View-Oriented Transactional Memory

Kai-Cheung Leung
Zhiyi Huang

University of Otago

P2S2 2011

Locks vs Transactional Memory (TM)

- Parallel programming is becoming mainstream
- Parallel programming models need to facilitate both performance and convenience
- In shared-memory models, Shared data generally manged either by:

  Locking Each shared object needed to be accessed atomically is protected by a lock. Lock acquired manually before access and released after access

  TM transactions used to access shared data atomically. All processes enter transactions freely and commit at the end of transactions, and if conflict occurs, one or more transactions abort and restart

- Problems in lock-based models:
  - Manually arranging fine-grain locks is tedious, and prone to errors such as deadlock and data race
  - Coarse grain locks has little concurrency
- Problems in TM models:
  - When conflict rare, encourage high concurrency, but...
  - When conflict high, transactions can abort each other and little progress is made

SAPERE • AUDE

Solution: Restricted Admission Control (RAC)

- Shared memory is like a room, and
- traditional TM models freely admits anyone into the room regardless of contention.
- RAC is like the doorman, who limits the number of people in the room depending on contention.
- RAC allows Q people in the room at a given time. $1 <= Q <= N$
- When $Q = N$, unrestricted admission, likes traditional TM
- When $Q = 1$, likes lock

Another problem...

- However contention in different places in a room is different
- e.g. many people fight for access to the PlayStation in the room,
- but few hard-working students are interested in accessing the bookself at the other side of the room
- However unreasonable to restrict access to the book because of high contention in the PlayStation, and would unnecessarily impede concurrency of the people (processes) wanting to read the books on the bookshelf

SAPERE·AUDE

Solution: View-Oriented Transactional Memory (VOTM)

- View-Oriented Parallel Programming (VOPP) a data-centric model which:
  - Variables private to the process by default
  - Each shared object must be explicited declared as “views”
  - Views must not overlap
  - Views are acquired before access and released after access
- VOTM is to control access to each view with TM, where:
  - A transaction begins when the view is accessed and ends when the view is released
  - Therefore shared data that can be accessed together can be put into the same view
  - Now each view is guarded by its own doorman (RAC) **individually** given the contention of the view
  - Therefore when admission to the popular PlayStation is restricted, access to the bookshelf is not affected

SAPERE•AUDE

Little instrumentation needed to parallelize existing code with VOTM

```c
typedef struct Node_rec Node;

struct Node_rec {
  Node *next;
  Elem val;
};

typedef struct List_rec {
  Node *head;
} List;

List *ll_alloc(vid_type vid) {
  List *result;
  create_view(vid, size, 0);
  result = malloc_block(vid, sizeof(result[0]));
  acquire_view(vid);
  result->head = NULL;
  release_view(vid);
  return result;
}
```

Figure: Code snippet of list allocation in VOTM

SAPERE•AUDE

```
void ll_insert(List *list, Node *node, vid_type vid) {
  Node *curr;
  Node *next;

  acquire_view(vid);

  if (list->head->val >= node->val) {
    /* insert node at head */
    node->next = list->head;
    list->head = node;
  } else {
    /* find the right place */
    curr=list->head;
    while (NULL != (next = curr->next) &&
           next->val < node->val) {
      curr = curr->next;
    }
    /* now insert */
    node->next = next;
    curr->next = node;
  }
  release_view(vid);
}
```

Figure: Code snippet of list insertion in VOTM

Performance

Table: Application runtime (s) at $N = 16$

Application	VOTM	TinySTM	Lock-based
TSP $Q = 1$	52.23	194.73	52.23
Intruder	43.05	127.70	100.62
Bayes	11.15	19.51	30.72
Genome	4.93	5.91	37.48
Labyrinth	35.60	35.08	331.28
Vacation	14.84	14.1	61.88
SSCA2	8.80	8.77	56.28

Table: Number of transactions and aborts at $N = 16$

Application	#transactions	VOTM	TinySTM
TSP $Q = 1$	3,925,092	0	4,150,852,440
Intruder	23,428,141	10,986,905	1,238,254,062
Bayes	1,751	4,591	522,972
Genome	2,472,907	83,273	64,595,381
Labyrinth	1056	196	202
Vacation	4,194,304	1,443	1,059
SSCA2	22,362,292	62	64

SAPERE·AUDE

Origin of performance gain - Use of multiple views

- Both Bayes and Intruder have a view for the main data structure, plus one or more queues that are not accessed together with the main data structure
- In VOTM, these queues are allocated in different views
- Performance of multiple-view VOTM surpasses TinySTM+RAC

Table: Performance of VOTM and TinySTM + RAC at $N = 16$

	Application	VOTM	TinySTM + RAC
time(s)	Bayes	11.15	11.97
	Intruder	43.05	59.50
#aborts	Bayes	4591	4587
	Intruder	10986905	10337777

Origin of performance gain - RAC

- Microbenchmarks from Eigenbench confirm RAC can find *Q* to optimize performance

Table: Performance of Adaptive RAC in TinySTM-ETL

Microbenchmark	*time*(s) (RAC)	*Q* (RAC)	#*aborts* (RAC)	*time*(s) (Q16)	#*aborts* (Q16)	*time*(s) (opt)	*Q* (opt)	#*aborts* (opt)
Highcon	25.6	4	9.96k	76.0	648k	25.4	4	7.17k
FutileStall	3.23	1	7.98	40.3	47.3m	4.21	1	0
StarvingElder	47.0	16	3.05m	46.8	3.02m	46.8	16	3.02m
StarvingWriter	22.1	16	33.2m	21.8	10.6m	21.8	16	10.6m

Table: Performance of Adaptive RAC in TinySTM-CTL

Microbenchmark	*time*(s) (RAC)	*Q* (RAC)	#*aborts* (RAC)	*time*(s) (Q16)	#*aborts* (Q16)	*time*(s) (opt)	*Q* (opt)	#*aborts* (opt)
Highcon	15.0	16	2.98k	15.0	3.01k	15.0	16	3.01k
FutileStall	3.33	1	46.5k	7.73	3.01m	4.86	1	0
StarvingElder	51.5	16	1.05m	51.2	1.05m	51.2	16	1.05m
StarvingWriter	19.2	16	3.83k	19.2	3.80k	19.2	16	3.80k

SAPERE · AUDE

Conclusion

- VOTM maximizes both progress and concurrency by allowing shared data with different access pattterns to be allocated into different views and use RAC to optimize of each view individualy according to **its** contention
- Experimental results confirm VOTM has superior performance to both TM and lock-based models
  - When a low-contention view held for long time, VOTM has concurrency of TM, but locks will have poor concurrency.
  - VOTM can reduce admission quota when view has high contention, thus improves progress while maximizes concurrency.

Current Work - RAC theoretical model

- We have developed a theoretical model for RAC, that suggests time spent in aborted and successful transactions should be used to calculate whether the admission quota Q needs to be adjusted:

$$\delta(Q) = \frac{CPUcycles_{aborted\_tx}}{CPUcycles_{successful\_tx} * (Q - 1)} \tag{1}$$

and if $\delta(Q) > 1$, then Q should be decreased

Future Works

- Will extend the theoretical analysis to show how splitting shared data in multiple views improve performance in VOTM
- Implement VOTM on a managed language so that view creation and acceess checked at compile time, to realize the automatic view access management in the Maotai 3.0 paper
  - Maotai 3.0: Automatic Detection of View Access in VOPP, Leung, K.C. and Huang Z., In Proceedings of the Eleventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2010). pp.138-147, IEEE Computer Society (2010), Wuhan.