When and how VOTM can improve performance in contention situations

Kai-Cheung Leung
Yawen Chen
Zhiyi Huang

University of Otago
New Zealand

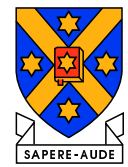
P2S2 2012

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 is acquired before access and released after access

  TM Transactions are 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 is rare, encourage high concurrency, but...
  - When conflict is high, transactions can abort each other and little progress is made

Solution: Restricted Admission Control (RAC)

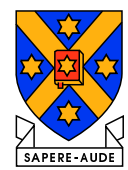
- Shared memory is like a room, and
- traditional TM models freely admits anyone into the room regardless of contention.
- RAC is like the doorman, who limits the number of people in the room depending on contention.
- RAC allows Q people in the room at a given time.
  $1 <= Q <= N$
- When $Q = N$, unrestricted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

Another problem...

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

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



Little instrumentation needed to parallelize existing code with VOTM

```
1  typedef struct Node_rec Node;
2
3  struct Node_rec {
4    Node *next;
5    Elem val;
6  };
7
8  typedef struct List_rec {
9    Node *head;
10 } List;
11
12 List *ll_alloc(vid_type vid) {
13   List *result;
14   create_view(vid, size, 0);
15   result = malloc_block(vid, sizeof(result[0]));
16   acquire_view(vid);
17   result->head = NULL;
18   release_view(vid);
19   return result;
20 }
```

Figure: Code snippet of list allocation in VOTM

```
1  void ll_insert(List *list, Node *node, vid_type vid) {
2    Node *curr;
3    Node *next;
4
5    acquire_view(vid);
6
7    if (list->head->val >= node->val) {
8      /* insert node at head */
9      node->next = list->head;
10     list->head = node;
11   } else {
12     /* find the right place */
13     curr=list->head;
14     while (NULL != (next = curr->next) &&
15            next->val < node->val) {
16       curr = curr->next;
17     }
18     /* now insert */
19     node->next = next;
20     curr->next = node;
21   }
22   release_view(vid);
23 }
```

Figure: Code snippet of list insertion in VOTM

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:

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

  and if $\delta(Q) > 1$, then Q should be decreased
- The RAC model can also be applied individually in each view in multiple-view cases.

VOTM-OrecEagerRedo on a 64-core machine

VOTM prevents livelocks and relieves high contention in application data by restricting access through RAC.

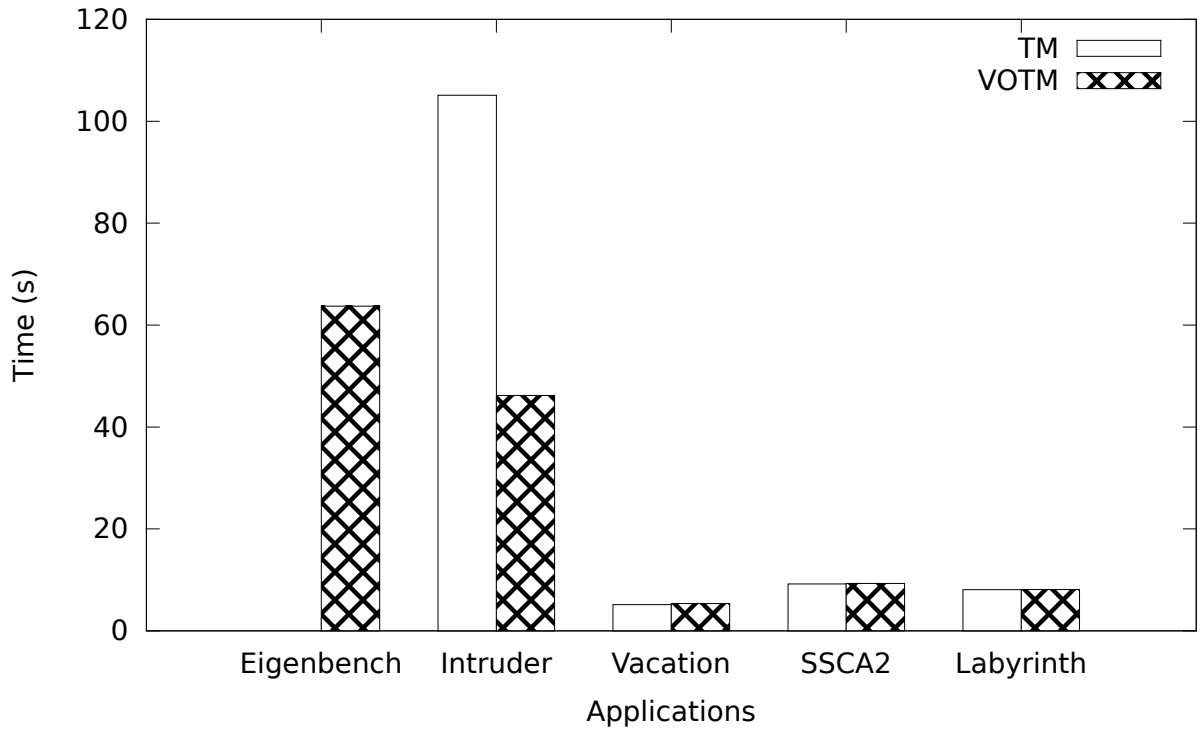
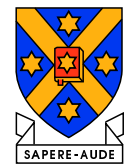


