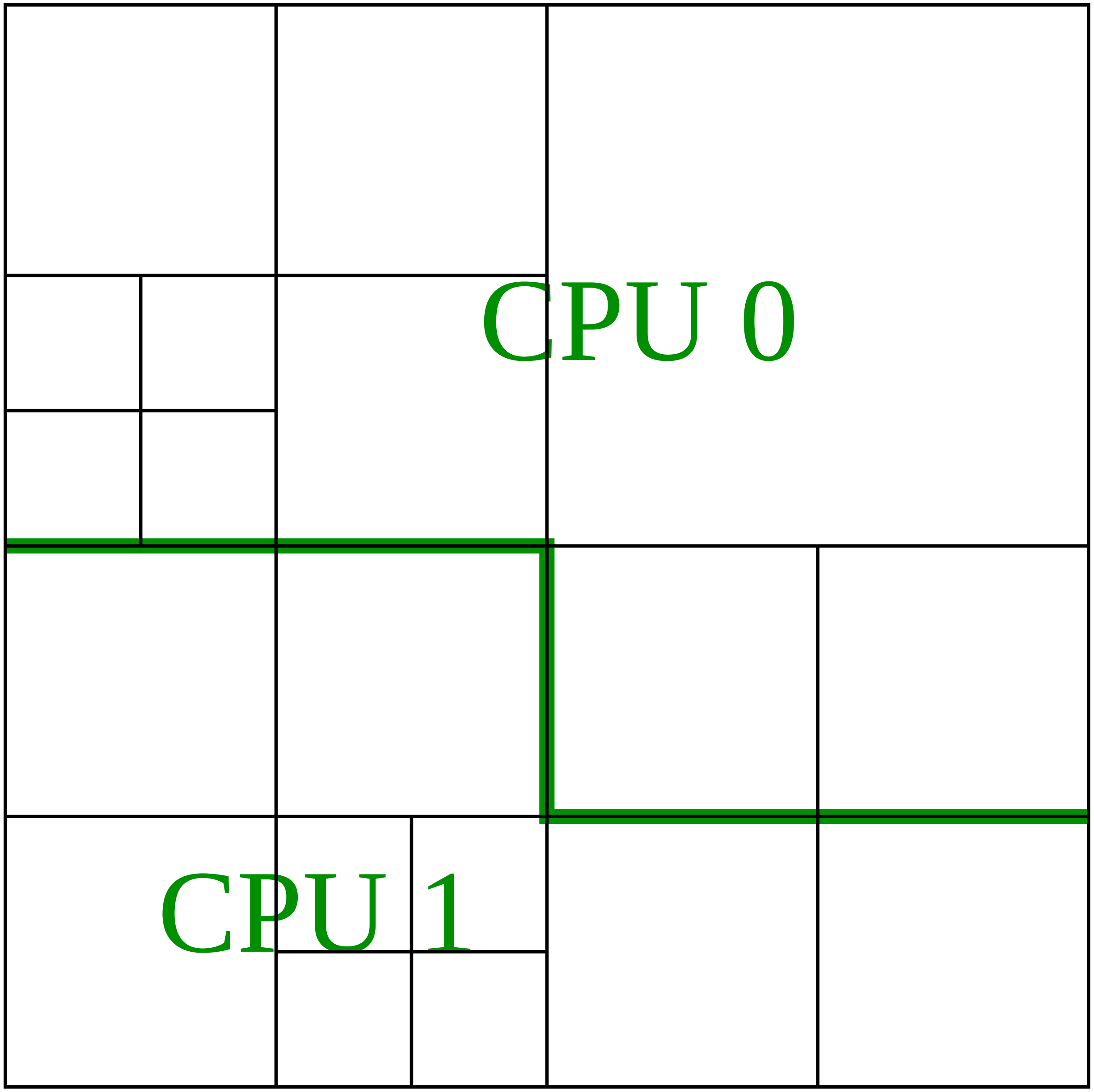
Gravity tree



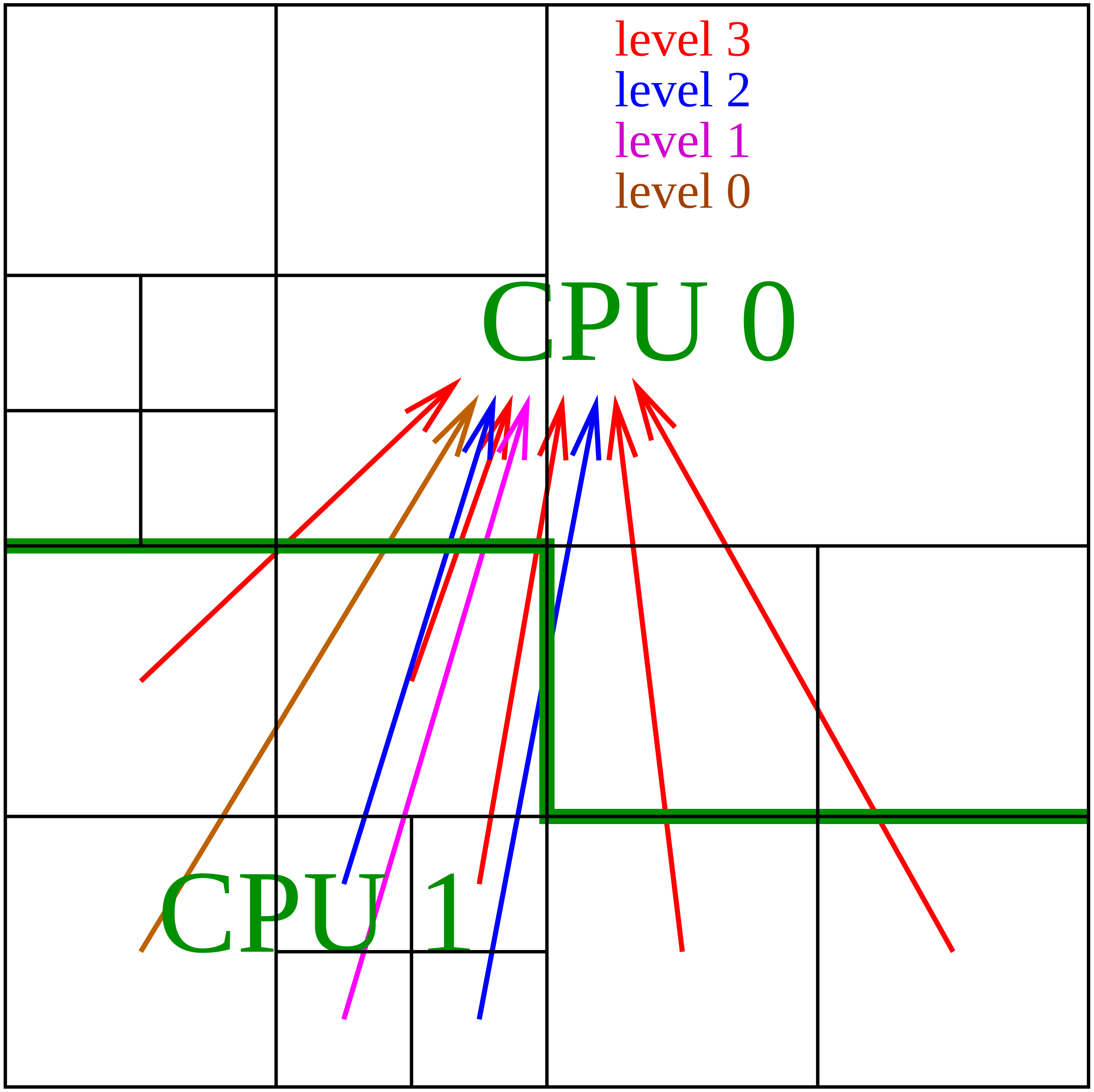
Gravity tree



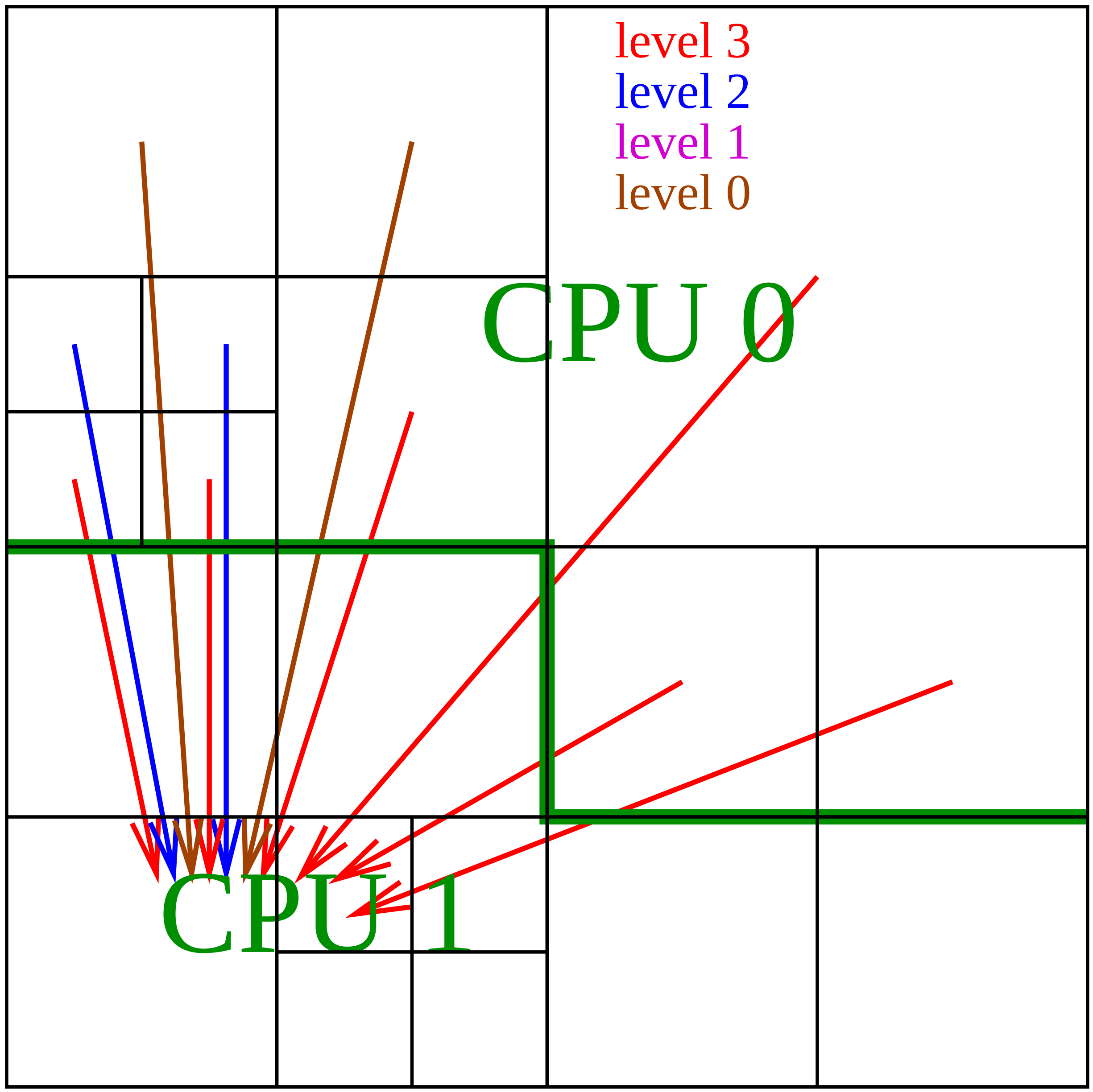
Gravity tree



Gravity tree



