ArTA: *A*daptive G*r*anularity in *T*ransactional *A*pplications

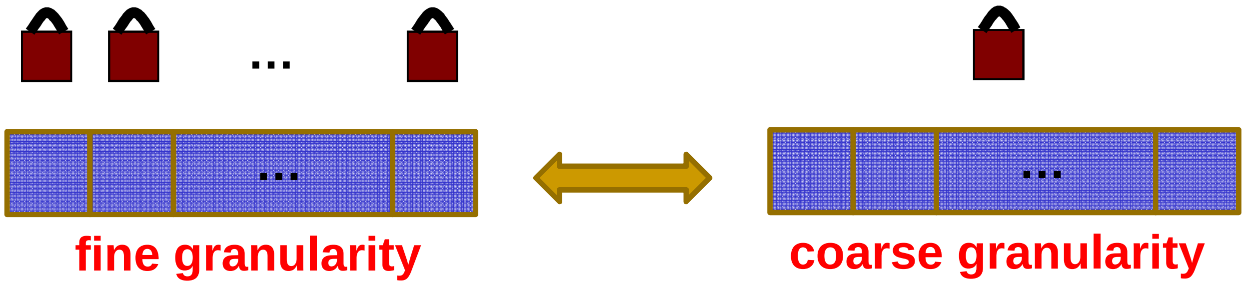
Ehsan Atoofian

Electrical Engineering Department

Lakehead University

Motivation

- Software Transactional Memory (STM) exploit locks to synchronize accesses to the shared memory locations



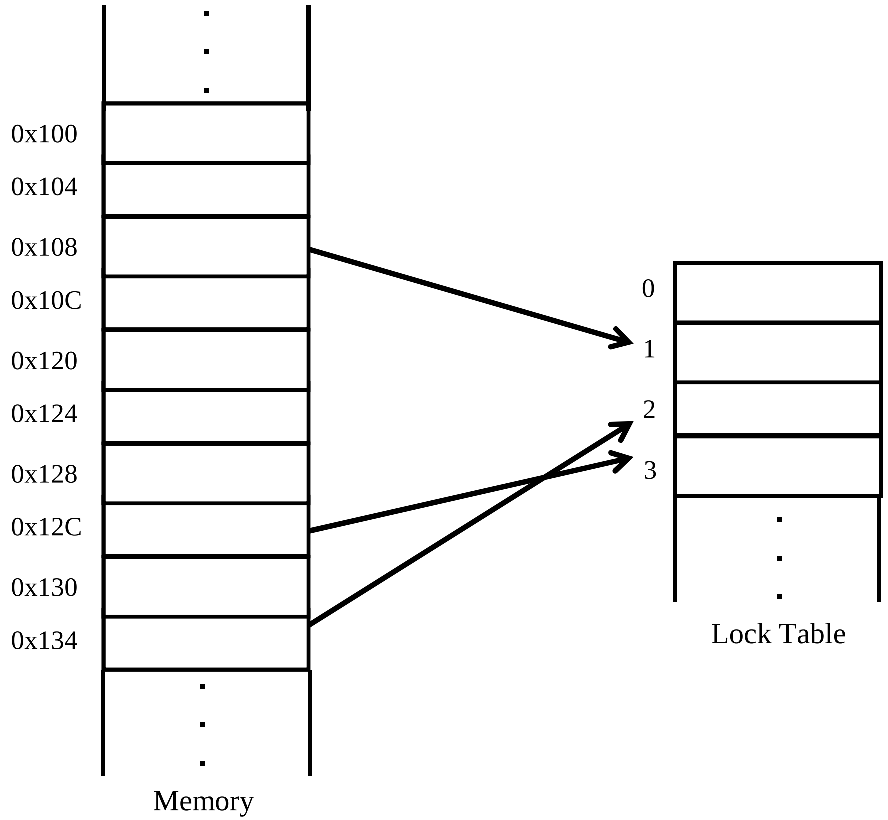
- *A*daptive G*r*anularity in *T*ransactional *A*pplications (ArTA): changes granularity of locks dynamically→Speedup: 27%

Outline

- Motivation
- ArTA
- Results
- Conclusion

Lock in STM

- Memory addresses, map to lock table, handle concurrent accesses to shared data
- Lock granularity specifies # of consecutive memory locations mapped to the same entry of the lock table

