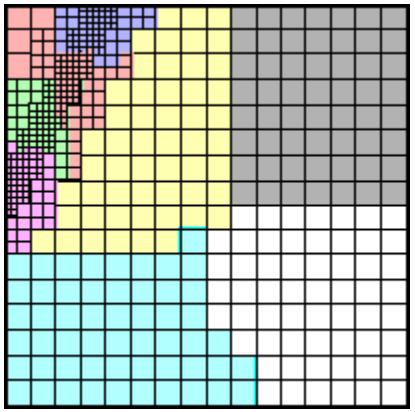
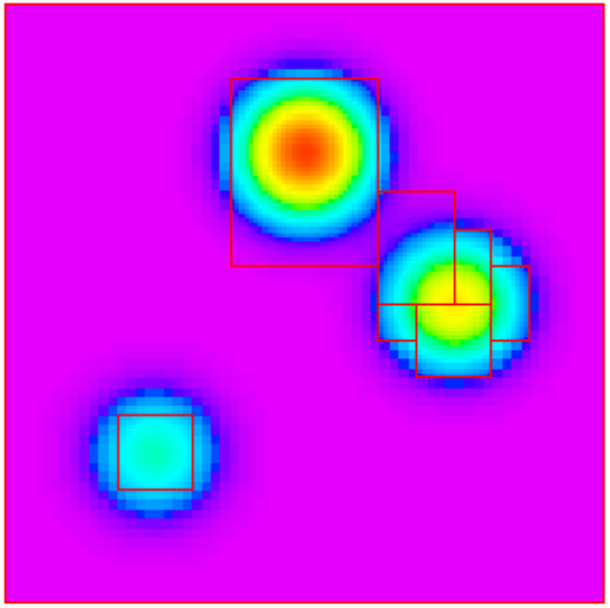
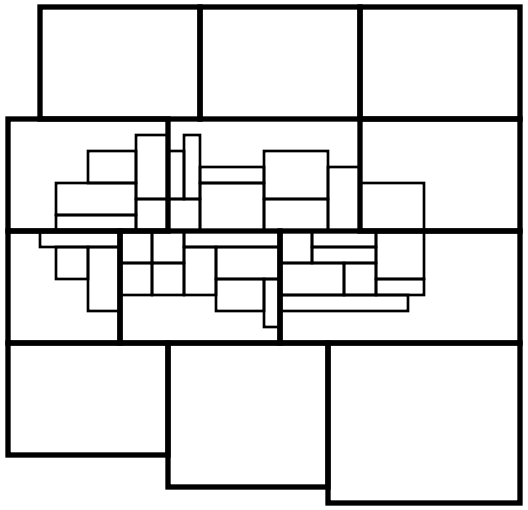
Evaluation and Optimization of a Titanium Adaptive Mesh Refinement



Amir Kamil
Ben Schwarz
Jimmy Su



Adaptive Mesh Refinement



- AMR uses hierarchy of grids, with fine grids nested inside coarse ones.
- Grids are refined only where extra resolution is needed.

- Grid-based V-cycle code written in Titanium
- Severely limited by communication overheads

  Communication is between levels on the V-cycle: Interpolation
  Communication within levels of a V-cycle: Boundary Exchange

Unix Processes vs. Pthreads

- Pthreads expected to perform better, due to shared memory communication.
- However, some parts of the code perform worse with Pthreads.

Time in seconds (processes/Pthreads per process)

	1/1	2/1	1/2	4/1	1/4	8/1	1/8
SolveAMR	235.4	164.0	140.3	147.3	91.05	128.3	87.47
Exchange	84.16	84.68	48.10	97.46	31.64	99.05	41.63
GSRB	59.97	29.69	30.58	14.69	15.29	7.22	7.12
CFInterp	45.4	24.28	35.02	16.03	29.61	10.10	30.61
Initialization	41.68	31.60	53.82	27.04	67.61	29.91	92.17

Static vs. Instance Accesses

- Static variable accesses are slow in Titanium.
- This slowdown is much greater with Pthreads (~7 times slower than with processes).
- Indirect accesses using wrappers can eliminate much of the cost.

