Technische Universiteit
Eindhoven
University of Technology

ILP Formulation and Solution Strategies for Memory Command Scheduling

Sven Goossens, Karthik Chandrasekar, Benny Akesson, and Kees Goossens

ES Reports
ISSN 1574-9517

ESR-2014-03
23 October 2014

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

ILP Formulation and Solution Strategies for Memory Command Scheduling

Sven Goossens[1], Karthik Chandrasekar[2], Benny Akesson[3], and Kees Goossens[1]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Delft University of Technology, Delft, The Netherlands
[3]Czech Technical University, Prague, Czech Republic

***Abstract*—A memory pattern is a small series of scheduled SDRAM commands with known length and function. This article describes an ILP formulation that generates predictable memory patterns that implement a close-page policy. The generated patterns are optimal in terms of length, i.e. there is no other permutation of this set of commands satisfying pattern scheduling rules and timing constraints resulting in a shorter pattern.**

I. Problem formulation

Memory patterns have been discussed in [1] as a means to create a real-time SDRAM controller. A pattern is a sequence of statically scheduled SDRAM commands. Patterns are dynamically scheduled at run-time by the memory controller. The corresponding worst-case analysis [2] provides performance bounds in terms of the worst-case execution time of a request, which can be used to derive higher-level performance bounds on bandwidth and response time [3]. In [4] we describe two heuristics that can generate memory patterns, and subsequently evaluate the quality of these heuristics with an Integer Linear Programming (ILP) formulation of the same problem. This article formalizes the ILP formulation, and should make it easier to understand the details of the associated scripts [5] that generate the formulations.

The ILP formulation should generate a memory command schedule that satisfies the following high-level constraints:

1) the schedule is a valid DRAM command schedule for the memory device under consideration, and
2) the schedule can be repeated after itself without violating timing constraints.

Note that this article makes a distinction between ILP constraints, i.e. the linear equations which are part of the ILP description, and timing constraints, which are defined by JEDEC and specify the minimum distance between pairs of SDRAM commands [6]–[11].

A. ILP variables

Which commands are included in a memory pattern is defined by:

1) the type of the pattern, i.e. either *read* or *write*. Refresh and switching patterns can be derived later based on the read and write pattern.
2) the number of banks interleaved, BI.
3) the number of bursts per bank, BC.

Each pattern contains BI activates, BI precharges, and BI · BC read or write commands, for read and write patterns, respectively. The ILP problem schedules two incarnations of the pattern, of which the first is complete, and the second is not. The second incarnation consists only of ACT commands and is used to express constraints regarding activate and precharge commands that span across patterns. We use this partial pattern to simplify the formulation. Note that by doing this, we ignore potential constraints related to read and write commands in the first incarnation which influence the second incarnation. This only works if the following two assumptions hold:

- The ACT-to-RD/WR constraint is always greater than or equal to the RD-to-RD and WR-to-WR constraints, which has been true for all memories introduced up to now.
- Switching patterns resolve all remaining constraints across read to write and write to read patterns boundaries.

For the purpose of this description, we regard commands as 3-tuples (c_t, c_b, c_n), consisting of a type $c_t \in \{\text{ACT, RD, WR, PRE}\}$, a bank $c_b \in \{0...\text{BI-1}\}$ and an incarnation id $c_n \in \{0, 1\}$. The set of all commands in the pattern is called C. All commands have c_n set to 0, except for the extra ACT commands representing the start of the second incarnation of the pattern. For those commands $c_n = 1$.

$$
\begin{aligned}
C_{\text{ACT}} &= \{(\text{ACT}, c_b, c_n) \mid c_b \in \{0...BI-1\}, c_n \in \{0,1\}\} \\
C_{\text{PRE}} &= \{(\text{PRE}, c_b, 0) \mid c_b \in \{0...BI-1\}\} \\
C_{\text{RW}} &= \bigcup_{0 \le i < BC} \{(\text{RD/WR}, c_b, 0) \mid c_b \in \{0...BI-1\}\} \\
C &= C_{\text{ACT}} \cup C_{\text{PRE}} \cup C_{\text{RW}}
\end{aligned}
$$

The algorithm that generates the ILP formulation determines a conservative lower bound (L_c) and upper bound (U_c) on the position of each command c in the pattern, as further explained in Section I-B. A set of ILP variables (V_c) is created for each command. It contains one boolean variable X_i^c for each integer position i in the interval $[L_c, U_c)$. If the variable X_i^c is *true*, then this means that the command c is scheduled in position or cycle i:

$$\forall c \in C, \qquad V_c = \{X_i^c \mid i \in \mathbb{Z}, L_c \le i < U_c\}$$

One way to visualize this set of variables is like a matrix, where each column corresponds to a command, and each row to a position in the pattern (see Fig. 1). The goal of the ILP formulation is to mark exactly one variable in each column as *true*. After "graying out" the options we know are invalid based on the lower and upper bounds, all remaining variables in a command column are contained in the V_c set of that command.

