(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 July 2016 (21.07.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/113223 A1

(51) International Patent Classification:
G06F 11/30 (2006.01)

(21) International Application Number:
PCT/EP2016/050385

(22) International Filing Date:
11 January 2016 (11.01.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
1500446.8 12 January 2015 (12.01.2015) GB

(71) Applicant: **THE UNIVERSITY OF MANCHESTER** [GB/GB]; Oxford Road, Manchester M13 9PL (GB).

(72) Inventors: **GHASEMPOUR, Mohsen**; C/o The University of Manchester, Oxford Road, Manchester M13 9PL (GB). **LUJAN MORENO, Miguel Angel**; C/o The University of Manchester, Oxford Road, Manchester M13 9PL (GB).

(74) Agents: **HACKNEY, Nigel** et al.; Mewburn Ellis LLP, City Tower, 40 Basinghall Street, London Greater London EC2V 5DE (GB).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

(54) Title: MONITORING DEVICE

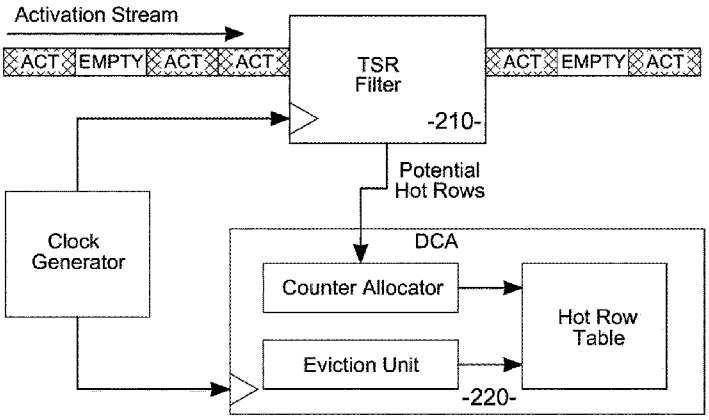

Figure 5

(57) Abstract: A monitoring device for monitoring the accessing of data items in a memory, the monitoring device including a first monitoring unit and a second monitoring unit. The first monitoring unit is configured to: monitor a stream of access requests/commands, wherein each access request/command is directed to a data item in the memory; determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access; and if it is determined that a data item meets the condition indicative of frequent access, notify a second monitoring unit of the data item. The second monitoring unit is configured to monitor the accessing of data items notified to the second monitoring unit by the first monitoring unit. The second monitoring unit may be configured to identify one or more frequently accessed data items (e.g. "hot rows") from the data items notified to it by the first monitoring unit. This may be useful for avoiding "row hammer" errors.

WO 2016/113223 A1

MONITORING DEVICE

This invention relates to a monitoring device for monitoring the accessing of data items in a memory, e.g. a DRAM memory. The monitoring device may be or may form part of a memory controller.

Despite non-volatile memory technologies starting to make inroads, the main memory market for computer systems is dominated by DRAMs. As DRAM manufactures continue to move to smaller technology nodes, they are trying to optimize manufacturing costs and memory performance without degrading reliability [20, 15]. The increased density and smaller storage cells make DRAM cells more susceptible to different type of noise like electromagnetic coupling effect between cells [11, 27, 35].

As described in more detail below, due to the volatile nature of DRAM technologies, memory controllers typically need to issue refresh commands at strict time intervals to avoid losing stored data. Another not so well known mechanism of losing or corrupting stored data within a refresh interval is to have a sequence of activation commands causing a row of data cells in a DRAM to be activated above certain frequency thresholds. This is referred to as the "row hammer" effect. In the case of the "row hammer" effect occurring, data corruption is caused by electrical disturbance and generally affects the DRAM rows that are adjacent to (i.e. neighbour) the frequently activated row. Data corruption caused in this way can be referred to as a "row hammer error". Thus, it is possible to modify the contents of a physical DRAM row (typical sizes 1KB-4KB for DDR3) by frequently activating an adjacent row. In other words, segmented memory or page protection cannot guarantee isolation of two or more programs when they are using memory mapped to adjacent physical rows.

In June 2014 Kim et al. [19] empirically demonstrated that despite the best effort to mitigate disturbance errors, a high percentage of DRAM modules suffer from this phenomenon (i.e. 110 modules out of 129 DDR3 modules from three different memory manufacturers were evaluated). In particular for the most recent modules, those dated in the last two years, all but one module were affected.

The reliability issue of DRAMs becomes even more critical considering the recent approach of datacentres to keeping entire databases in DRAM like RAMCloud (e.g. 64 TB of DRAMs) [1, 30, 29]. In general, the recent industrial and academic trend toward the big-data analytics based projects increases the demand for data security and reliability.

Moreover, the scalability of emerging memory technologies demands a more scalable and robust fault detection technique. Hybrid Memory Cube ("HMC") is one examples of such emerging memory technologies introduced by Micron [23]. The 3D structure of HMC, having more banks than traditional DRAM systems, provides 15x more bandwidth than conventional DDR3 modules and requires 70% less energy and 90% less space than existing modern memory technologies.

To address the reliability issue, especially for datacentres and servers, a significant cost and effort is required. For instance, using traditional Error Correction Code

("ECC") modules for server-grade systems imposes a 12.5% capacity overhead to memory systems [19]. Even in this situation, ECC modules only can correct single-bit errors which is not efficient in the case of multi-bit disturbance errors. Therefore, considering the emergence of new memory technology and an industrial approach toward developing hardware support for big-data analytics, the present inventors believe it is time to address disturbance errors more seriously.

The present invention has been devised in light of the above considerations.

US2014/0006704 discloses a system that monitors data accesses to specific rows of a memory to determine if a row hammer condition exists. The system can monitor accessed rows of memory to determine if the number of accesses to any of the rows exceeds a threshold associated with risk of data corruption on a row of memory physically adjacent to the row with high access. Based on the monitoring, a memory controller can determine if the number of accesses to a row exceeds the threshold, and indicate address information for the row whose access count reaches the threshold.

The following patent applications may be also considered in relation to the row hammer effect:

- US20140281206
- US2014/0059287
- US2014/0089576
- US2014/0095780
- US2014/0177370
- US2014/0281207
- US2014/0189228
- US2014/0085995
- US2014/0006703
- US2014/0156923
- US2014/0095947
- US2008/0253212
- US2010/0074042
- US2010/0172200
- US2011/0219197
- US2014/0003173
- US2014/0095780
- WO2014/004111
- WO2014/004748
- WO2014/084917
- WO2014/099031

The row hammer effect has been discussed in various recent sources [37-54].

