Enzo-E/Cello astrophysics and cosmology

algorithmic improvements for scalability and robustness

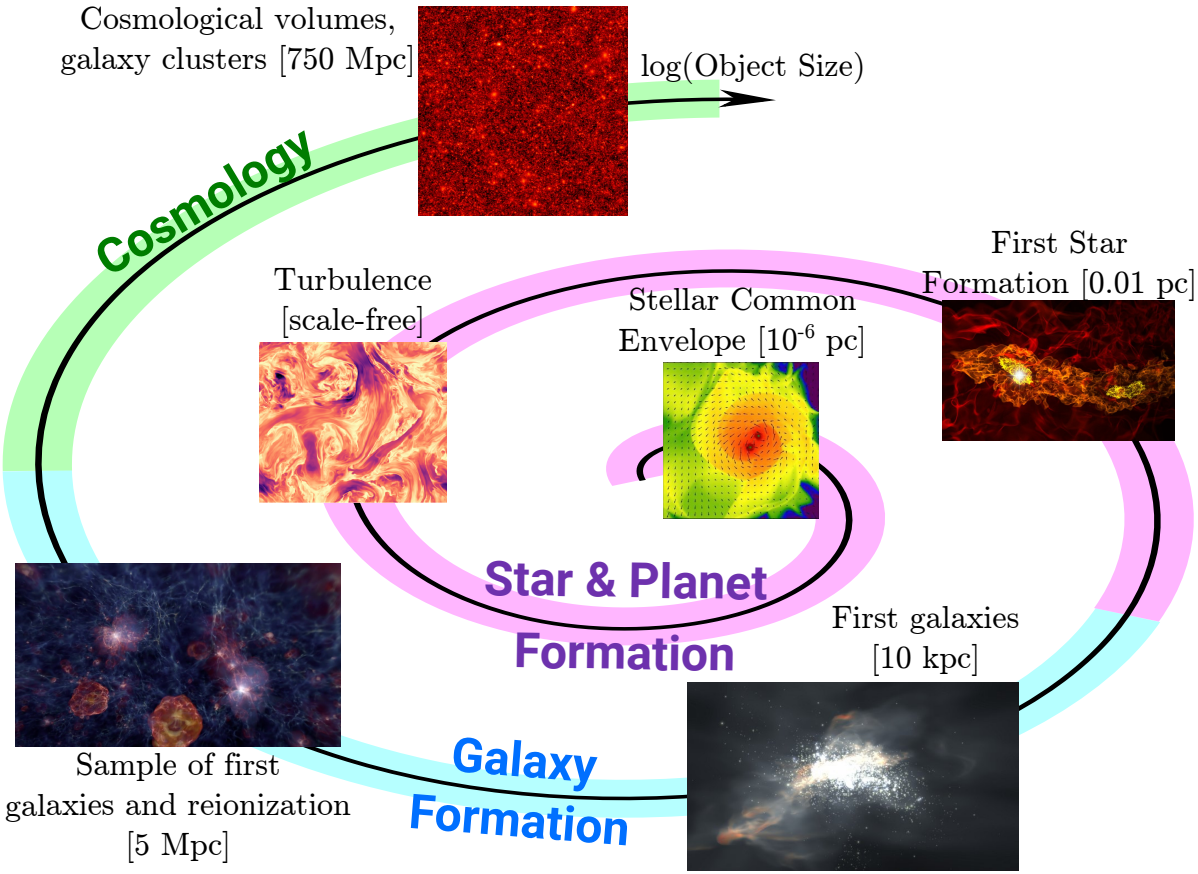
James Bordner, Michael L. Norman

University of California, San Diego
San Diego Supercomputer Center

19th Annual Workshop on
Charm++ and Its Applications
2021-10-18/19

Scientific motivation

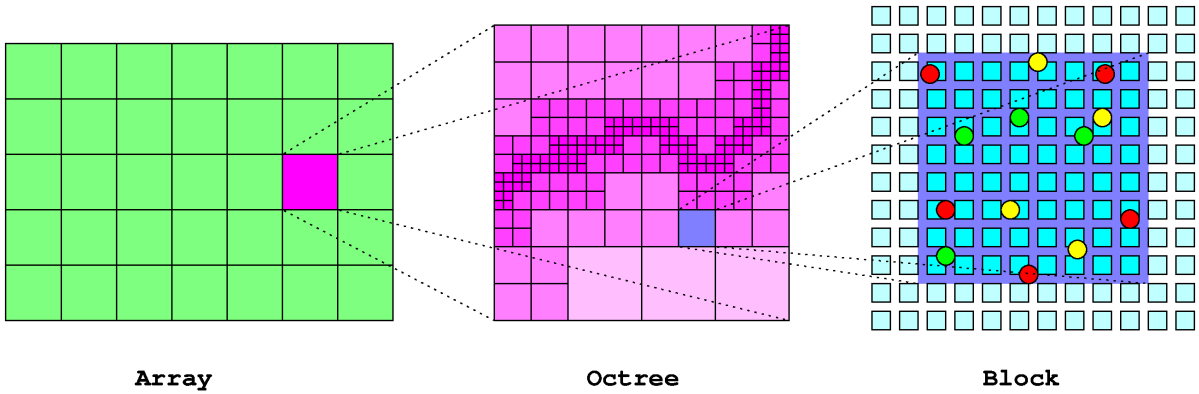
Astrophysics and cosmology



[Image credit: John Wise]

Enzo-E/Cello Software layers

Cello adaptive mesh refinement framework



Parallel datastructure

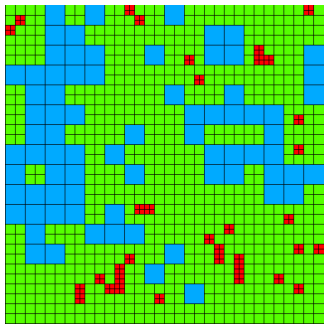
- array-of-octree AMR
- fixed shaped blocks
- field data (arrays)
- particle data
- single chare array
- fully distributed

AMR operations

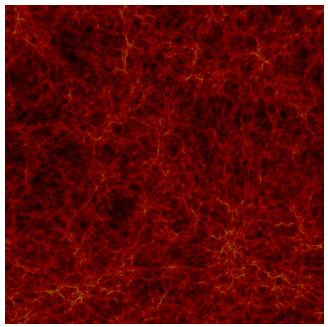
- adaptive remeshing
- apply IC's/ BC's / Methods
- refresh ghost zones
- migrate particles
- inter-level operations
- HDF5 input/output

Enzo-E/Cello Software layers

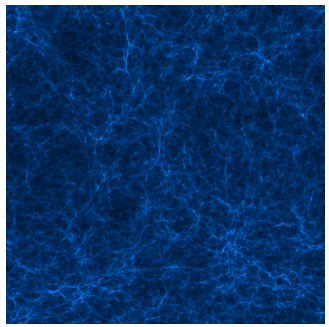
Enzo-E computational astrophysics and cosmology application



array-of-octrees



baryonic matter



dark matter

Physics methods

- PPM hydrodynamics
- PM gravity
- MHD VL-CT (M. Abruzzo)
- Grackle chemistry/cooling