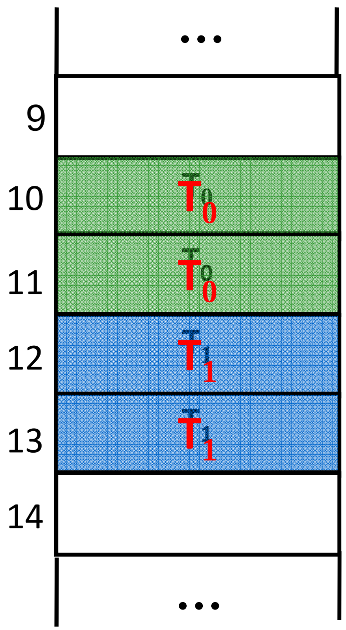


Fine vs. Coarse Granularity Locks

- Fine granularity
  - Pros: increases concurrency
  - Cons: Increases overhead

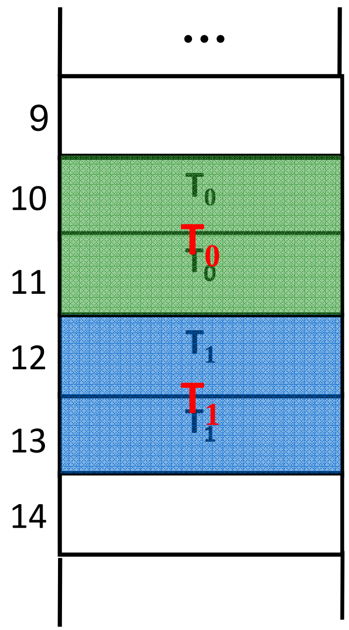
- Coarse granularity
  - Pros: reduces overhead
  - Cos: Increases false conflict