A first aspect of the invention may provide:

A monitoring device for monitoring the accessing of data items in a memory, the monitoring device including:

a first monitoring unit configured to:

monitor a stream of access requests/commands, wherein each access request/command is directed to a data item in the memory;

determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access; and

if it is determined that a data item meets the condition indicative of frequent access, notify a second monitoring unit of the data item; and

the second monitoring unit, the second monitoring unit being configured to:

monitor the accessing of data items notified to the second monitoring unit by the first monitoring unit.

Thus the first monitoring unit, based on the stream of access requests/commands, determines whether any of the data items to which access requests/commands are directed (by the stream of access requests/commands) meet a condition indicative of frequent access. In other words, the stream of access requests/commands is tested to determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access.

By appropriately setting the condition indicative of frequent access, only a subset of data items to which access requests/commands are directed by the stream of access requests/commands (i.e. the subset of data items that meets the condition indicative of frequent access) need be notified to and monitored by the second monitoring unit. In other words, the second monitoring unit does not need to monitor the accessing of the subset of data items to which access requests/commands are directed but do not meet the condition indicative of frequent access.

Since the second monitoring unit does not need to monitor the accessing of all data items to which access requests/commands are directed, the burden on the second monitoring unit is reduced, and the method is rendered more efficient than a method in which the accessing of all data items to which access requests/commands are directed is monitored in the same way.

Herein, an access request/command directed to a data item in a memory may be viewed as any request or command for initiating a process involved in accessing data contained in the data item.

For the avoidance of any doubt, a data item in the memory could be an individual data cell or a group of such data cells, such as an address line (e.g. row) of data cells.

Typically, when a processor sends a read/write request to a memory controller, the controller will produce multiple commands to service that request. For example, to service a read request, a memory controller of a DRAM may issue an activation command, a read command, and possibly a pre-charge commands.

Thus, for a typical memory (e.g. a DRAM), there are various types of request/command that could be viewed as an access request/command.

For example, any of the following could be viewed as an access request/command:

an activation command for activating an address line (e.g. a row) in the memory (here, the address line can be viewed as the data item to which the activation command is directed);

a write request/command for writing data to one or more addressable data cells in the memory (here, the one or more addressable data cells being written to, or an address line containing such data cells, can be viewed as the data item to which the write request/command is directed);

a read request/command for reading data from one or more addressable data cells in the memory (here, the one or more addressable data cells being read from, or a row containing such data cells, can be viewed as the data item to which the read request/command is directed).

It follows that the stream of access requests/commands may be any of:

a stream of activation commands, wherein each activation command is for activating an address line (e.g. a row) in the memory;

a stream of write requests/commands, wherein each write request/command is for writing data to one or more addressable data cells in the memory;

a stream of read requests/commands, wherein each read request/command is for reading data from one or more addressable data cells in the memory.

Preferably, the stream of access requests/commands is a stream of activation commands, since as discussed in more detail below, this is particularly useful for identifying "hot rows". However, as discussed in more detail below, the inventors believe that the same principles may find use with various other types of access request/command.

Preferably, the condition indicative of frequent access is met by a data item to which an access request/command is directed, if a portion of the stream of access requests/commands corresponding to a recent time window includes more than a predetermined number of access requests/commands (e.g. more than one access request/command) directed to that data item.

Thus, the first monitoring unit may be configured to determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access by testing each data item to which an access request/command is directed so as to determine whether a respective portion of the stream of access requests/commands corresponding to a recent time window includes more than a predetermined number of access requests/commands (e.g. more than one access request/command) directed to that data item.

In some embodiments, each data item to which an access request/command is directed may be tested using a different portion of the stream of access requests/commands (e.g. as is the case with the specific example described below with reference to Fig. 6). However, in other embodiments, multiple data items to which access requests/commands are directed may be tested using the same portion of the stream of access requests/commands.

In some embodiments, each data item to which an access request/command is directed may be tested using a respective portion of the stream of access requests/commands that includes an access request/command directed to that data item (e.g. as is the case with the specific example described below with reference to Fig. 6, where the left most memory element stores an identifier of the data item being tested).

The predetermined number of access requests/commands (used in determining whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access) may be one (e.g. as is the case with the specific example described below with reference to Fig. 6). However, a skilled person will appreciate that a different predetermined number of access requests/commands could be used, e.g. depending on the type of access request/command being monitored, the size of the recent time window, and the purpose of the monitoring.

Note that the predetermined number of access requests/commands (used in determining whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access) may be a fixed number, or may be dynamically adjusted (e.g. dynamically increased during a refresh interval of the memory, as described in connection with a specific example described below with reference to Fig. 6).

Preferably, the/each portion of the stream of access requests/commands corresponding to a recent time window is divided into N time intervals of predetermined length, where N is an integer.

The predetermined length of each of the N time intervals may correspond to a minimum time interval to perform an access request/command. For example, if the stream of access requests/commands is a stream of activation commands, then predetermined length of each of the N time intervals may correspond to a minimum time interval to perform an activation command.

In this way, if a given portion of the stream of access requests/commands corresponding to a recent time window includes an access request/command directed to a given data item in more than a predetermined number (e.g. in more than one) of the N time intervals, then it can be inferred that the portion of the stream of access requests/commands corresponding to the recent time window includes more than the predetermined number of access requests/commands directed to that data item (e.g. thereby helping to determine if a condition indicative of frequent access as proposed above has been met).

Preferably, the first monitoring unit includes a register with N memory elements configured to store a portion of the stream of access requests/commands corresponding to a recent time window divided into N time intervals of predetermined length, where N is an integer. Preferably, each memory element in the register corresponds to a respective one of the N time intervals so that if the respective time interval includes an access request/command directed to a data item, then an identifier of that data item is stored as an entry in the memory element. In this way, if an identifier of a given data item is stored in more than a predetermined number (e.g.

more than one) of the N memory elements, then it can be inferred that the portion of the stream of access requests/commands corresponding to the recent time window stored in the register includes more than the predetermined number of access requests/commands for that data item (e.g. thereby helping to determine if a condition indicative of frequent access as proposed above has been met).

Preferably, the first monitoring unit is configured to update the portion of the stream of access requests/commands stored in the register at regular intervals, e.g. each time a further time interval of the predetermined length elapses, e.g. so that a different portion of the stream of access requests/commands is stored in the register each time a further time interval of the predetermined length elapses.

Preferably, the register is a shift register. In this case, the first monitoring unit may be configured to update the portion of the stream of access requests/commands stored in the register each time a further time interval of the predetermined length elapses by:

shifting entries stored in the memory elements of the shift register by one memory element away from a memory element at an input end of the shift register, thereby shifting out an entry stored in a memory element from an output end of the shift register; and

if a portion of the stream of access requests/commands corresponding to the further time interval includes an access request/command directed to a data item, then an identifier of that data item is stored as an entry in the memory element at the input end of the shift register.