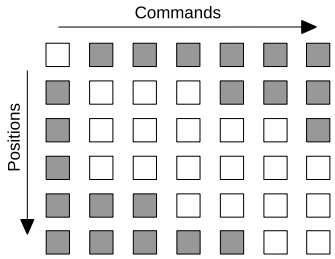

Fig. 1. Visualization of the ILP variables in matrix form.

The ILP constraints and objective function are constructed as linear equations based on these variables. For ease of notation, we define a function $pos(V_c)$, which returns a sub-expression representing the position of a command c in the pattern. The pos function works based on the assumption that only *one* of the variables in each set V_c can be *true* in a valid solution. In Section I-C, we show how this is enforced with additional constraints:

$$pos(V_c) = \sum_{X_i^c \in V_c} i \cdot X_i^c$$

B. Determining lower and upper bounds

It is essential to bound the number of possible positions for each command to reasonable ranges to limit the problem size, and with that, the computation time of the ILP solver. Fig. 2 shows how the bounds of a command are determined. Based on the two patternGen heuristics, an upper bound $N_{\text{heuristic}}$ on the optimal pattern length can be found. The ILP problem schedules two incarnations of the pattern (one complete, one partial consisting only of ACT commands), and hence the range of command positions that has to be considered is twice as large as $N_{\text{heuristic}}$.

We further limit the range on a per-command basis, by considering which other commands *have* to precede it based on the pattern generation rules and the SDRAM state machine, and which commands directed towards the same bank still need to be scheduled after it. Lower bounds on how many cycles it takes to schedule a series of commands are determined based on a very simplistic ASAP scheduler. Considering just two commands at a time, it sums the SDRAM timing constraints between each command pair, until it reaches the end of the series. The scheduler fails to spot constraints between commands separated by other commands, and is hence too optimistic, but it is sufficient to quickly find a lower bound on the duration of a sub-schedule.

Tighter lower and upper bounds could be potentially be found by improving this ASAP scheduler, and by using $N_{\text{heuristic}}$ as a secondary reference point for commands that necessarily happen in the first incarnation of the pattern, further reducing the computation time of the solver. However, the bounds that are currently derived limit the problem size sufficiently such that the largest problems we consider (32 bursts, to 8 different banks) are solved within a few hours.

C. ILP Constraints

This section reiterates the list of constraints that was mentioned in the journal article [4], and describes how each of them translates into an ILP constraint.

1. *An ACT to bank 0 is scheduled in cycle 0.*
$$X_0^c = 1 \mid c = (\text{ACT}, 0, 0)$$
2. *Only one command may be scheduled in each cycle.* Precharge commands are exempted from this rule, since they are executed using auto precharge flags and do not require a slot in the schedule:
$$\forall i \in \mathbb{Z}, \quad \sum_{X_i^c \in V_c} X_i^c \leq 1 \mid c \in C, c \notin C_{\text{PRE}}$$
3. (Not in journal article) *Each command is scheduled exactly once*:
$$\forall c \in C, \quad \sum_{X_i^c \in V_c} X_i^c = 1$$
Based on these constraints, all sets V_c are Special Ordered Sets (SOS) of type 1, i.e. a set of variables of which at most one can take the value *true*. These constraints are explicitly described as SOS in the ILP formulation, since this helps guide the ILP solver in finding a solution. Each variable in such a set needs a relative weight that represents its costs. We assign weights to the elements according to the position (i) within the pattern the element corresponds to.
4. Two of the constraints in the article are related to timing constraints:
   a. *The relative delays between any pair of commands is at least as large as prescribed by the timing constraints of the SDRAM.*
   b. *There are at most four ACT commands in each Four Activate Window (*FAW*) window.*