Figure: Single-view applications in VOTM-OrecEagerRedo (Eigenbench on TM is not shown due to livelock)



VOTM can further improve performance by splitting shared data into multiple views, which allows fine-grain access optimization by RAC on each view.

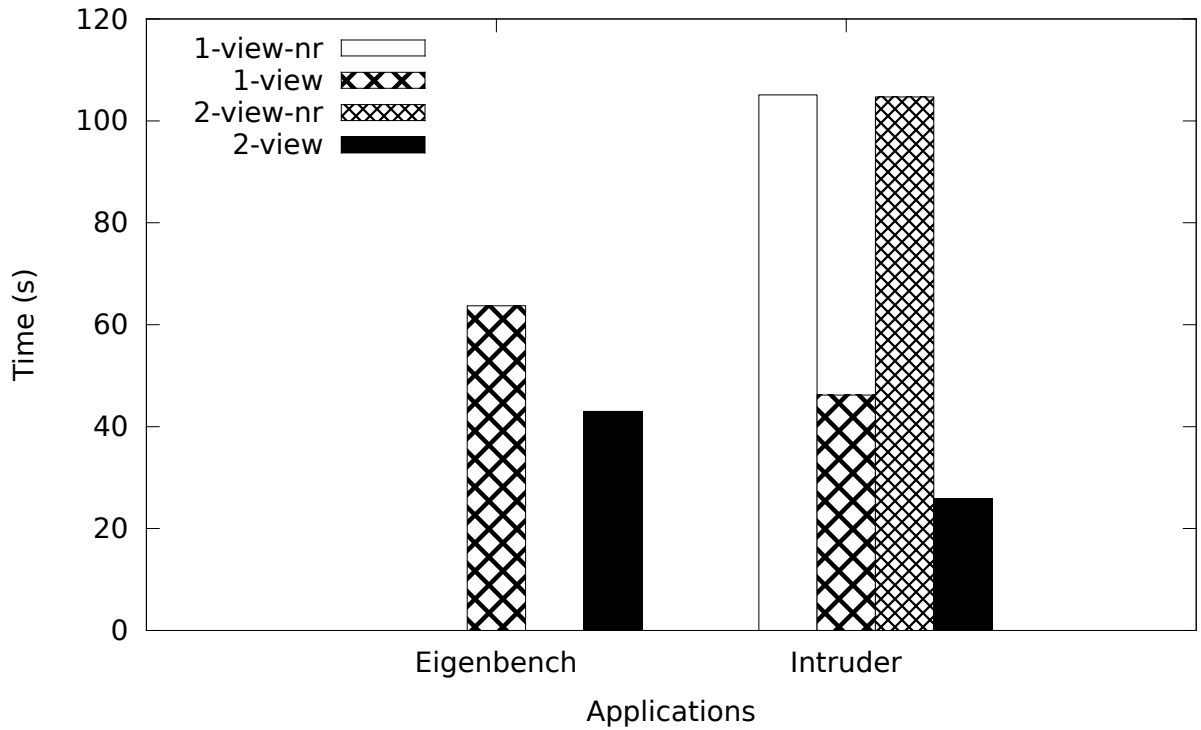


Figure: 2-view based applications on VOTM-OrecEagerRedo. For Eigenbench, its 1-view-nr and 2-view-nr versions have livelock.