Fine vs. Coarse Granularity Locks



Fine granularity

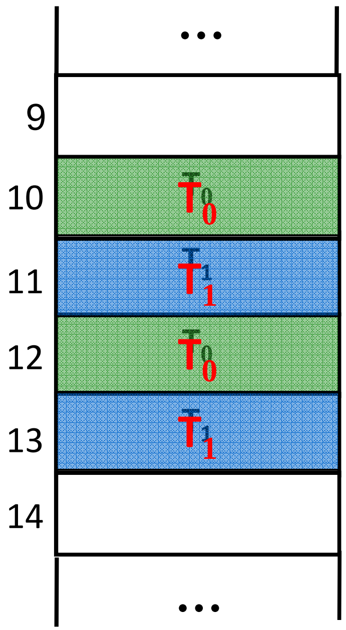
# of locks: 4
Execute concurrently



Coarse granularity

# of locks: 2
Execute concurrently

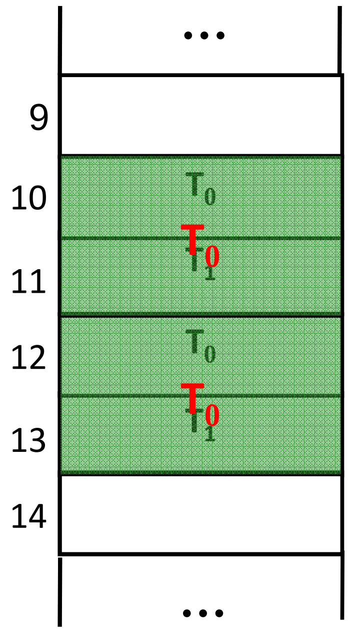
Fine vs. Coarse Granularity Locks



Fine granularity

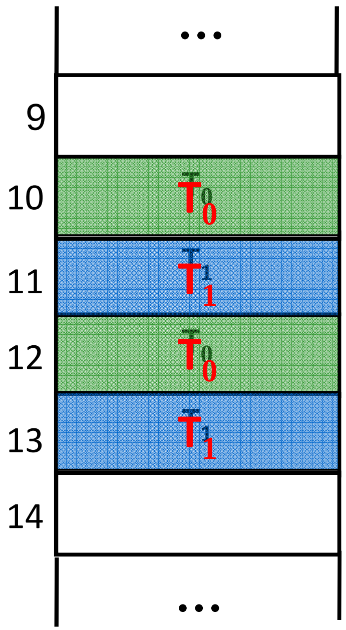
# of locks: 4

Execute concurrently



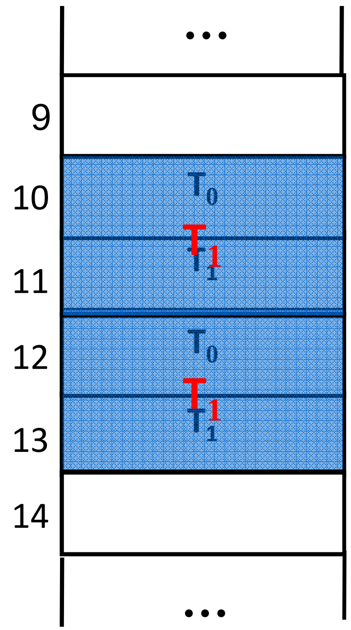
Coarse granularity

Fine vs. Coarse Granularity Locks



Fine granularity

# of locks: 4
Execute concurrently

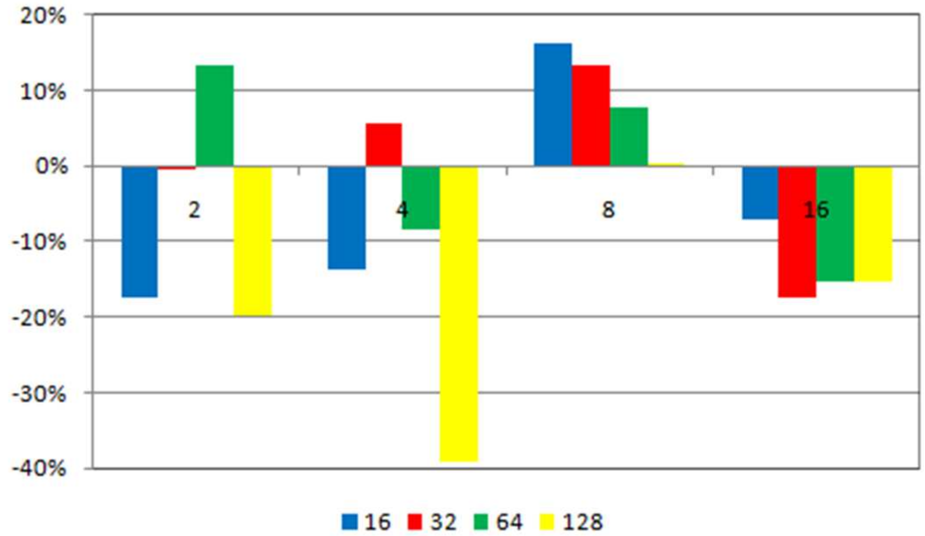


Coarse granularity

# of locks: 4
Execute serially

Variable Granularity Locks

- Fine vs. coarse granularity in Labyrinth
- # of threads changes 2~16
- Lock granularity changes 16~128
- Performance varies -39%~16%

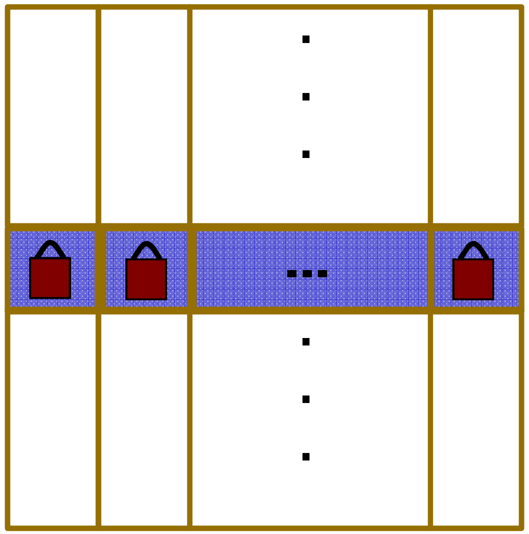


Outline

- Motivation
- ArTA
- Results
- Conclusion

Lock Granularity in Kmeans

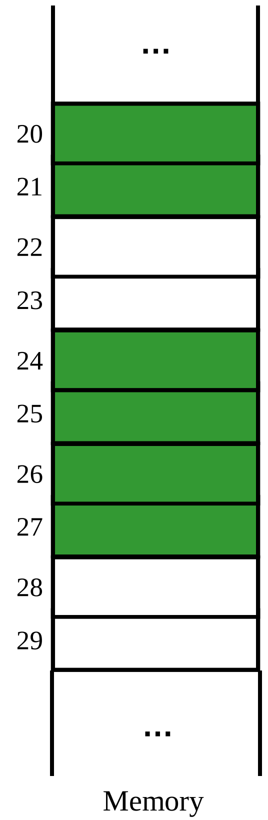
```
float **new_centers;
…
TM_BEGIN();  //start of transactional section
…
for (j = 0; j < 32; j++) {
     TM_SHARED_WRITE( new_centers[index][j],  …);
    }
TM_END();  //end of transactional section
```



