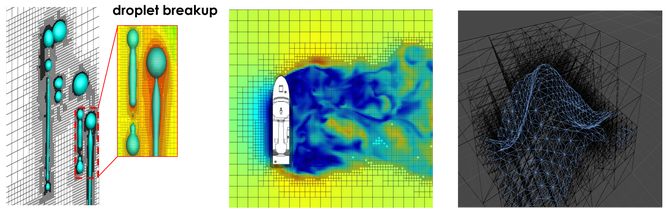
Large-scale Adaptive Mesh Simulations Through Non-Volatile Byte-Addressable Memory

Bao Nguyen **Hua Tan** **Xuechen Zhang**

Washington State University Vancouver

Octree-based Meshing is Widely Used

- **The simulations of complex physical phenomena**



(a) Droplet ejection of injet (b) Air flow distortion simulation (c) 3D computer graphic

- Problems with current in-core solutions:
  - Memory demands are significant
  - DRAM does not scale well

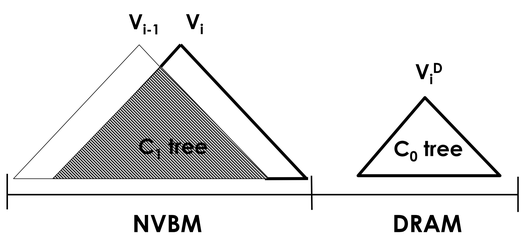
Challenges of Octree-based Meshing on NVBM

- **Using NVBM for adaptive meshing is promising:**
  - Non-volatile
  - Byte-addressable
  - Power-efficient
  - Latencies comparable to DRAM
- **Challenges**:
  - NVBM writes incur high latency while Octree meshing operations is write-intensive.
  - Existing octree data structure is not durable for NVBM.
  - Complicated handling for special pointers during failure recovery between persistent and volatile octants.

Existing Solution

- **Consistent data structures**: CDDS B-Tree [FAST'11]
  - Principles: A multi-version data structure
  - Limitations: Designed for B-tree not Octree
- **In-core meshing algorithms**: top-down[SC' 05] and bottom-up [SC' 07][SIAM' 08]
  - Principles: Parallel meshing in DRAM
  - Limitations: Data structure is not durable upon failures
- **Out-of-core algorithms**: Etree [SC' 04]
  - Principles: Store octants as pages on disks
  - Limitations: Designed for slow storage devices

Our Solution: Persistent Merged Octree

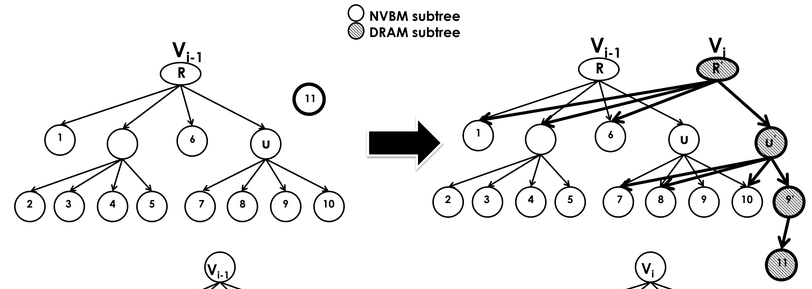


V_{i-1} : root node of persistent tree versions at time T_{i-1}
V_i : root node of active tree versions at time T_i
V_i^D : root node of the C_0 subtree
C_0 : frequently accessed subtree stored in DRAM
C_1 : less frequently accessed subtree stored in NVBM

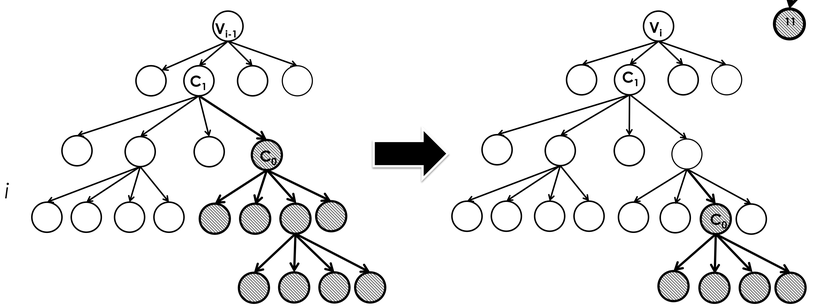
Effectively exploit NVBM for memory extension; Near-instantaneous failure recovery achieved by handling multi-version tree.