Specialized operations

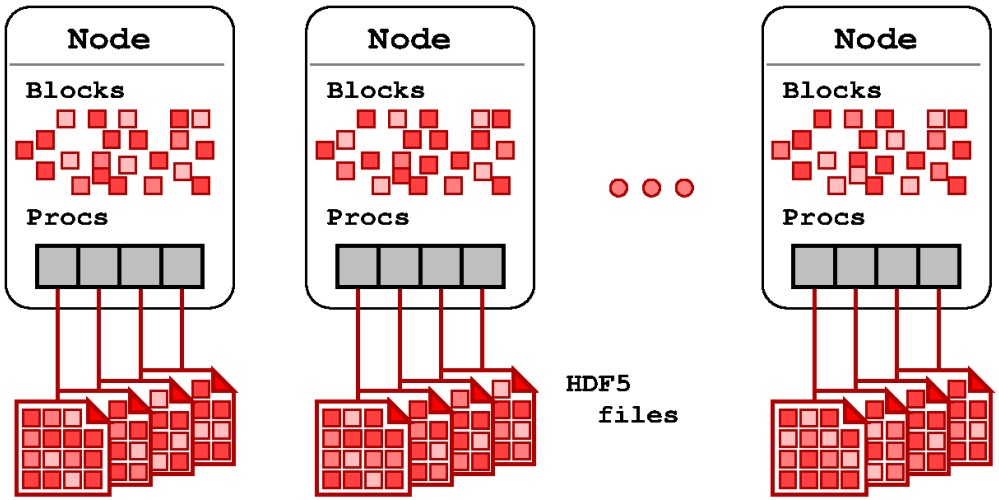
- interpolation methods
- refinement criteria
- "problem generators"
- linear solvers

Outline

1. I/O: writing (reading) data to (from) disk
   - previous approaches
   - new approach
   - preliminary performance
   - possible future directions
2. AMR: adaptive mesh refinement
   - remeshing issue: balancing the mesh
   - previous approach
   - new approach

HDF5 Input / Output

Previous implementation: **writing data**



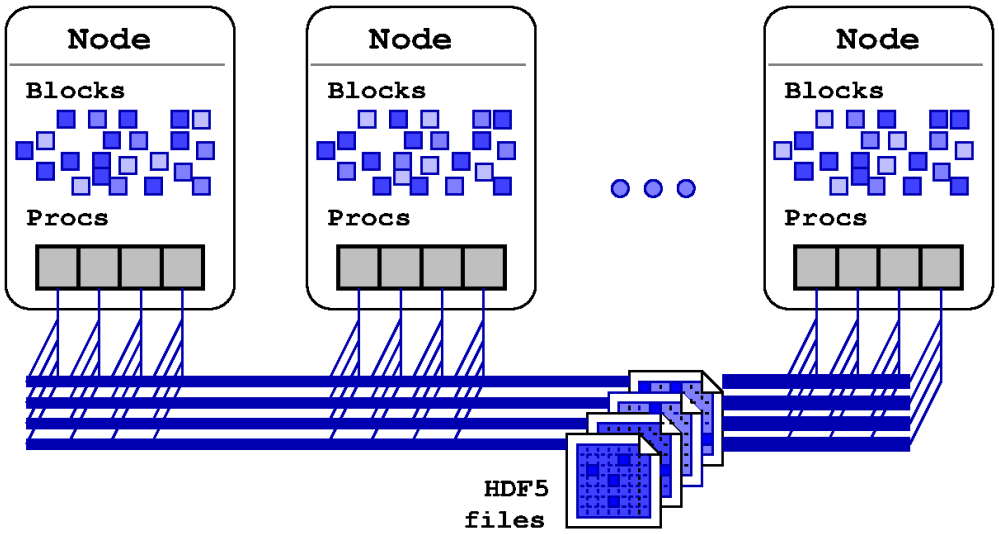
- One file per process
- (SMP mode: one per node)
- One open/close per file
- One data write per Block
- No inter-block comm.

- Synchronization required
  - open before *any* writes
  - close after *all* writes
- simple reduction using `contribute()`

- Moderate load on MDS
- Reduced using SMP mode
- Limited flexibility
- Limited scalability

HDF5 Input / Output

Previous implementation: **reading data**



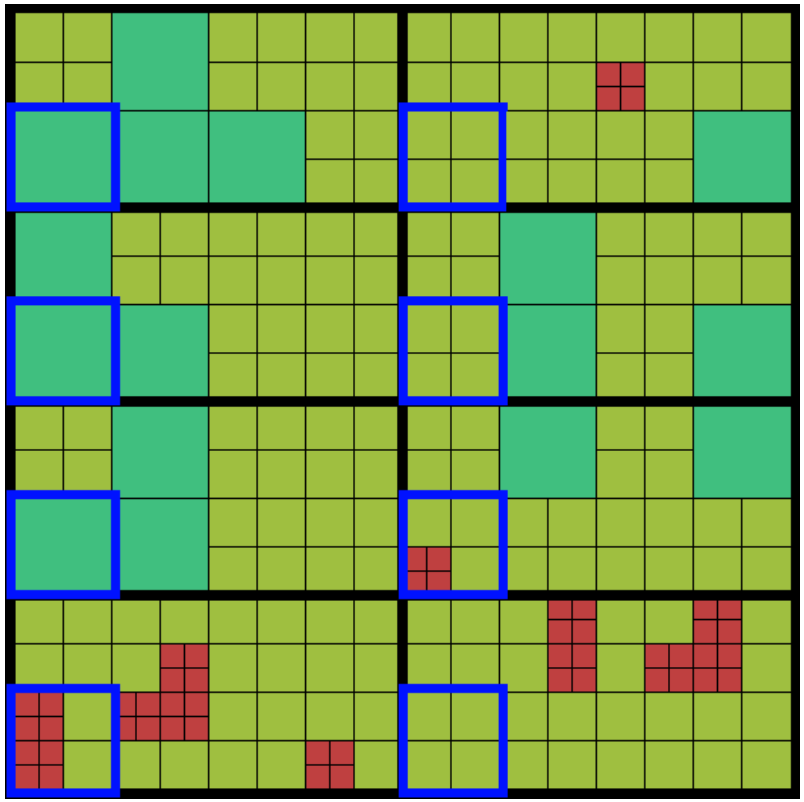
- Root-level blocks only
- One file per field/attribute
- One open/close per *Block*
- One data read per Block
- No inter-block comm.