Fine Granularity

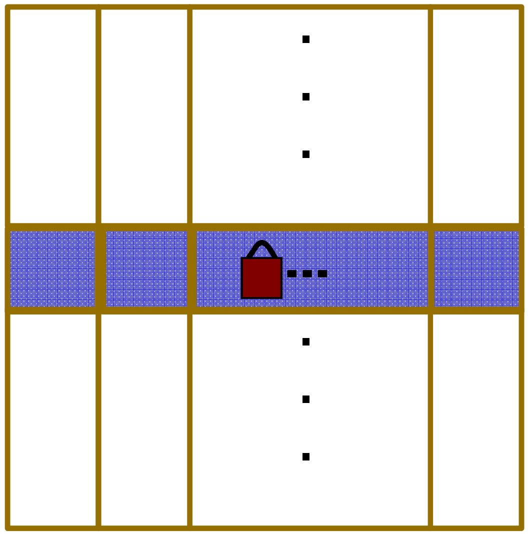
ArTA

- Monitors transactional write operations
- Continuous memory accesses form a group
- ArTA selects the smallest group for granularity of the lock



ArTA in Kmeans

```
float **new_centers;
…
TM_BEGIN();  //start of transactional section
…
for (j = 0; j < 32; j++) {
    TM_SHARED_WRITE( new_centers[index][j],  …);
  }
TM_END();  //end of transactional section
```



- ArTA sets lock granularity to row size in new_centers[][]

ArTA in TXs with Different Granularities

```
while (1)
{
...
TM_BEGIN();  //first transaction
coordinatePairPtr = (pair_t*) TMQUEUE_POP (headPtr);
TM_END();

...

TM_BEGIN();  //second transaction
for (i = 0; i <n; i++) {
        TM_SHARED_WRITE(&(vectorPtr[i]) , …);
}
TM_END();
...
}
```

Labyrinth Benchmark

Saturating Counter

- A Saturating Counter (SC) improves confidence of prediction in ArTA
- SC
  - Incremented, if granularity of two consecutive transactions are the same
  - Reset to zero, otherwise
- ArTA is trusted only if SC>threshold

ArTA in Labyrinth

```
while (1)
{
...
TM_BEGIN();  //first transaction
coordinatePairPtr = (pair_t*) TMQUEUE_POP (headPtr);
TM_END();

...

TM_BEGIN();  //second transaction
for (i = 0; i <n; i++) {
            TM_SHARED_WRITE(&(vectorPtr[i]) , ...);
}
TM_END();
...
}
```

-Different Granularities
-SC=0

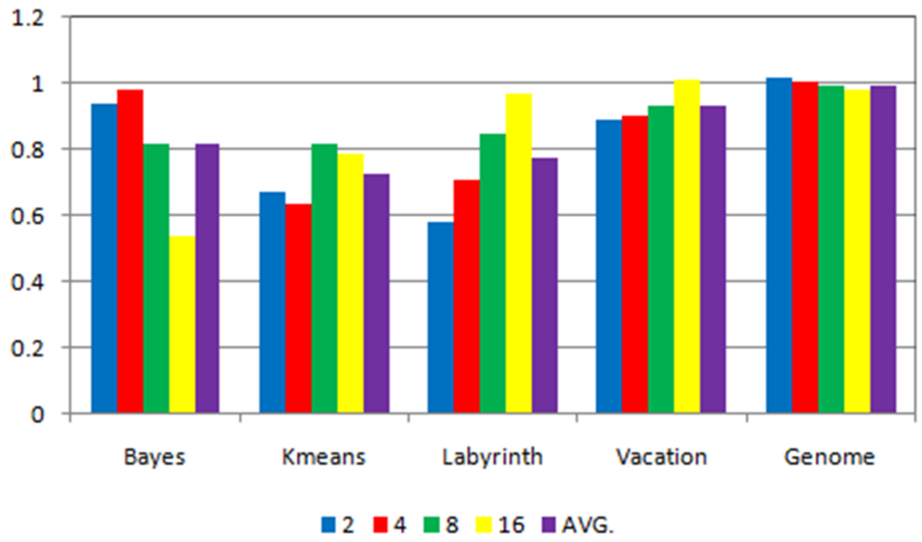
Labyrinth Benchmark

Experimental Framework

- Benchmarks: Stamp v0.9.7
  - Run up to competition
  - Measured statistics over 10 runs
- TL2 as an STM framework
  - Lock table with 1M entries
- Two Intel Xeon E5405, quad core processors

Speedup in ArTA

- 2-bit saturating counter with threshold=1
- Kmeans, 27% improvement on average
- Genome, less than 1%
- Vacation, Bayes, Labyrinth, 7%, 18%, 22%,



Conclusion

- Applications react differently to lock granularity
- ArTA is a speculative approach and dynamically changes lock granularity
- ArTA improves performance of STMs up to 27% on average

Thank You!

Questions?