Gravity tree



The code scheme

gr_solversInit − initialization, allocation of arrays, setting constants, etc.

Grid_solvePoisson (in each time−step):

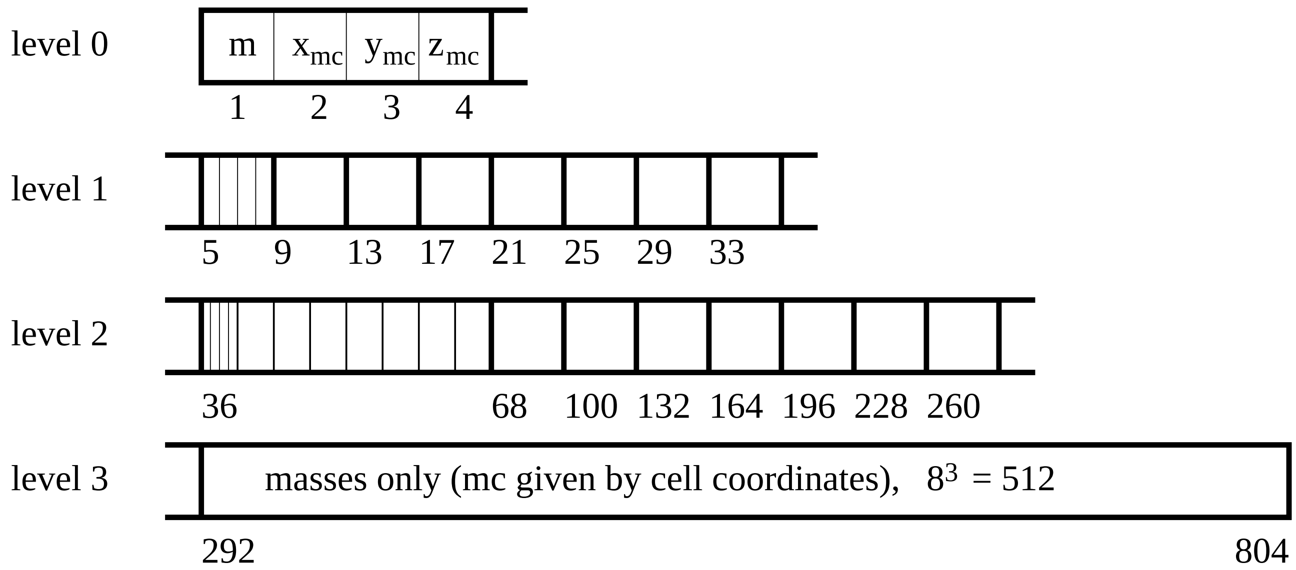
- gr_treeBuildTree ← gr_treeBuildTreeBlock ← { gr_treeGetTreePos, gr_treeGetTreeSize }
- gr_treeExchangeTrees
- gr_treePotential ← gr_treePotentialBlock
- gr_treeDestroyTree