Time in seconds (processes/Pthreads per process)

	10M accesses		100M accesses	
	8/1	1/8	8/1	1/8
Instance	0.0342	0.0344	0.335	0.335
Static	1.68	11.6	16.8	115.4
Indirect	0.241	0.242	2.41	2.42

- This only accounts for a small fraction of the slowdown in AMR.

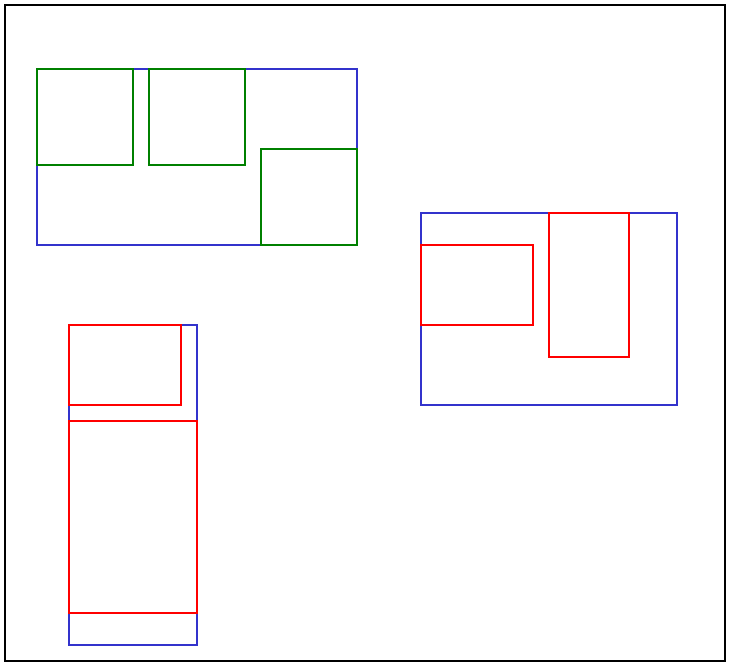
Cache Misses

- Since Pthreads share the heap segment of memory, false sharing can occur, resulting in many more cache misses.
- Slowest two lines of code in initialization show many more data cache misses with Pthreads.

Time and cache misses (processes/Pthreads per process)

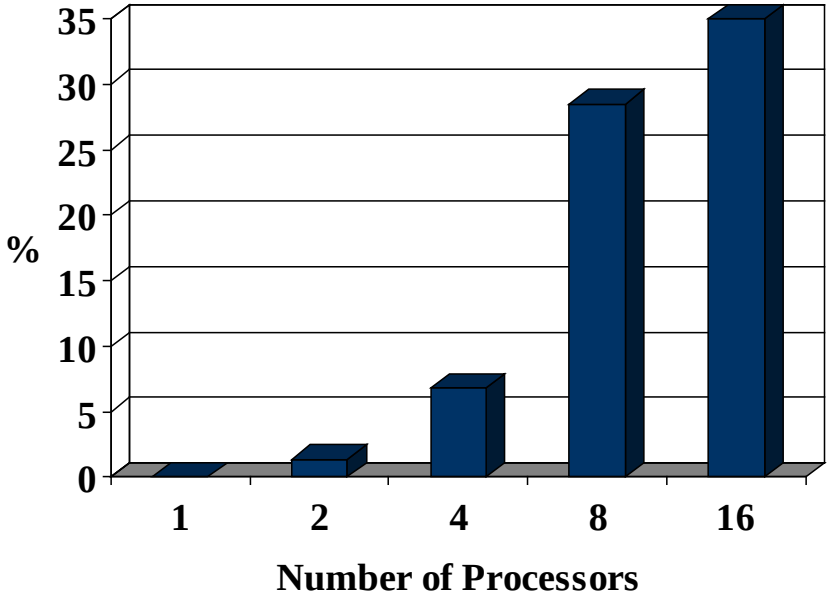
	Time (seconds)		Cache misses (millions)	
	8/1	1/8	8/1	1/8
Line 1	5.67	16.72	7.54	48.6
Line 2	6.49	21.54	8.41	54.1

