Appendix to *DRAMbulism*: Balancing Performance and Predictability through Dynamic Pipelining

This document provides the appendix to:
Reza Mirosanlou, Mohamed Hassan and Rodolfo Pellizzoni, *DRAMbulism*: Balancing Performance and Predictability through Dynamic Pipelining. Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium, Sydney, Australia, April 2020.

A. *Proofs of Section V*

Proof: [Lemma 1] Let k to denote the number of PRE commands of other banks that conflict with the cua. Since there are $b-1$ other banks in the system and due to RR arbitration in Rule 6, k is at most $b-1$. We next focus on the number of conflicting ACTs. Due to the t_{RRD} constraint between successive ACTs, the number of conflicting ACTs is bounded by $\lceil\frac{L_{PRE}+1}{t_{RRD}}\rceil$ (one ACT every t_{RRD} cycles; note that adding one to L_{PRE} is required to account for the clock cycle when the cua is issued). Also note that due to intra-bank constraints, an ACT must follow a PRE to the same bank by at least t_{RP}, and a PRE must follow the previous ACT to the same bank by at least $t_{RCD}+t_{RTP}$. Hence, the number of interfering ACTs belonging to the k banks with PRE commands is also bounded by $\lceil\frac{L_{PRE}+1-t_{RP}}{t_{RRD}}\rceil+\lceil\frac{L_{PRE}+1-t_{RCD}-t_{RTP}}{t_{RRD}}\rceil$. Noting that such constraint does not apply to the remaining $b-1-k$ banks, we obtain that the number of conflicting ACT commands is at most $\min(\lceil\frac{L_{PRE}+1}{t_{RRD}}\rceil,\lceil\frac{L_{PRE}+1-t_{RP}}{t_{RRD}}\rceil+\lceil\frac{L_{PRE}+1-t_{RCD}-t_{RTP}}{t_{RRD}}\rceil+b-1-k)$.

Finally, note that the distance between any two CASes is at least t_{CCD}; the distance between a PRE and a CAS command to the same bank is at least $t_{RP}+t_{RCD}$; and between a CAS and a PRE t_{RTP}. Hence, we similarly obtain a bound for the number of conflicting CASes of $\min(\lceil\frac{L_{PRE}+1}{t_{CCD}}\rceil,\lceil\frac{L_{PRE}+1-t_{RTP}}{t_{CCD}}\rceil+\lceil\frac{L_{PRE}+1-t_{RP}-t_{RCD}}{t_{CCD}}\rceil+b-1-k)$. Adding together the number of conflicting commands of each type yields the proof. ■

Proof: [Lemma 2] Assuming that the current round has the same direction as the previous round, regardless of the direction, the inter-bank constraint between two CASes is t_{CCD}; since the last CAS of the previous round is issued one clock cycle before the end of the round, the maximum value of $CAStimer_{init}$ is $t_{CCD}-1$. If the current and previous rounds have different directions, assuming that the previous round was read, the earliest time that a write CAS can be issued is after t_{RTW} cycles and $CAStimer_{init}$ is thus at most $t_{RTW}-1$. Similarly, if the previous round was write, the earliest time a read CAS can be issued is after t_{WtoR} cycles, and the $CAStimer_{init}$ is at most $t_{WtoR}-1$. This concludes the proof. ■

Proof: [Lemma 3] The inter-bank ACT constraints are t_{RRD} and t_{FAW}. For the t_{FAW} constraint to delay the first ACT in a round, we need four ACTs in the previous round. Since we want to determine the maximum value of $ACTtimer$, we assume that those ACTs are issued as late as possible, which means t_{RRD} after each other. The last ACT in previous round needs to issue its CAS command which incorporates ACT to CAS delay t_{RCD}. Therefore, the total time from issuing the first ACT of the sequence to the time the previous round finishes is $3\cdot t_{RRD}+t_{RCD}+1$. Hence, the maximum delay that t_{FAW} can cause to an ACT at the beginning of the round is $t_{FAW}-(3\cdot t_{RRD}+t_{RCD}+1)$. We next consider t_{RRD}. Since for all devices described in Table I, $t_{RCD}>t_{RRD}$, it follows that the t_{RRD} constraint generated by the last ACT in the previous round cannot delay the first ACT of the current round. This yields the lemma. ■