   The ILP constraints required to implement *a.* can be split in two categories: those for which the ordering of the commands involved is irrelevant for the timing constraint, and those for which the order matters. Starting with the first category, the possible command combinations are limited to: 1) two activate, 2) two read or 3) two write commands. Since commands for the same bank are required to happen in a fixed order according to the bank state machine (i.e. activate, then read/write bursts, followed by a precharge), we discard constraints that only apply when the associated commands target the same bank for now, and treat them later when we enforce the command ordering at Constraint 5.
   This only leaves the 3 previously mentioned pairs, directed at different banks and potentially different bank groups, i.e. 6 different timing constraints. Each timing constraint related to (pairs of) commands of the same type can be interpreted as a *window* in which only a specific number K of those commands may be scheduled. We use this interpretation to capture both the FAW constraint (where $K = 4$) and the narrow set of order-agnostic constraints ($K = 1$) within the same ILP constraint template. The window sizes are equal to the values in the constraint Table 1 and 2 in the article [4].
   First, we deal with timing constraints for which the bank group to which the command is sent does not influence the constraint value. For all considered memory types except DDR4, all constraints don't care about the bank group. For

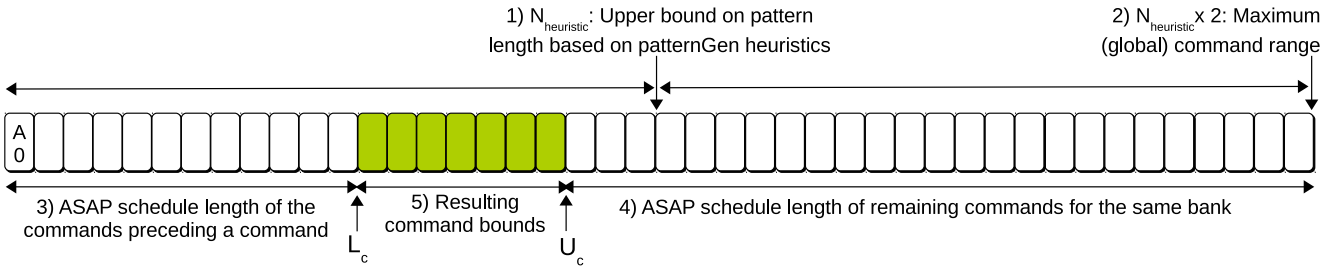

Fig. 2. Finding lower and upper bounds on the position of a command.

DDR4, in the constraints with an _X postfix, _X is substituted by _L, since the *long* constraints have to be satisfied across all bank groups. The values of the timing constraints between two commands of type tp, $tp \in \{\text{ACT, RD, WR}\}$, are denoted as TC_{tp}. Each TC_{tp} is an integer number of clock cycles, $TC_{tp} \in \mathbb{Z}$.

A constraint has to be added for all windows of size TC_{tp} in the valid command range, i.e. between 0 and $N_{\text{heuristic}}$:

$$\forall j \in \{0..2 \cdot N_{\text{heuristic}} - 1\}, \forall\ tp \in \{\text{ACT, RD, WR}\}, \sum_{X_i^c \in V_c} X_i^c \leq K \mid c \in C, c_t = tp, j \leq i < j + TC_{tp} \quad (1)$$

For DDR4, certain timing constraints only need to be satisfied when the associated commands target the *same* bank group. We refer to them as TC'_{tp}, and the total number of bank groups as nbg. We again iterate over all possible windows, but additionally limit the commands we include in the ILP constraints by their bank group, which is given by their bank id c_b modulo nbg:

$$\forall j \in \{0..2 \cdot N_{\text{heuristic}} - 1\}, \forall bg \in \{0..nbg - 1\}, \forall\ tp \in \{\text{ACT, RD, WR}\}, \sum_{X_i^c \in V_c} X_i^c \leq K \mid \begin{array}{r} c \in C, c_t = tp, \\ mod(c_b, nbg) = bg, \\ j \leq i < j + TC'_{tp} \end{array} \quad (2)$$

In the implementation, constraints that are trivially satisfied because the number of selected X_i^c variables in the equation is smaller than K are not added to the problem description. Fig. 2 shows an example of the variables that are selected for each window based on Eq. (1) or Eq. (2).

5. *The commands for each bank are executed in the proper order, i.e. start with an activate, followed by BC read or write commands, followed by a precharge.* For each pair of commands c_0 and $c_1 \in C$, which are constrained by a timing constraint $T \in \mathbb{Z}$, and for which the required order is known, such that $pos(V_{c_1}) > pos(V_{c_0})$ in a valid solution of the ILP problem, we add the following ILP constraints:

$$pos(V_{c_1}) - pos(V_{c_0}) \geq T \quad (3)$$

6. The relative order of commands across pattern incarnations is constrained by the following rules:
   a. *Commands for the second instance of the pattern cannot be scheduled before the extra activate to bank 0.* This constraint only refers to the activate commands in the second pattern incarnation. They are dealt with in Constraint 7.
   b. *Commands for the first instance need to be scheduled before the extra activate to bank 0.*