The size of the recent time window (and/or the number of time intervals/memory elements, N) may depend, for example, on the type of access request/command being monitored, as well as the purpose of the monitoring.

The steps involved in monitoring (at the second monitoring unit) the accessing of the data items notified to the second monitoring unit may depend, for example, on the type of access request/command being monitored as well as the purpose of the monitoring. For the avoidance of any doubt, monitoring the accessing of the data items notified to the second monitoring unit may involve monitoring any process that involves accessing the data items.

Preferably, the second monitoring unit is configured to identify one or more frequently accessed data items (which items may be referred to as "hot" data items, such as "hot rows") from the data items notified to it by the first monitoring unit. As discussed in more detail below, this may be useful for avoiding "row hammer" errors (if the stream of access requests/commands being monitored is a stream of activation commands).

Preferably, the second monitoring unit is configured to: for each data item notified to the second monitoring unit, count (e.g. in a respective access counter corresponding to the data item) the number of times the stream of access requests/commands includes an access request/command directed to that data item.

In this case, the second monitoring unit may be configured to: for each data item notified to the second monitoring unit, identify the data item as a frequently accessed

data item if the counted number of times the stream of access requests/commands includes an access request/command directed to that data item (e.g. as stored in the corresponding access counter) reaches a predetermined threshold.

However, a skilled person would appreciate that counting is not essential to the invention, since the accessing of the data items notified to the second monitoring unit could be monitored in other ways. For example, the data items notified to the second monitoring unit could be recorded in a table, with data items being promoted up the table each time the stream of access requests/commands includes an access request/command directed to that data item. This would still enable one or more frequently accessed data items to be identified according to table position, without necessarily counting the number of times each candidate address line is accessed.

Identifying data items notified to the second monitoring unit as frequently accessed data items could have various uses in the context of a memory.

For example, the second monitoring unit may be configured to: if a data item is identified by the second monitoring unit as a frequently accessed data item (e.g. as a "hot" data item, such as a "hot row"), flag the data item as a frequently accessed data item for use by a memory buffer configured to temporarily store data from the memory.

The second monitoring unit may therefore be configured to notify the memory buffer of a data item it has flagged as a frequently accessed data item.

Preferably, the memory buffer is configured to determine its content based on a data item flagged as a frequently accessed data item by the second monitoring unit. For example, the memory buffer may be configured to determine its content based on a data item flagged as a frequently accessed data item by:

storing the frequently accessed data item or a block of data including at least part of the frequently accessed data item in the memory buffer; and/or

retaining the frequently accessed data item or a block of data including at least part of the frequently accessed data item in the memory buffer.

In this way, at least part of the frequently accessed data item (preferably the entire frequently accessed data item) can be accessed directly from the memory buffer, i.e. without having to retrieve the frequently accessed data item from the memory (which may take considerably longer, e.g. in the case of a DRAM). As discussed below in more detail, this is one possible route to avoiding a row hammer error (in the case that the stream of access requests/commands being monitored is a stream of activation commands).

By way of example, the memory buffer could be a CPU cache, e.g. a cache of a CPU in a computer system containing the memory. In this case, the CPU cache could be configured to determine its content based on a data item flagged as a frequently accessed data item by:

storing one or more cache lines that include at least a part of the frequently accessed data item in the CPU cache;

retaining one or more cache lines that include at least a part of the frequently

accessed data item in the CPU cache (e.g. that would otherwise have been evicted from the CPU cache, e.g. according to an eviction policy of the cache).

Alternatively, the memory buffer could be a "dedicated" memory buffer configured to store data items identified by the second monitoring unit as frequently accessed data items. In this case, the dedicated memory buffer could be configured to determine its content based on a data item flagged as a frequently accessed data item by:

storing the frequently accessed data item in the dedicated memory buffer;

retaining the frequently accessed data item in the dedicated memory buffer.

The dedicated memory buffer could, for example, be included in the monitoring device (which may be a memory controller, see below), along with the first monitoring unit and second monitoring unit.

Preferably, the second monitoring unit is configured to: for each data item notified to the second monitoring unit, if the data item meets a condition indicative of non-frequent access, flag the data item as a candidate for eviction from the second monitoring unit or evict the data item from the second monitoring unit. This may be helpful to avoid false identification of frequently accessed data items and/or reduce the processing burden on the second monitoring unit.

The condition indicative of non-frequent activation could be implemented by having, for each data item notified to the second monitoring unit, a respective timer corresponding to the data item that is reset to an initial value each time the data item is notified to the second monitoring unit, e.g. with the condition indicative of non-frequent activation being met by a data item if the timer corresponding to the data item reaches a value corresponding to a period of time of predetermined length).

Preferably, the second monitoring unit is configured to: for each data item notified to the second monitoring unit, and for each of a plurality of periods of time of predetermined length: count in a respective credit counter corresponding to the data item the number of times the address line is accessed more than a predetermined number of times (e.g. more than twice) in the period of time of predetermined length.

Here, the period of time of predetermined length may correspond to the size of the recent time window noted above.

Preferably, the second monitoring unit is configured to: for each data item notified to the second monitoring unit, and for each of the plurality of predetermined periods of time, decrement the credit counter if the data item is not activated in the predetermined period of time.

As described in the example discussed below, the credit counter may be useful to account for non-uniform distributions in how the address lines are activated. For example, the credit counter corresponding to a data item flagged as a candidate for eviction may be used in a decision to determine whether to evict that data item.

Preferably, the monitoring device is configured to perform the above mentioned steps after each refresh of the memory (as is typically needed by volatile memories).

Preferably, the monitoring device is or forms part of a memory controller for controlling the memory.

For the avoidance of any doubt, the first monitoring unit and second monitoring unit may be separate physical units within the monitoring device (which may form part of a memory controller, see above) or could be implemented as software within a computational device (e.g. a microcontroller) that forms part of the monitoring device.

The memory may include a plurality of addressable memory cells, which may be arranged in a plurality of address lines (e.g. rows and/or columns).

The monitoring device may be included in a memory architecture including the device for monitoring data items and the memory. The memory architecture may further include a CPU cache as described above.

The memory may be a volatile memory.

The memory may be a random access memory, i.e. a RAM.

The memory may be a dynamic random access memory, i.e. a DRAM. As discussed below, "row hammer" is a known problem for DRAMs. A DRAM may include a plurality of addressable memory cells arranged in a plurality of address lines (e.g. rows and/or columns), where each addressable memory cells includes a capacitor.

However, this is only an example, as the device for monitoring data items could be used with various different types of memory.

The first aspect of the invention may provide a monitoring method performed by a monitoring device as set out above. The monitoring method may include a method step corresponding to any one or more features the monitoring device is configured to perform, as described above or below.