gr_solversFinalize − deallocation of arrays

Interfaces:

gr_treeData − global variables of the module
gr_treeInterface − prototypes of subroutines

Tree in RAM



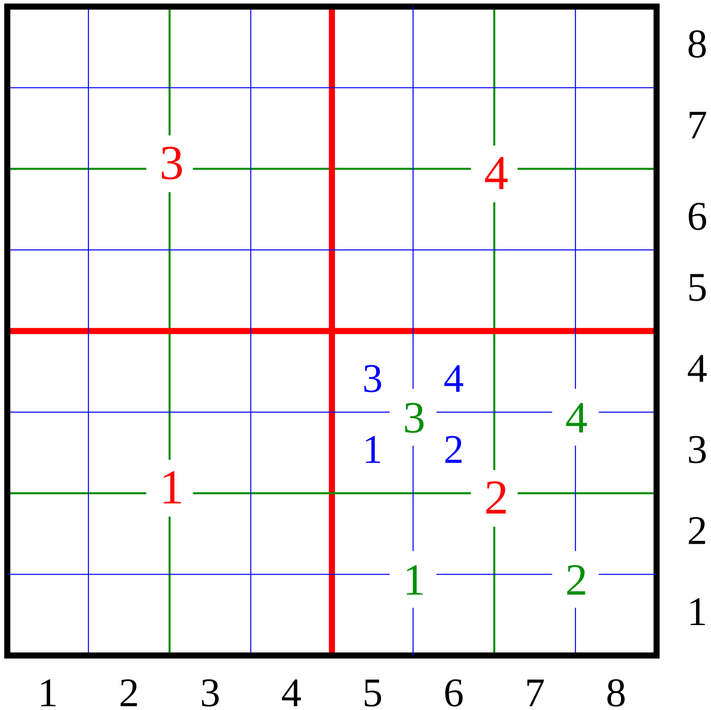
L . . . number of the lowest level (typically 3)

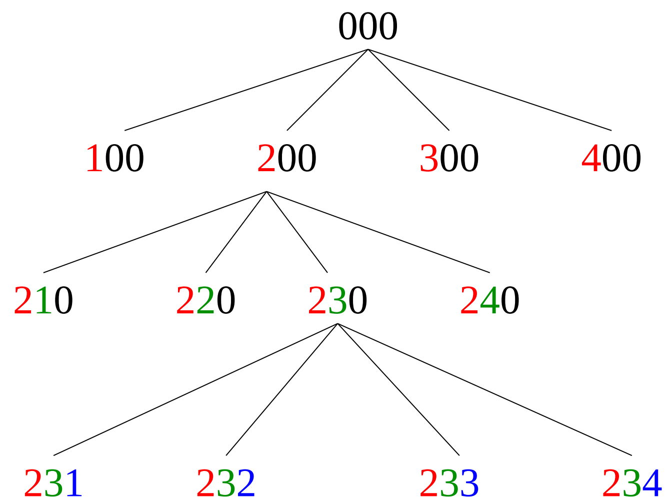
$$\text{Tree size} = 8^L + 4\sum_{i=0}^{L-1} 8^i = 8^L + 4\frac{8^L - 1}{7}$$

tree nodes identified by multi-index - integer array of size L: (l_1, l_2, l_3); $l_i =$

- *1-8 . . . number of node on i-th level*
- *0 . . . multi-index (i.e. node) is of level i-1*

Multi-index - ID of the node





coordinates → node multi-index

```
do l = 1, tr_levels
  fac = 2**(tr_levels - l)
  multi(l) = 1 + mod((i-1),2*fac)/fac  &
           + 2 *(mod((j-1),2*fac)/fac) &
           + 4 *(mod((k-1),2*fac)/fac)
enddo
```

node multi-index → coordinates

```
(i,j,k) = (1,1,1)
do l = 1,tr_levels
  fac = 2**(tr_levels - l)
  i = i + (mod(multi(l)-1,2))     * fac
  j = j + (mod((multi(l)-1)/2,2)) * fac
  k = k + (mod((multi(l)-1)/4,2)) * fac
enddo
```

