TM Program Concurrency Analysis: Parallelism versus Number of Schedules

Miroslav Popovic, Miodrag Djukic, Nenad Cetic

Contents

- Introduction
  - Motive, Goal, Related Work
- Our work-span methodology for TM Programs
- Analysis of Special Cases
  - Two Write Skewed TxNs
  - Simple Bank Special Cases
- Conclusion

Motive

- Intel Haswell and IBM Blue Gene/Q Compute Chip are here!
  - Skeptic A: The first swallows or COTS?
  - Skeptic B: How about TM performance? Is it good?

- Clearly we need a convincing methodology for estimating TM program performance. Answers?
  - Some stick to tradition…
  - Some are advocating brand new approaches…
  - Some are taking evolutional approaches…

Selected Related Work

- Three kinds of approaches exemplified:
  - Diegues and Cachopo use *throughput* and *complexity* in their recent technical report [1].
  - Gramoli *et al.* [2] proposed that the *set of schedules* (interleavings of steps of the sequential program), should be used as a measure of the "*amount of concurrency*".
  - Popovic *et al.* [3] adapted the well-known *work-span methodology*, which is based on modeling programs as DAGs, and calculating their *work*, *span*, *parallelism*, and *speedup*.

Goal

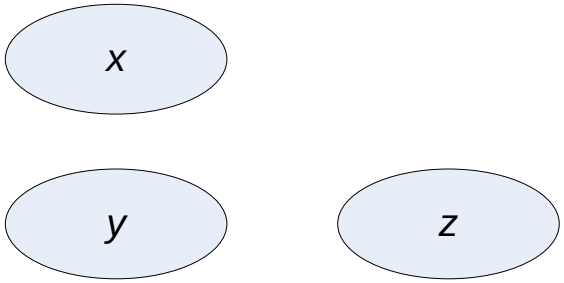
- Aims of this work:
  - Compare No of Interleavings [2] and Parallelism [3]
    - On some special cases at hand
    - See their properties and insights they provide
  - Make some recommendations
    - if possible

Work-Span for TM Programs [3]

- High level DAGs
  - if u and v are in parallel, P(u, v), v is drawn bellow u,
  - if u and v are in series, S(u, v), v is drawn to the right from u.

- Defs
  - The *work*, T_1, is the time to execute the program on a single processor, i.e. the sum of individual execution times of all the TxNs.
  - The *span*, T_∞, is the critical path in the DAG. Practically WCET for group of TxNs on ideal parallel comp.
  - The *parallelism* Ψ is the ratio T_1 / T_∞, i.e. $\Psi = T_1 / T_\infty$.

Simple Example



- DAG for the composition S(P(x,y), z).
- *Work* and *span*
  - $T_1 = 2$ (two initial TxNs, x and y)
  - $T_\infty = 2$ (two TxNs on the critical path, y and z)
- Parallelism & Speedup
  - $\Psi = 2/2 = 1$
  - $S_P \leq 1$

TM Work-Span Theorems [3] (1/3)

- T1 (Read TxNs)
  - For a group of n strictly concurrent unity (SCU) TxNs that share at least one t-variable, by reading them only, parallelism is $\Psi = n$.

- T2 (Write TxNs)
  - For a group of n SCU TxNs that share at least one t-variable, by writing only, parallelism is $\Psi = 1$.

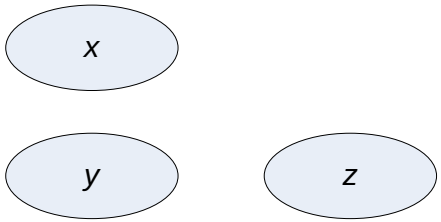
TM Work-Span Theorems [3] (2/3)

- T3 (Read-Write TxNs, Low Ψ)
  - The *lower bound* on parallelism for a group of n read-write transactions where k of them are conflict-free, $(n - 2) \geq k \geq 0$, $n \geq 2$, is: $\Psi \geq n / (n - k)$.

- T4 (Read-Write TxNs, High Ψ)
  - The *upper bound* on parallelism for a group of n read-write transactions where k of them are conflict-free, $(n - 2) \geq k \geq 0$, $n \geq 2$, is: $\Psi \geq n / 2$.

TM Work-Span Theorems [3] (3/3)

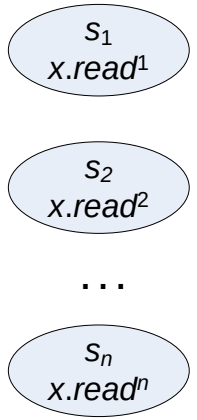
- T5 (Read and Write TxNs Mix)
  - For a group of n SCU TxNs that share at least one t-variable, where m TxNs are reading shared t-variables ($m<n$) and (n-m) TxNs are at least writing them, parallelism is: $\Psi \geq n / (n - m + 1)$.