Proof: [Lemma 4] Since we assume that all other banks issue a transaction in Round 2, by Rule 6, the tua has the highest RR priority in Round 3. Hence, for a closed tua, its ACT will be issued $ACTtimer_{init}$ after the beginning of Round 3. The CAS of the tua will then become intra-ready t_{RCD} after the ACT is issued. We now have two cases: 1) the CAS becomes intra-ready at or before $CAStimer_{init}$ after the beginning of the round; or 2) the CAS becomes intra-ready after $CAStimer_{init}$. In case 1), the CAS will be issued no later than $CAStimer_{init}+1$ (note that, an ACT could cause command bus conflict at $CAStimer_{init}$), resulting in a latency bound $L^{CR}_{R3}=CAStimer^{\max,R}_{init}+1$. In case 2), depicted in Figure 4, the CAS could be delayed by another (lower priority) read CAS in Round 3; in the worst-case such CAS could be issued the clock cycle before the CAS under analysis becomes intra-ready, resulting in an added delay of t_{CCD} when accounting for bus conflict. This results in $L^{CR}_{R3}=ACTtimer^{\max}_{init}+t_{RCD}+t_{CCD}$. Taking the maximum of Case 1) and 2) results in Equation 11. ■

Proof: [Lemma 5] Let $\bar{t}$ be the time at which transactions become pipe-blocked in the round, and $\overline{CAStimer}$ be the value of $CAStimer$ at $\bar{t}$. First note that based on Rule 4, no ACT command can be issued after $\bar{t}$; this means that CASes issued after the blocking point cannot suffer bus conflict. Furthermore, the number of CAS commands issued after $\bar{t}$ is exactly equal to the number of pending requests N^{pend} at that time; and it must be $N^{pend}\geq 1$, otherwise the round would have ended by Rule 1. We analyze two cases: 1) all N^{pend} CAS commands are delayed (Figure 6(a)); 2) at least one of the N^{pend} CASes is not delayed (Figure 6(b)). For Case 1), considering that each CAS delays the next one by t_{CCD} since there are no bus conflicts, and that the round ends one cycle after $\tau_N.C$, we immediately obtain:

$$L_{pipe}=\overline{CAStimer}+(N^{pend}-1)\cdot t_{CCD}+1. \tag{18}$$

Since the third condition in Rule 4 must not hold at $\bar{t}$, we have the following inequalities:

$$\begin{aligned}
&\overline{CAStimer}+N^{wait}\cdot t_{CCD}-t_{RCD}-1<0 &(19)\\
&\implies(\overline{CAStimer}+N^{wait}\cdot t_{CCD}-t_{RCD}-1)\\
&+t_{RCD}-t_{CCD}+2\leq t_{RCD}-t_{CCD}+1\\
&\implies\overline{CAStimer}+(N^{wait}-1)\cdot t_{CCD}+1 &(20)\\
&\leq t_{RCD}-\ t_{CCD}+1 &(21)
\end{aligned}$$

Since $N^{pend}\leq N^{wait}$, we have:

$$\begin{aligned}
L_{pipe}=&\quad\overline{CAStimer}+(N^{pend}-1)\cdot t_{CCD}+1\\
\leq&\quad(\overline{CAStimer}+(N^{wait}-1)\cdot t_{CCD}+1\\
\leq&\quad t_{RCD}-t_{CCD}+1, &(22)
\end{aligned}$$

which is the first term of the max in Equation 12.

We now analyze Case 2). Let $\tau_j.C$, with $j \leq N$, be the command with the largest index in N^{pend} that is not delayed; then CASes must be issuable at $\tau_j.C.t - 1$ since there are no bus conflicts. Hence, $\tau_j, \ldots, \tau_N$ must all be close transactions, otherwise they would be intra-ready at $\bar{t}$ and their CASes would be issued at or before $\tau_j.C.t - 1$. Then, since the minimum time that an ACT can arrive after the previous ACT is t_{RRD}, and $ACTtimer$ must be zero the clock cycle before the blocking point, it follows that $\bar{t} \geq \tau_j.A.t + (N - j + 1) \cdot t_{RRD} + 1$; while the round ends at $\tau_j.A.t + t_{RCD} + (N - j) \cdot t_{CCD} + 1$. Hence, we obtain:

$$\begin{aligned} L_{pipe} = &\quad \tau_j.A.t + t_{RCD} + (N-j) \cdot t_{CCD} + 1 - \bar{t} \\ \leq &\quad t_{RCD} + (N-j) \cdot t_{CCD} - (N-j+1) \cdot t_{RRD} \\ = &\quad t_{RCD} - t_{RRD} + (N-j) \cdot (t_{CCD} - t_{RRD}); \end{aligned} \tag{23}$$

but since $t_{RRD} \geq t_{CCD}$ for all devices, Equation 23 is maximized for $j = N$, yielding the second term of the max in Equation 12. This completes the proof. ■