VOTM-NOrec

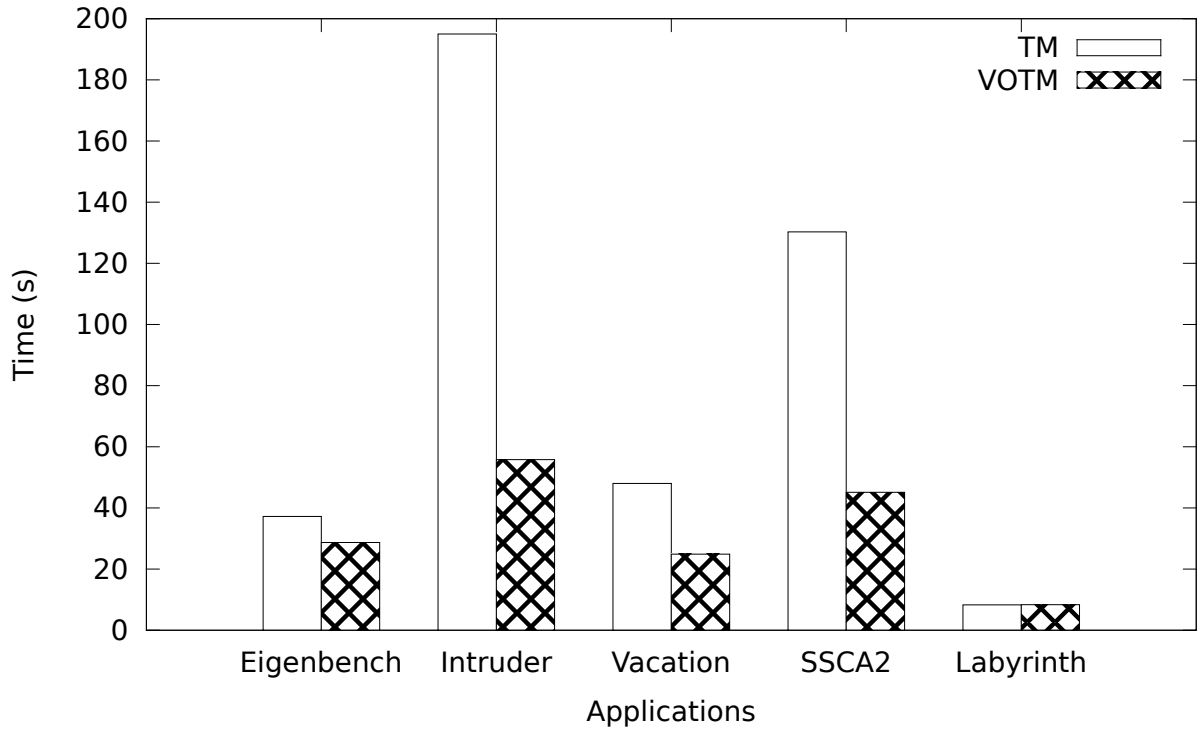
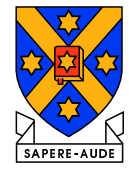