- Less synchronization required
- Each block opens/reads/closes
- Easy to implement

- Extreme load on MDS
- Crashed 1K Frontera nodes(!)
- Limited flexibility
- Limited scalability

HDF5 Input / Output

New implementation: regular domain partitioning

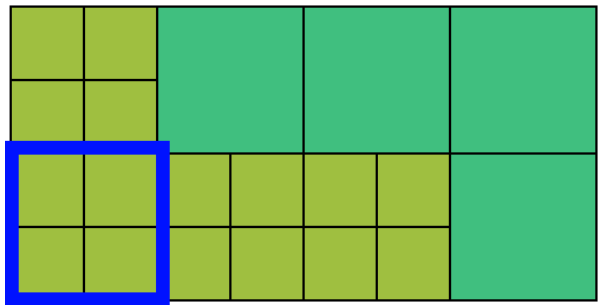


- Regular partitioning of root Blocks
- One file per partition
- One ***writer*** Block B_0 per partition
- Proceeds by depth-first traversal
- Involves inter-block comm.
- Synchronized by token passing

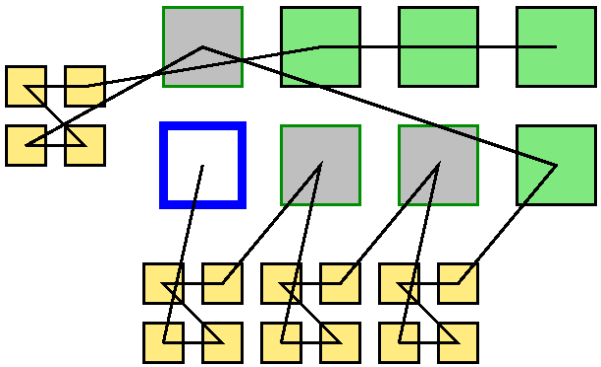
HDF5 Input / Output

New implementation: depth-first tree traversal

$(4, 2)$-blocking partition



Block partition traversal



- Alternates request / send data
  - B_0 requests data from B_i
  - B_i sends data to B_0
- Next block $i + 1$ only known at B_i

B_0 `opens file`
B_0 `writes own data if leaf`
`for` $i = 1, 2, \ldots$
    B_0 `requests data from` B_i
    B_i `returns data and` B_{i+1} `index to` B_0
    B_0 `writes` B_i `data`
B_0 `closes file`

HDF5 Input / Output

New implementation: preliminary performance

HPC Platform	Test Problem
TACC Frontera	$N_0 = 2048^3$ cosmology
4096 nodes (229K cores)	2.1M 16^3 (24^3) blocks
LUSTRE 32 OST's	$(8, 8, 8)$ `partitions`
120GB/s peak	Nine `double` fields

blocks (M)	time(s)	rate (GB/s)	% peak
2.10	110.09	17.6	14.7%
2.10	66.26	29.3	24.4%
2.12	69.11	28.5	23.8%
2.30	80.50	26.5	22.1%

HDF5 Input / Output

Future directions

There are some remaining issues with the new implementation

1. not load-balanced
2. single writer per file

Future possible directions may include

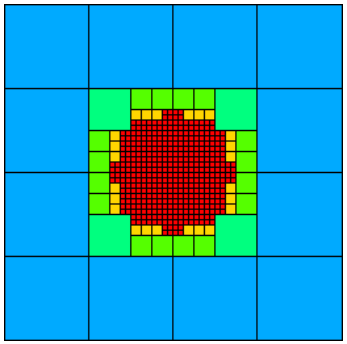
1. partitioning using space-filling curves
2. multiple writers per file

Adaptive mesh refinement algorithm

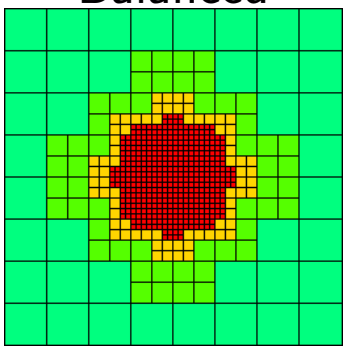
Remeshing is used to adapt to evolving simulation features.

- Block-local refinement criteria
  - tag `refine`, `coarsen`, or `same`
  - relative slope, mass density, etc.
- Change at most one level per cycle
  - $|L_i^k - L_i^{k+1}| \leq 1$
- Want to prohibit "level-jumps"
  - $|L_i^{k+1} - L_j^{k+1}| \leq 1$
  - improves mesh "quality"
  - simplifies inter-level data transfers
- Requires "balancing" step
  - refine (don't coarsen) if needed