B. Latency Analysis for Open Read Requests

In this section we provide the equations for the latency analysis of an open read requests. For the open request, the latency decomposition can be shown in Figure 11. Therefore, the open read request consists of the following timing elements:

$$L^{ORpR}_{req} = L^{ORpR}_{tran} + t_{RL} + t_{BUS}. \tag{24}$$

Notice that in order to have an open read, the request must be preceded by a read as discussed in Section V. The transaction latency can be achieved from:

$$L^{ORpR-self}_{tran} = L^{OR}_{self} + L_{round}(b-1, CAStimer^{\max,W}_{init}, ACTtimer^{\max}_{init}) + L^{OR-self}_{R3} \tag{25}$$

$$L^{ORpR-pipe}_{tran} = L_{pipe} + L_{round}(b-2, CAStimer^{\max,W}_{init}, ACTtimer^{\max}_{init}) + L^{OR-pipe}_{R3} \tag{26}$$

$$L^{ORpR}_{tran} = \max(L^{ORpR-self}_{tran}, L^{ORpR-pipe}_{tran}). \tag{27}$$

Note that in order to happen a pipe-blocking when we want to calculate $L^{ORpR-pipe}_{tran}$, there must be a close read transaction that was blocked in the first read round according to Rule 4. That is, in the second round, apart from the tua, we need to factor out the request that caused the pipe-blocking resulting in b-2 requests in the second round.

The maximum amount of time that an open transaction can be self-blocked in a round is:

$$L^{OR}_{self} = \quad L_{round}(b, 0, 0) - t_{RL} - t_{BUS}. \tag{28}$$

The worst-case scenario for L^{OR}_{self} happens when a requestor sends another request while it was serviced in the same round. Note that the blocking time can be obtained as the difference between the length of Round 1, and the time that the data of the previous request of the same bank is transferred. Hence, to maximize L^{OR}_{self}, we maximize the length of Round 1, assuming that the previous request completes as soon as possible, which

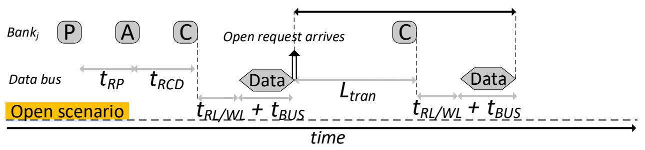

Fig. 11. Increasing the number of requestor in the system with DDR3 2133L.

is $t_{RL} + t_{BUS}$ after the beginning of the round (assuming $CAStimer_{init} = 0$).

The maximum times required to issue the CAS command in Round 3 for open read transaction are:

$$L^{OR-self}_{R3} = CAStimer^{\max,R}_{init} + 1, \tag{29}$$

$$L^{OR-pipe}_{R3} = CAStimer^{\max,R}_{init} + t_{CCD} + 2. \tag{30}$$

For an open tua with pipe-blocking, there must be a close transaction that was also pipe-blocked in Round 1, otherwise the tua would not be pipe-blocked. This close transaction must be issued in Round 3. Therefore, we consider two cases: 1) if $CAStimer_{init} + 1 < ACTtimer_{init} + t_{RCD}$, the open CAS will be issued at either $CAStimer_{init}$ or $CAStimer_{init} + 1$. 2) If $CAStimer_{init} + 1 \geq ACTtimer_{init} + t_{RCD}$, then the CAS of the close transaction might be issued first (since it might have higher priority) at $CAStimer_{init} + 1$, delaying the CAS of the tua for an extra $t_{CCD} + 1$ clock cycles, so that it is issued at $CAStimer_{init} + 1 + t_{CCD} + 1$. Taking the maximum of the two cases and noting that other requestors have lower priority in RR queue results in $CAStimer_{init} + t_{CCD} + 2$. For an open tua with self-blocking, we only consider the first case since there is no such outstanding close request in Round 3.

C. Latency Analysis for Write Requests

For the write request, we follow the same steps as read request but we need to consider the write related timers and timing constraints. After applying the appropriate changes, we obtain the following formulas for the write requests:

$$L^{CWpR}_{req} = t^{pR}_{\alpha} + L_{PRE} + t_{RP} + L^{CWpR}_{tran} + t_{WL} + t_{BUS}, \tag{31}$$

$$L^{CWpW}_{req} = t^{pW}_{\alpha} + L_{PRE} + t_{RP} + L^{CWpW}_{tran} + t_{WL} + t_{BUS}; \tag{32}$$

where the transaction latencies can be achieved from:

$$L^{CWpW}_{tran} = \max(L_{pipe}, L^{CW}_{self}) + L_{round}(b-1, CAStimer^{\max,R}_{init}, ACTtimer^{\max}_{init}) + L^{CW}_{R3} \tag{33}$$

$$L^{CWpR}_{tran} = L_{pipe} + L_{round}(b-1, CAStimer^{\max,R}_{init}, ACTtimer^{\max}_{init}) + L^{CW}_{R3}; \tag{34}$$

and L^{CW}_{R3} is:

$$L^{CW}_{R3} = \max(ACTtimer^{\max}_{init} + t_{RCD} + t_{CCD}, CAStimer^{\max,W}_{init} + 1). \tag{35}$$

The maximum amount of time a close write transaction can be self-blocked in a round is:

$$L^{CW}_{self} = \quad L_{round}(b, 0, 0) - t^{pW}_{\alpha} - L_{PRE} - t_{RP} - t_{WL} - t_{BUS}. \tag{36}$$