A second aspect of the invention may provide a monitoring device (or monitoring method) according to the first aspect of the invention, wherein the stream of access requests/commands for data items within a memory is a stream of activation commands, wherein each activation command is for activating an address line (e.g. a row) in the memory.

In this context, the term "access" (and derivatives thereof) in the first aspect of the invention may be replaced by the term "activation" (and derivatives thereof) in the second aspect of the invention since, as discussed above, an activation command is one type of access request. Similarly, the term "data item" may be replaced with "address line", since in the case of an activation command, the address line can be viewed as the data item to which the activation command is directed.

Thus, the second aspect of the invention may provide:

A monitoring device for monitoring the activation of address lines in a memory, the monitoring device including:

a first monitoring unit configured to:

monitor a stream of activation commands, wherein each activation command is directed to an address line in the memory;

determine whether any of the address lines to which activation commands are directed meet a condition indicative of frequent activation; and

if it is determined that an address line meets the condition indicative of frequent activation, notify a second monitoring unit of the address line; and

the second monitoring unit, the second monitoring unit being configured to:

monitor the activation of address lines notified to the second monitoring unit by the first monitoring unit.

The monitoring device according to the second aspect of the invention may include any feature described in connection with the first aspect of the invention, except that the stream of access requests/commands is a stream of activation commands and each data item is an address line.

The following discussion provides some preferred features where the stream of access requests/commands is a stream of activation commands.

If the/each portion of the stream of activation commands corresponding to a recent time window is divided into N time intervals of predetermined length, where N is an integer, then the number of time intervals N may be derived from an activation threshold (ACT_{th}), which may be taken as a minimum number of activations required to induce a row hammer error, and/or a maximum number of activations per refresh interval of the memory (MAX_{ACT}).

In an example described below, the number of time intervals N included in the recent time window is chosen to be $MAX_{aggressor} + 1$, where $MAX_{aggressor}$ is a maximum number of aggressor rows. $MAX_{aggressor}$ can be derived from ACT_{th} and MAX_{ACT} as shown in Equation (2).

Preferably, the/each portion of the stream of activation commands corresponds to a recent time window that is at least the size of a time window (e.g. a Hot Time Window or "HTW") in which an address line must be activated at least once to be identified as a potential hot row (e.g. assuming activation commands are uniformly distributed).

The monitoring device may be configured to, if an address line is identified as a frequently activated address line (e.g. as a "hot row") by the second monitoring unit, initiate the refreshing of one or more address lines that are adjacent to the frequently activated address line. As discussed below, this is another possible route to avoiding a row hammer error (which may be used as an alternative to or in addition to the memory buffer solution described above).

The invention also includes any combination of the aspects and preferred features described except where such a combination is clearly impermissible or expressly avoided.

Examples of the present proposals are discussed below, with reference to the accompanying drawings in which:

Fig. 1 shows a proposed memory architecture.

Fig. 2 shows a) a DRAM Rank containing DRAM devices, b) a DRAM device containing DRAM banks, DRAM arrays and DRAM row buffers and c) a DRAM cell as it exists in a DRAM array.

Fig. 3 shows the row hammer phenomenon, as it occurs in the array of cells of a typical DRAM memory bank. Row victims necessarily exist adjacent to aggressively activated rows (row aggressors).

Fig. 4 shows the arrangement of 460ns hot time windows within an interval of a 64ms refresh cycle.

Fig. 5 shows an overview of a proposed ARMoR architecture, containing a clock generator, time-based shift register and dynamic counter allocator. The dynamic counter allocator is further comprised of a counter allocator, eviction unit and hot row table.

Fig. 6 shows a time-based shift register filter as used for filtering out non-potential hot rows within the proposed ARMoR architecture. Comparison of the last entry of the shift register with all subsequent entries is shown.

Fig. 7 shows the proposed ARMoR hot row table as it exists in the dynamic counter allocator. The hot row tables consists of N entries sorted by, full flag, update flag, update timeout, credit counter, expiry counter, ACT counter and ACT address.

Fig. 8 shows the average activation interval across 8 DRAM banks for each benchmark workload within ARMoR.

Fig. 9 shows the induced unique number of row aggressors for different access distributions within ARMoR.

Fig. 10 shows the total number of row aggressors during extraction time, according to different access distributions within ARMoR.

Fig. 11 shows the maximum activations per refresh interval of a row within a refresh interval for each benchmark workload within ARMoR

Fig. 12 shows the reduced performance overhead of ARMoR compared with PARA.

Fig. 13 shows performance overhead of ARMoR and PARA for different 8-thread workload mixes.

Fig. 14 shows the ARMoR performance overhead for synthetic kernels with various access distributions.

Fig. 15 shows the PARA performance overhead for synthetic kernels.

Fig. 16 shows the PARA miss-rate for synthetic kernels.

Fig. 17 shows the ARMoR MT-fluid hot row table performance in terms of detection accuracy and prediction error.

Fig. 18 shows the required number of table entries for ARMoR to detect all the possible row hammer errors for different access distributions.

Fig. 19 shows the ARMoR storage overhead for different memory capacities.

Fig. 20 shows the ARMoR execution time improvement considering buffering entire row(s) for different access distributions.

Fig. 21 shows the ARMoR execution time improvement considering buffering cache lines for different access distributions.

The proposed memory architecture 100 shown in Fig. 1 includes a proposed memory controller 102 for controlling a memory 104.

The memory 104 preferably includes a plurality of addressable memory cells which are arranged in a plurality of address lines (rows and columns).

The memory controller 102 includes a first monitoring unit 110 and a second monitoring unit 120 which, for the avoidance of any doubt, may be separate physical units within the memory controller 102 or could be implemented as software within a computational device (e.g. a microcontroller) that forms part of the memory controller 102.

The first monitoring unit 110 is preferably configured to monitor a stream of access requests/commands, wherein each access request/command is directed to a data item in the memory 104; determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access; and if it is determined that a data item meets the condition indicative of frequent access, notify the second monitoring unit 120 of the data item.

The second monitoring unit 120 is preferably configured to monitor the accessing of data items notified to the second monitoring unit 120 by the first monitoring unit 110.

Preferred features of the memory controller 102, the first monitoring unit 110 and the second monitoring unit 120 have already been discussed above.

As discussed in detail above, the proposed memory architecture 100 could usefully be applied to various different types of access request/command, e.g. to activation commands, write requests/commands or read requests/commands.

However, the following discussion focusses on specific examples in which the stream of access requests/commands is a stream of activation commands, and the memory architecture 100 is implemented for the purpose of identifying frequently activated address lines ("hot rows") in a DRAM, with a view to avoiding row hammer errors. The specific examples discussed below are sometimes referred to for simplicity as "ARMoR", which stands for A Run-time Memory hot row detector to prevent Row hammer data corruption in DRAMs.