Non-balanced



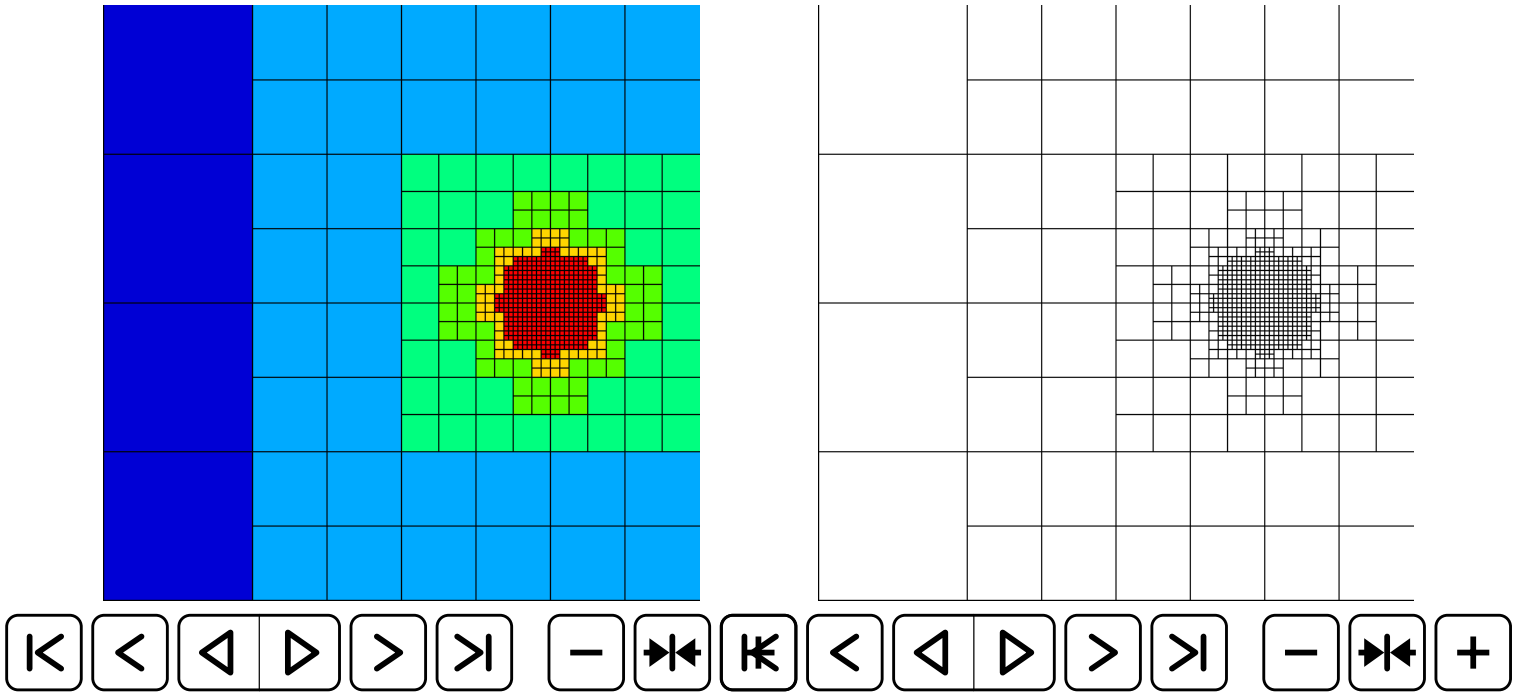
Balanced



Adaptive mesh refinement algorithm

Mesh balancing complications

Mesh balancing can lead to a cascade of induced refinements



Adaptive mesh refinement algorithm

Previous implementation

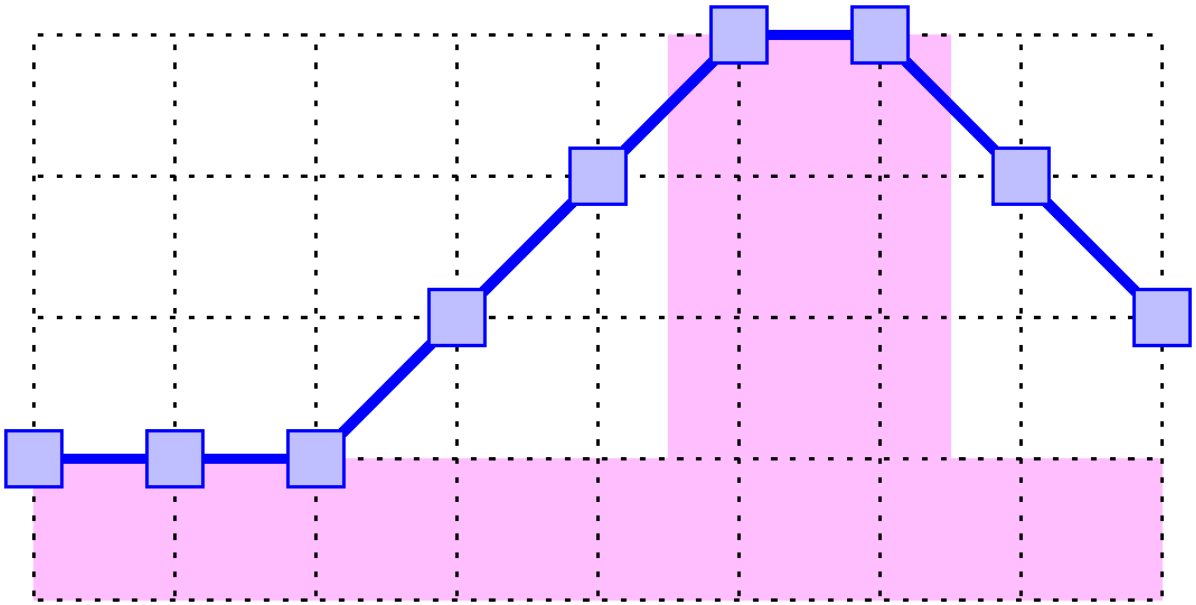
Based on "Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement"

Akhil Langer, Jonathan Lifflander, Phil Miller, Kuo-Chuan Pan, Laxmikant V. Kale, Paul Ricker

1. Blocks evaluate local refinement criteria
2. Notify neighbors with current desired level
3. Update level as needed to avoid level-jumps
4. Re-notify neighbors with updated level as needed
5. Balancing terminates with quiescence detection

Adaptive mesh refinement algorithm

Previous implementation

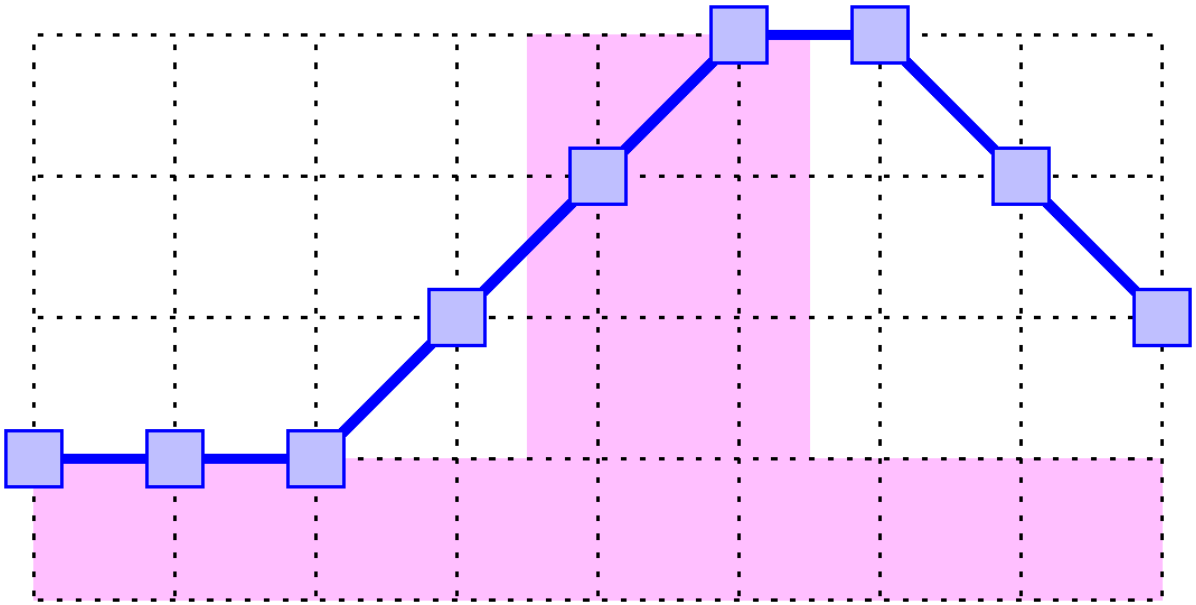


current “mesh”