A Write Skew Case

- May be modeled with DAG on top.
  - $\Psi = 1$ and $\Psi \neq \Psi(P)$, P - number of processors
- No of Interleavings (I)?
  - I is a function of P, i.e. $I = I(P)$. Consider two cases:
    - P=1. Then there are 2 interleavings: $<t_1, t_2>$ and $<t_2, t_1>$, so I=2. Why? Because OS scheduler may selected either of 2.
    - $P \geq 2$. Then there are 2 interleavings $< t_1 \mid\mid t_2, t_1>$, and $< t_1 \mid\mid t_2, t_2>$, so I=2. Why? Because $t_1 \mid\mid t_2$ run in parallel on two different cores, therefore in history (log file) this is happening at the same time, thus $<t_1 \mid\mid t_2>$ is a single item of interleaving.
  - Remarks: (1) $\Psi \neq I$, (2) I depends on P, i.e. $I = I(P)$.

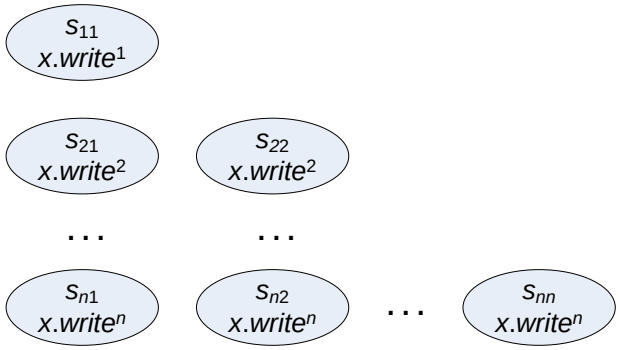
Group of *N* Read SCU TxNs



- May be modeled with DAG on top.
  - $\Psi = n$

- No of Interleavings (I)?
  - I depends on P, $I = I(P)$. Consider just two cases:
    - $I(P=1) = n!$ Why? Because OS scheduler may run these n TxNs sequentially in $n!$ orders.
    - $I(P \geq n) = 1$. Why? Because $<t_1 \mid\mid t_3 \mid\mid \ldots \mid\mid t_n>$ is a single item of interleaving in the history (log file).

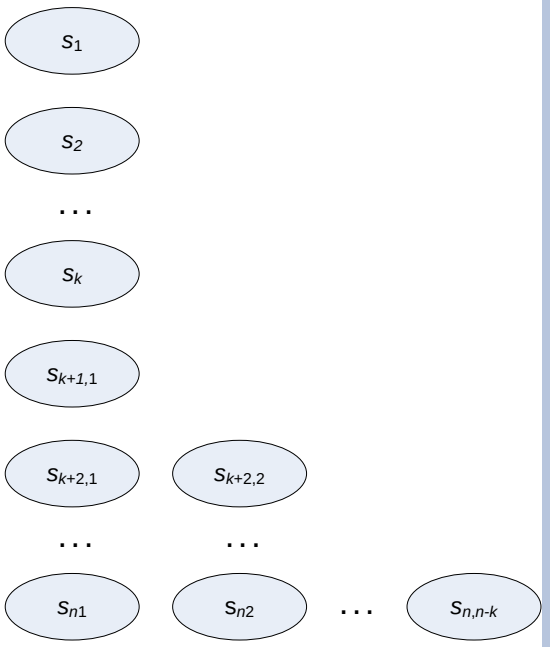
Group of *N* Write SCU TxNs



- May be modeled with DAG on top.
  - $\Psi = 1$

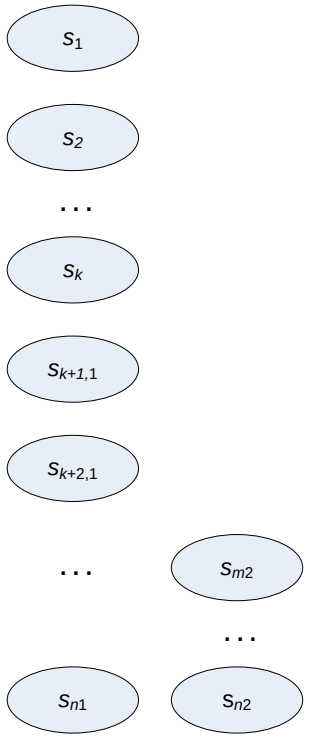
- No of Interleavings (I)?
  - I depends on P, $I = I(P)$. Consider just two cases:
    - $I(P=1) = n!$ Why? Same as in the previous case.
    - $I(P \geq n) = n!$ Why? Because when n TxNs run in parallel, TM will allow only one of them to get committed, (n-1) TxNs will be aborted. TM may select any TxN out of these n TxNs to be committed, so there is n such selections. Similarly, when (n-1) TxNs run in parallel, TM may make (n-1) selections, and so forth… Thus the total number of these selections is $I(P \geq n) = n(n\text{-}1)(n\text{-}2) \ldots 2 = n!$

Group of *N* Read-Write SCU TxNs, Worst Case



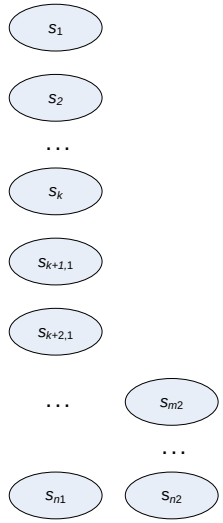
- May be modeled with this DAG.
  - $\Psi = n / 2$
- No of Interleavings (I)?
  - I depends on P, $I = I(P)$. Consider just two cases:
    - $I(P{=}1) = n!$ Why? Same as in the previous case.
    - $I(P{\geq}n) = (n - k)!$ Why? Because the k conflict-free TxNs will be successfully completed in the first round. The remaining $(n - k)$ conflicting TxNs execute effectively as a group of write TxNs in the case T2.