Linear index in the tree array

```
integer function get_tree_pos(level, mi)
  use gr_treeData, ONLY : tr_levels
  integer,intent(in) :: mi(1:tr_levels)
  integer            :: pos, fs

  ! field size: 1 for the lowest level (only masses),
  !             4 for the higher levels (mass, position of mass centre)
  if (level == tr_levels) then
    fs = 1
  else
    fs = 4
  endif

  pos = 1 + 4*(8**level - 1)/7 ! offset of the level
  do l = level,1,-1
    pos = pos + (mi(l)-1)*fs
    fs = fs * 8
  enddo
  get_tree_pos = pos
end function get_tree_pos
```

Building tree

- create the lowest level (3)
  - *last segment of tree array filled with cell masses*
  - *the 2nd lowest level prepared -* m_i, $m_i \times \mathbf{r}_i$ *accumulated*
- finish the 2nd lowest level
  - $\mathbf{r}_{mc} = \sum m_i \times \mathbf{r}_i / m$
- create the higher levels
  - *loop over levels (second lowest and higher)*
  - *loop over nodes of the given level*
  - *collect contribution from each octet and write it into parent node*

Communication - data structures

- communication arrays that are synchronized among all CPUs

```
integer tr_activeBlocks(MAXBLOCKS, #CPUs) - 1 denotes LEAF block
real tr_BBoxes(MAXBLOCKS, 2, DIM, #CPUs) - bounding boxes
real tr_Coords(MAXBLOCKS, 2, BLOCK_SIZE, #CPUs) - coordinates
real tr_Diag(tr_levels, MAXBLOCKS, #CPUs) - node sizes
real tr_sentTreeLevels(MAXBLOCKS, #CPUs (to), #CPUs (from))
```

- trees - array of pointers

```
! array of pointers to trees 2D = (#CPUs, MAXBLOCKS)
type p_tree
  real, dimension(:), pointer :: p
end type p_tree
type(p_tree), save, dimension(:,:), allocatable :: tree_array
```

- communication costs:
  - ▷ *ping roundtrip on coma* ~ 0.2 *ms* (*→ establishing TCP connection* ~ 1 *ms*)
  - ▷ *1Gb ethernet = 128 MB/s ⇒ 128 KB/ms*
  - ▷ *establishing connection ⇔ transmitting array with 16000 elements*

Communication scheme

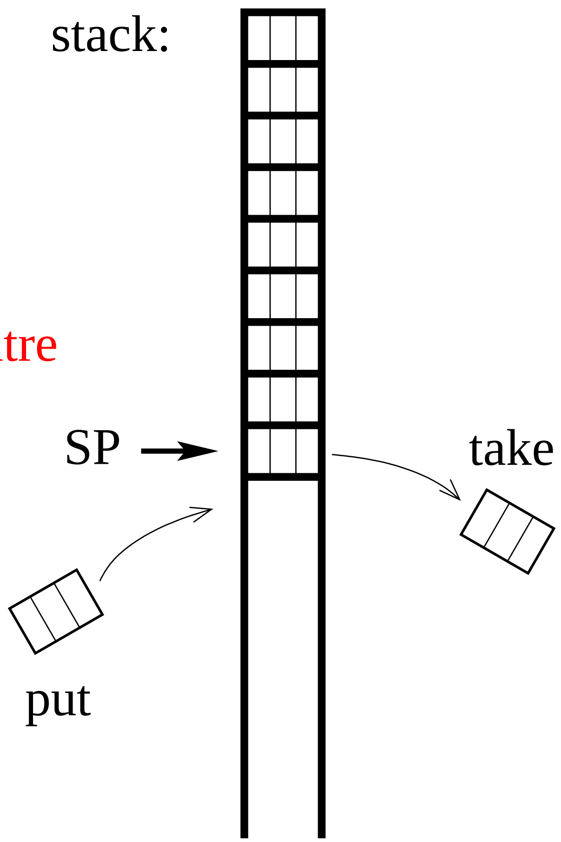
- synchronize global block describing arrays