Nonetheless, a skilled person will readily appreciate that the specific examples described below represent just one mode of implementing the present invention, and that the present invention could be implemented to other types of access request/commands (i.e. not just activation commands) and for purposes other than detecting row hammer errors. Therefore the following discussion and corresponding drawings should be considered as being illustrative and not limiting on the scope of the invention.

1. Overview

The specific examples described below (referred to for brevity as "ARMoR") are believed to provide an efficient and scalable technique for overcoming disturbance errors in DRAMs and future memory technologies. To the inventors' knowledge, this is the first proposal for detecting "hot rows" at run time with high level of accuracy to prevent row hammer errors (a specific type of disturbance error) and improve the reliability of overall memory systems.

ARMoR can be viewed as a novel hardware technique which may improve memory controllers by detecting which specific rows are at risk of row hammer, without a need to stop any programs. In addition, ARMoR may be able guarantee 100% or near 100% detection and prevention of row hammer effect with minimal execution time overhead and hardware requirements. In experimental results discussed below, over 18 standard workloads and 36 synthetic memory intensive kernels show that ARMoR incurs virtually no execution time overhead for the workloads; a maximum of $3.5x10^{-3}$ %. When buffers are added to the memory controller to capture rows at risk of row hammer, the execution times are sped up with small hardware costs.

2. Background and Motivation

This section presents general background on the internal structure and operation of DRAMs to explain the "row hammer" data corruption effect.

2.1. DRAM Organization

The typical smallest storage unit of a DRAM device is a DRAM cell that is depicted in Fig. 2c. A typical DRAM array consists of DRAM cells arranged in address lines normally referred to as rows and columns. A typical DRAM bank is composed of a DRAM array and a set of sense amplifiers (or Row Buffer). A DRAM device (Fig. 2b) typically includes multiple banks which work independently. However, since there are typically shared resources between banks (e.g. a data bus), usually only one bank can used for a read or write operation at a given time. Typically, each DRAM device has a narrow data bus (e.g. 4-16 bit). Thus, multiple DRAM devices normally work together in parallel within a rank (Fig. 2a) to support a required bandwidth.

2.2. DRAM Operation

To perform a read or write operation in response to a read/write request, a few steps orchestrated by the memory controller are required. First, a target row must be opened using an activation command. Issuing this command brings the target row into a row buffer. Then a read or write command can be issued to access the desired data. Since, there is only one row buffer per bank, to access a different row within a

bank the opened row must normally be closed using a pre-charge command. This command prepares the row buffer to accept a new row for future access requests [14]. Consecutive accesses to different rows within a bank are called row misses and impose extra power and time overheads. We use these constraints to evaluate the overhead of our proposal in Section 7.

Fig. 2c presents a low level structure of an example DRAM cell. Each DRAM cell typically consists of a transistor and a capacitor which are connected to the bitline wire and wordline wire. Typically, a fully charged capacitor holds logical '1' and fully discharged capacitor holds logical '0' value. A wordline is connected to all cells located in a row and a bitline is connected to all rows in a column. To access a row the corresponding wordline will be raised to a high voltage. This operation enables all the transistors in a target row and, as a result, the capacitors will be connected to their corresponding bitlines. In this way, the row data, which are the capacitors' charges, will be transferred to the row buffer and the data will be available from there. This entire operation is usually called row activation, and is typically performed in response to an activation command. In this process the capacitors first will be discharged and then resorted to their former value. At the end, by lowering the wordline the transistors will be turned off and capacitors will be disconnected from bitlines [14, 13, 17].

2.3. DRAM Refresh

Considering the electronic circuit of an example DRAM cell (Fig. 2c) the charge stored in these cells is not persistent. This is because capacitors will lose their electric charge over the time due to various types of leakage [32, 31]. Thus, each DRAM cell has a limited retention time. According to the DDR3 DRAM specification [33], each DRAM cell should have a retention time of 64 ms, which means after 64 ms, DRAM cells are susceptible to lose their data. Therefore, all the DRAM cells should be refreshed every 64 ms to sustain data reliability. To refresh a DRAM cell the memory controller issues a refresh command at specific time intervals to make sure that all the rows within the system are refreshed at least every 64 ms. Originally, one refresh command was issued by memory controller to every row in the system. However, as the size of memory and number of rows increase in modern DRAM devices issuing one refresh command per row is not practical anymore. Thus a typical modern memory controller only issues a fixed number of refresh commands within a refresh period (e.g. 8192 Refresh Commands) and each refresh command refreshes multiple rows at the same time.

2.4. Row hammer Effect - Corrupting Data without Writing

The row hammer effect is not well known, but was highlighted by a test equipment company called Teledyne LeCroy [26, 21] in the context of DDR4 DRAM. They observed that 'aggressive' activations of a specific row in a DRAM can corrupt adjacent rows' data. In reality this phenomenon is a specific type of disturbance error that occurs due to intensive interaction between electronic components that are supposedly isolated from each other [19].

The row hammer effect in DRAMs can occur when a specific wordline of a DRAM cell is activated repeatedly within one refresh interval. In this situation the neighbouring

cells leak charge at faster rate than expected. Thus, the retention time of such cells is less than 64 ms which means that these cells may lose their data (charge) before refresh happens. Therefore, in refreshing these corrupted cells the wrong data will be read and written back again to the DRAM cell (Fig. 3). Moreover, ECC modules are not very efficient in this situation since they cannot detect multi-bit errors.

Kim et al. [19] provided the first empirical study in a peer-reviewed comprehensive study that demonstrated the existence of disturbance errors and more specifically the row hammer effect in commodity DRAM devices. In the specific examples discussed below, the focus is on proposing a hardware solution to overcome the row hammer effect instead of proving of its existence.

A possible straightforward solution considered by the inventors to mitigate row hammer was to increase the refresh rate for all the rows in the memory system. However, although this approach might alleviate the row hammer effect, it would also impose an unnecessary power and performance overhead to the system. Another solution considered by the inventors was to detect the rows with high activation values (i.e. 'hot rows') and refresh their (physical) neighbours. A simple method to recognize hot rows in a DRAM is to dedicate a counter per row to keep track of number of activations of each row. However, having one counter per row induces a significant area and power overhead to memory system. Kim et al. [19] also investigated different general techniques used to find frequent items from stream of items, such as: Bloom Filters [5, 9], Morris Counters [28] and some other standard techniques [16] to identify the hot rows. However, none of the mentioned techniques is scalable as we increase the size of main memory or cannot guarantee 100% successful row hammer identification/correction.

In the specific examples discussed below, there is proposed an architecture for achieving native memory hot row identification/detection, instead of trying to apply traditional techniques to find hot rows (note in some literatures, hot rows may be referred to as "hot pages").