its associated refinement criteria

Adaptive mesh refinement algorithm

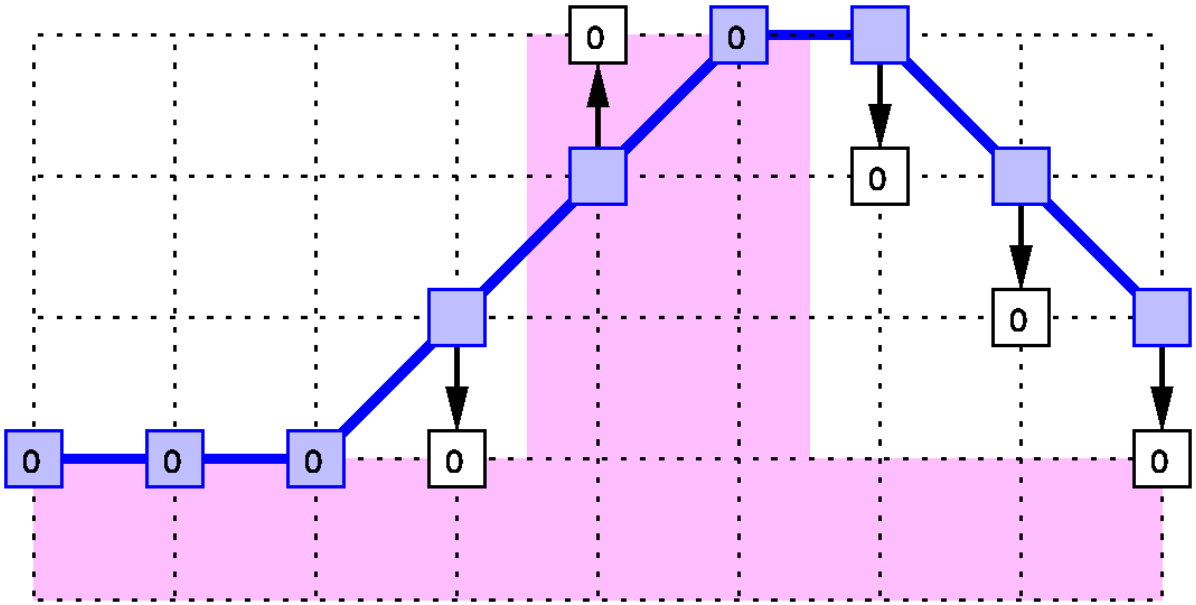
Previous implementation



updated refinement criteria

Adaptive mesh refinement algorithm

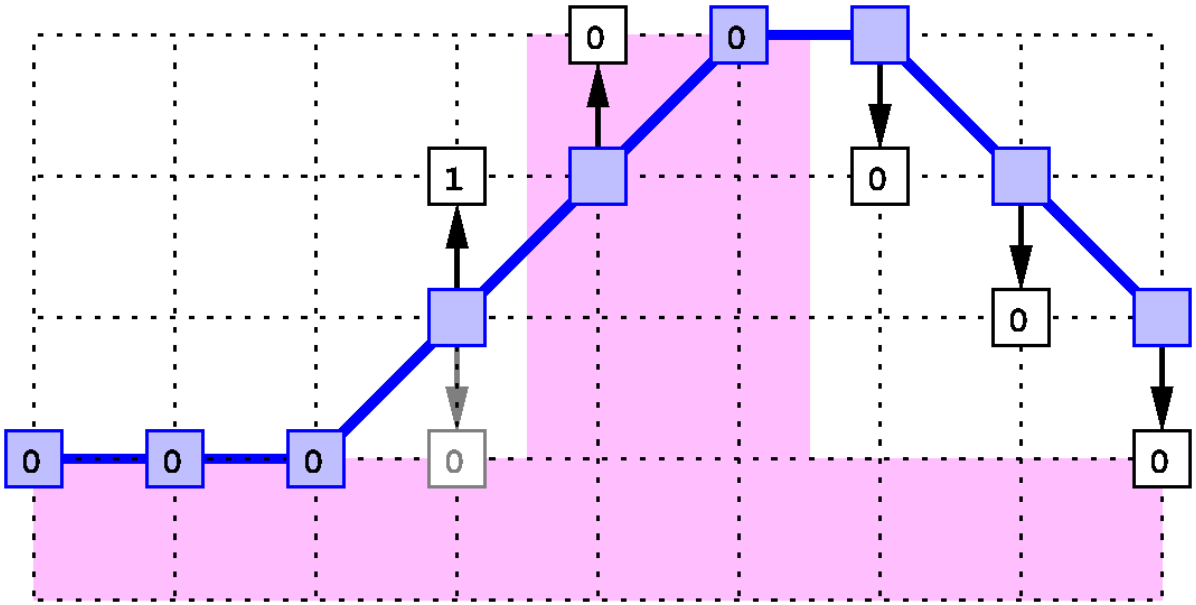
Previous implementation



evaluate local refinement criteria
share levels with neighbors

Adaptive mesh refinement algorithm

Previous implementation

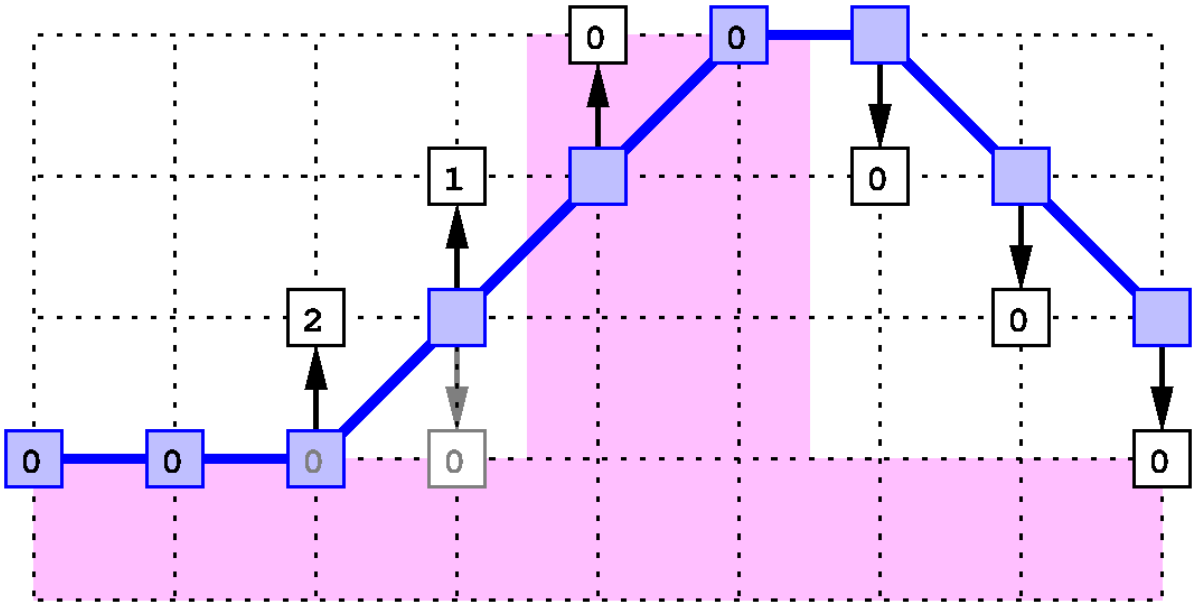


update levels based on current neighbor levels

share updated levels with neighbors

Adaptive mesh refinement algorithm

Previous implementation

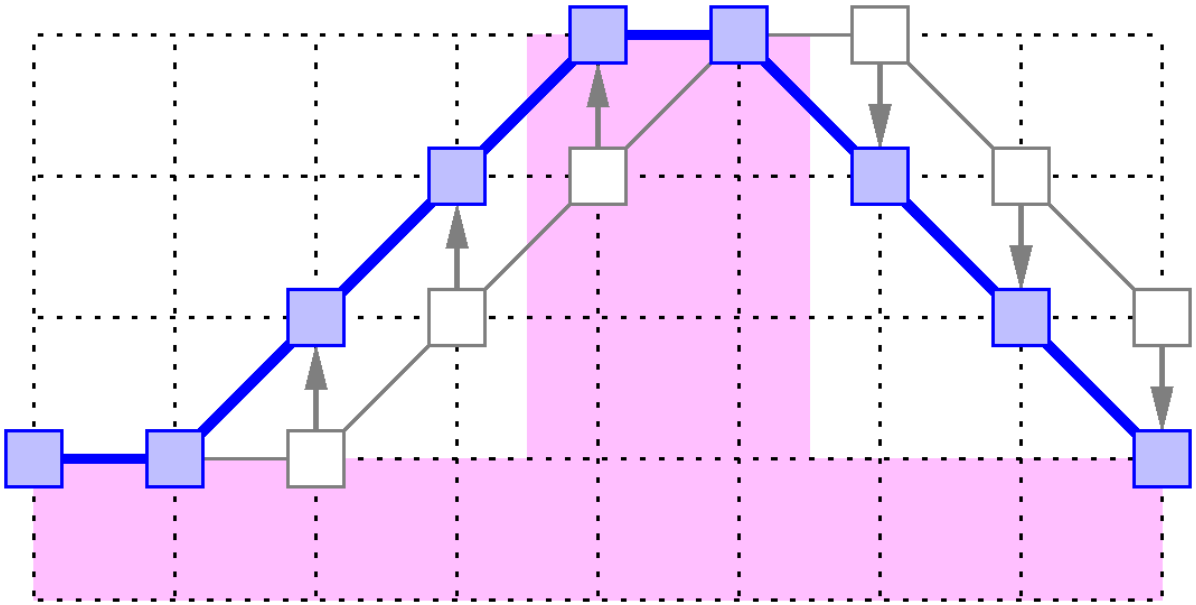


update levels based on current neighbor levels

share updated levels with neighbors

Adaptive mesh refinement algorithm

Previous implementation



terminate with quiescence detection

Adaptive mesh refinement algorithm

Previous implementation

Unfortunately this approach doesn't always work

- Can fail with "level-jump detected" error
- Occurs non-deterministically
- Cause unknown (code? algorithm?)
- Suspect related to quiescence detection

> Charm++ Frequently Asked Questions:
>
> ```
> 6.3.22. Should I use quiescence detection?
>
> Probably not. ...
> ```

- Our revised approach is designed to avoid QD