Basic Operations of Persistent Merged Octree (PM-Octree)

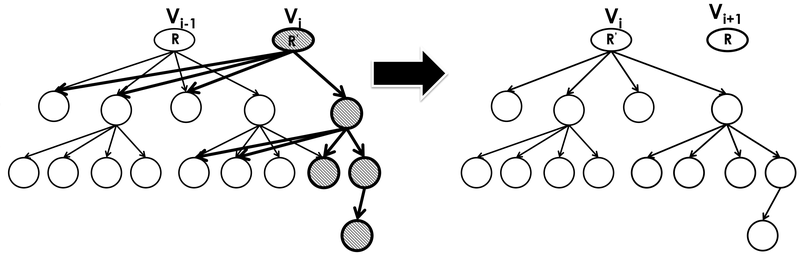
- **Insert Octant 11**
  - Only keep track of 2 tree versions V_{i-1} and V_i.
  - V_{i-1} is use for recovery when failure occurs.
  - Copying overhead significantly is reduced.



- **Merge C_0 with C_1**
  - Subtree of C_0 in DRAM is trimmed and merged to tree C_1 NVBM when $Threshold_{DRAM}$ is reached or simulation of time step *i* is completed.

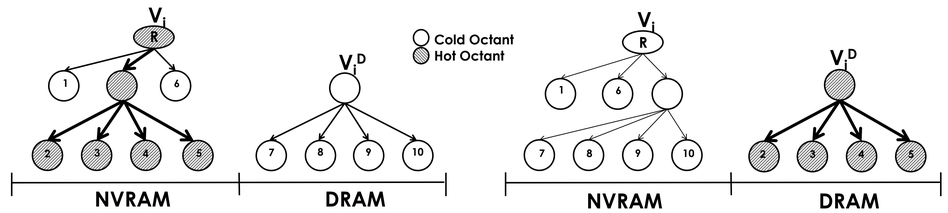


- **Create a persistent tree**
  - Persistent V_i tree will be stored in NVBM.
  - Garbage Collector deleted marked octants in V_{i-1}.
  - V_{i-1} will be V_{i+1}, ready for next simulation round.



Dynamic Layout Transformation for Reducing Write Latency on NVBM

- Transformation is carried out by the end of each time step



(a) Locality oblivious layout w/o transformation (b) Locality aware layout with transformation

A feature-directed sampling approach is used to identify hot octants; NVBM induced additional memory latencies can be effectively shielded.

Program Interface of PM-Octree Data Structure

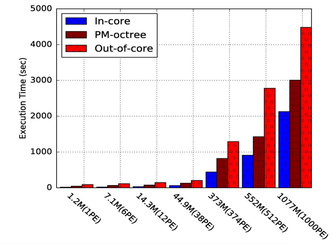
Routine	*Description*
pmoctree * pm_create(octree * tree)	create a new PM-octree; return a pointer to V_i
void pm_persistent(pmoctree * tree)	create a persistent version of octree
pmoctree * pm_restore(void)	restore a PM-octree; return a pointer to V_i
void pm_delete(pmoctree * tree)	delete all octants on NVBM and DRAM

Evaluation

- Titan supercomputer at Oak Ridge National Laboratory
- Cray Linux Environment - 16-core AMD Opteron-6274
- 18,688 nodes interconnect by Gemini network
- Emulation of NVBM on DRAM
- Gerris Flow Solver (GFS) application

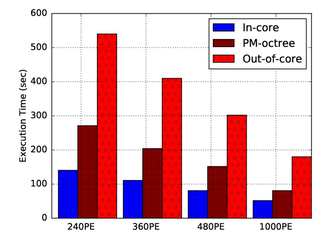
Scalability

- **Weak Scalability**
  - Problem size increases from 1.2M to 1077M elements.
  - Number of PEs increases from 1 to 1000 PEs.
  - Number of element on each PE: ~1 Million.



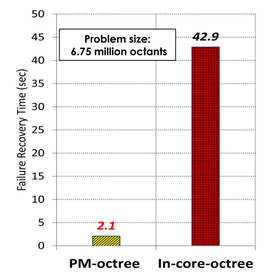
Execution time of PM-octree increases just as a logarithm of the problem size.

- **Strong Scalability**
  - Problem size is kept constant [150 millions elements].
  - Number of PEs increases from 1 to 1000 PEs.



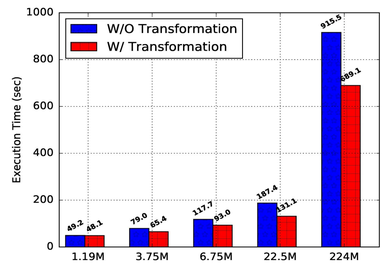
PM-octree performance is close to an ideal speed-up; Similar scalability as in-core-octree.

Failure Recovery



Guarantee data consistency after failures.

Transformation



Execution time is reduced by ~25%; No. of write to NVBM is reduced by ~30%.

Conclusion and Future Work

- **Persistent merged octree**
  - It has good scalability and failure recovery through NVBM.
  - It has easy-to-program interface and frees users from error-prone and tedious tasks of persistent pointer management.
- **Future works:**
  - Automate $Threshold_{DRAM}$ for C_0 tree setting for better efficiency and PM-octree testing with other flow solvers.