Group of *N* Read-Write SCU TxNs, Best Case (1/2)



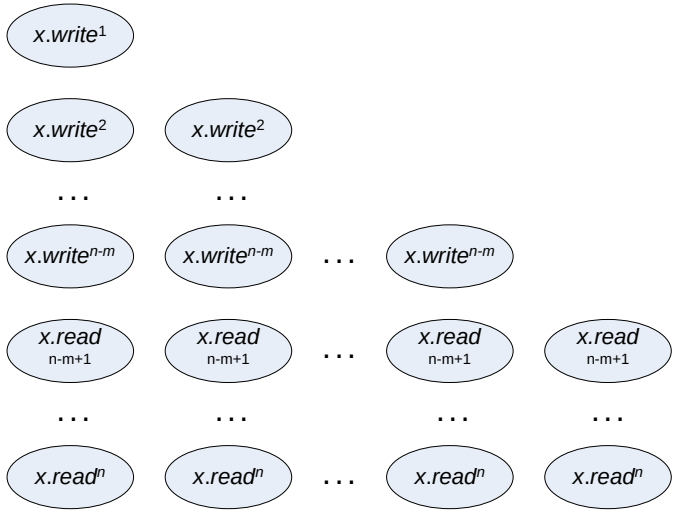
- May be modeled with this DAG.
  - $\Psi = n / (n - k)$
- No of Interleavings (I)?
  - I depends on P, $I = I(P)$. Consider just two cases:
    - $I(P=1) = n!$ Why? Same as in the previous case.
    - $I(P \geq n) = 2^m$, where $m = \lfloor (n - k)/2 \rfloor$ Why? Well, we assume TM arbiters on conflicting TxNs such that it randomly selects a TxN that commits.
      - Further on, there are two cases, when the number of conflicting TxNs, $(n - k)$, is even and when it is odd, see the next slide.

Group of *N* Read-Write SCU TxNs, Best Case (2/2)



- $I(P \geq n) = 2^m$, where $m = \lfloor (n - k)/2 \rfloor$, 2 cases:
  - If $(n - k)$ is even, there is m conflicting pairs of TxNs, and TM may allow either the 1st or the 2nd TxN from the pair to get committed (the other will get aborted).
  - If $(n - k)$ is odd, there is a trio of TxNs such that the first two TxNs are in the conflict with the third TxN, but there is no conflict between the first two TxNs.
    - Thus, either the first two TxNs get committed in the 1st round or in the 2nd round (the 3rd TxN is either aborted in the 1st round or in the 2nd round). This makes 2 interleaving scenarios for the TxN trio.
    - TM manages the remaining (m – 1) pairs of TxNs the same way as in the previous case when $(n - k)$ was even, yielding 2^{m-1} interleaving scenarios, so there is a total of $2 \times 2^{m-1} = 2^m$ interleaving scenarios.

Mix of Read and Write SCU TxNs



- May be modeled with this DAG.
  - $\Psi = n / (n - m + 1)$

- No of Interleavings (I)?
  - I depends on P, $I = I(P)$. Consider just two cases:
    - $I(P{=}1) = n!$ Why? Same as in the previous case.
    - $I(P{\geq}n) = (n - m)!$ Why? Because there is $(n - m)$ Write TxNs, and they will behave as in the case T2, so there is $(n - m)!$ interleaving scenarios all together. The remaining m Read TxNs always get executed the same way (in parallel), so there execution does not increase the number of possible scenarios.

Conclusions

- Based on the previous analysis, we may conclude:
  - The *parallelism* and the *number of interleavings* are completely different measures,
  - The *number of interleavings* is an absolute measure that does not provide any information on *parallelism* that exists in a concurrent program,
    - nor the *speedup* that a concurrent program may achieve.
  - The *work*, *span*, *parallelism*, and *speedup* are measures that are more meaningful then the number of interleavings.

- Recommendation
  - We suggest using the *work-span methodology* when analyzing TM program concurrency and performance.

THANK YOU!

- Qs?

References

- [1] N. Diegues and J. Cachopo, “Exploring parallelism in transactional workloads”, INESC-ID Lisbon, Tech. Rep. RT/16/2012, June 2012.
- [2] V. Gramoli, P. Kuznetsov, and S. Ravi, “Sharing a Sequential Data Structure: Correctness Definition and Correctness Analysis”, The 4th WTTM, 2012.
- [3] M. Popovic, I. Basicevic, M. Djukic, and N. Cetic. “Estimating Parallelism of Transactional Memory Programs”. 3rd IEEE International Conference on Information Science and Technology, Yangzhou, Jiangsu, China, 2013.