3. Row Hammer: Analytical Analysis

To develop an optimized solution for the row hammer effect, it is important to understand when this phenomenon might happen. There are a few effective factors that should be taken into consideration when analysing row hammer problems that are briefly explained next, though the inventors do not wish to be bound by any theory stated herein.

Refresh Interval (RI): it has been discussed that every 64 ms a typical DRAM must refresh all the rows in the system. Therefore, RI may be taken as the maximum time period that row hammer problems can be accrued.

Activation Threshold (ACT_{th}): may be taken as the minimum number of activations that are required to induce a row hammer error. Kim et al. [19] have done an extensive evaluation of row hammer effect on a wide range of DRAM devices (e.g. 129 DRAM Modules) from three major DRAM manufacturers and they ended up with three different values for ACT_{th} for this set of DRAM devices which is presented in Table 1. In the experimental work set out below, ACT_{th} was assumed to be 139K.

Modules	ACT_{th}
Module A	139K
Module B	155K
Module C	284K

Table 1: ACT_{th} for different evaluated DRAM modules in [19].

Minimum Activation Interval (MIN_{AI}): may be taken as the minimum time interval to activate a row within the DRAM device. This value is typically limited to TRC – that is the time interval between accessing a row and restoring data back to DRAM array plus the pre-charge time. In the experimental work described below, TRC and therefore MIN_{AI} is around 49 ns.

Maximum Possible ACT per RI (MAX_{ACT}): may be taken as the maximum possible number of activation commands that can be issued to a bank within RI, which based on the above definitions may be taken as:

$$MAX_{ACT} = \frac{RI}{MIN_{AI}} = \frac{64ms}{49ns} \approx 1.3\,M \qquad (1)$$

Aggressor row or hot row: may be taken as a repeatedly activated/opened row that causes a row hammer error in neighbours by the row hammer effect.

Victim row: may be taken as a neighbour of an aggressor row that is affected by row hammering.

Maximum Possible Aggressors Per RI ($MAX_{aggressor}$): may be taken as the maximum possible number of aggressor rows per RI which, considering the MAXACT and the ACT_{th}, and assuming ACT_{th} =139K, can be calculated from Equation (2) as follows:

$$MAX_{aggressor} = \frac{MAX_{ACT}}{ACT_{th}} = \frac{1.3M}{139K} \approx 10 \qquad (2)$$

$MAX_{aggressor}$ suggests that we are looking for few aggressor rows (approx. 10 aggressor rows) per bank per RI out of millions of existing rows in the system. This leads the inventors to the conclusion that there should be a specific behaviour in the activation stream (stream of activation commands) that creates aggressor rows, which is one of the main principles behind ARMoR.

4. ARMoR: A Run-time Memory Hot Row Detector

4.1. ARMoR - Basic Principles

In section 3 above, it was derived that there are typically only a few rows that can be a potential aggressor row within a typical RI (e.g. $MAX_{aggressor}$ is approx. 10). Thus, in the proposed architecture discussed below, only the activation behaviour of potential hot rows is monitored in detail, rather than monitoring the activation of all the

available rows in the system. Devising an architecture capable of efficiently detecting potential hot rows to achieve this end led to the proposals set out herein.

In Section 3, above, it was suggested that there might be a specific behaviour in the stream of activation commands issued to a bank to cause the row hammer effect. To identify such a pattern of activation commands at run time, let's assume that the stream of activation commands directed to hot rows follows a uniform distribution. This would mean that there is a fixed time interval between each activation command directed to a hot row in a DRAM's bank. In this situation, it is possible to define a minimum requirement that each row must satisfy to be identified as frequently activated, i.e. "hot".

Potential Hot rows Condition: considering our previous assumptions (e.g. of uniform distribution of activation commands directed to a hot row), we may define a time window, e.g. Hot Time Window (HTW), in which a row must be activated at least once to be identified as a potential hot row within RI (Fig. 4), which may be calculated as follows:

$$HTW = \frac{RI}{ACT_{th}} = \frac{64ms}{139K} \approx 460\ ns \qquad (3)$$

As Equation (3) shows, the RI is divided to 139K different HTWs (of 460 ns each) which means that for a given row to become a hot row within a given RI, there should be at least one activation command directed to that specific row in each HTW, otherwise the row is not going to reach the ACT_{th}. Therefore, on the assumption of a uniform distribution, if there are no activation commands directed to a given row within a given HTW, then that row is not going to be a hot row within the RI. However, if there are one or more activations commands directed to a given row within an HTW then the target row might be a potential hot row depending on the activation stream behaviour for the future HTW time slots. In this way a simple structure can be developed to filter potential hot rows at run time.

So far, a uniform distribution of the activation stream has been assumed for the sake of simplicity to explain the basic principles behind ARMoR. However, this is not always a valid assumption. The activation stream completely depends on the application behaviour and memory access pattern. Considering multithread and multicore systems the application access pattern – and as a result the activation stream – is most likely 'random'. The next section explains how to use similar principles to detect hot rows at run time for uniform and non-uniform distributions in the stream of activation commands using a simple counter (a credit counter).

4.2. ARMoR - Overview of an Example Architecture

In this section, we use the concepts presented so far to propose a novel and low-cost architecture to identify hot rows in DRAMs.

Fig. 5 shows an overview of a preferred ARMoR architecture that includes a first monitoring unit 210 and a second monitoring unit 220.

The first monitoring unit 210 is preferably configured to monitor a stream of activation commands (an "Activation Stream") directed to a DRAM (not shown); determine whether any of the address lines to which activation commands are directed meet a condition indicative of frequent activation; and if it is determined that an address line meets the condition indicative of frequent activation, notify a second monitoring unit of the address line.

The second monitoring unit 220 is preferably configured to monitor the activation of address lines notified to it by the first monitoring unit 210.

In the following discussion, the first monitoring unit 210 is implemented by a Time-based Shift-Register Filter ("TSRF") and the second monitoring unit 220 is implemented by a Dynamic Counter Allocator ("DCA").

At a high level of abstraction there are two main phases to identifying hot rows using the ARMoR architecture of Fig. 5. In a first phase, the first monitoring unit 210 (TSRF) determines whether any of the rows to which activation commands are directed meet a condition indicative of frequent activation and notifies the second monitoring unit 220 (DCA) of any rows determined to meet this condition (for brevity, any row meeting this condition are described herein as a "potential hot row"). In a second phase, the second monitoring unit 220 (DCA) monitors the activation of the potential hot rows notified to it by allocating a respective counter to each potential hot row to keep track of the number of activations to these rows, so that the DCA 220 can identify hot rows based on the counted number of activations. Moreover, the DCA 220 is preferably able to evict potential hot rows (e.g. by de-allocating counters) when they lose their eligibility to be a hot row at run time.