Adaptive mesh refinement algorithm

Revised implementation: maintain bounds on levels

- Our revised approach[1] avoids QD by maintaining *level bounds*
  - initial lower bound $\underline{L}_i^{k+1}$ initialized to $\hat{L}_i^{k+1}$ (refinement criteria)
  - initial upper bound $\overline{L}_i^{k+1}$ initialized to $L_i^k + 1$
- based on current bounds of neighbors
  - lower-bound raised if needed
  - upper-bound lowered if possible
- block is done ("committed") when $\underline{L}_i^{k+1} = \overline{L}_i^{k+1}$
- terminate when all blocks are committed
  - global reduction via `contribute()`

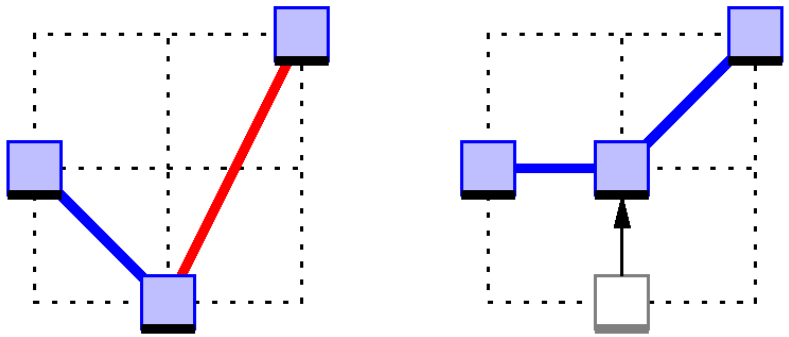
[1] [Edit: This algorithm was originally developed by Phil Miller and presented in his Ph.D. thesis *Reducing synchronization in distributed parallel programs* (2016).]

Adaptive mesh refinement algorithm

Revised implementation: updating level bounds

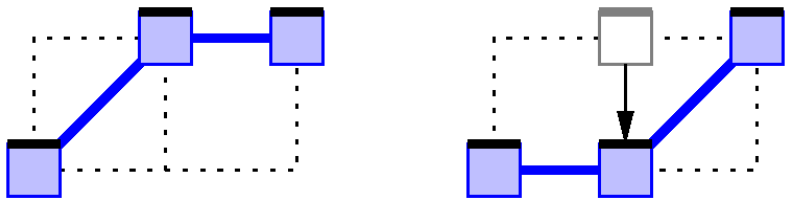
Increase lower bound if conflict with **any** neighbor lower bound

- $\underline{L}_i^{k+1} \leftarrow \max \underline{L}_i^{k+1}, (\underline{L}_j^{k+1} - 1)$



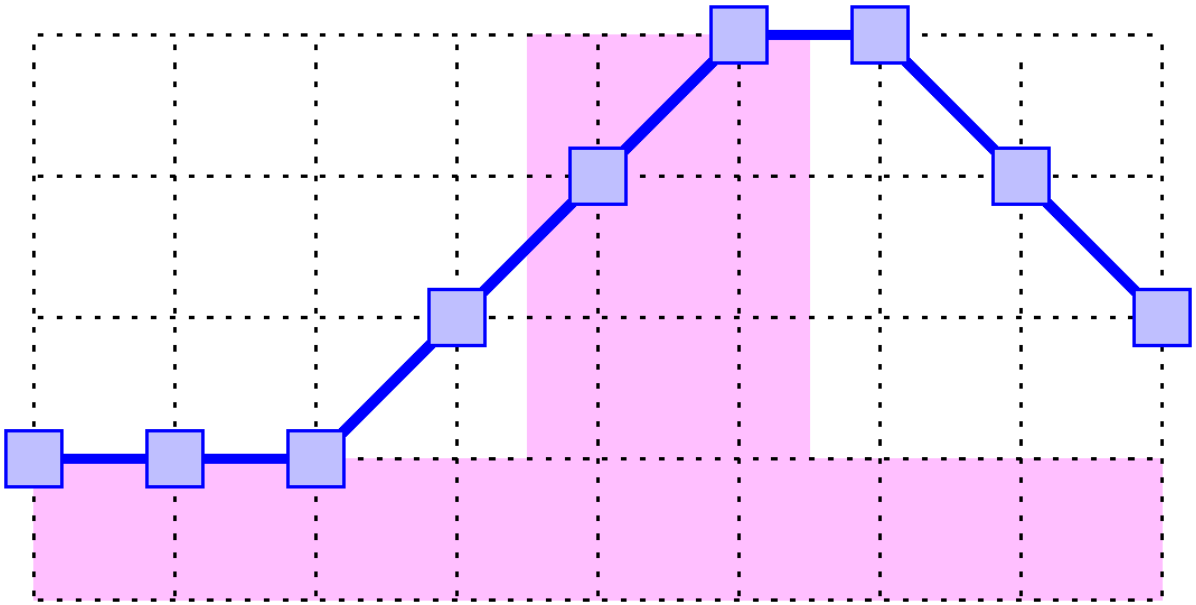
Decrease upper bound if consistent with **all** neighbor upper bounds