Figure: Single-view applications in VOTM-NOrec



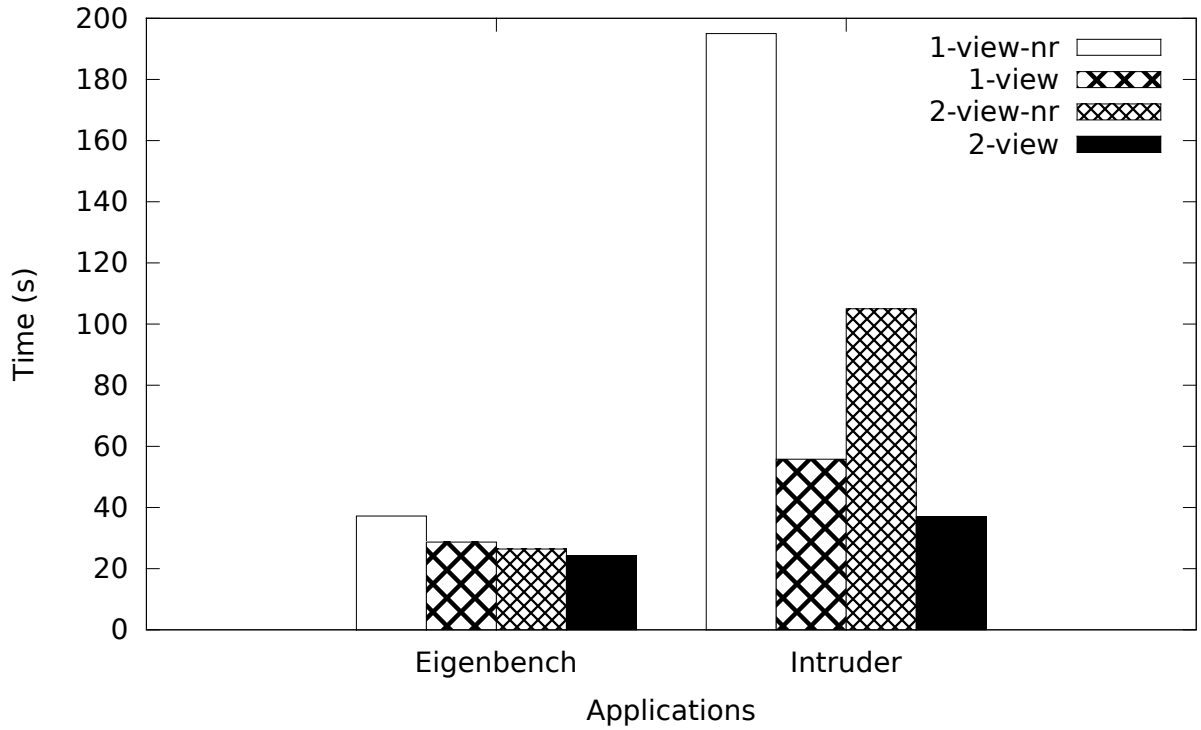


Figure: Two-view applications in VOTM-NOrec

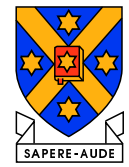


Table: Performance of VOTM Intruder

	2-view-nr				2-view			
Version	*time*	*#cmiss*	δ_1	δ_2	*time*	*#cmiss*	Q_1	Q_2
OrecEagerRedo	107.6	15.5G	0.95	0.003	25.8	8.1G	8	64
NOrec	105.2	18.5G	0.004	0.004	37.0	4.7G	16	16

Table: Single-view applications in VOTM-OrecEagerRedo

	TM			VOTM		
Application	time	δ	*cachemiss*	time	Q	*cachemiss*
Vacation	5.16	0.002	3.65G	5.36	64	3.69G
SSCA2	9.21	0.00001	2.07G	9.31	64	2.21G
Labyrinth	8.09	0.03	6.73G	8.13	64	6.74G

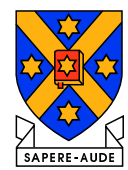
Table: Single-view applications in VOTM-NOrec

	TM			VOTM		
Application	time	δ	*cachemiss*	time	Q	*cachemiss*
Vacation	48.0	0.00002	25.5G	24.9	16	5.93G
SSCA2	130.3	0.00004	4.37G	45.1	16	3.88G
Labyrinth	8.32	0.03	6.79G	8.35	64	6.81G

View partitioning can relieve TM metadata contention

Table: MultiRBTree in VOTM-NOrec

version	#tx	#abort	#cachemiss
1-view-nr	32m	329k	11.6G
1-view	32m	180	4.76G
2-view-nr	32m	88.1k	7.30G
2-view	32m	388	4.63G
4-view-nr	32m	26.4k	4.75G
4-view	32m	2.02k	4.52G
8-view-nr	32m	41.1k	4.36G
8-view	32m	32.4k	4.26G