4.2.1. Time-based Shift-Register Filter ("TSRF")

Preferably, the first phase in identifying hot rows involves determining whether there are any rows that meet a condition indicative of frequent activation (referred to herein as "potential hot rows") that have the potential reach the ACT_{th} and notifying the DCA 220 of such potential hot rows. This phase could be characterised as involving filtering out non-potential hot rows to reduce the number of rows that need to be tracked at the DCA 220.

To implement the TSRF 210, there is proposed a simple time-based shift resister structure, as shown in Fig. 6. As its names implies, the contents of a shift register 212 is shifted (in this example to the left) every specific time interval. This time interval is chosen to be equal to MIN_{AI} to capture activation stream behaviour in fine-grain time windows. Preferably, the TSRF 210 checks every MIN_{AI} if an activation command has been issued in that period; if it has, an identifier (e.g. the address) of the requested row will be stored in the last location of shift register and, if there is no activation command in that time period, the last location will be marked as empty and the contents of shift register will be shifted to the left by one location.

In this way, the shift register 212 stores a portion of the stream of activation commands corresponding to a recent time window, with the portion of the stream of activation commands stored in the shift register 212 being updated (to store a

different portion of the stream of activation commands) each time a further time interval of duration MIN_{AI} elapses.

Each memory element in the shift register is preferably made big enough to hold an identifier (e.g. an address-bit) representing a requested row.

The shift register 212 has N memory elements, where N is an integer. Thus, the shift register 212 stores a portion of the stream of activation commands corresponding to a recent time window divided into N time intervals of predetermined length (in this case, the predetermined length of the time intervals is equal to MIN_{AI}).

The number of memory elements N ("TSRF Size" in Equation (4), below) in the shift register (i.e. the shift register length) is preferably chosen in such a way that the recent time window (represented by the content of the shift register) is at least as large as an HTW, whilst also being able to accommodate the maximum number of possible hot rows within RI (or $MAX_{aggressor}$) plus one, as shown in Equation (4) as follows:

$$TSRF\ Size = MAX_{aggressor} + 1 = 10 + 1 = 11 \qquad (4)$$

Note that a shift register with N = 11 memory elements will represent a time window of 539ns (11 x 49ns), which is bigger than HTW (460ns, see above), so the identifier (e.g. physical address) of a requested row will be kept in the queue for at least the length of an HTW.

Every MIN_{AI}, before discarding the identifier of a requested row stored in the left-most memory element of the shift register 212, the identifier stored in the left-most memory element of the shift register 212 is compared (in a comparator) to all the other entries in the shift register 212 to see if any of these other entries contains the same identifier as that stored in the left-most memory element of the shift register 212. If there is a match, then it can be inferred that the portion of the stream of activation requests corresponding to a recent time window stored in the shift register 212 includes more than one activation command directed to the row indicated by the identifier stored in the left-most memory element of the TSRF shift register.

In this way, the first monitoring unit 210 is able to determine whether any of the rows to which activation commands are directed meet a condition indicative of frequent activation by testing each row to which an activation request is directed so as to determine whether a respective portion of the stream of activation requests corresponding to a recent time window includes more than one activation command directed to that row.

The above-described TSRF 210 represents just one way in which to implement a condition indicative of frequent activation that is met by an address line to which an activation command is directed, if a portion of the stream of activation commands corresponding to a recent time window includes more than a predetermined number of activation commands (in this case more than one activation command) directed to that address line. Of course, such a condition could be implemented in various other ways, as could be envisaged by a skilled person.

If there is a match as described above (i.e. such that the condition indicative of activation is met by the row indicated by the identifier stored in the left-most memory element of the TSRF shift register), then the row indicated by the identifier stored in the left-most memory element of the TSRF shift register is marked as a potential hot row and notified to the DCA 220, otherwise it will be discarded.

Since the shift register is updated every MIN_{AI}, and since the number of memory elements N in the shift register is chosen to accommodate the maximum number of possible hot rows within an RI (or $MAX_{aggressor}$) plus one, this queue length should guarantee capture of all potential hot rows, even if those potential hot rows all happen in the same HTW (approx. 460 ns).

Note that it is possible for the maximum number of possible hot rows within an RI ($MAX_{aggressor}$) to be present, but for activations to those rows to be distributed in different HTWs. Therefore, the shift register (i.e. the shift register length) is preferably chosen in such a way that the recent time window (represented by the content of the shift register) covers a minimum time equal to HTW, whilst being at least as big as $MAX_{aggressor} + 1$.

Note that the one extra memory element ("plus one") in the shift register is to keep the desired time interval between the first and last item in the queue. In more detail, there is a time limit that for which two consecutive activation commands can be directed to a DRAM bank (MIN_{AI} = 49ns, see above). This means that after each activation, it is necessary to wait 49ns before an activation command can be issued to a different row. Using this concept, it was concluded above that there was maximum number of possible row aggressors that can happen within a given RI ($MAX_{aggressor}$ = 10, see above). Now if by any chance activation commands are issued for all of these row aggressors in the same HTW (note this is a worst case scenario, which might not happen in practice, depending e.g. on the size of the HTW), the one extra memory element still allows all the potential hot rows to be identified.

Although, in this way, an initial filtering is carried out on the stream of activation commands, some rows identified as potential hot rows to the DCA 220, might not subsequently be actually be identified as actual hot rows by the DCA 220. For example, a lot of activation commands to a particular row might result in that row being identified as a potential hot row by the TSRF 210, but that potential hot row might not necessarily be subsequently identified by the DCA 220 as an actual hot row (e.g. if the counted number of activations does not reach ACT_{th}, or if the potential hot row is evicted from the DCA table because it meets a condition for eviction – see below)

Further Filtering Improvements: The experimental results set out below show that the structure of TSRF 210 described above delivers satisfactory results. However, the performance of the TSRF 210 could be improved further considering a dynamic threshold to detect the hot rows. It is been discussed that having a minimum of two activations to the same row within HTW would be a reasonable threshold to consider a row as a potential hot row. However, this threshold is calculated based on the overall RI of 64 ms. This means that, as time progresses, the condition for a row to be a potential hot row will change as well. For instance if 32 ms of RI has passed then, considering our uniform distribution assumption, the minimum number of row

activation commands within the remaining HTW must be more than two (e.g. four) if the target row is to reach the ACT_{th} and be a hot row.

4.2.2. Dynamic Counter Allocator ("DCA")

The DCA 220 is in charge of allocation and de-allocation of counters to potential hot rows notified to the DCA 220 by the TSRF 210. In this example, the DCA 220 comprises an allocator unit 222, an eviction unit 224 plus a potential hot row table 226. An overview of the DCA structure is presented in Fig. 7.

As soon as a potential hot row is identified by TSRF 210, it is notified to the DCA 220 by sending its physical address to the DCA 220. The DCA consists of simple counters, timers and registers which are explained as follows.