- $\overline{L}_i^{k+1} \leftarrow \max \underline{L}_i^{k+1}, (\overline{L}_j^{k+1} - 1)$



Adaptive mesh refinement algorithm

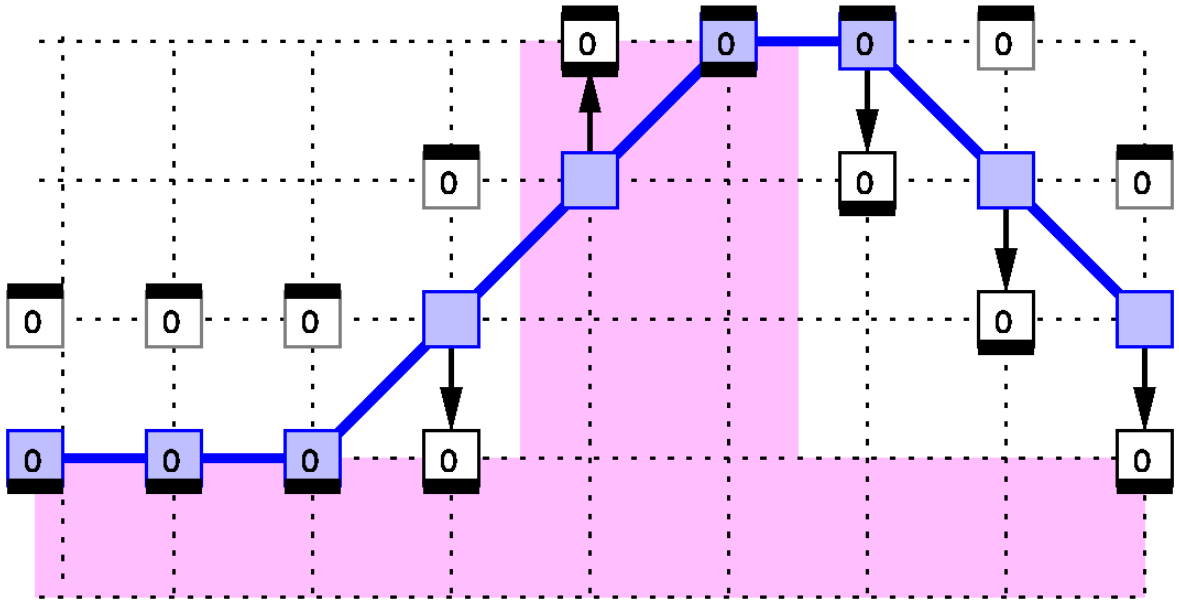
Revised implementation



updated refinement criteria

Adaptive mesh refinement algorithm

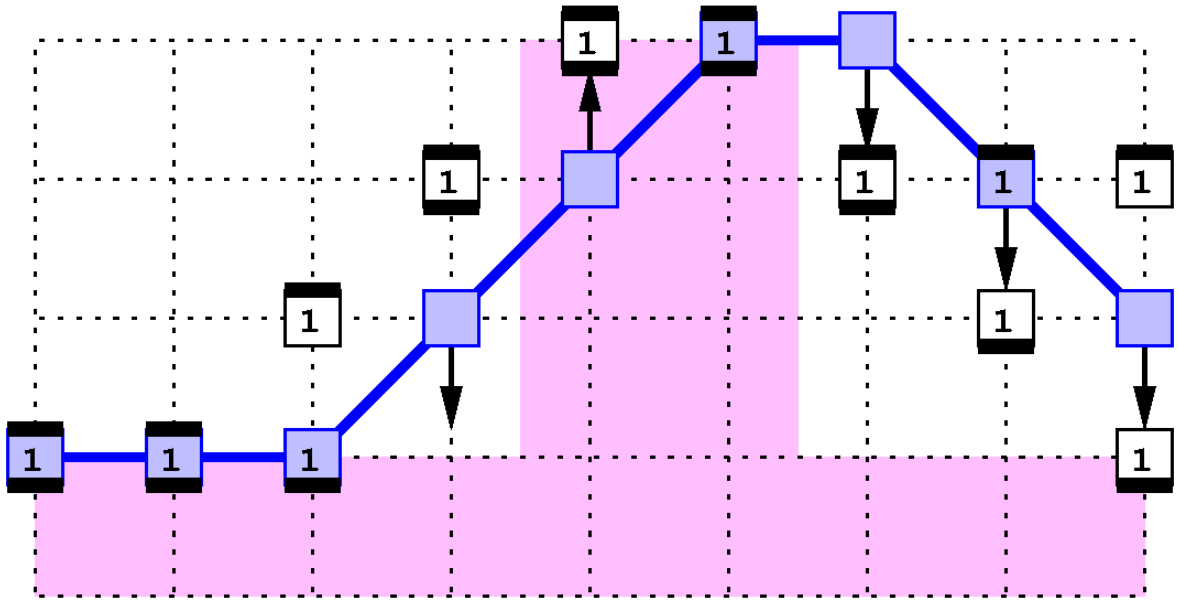
Revised implementation



evaluate local refinement criteria
initialize local lower and upper bounds
share bounds with neighbors