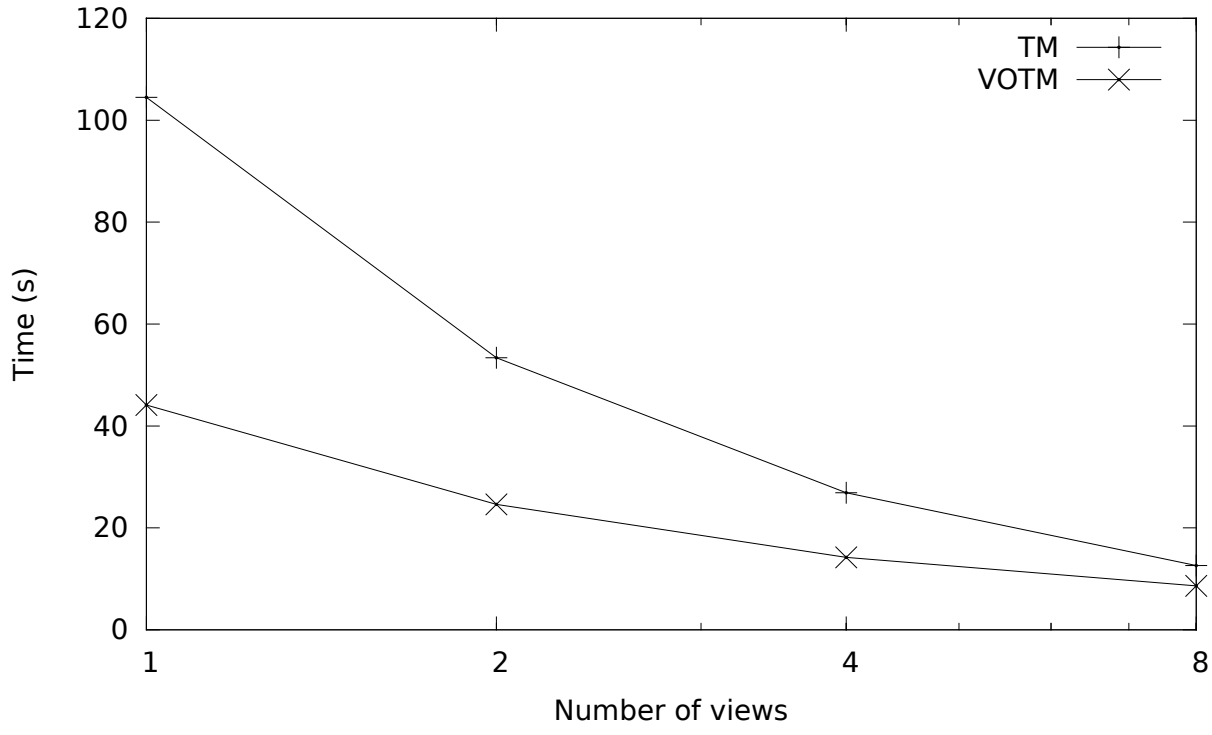


Figure: MultiRBTree in VOTM-NOrec

- Both Eigenbench and Intruder show view partitioning can improve performance by allowing fine-grain contention control of each view by RAC.
- Also in Intruder, δ_1 is large, which suggests high contention, and performance is improved by decreasing Q_1. δ_2 is very low, so the theorem correctly predicts that Q_2 should stay at 64.
- In Vacation, SSCA2 and Labyrinth, the theorem correctly predicts that Q should not be reduced in VOTM-OrecEagerRedo.
- In VOTM-NOrec, the very low δ scores suggests low *application data* contention, but results show further performance improvements by restricting Q due to reduction of metadata contention (indicated by the reduction of cache misses).
- Similarly, MultiRBTree shows view partitioning alone can also improve performance by alleviating the contention on TM metadata.

Conclusions

- VOTM improves both progress and concurrency by allowing shared data with different access patterns to be allocated into different views and use RAC to optimize each view individualy according to its contention
- VOTM can also relieve TM metadata contention by RAC and fine-grain views
- The current dynamic adjustment algorithm only takes into account of the application data contention. This algorithm needs to be refined to take care TM overheads, e.g., TM metadata contention