`tr_activeBlocks, tr_BBoxes, tr_Coords, tr_Diag`

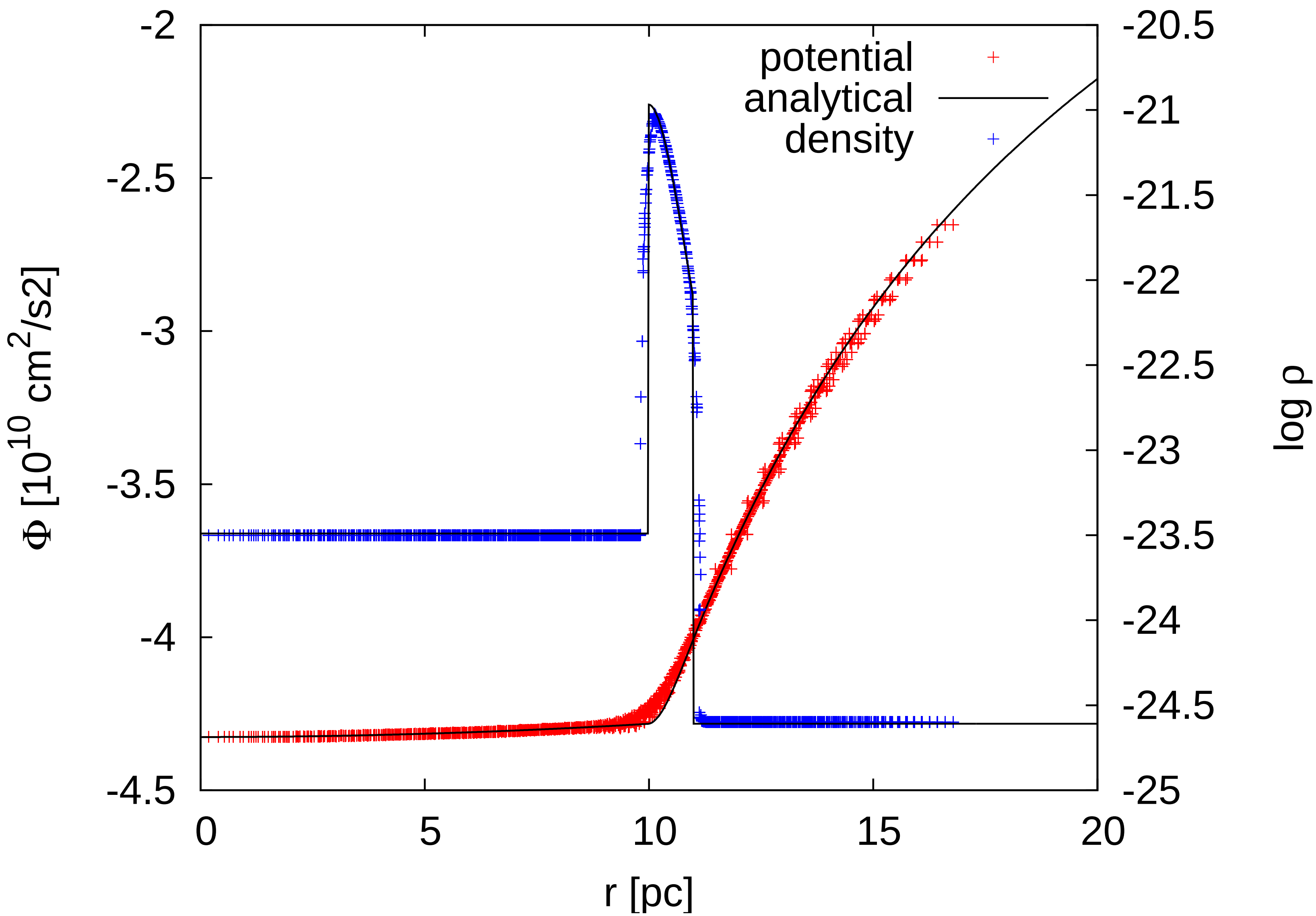
- determine tr_sentTreeLevels
  - ▷ *compare BBox of each block with BBoxes of all blocks on to-CPU*
  - ▷ *searches for the minimum distance of corners (cmp. all pairs)*
- synchronize tr_sentTreeLevels
- prepare messages
  - ▷ *for each CPU 1D array which contains trees to the necessary level*
  - ▷ *allocate arrays for messages to be recieved*
- exchange messages - non-blocking communication (MPI_Isend, MPI_recv)
- copy trees from messages to tree_array

Potential computation

```
for each point
  for each tree(cpu & block)
    put root node (0,0,0) on stack
    do
      take a node from stack
      dist = distance between point and node mass centre
      if block size / dist < limit_angle then
        potential += node mass / dist
      else
        put all children of the node on the stack
      endif
    until stack empty
  endfor
endfor
```