   Since Constraint 5. enforces commands per bank to be ordered, it is sufficient to enforce that the last read or write command to each bank happens before the extra activate command to bank 0. Because Constraint 5. already asserts this property for bank 0, we further limit the set of commands by only including read or writes to banks > 0. We then simply use the template given by Eq. (3) on all commands fitting these criteria, substituting them for c_1, and using $c_0 = (ACT, 0, 1)$ and $T = 0$.
7. *The first and second incarnation of the pattern should be the same.* First we define a set a' containing the V_c sets related to the ACT commands of the second pattern incarnation. Bank 0 is excluded, since it is used as the reference point from which the second pattern incarnation starts.

$$a' = \{V_c \mid c \in C, c_b > 0, c_n = 1\}$$

A set of constraints enforces that the distance between the extra activate command to a bank larger than 0 ($pos(V'_c)$) and the start of the second pattern incarnation ($pos(V_{(ACT,0,1)})$), is equal to the distance between the first activate command to that bank ($V_{(ACT,c_b,0)}$) and cycle 0.

$$\forall V'_c \in a', \quad pos(V'_c) - pos(V_{(ACT,0,1)}) = pos(V_{(ACT,c_b,0)}) - 0 \quad (4)$$

Note that this constraint is stronger than Constraint 6. since $pos(V_{(ACT,c_b,0)})$ is guaranteed to be > 0, and hence it

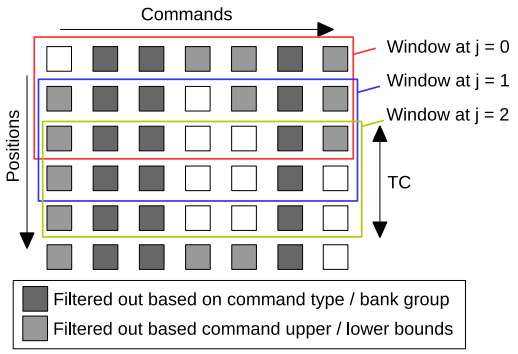

Fig. 3. Visualization of window-based constraints. Only the white boxes in each window are included in the sum in Eq. (1)).

forces all (remaining) commands of the second instance of the pattern to happen after the second activate to bank 0.

D. Objective function

The objective of the ILP formulation is to minimize the pattern length. This can be achieved by minimizing $pos(V_{(ACT,0,1)})$. There may be more than one possible optimal pattern; all commands in the pattern could be postponed as long as the pattern length is not influenced by it. To eliminate some equivalent optimal solutions, we add an extra element to the objective function which attempts to minimize the unnecessary postponement of commands, by adding the position of the last precharge in the pattern to the cost function. This makes it easier to visually compare the output to that of the patternGen heuristics. A helper variable $\hat{\text{PRE}}$ is introduced to represent the position of the last precharge in the pattern. We force it to be greater than or equal to the position of the last precharge in the pattern:

$$\forall c \in C_{\text{PRE}}, \quad \hat{\text{PRE}} - pos(V_c) \geq 0$$

A sufficiently large scaling factor s is applied to make sure the pattern length remains the primary optimization goal. The optimization goal is then set to:

$$\textbf{minimize } s \cdot pos(V_{(ACT,0,1)}) + \hat{\text{PRE}}$$

REFERENCES

[1] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*, ser. Embedded Systems Series. Springer, 2011.

[2] B. Akesson, W. Hayes Jr., and K. Goossens, "Classification and analysis of predictable memory patterns," in *Embedded and Real-Time Computing Syst. and Applicat. (RTCSA)*, 2010, pp. 367–376.

[3] B. Akesson and K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2011, pp. 1–6.

[4] S. Goossens *et al.*, "Power/performance trade-offs in real-time SDRAM controllers," *IEEE Trans. Comput.*, under review.

[5] S. Goossens, "Power/performance trade-offs in real-time SDRAM controllers - code and datasets," http://www.es.ele.tue.nl/~sgoossens/sdram_trade_offs.

[6] *DDR2 SDRAM Specification*, JESD79-2F ed., JEDEC Solid State Technology Association, 2009.

[7] *DDR3 SDRAM Specification*, JESD79-3E ed., JEDEC Solid State Technology Association.

[8] *DDR4 SDRAM Specification*, JESD79-4 ed., JEDEC Solid State Technology Association.

[9] *Low Power Double Data Rate Specification*, JESD209B ed., JEDEC Solid State Technology Association.

[10] *Low Power Double Data Rate 2 Specification*, JESD209-2D ed., JEDEC Solid State Technology Association.

[11] *Low Power Double Data Rate 3 Specification*, JESD209-3B ed., JEDEC Solid State Technology Association.