Adaptive mesh refinement algorithm

Revised implementation

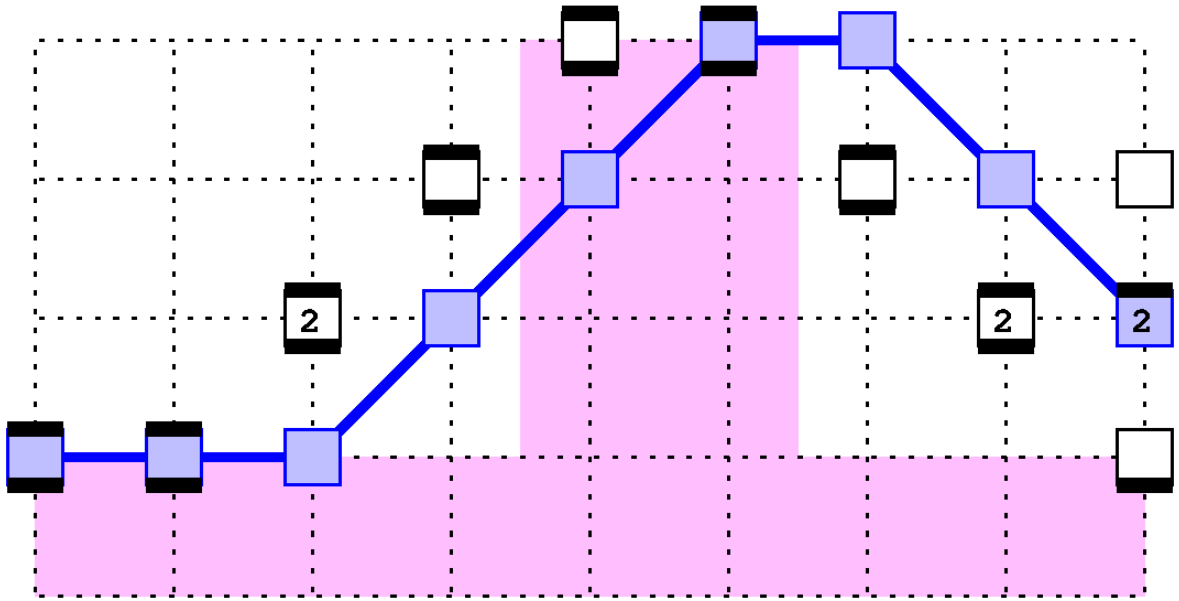


update bounds based on current neighbor levels

share updated bounds with neighbors

Adaptive mesh refinement algorithm

Revised implementation

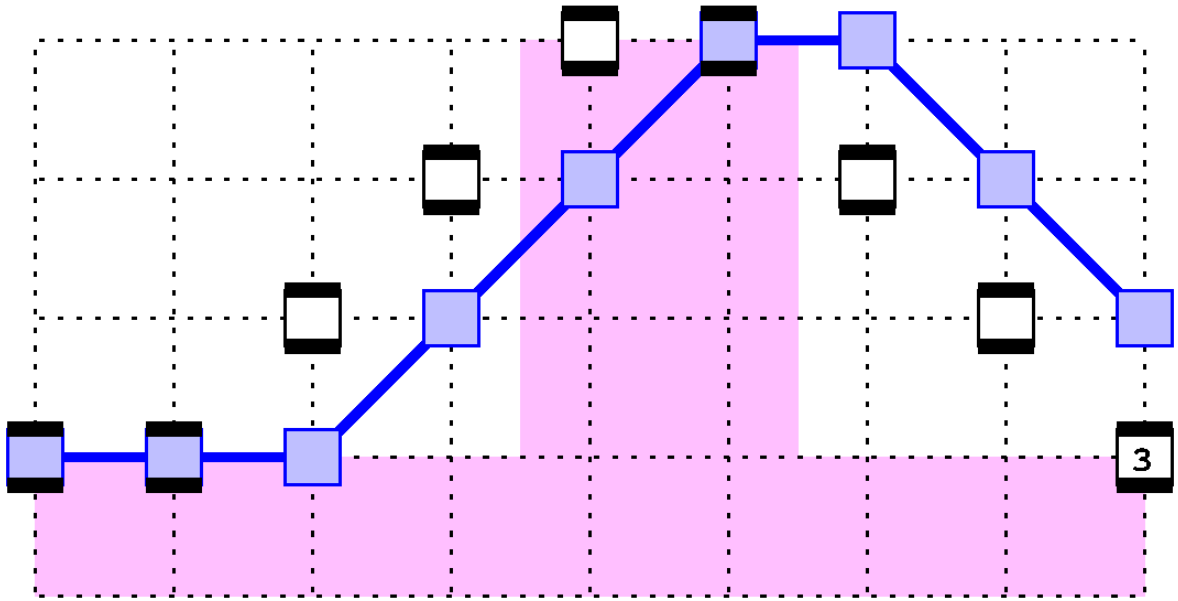


update bounds based on current neighbor levels

share updated bounds with neighbors

Adaptive mesh refinement algorithm

Revised implementation

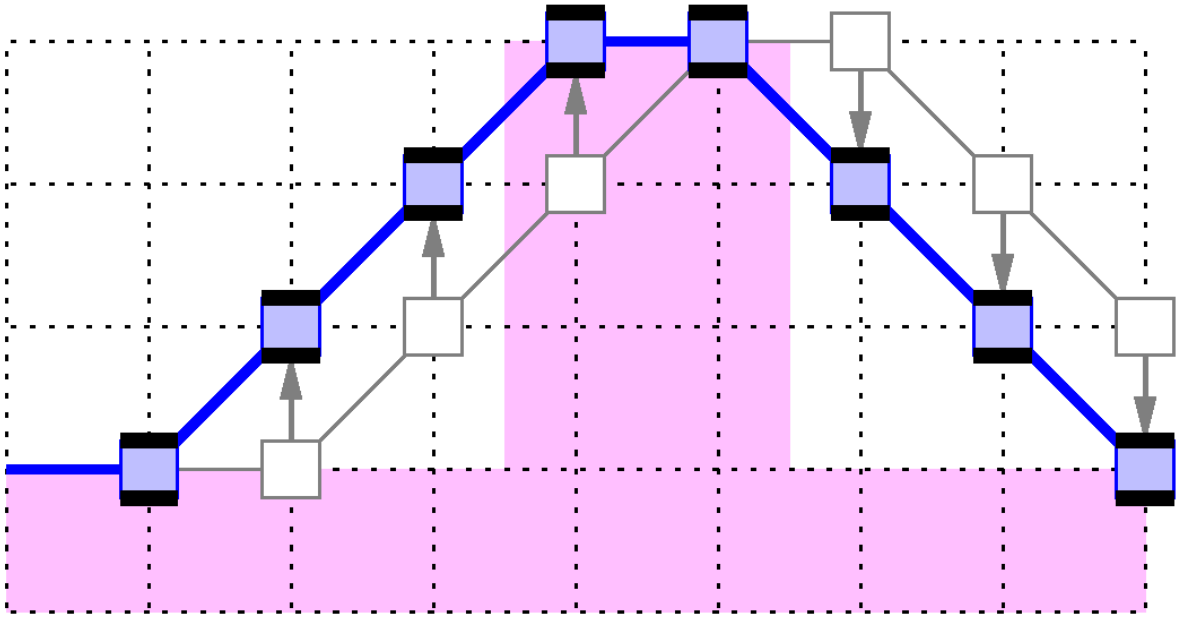


update bounds based on current neighbor levels

share updated bounds with neighbors

Adaptive mesh refinement algorithm

Revised implementation



terminate when all block levels are committed

`contribute()`

Conclusions

- Enzo-E/Cello HDF5 I/O scaling has been greatly improved
  - Not restricted to one file-per-processor
  - Much reduced load on MDS for HDF5 input
  - Plans for further improvement
    - improve I/O load-balancing via space-filling curves
- Enzo-E/Cello mesh refinement robustness improvement in progress
  - Current implementation can fail with "level-jump" errors
  - Updated algorithm formulated to bypass need for QD

Funding for development of Enzo-E/Cello has been provided by the National Science Foundation grants OAC-1835402, SI2-SSE-1440709, PHY-1104819, and AST-0808184. http://cello-project.org/

□