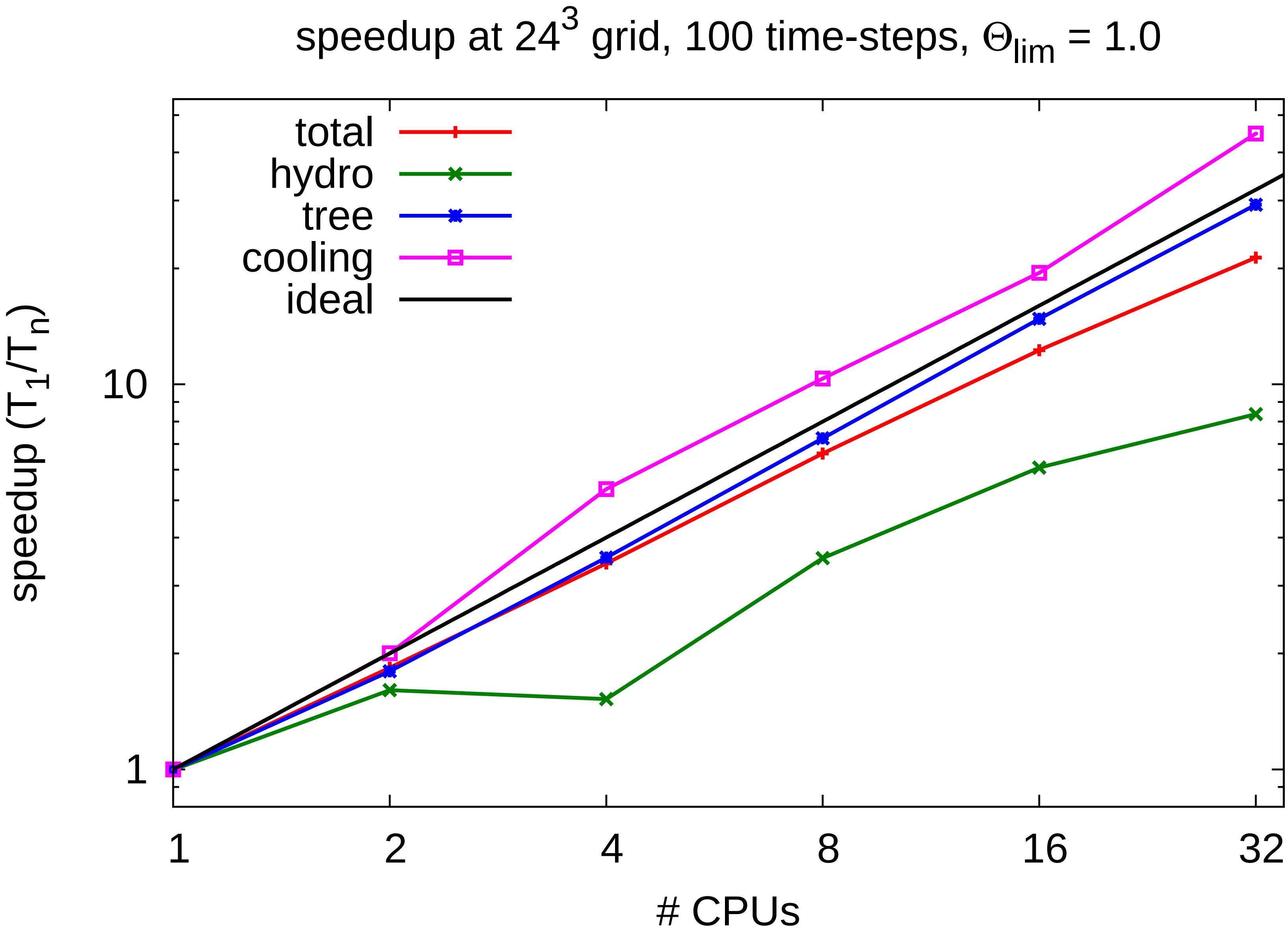
Accuracy test



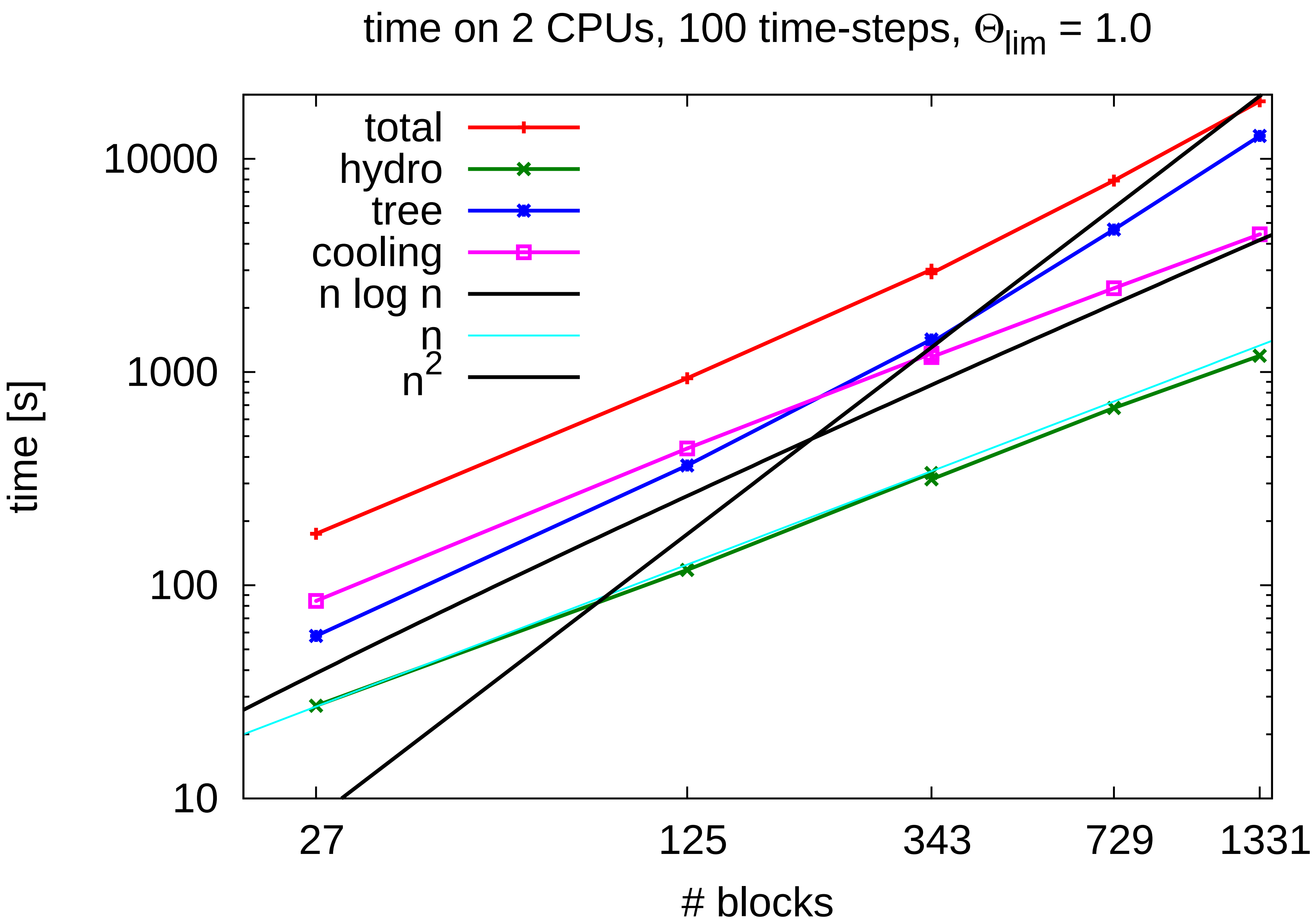
Performance

```
Number of processors:              8
Dimensionality:                    3
Max Number of Blocks/Proc:      1500
Number x zones:                    8
Number y zones:                    8
Number z zones:                    8
[GRID] min blks      158    max blks      159    tot blks      1267
[GRID] min leaf blks 138    max leaf blks 140    tot leaf blks 1112
            beginning / ending   : 10-11-2007  17:15.51 / 10-12-2007  03:45.39
seconds / number of subintervals :            37788.063 / 564
------------------------------------------------------------------------------
accounting unit                       time sec  num calls   secs avg  time pct
------------------------------------------------------------------------------
initialization                         125.469      1        125.469     0.332
evolution                            37660.289      1      37660.289    99.662
 gravity                             31560.410   1128         27.979    83.520
  poisson_tree                       31529.262   1128         27.951    83.437
   build_tree                           54.137   1128          0.048     0.143
   exchange_tree                       630.207   1128          0.559     1.668
   potential                         30844.047   1128         27.344    81.624
   destroy_tree                          0.840   1128          0.001     0.002
 hydro                                5116.965   1128          4.536    13.541
  guardcell internal                  2939.414   3384          0.869     7.779
  hy_sweep                            2177.516   3384          0.643     5.762
   hy_block                           1835.227 466992          0.004     4.857
    hydro_1d                           595.941 ******          0.000     1.577
   hy_updateSoln                        34.543 933984          0.000     0.091
 Grid_updateRefinement                 523.840    564          0.929     1.386
 io                                    445.289    564          0.790     1.178
```

Scaling with # of CPU



Scaling with # of blocks



What next

- higher tree levels (to be less bush-like :))
  - *in theory simple: just to add $m, \mathbf{r}_{mc}$ for non-LEAF blocks*
- some optimalization ($1/x$, inline function)
- quadrupole moments
- any ideas?