Load Balancing



- Rough estimate: Time spent waiting at barriers

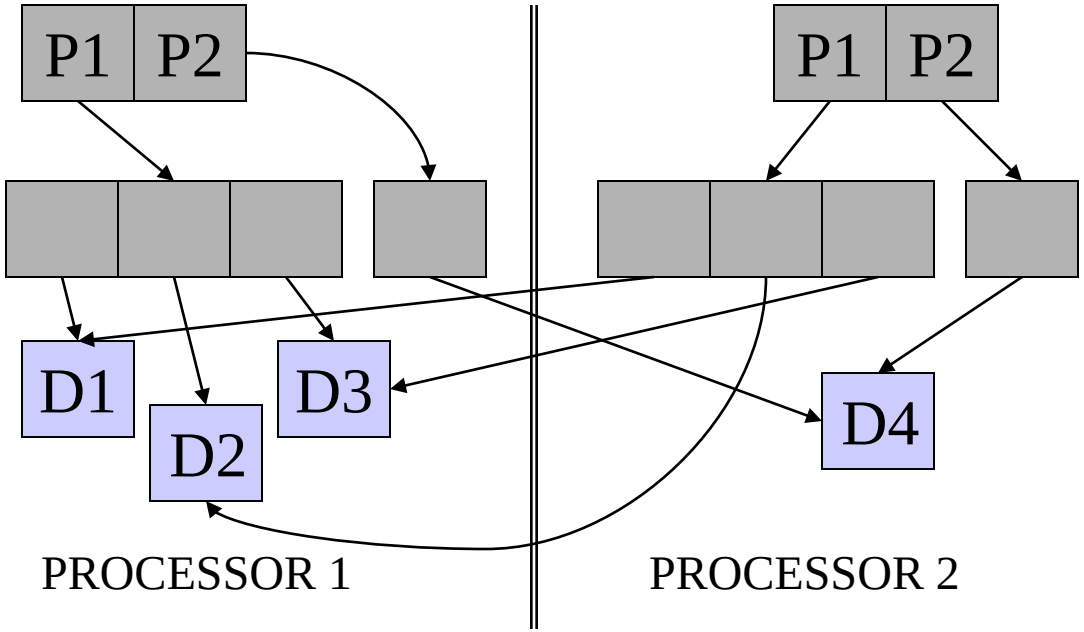
  Different data sets for each configuration



- Conflicts with optimizing to reduce communication
  - Communication between coarse grids and fine grids for interpolation. Favors allocation of overlapping grids to a single processor so interpolation requires no communication.

Optimizing with Communication Primitives

- Titanium *exchange* over *broadcast* during initialization



Distributed Data Structure

Broadcast

Broadcast each data block (D) to everyone else. Broadcasts execute serially.

Exchange

Exchanges can happen in parallel.

* Results Pending

Exchange

- The exchange operation copies the elements in the overlap sections between boxes.
- The operation accounts for 30% of total running time for the sequential backend.
- The original code has two implementations of the exchange operations.
  - Array copy
  - Foreach loop over the intersection domain

Exchange Optimizations

- Domain intersection calculation accounts for 46% of the running time of exchange.
- Amortize the intersection cost by caching the calculation
- About 20% of the intersections only has a single element in it.
  - Most resulted from the intersections of corners of two boxes, which are not needed in the calculation.
  - 44% speedup in exchange by checking for this case

Box Overlap Representation

- Both the boxes and box intersections are static throughout the execution of the program.
- Box overlaps are currently represented as intersections of Titanium arrays.
- Even with caching of intersection calculations, some overheads are unavoidable.
  - Loop overhead for small intersections
  - Translation of points in a domain to virtual addresses

Experimental Representation

- Represent all intersections as an array of source/dest C pointer pairs
- 40% speedup over the intersection caching version
- Improving on the initial version to reduce the size of the representation
  - Represent small intersections pointer pairs
  - Represent large intersections as a base pointer and strides

Conclusion

- Faster than C++/MPI implementation of Chombo
- More speedups pending due to work in progress