Full Flag: is a 1-bit flag to specify if the current entry of hot row table is accepted or free.

Hot row Address: is a register to keep the address of potential hot row.

ACT Counter: is a counter allocated to a potential hot row to keep track of number of activations of that specific row.

Update Flag: is 1-bit flag to show if the corresponding potential hot row has been activated in the last MIN_{AI} or not.

Update Timeout: is a timer that keeps track of the HTW period. If there is no activation to a potential hot row within the HTW then the potential hot row become a candidate to be evicted from the hot row table. This timer will be reset to its initial value after every activation to the corresponding potential hot row.

Credit Counter: The presented architecture so far can only detect hot rows with a uniformly distributed activation stream. The Credit Counter is a simple, intelligent way to extend the design to be able to detect a hot row with non-uniform distribution of an activation stream. It has been discussed that one necessary requirement (but not the only one) for a specific row in DRAM to be a hot row is to have at least two activations per HTW (i.e. every 460 ns) otherwise the potential row becomes a candidate for eviction from the hot row table. However, depending on the activation stream distribution, it might be a case that there are several activations to the target row within one HTW but no activations for the next few HTWs (non-uniform distribution). In this situation, the target row still has a potential to reach the ACT_{th} and become a hot row. In this case, the Credit Counter comes into play. This counter keeps track of all the extra activations (i.e. more than two activations) within the HTW. This extra credit specifies the number of future HTW that the target row can still remain a potential hot row even if there is no activation to that row. This behaviour can be implemented simply by decrementing the credit counter every time that that there is no activation to the potential hot rows within HTW (i.e. Update Timeout = 0).

Of course, this represents just one methodology for handling a non-uniform distribution of activation commands. Alternative alternate methodologies for handling

a non-uniform distribution of activation command could also be envisaged by a skilled person.

Expiry Counter: is a counter that prioritises the order of eviction from the hot row table if there are several potential hot rows which are flagged as candidates to be evicted. In this way, a candidate row with higher number of activations will be evicted later than one with a lower number of activations. It provides extra time credit to the row with higher number of activations to remain a potential hot row if there are more activation commands to this row.

Finally, if the ACT Counter corresponding to a given potential hot row reaches ACT_{th}, then that potential hot row may be identified (e.g. flagged) by the DCA 220 as a hot row. If a potential hot row is identified as a hot row, then preferably a corresponding signal is sent to the memory controller for further actions which will be described in the next section. At the end of the RI, the potential hot row will be evicted from hot row table.

Hot row Table Size: considering the possibility of having maximum hot rows within RI (e.g. $MAX_{aggressors}$ approx. 10) the 5 maximum required hot row table size to accommodate all the possible hot rows is equal to $MAX_{aggressors}$ per each bank. Thus, a maximum of 80 entries in the hot row table (820 Byte of storage) is required to support a 4 GB DRAM system populated in 8 banks. However, this number is calculated for the worst case scenario and our experimental results show that the possibility of having $MAX_{aggressors}$ per RI per bank is negligible. Therefore, a much smaller hot row table would be enough to detect hot rows in most of situations.

The experimental results presented in Section 7 show that ARMoR can identify all the possible hot rows (based on a desired ACT_{th}) in the system with high level of accuracy of number of activation to each hot rows (i.e. 99.99%) .

5. ARMoR – Example Applications

In the previous section, an ARMoR architecture was proposed to detect hot rows in DRAMs. In general, having a knowledge of which are the hot rows in a memory architecture provides an opportunity to improve different aspects of functionality of the memory architectures. In this section we discuss how ARMoR could potentially be used to improve reliability and performance of DRAMs.

5.1. Prevent Row hammer

To recap, if a specific DRAM row is activated repeatedly – more than a certain threshold (ACT_{th}) within RI – the adjacent rows' data may be corrupted (i.e. the row hammer effect). It was mentioned in previous sections that the most intuitive way to solve this issue is to detect the hot rows and refresh their neighbouring rows. Since ARMoR provides the solution to detect hot rows in DRAMs, then the only step remaining is to refresh the victim rows.

It would be straightforward to refresh the victim rows for a given DRAM, provided the mapping scheme of logical to physical rows is known for that DRAM (this information would be known to the DRAM manufacturer, but such information is not always made public by the manufacturers). Knowledge of this mapping scheme would therefore

enable the memory controller to issue a refresh command, or a simple activation command, to the physically adjacent rows. This technique could be used with any hot row detection technique, as well as the presented work in [19]. The performance overhead imposed by refreshing victim rows with the ARMoR architecture is discussed below.

Performance overhead: one of the main advantages of the ARMoR architecture compared with the solution presented by Kim et al. [19] is that, since ARMoR detects the exact row aggressors, only the actual victim rows can be refreshed. This means that, in the worst case, for each identified hot row, a maximum of two adjacent rows need refreshing. We can say that it imposes a maximum of two extra page-misses cost (i.e. approx. 2 x TRC) to the memory system. While, in the solution presented in [19], the refreshed rows might or might not be row victims. This imposes an unnecessary overhead to performance and power of the memory system. This extra overhead is evaluated in Section 7, below. In Section 5.2, below, there is presented an alternative approach to overcome the problems caused by the row hammer effect, using some unique capabilities provided by the ARMoR architecture.

5.2. Performance Improvement

The experimental results and analytical analysis provided below shows that for most workloads, only a small number of rows tend to be repeatedly flagged as hot rows in consecutive refresh intervals.

The inventors have observed that accessing DRAMs requires following a fairly tight timing limitation since, in general it is necessary to activate a row, wait for a while then issue read command, wait for a while and so on. Typically, a DRAM has millions of rows. Now, since only 10-100 of DRAM rows are frequently accessed, among millions of rows, the inventors have observed that these hot rows could be stored/retained in a memory buffer outside the DRAMs, which would not need a complicated access protocol. By using such a memory buffer to store/retain hot rows, future requests to these hot rows could be serviced much faster than if the hot rows had to be retrieved from DRAMs. Moreover, using such a memory buffer could reduce or even eliminate row hammer errors, since the corresponding requests to these hot rows will not need to go to a DRAM anymore. Note that this solution to row hammer would not require the refreshing of victim rows, and therefore knowledge of the mapping scheme of logical to physical rows in DRAMs would not be required in order to implement this solution.

The following two different solutions incorporating the idea of using a memory buffer to store/retain hot rows were considered:

1) Notifying a Last Level Cache (LLC) of a CPU of the hot rows, e.g. so to avoid evicting from the LLC any cache lines that include at least a part of (i.e. are located in) the hot rows.

2) Storing the hot rows in a dedicated memory buffer located in the memory controller, i.e. so as to cache the hot rows.

In both of these solutions, the number of activation commands to the hot rows and, as a result, the number of possible row hammer occurrences, will be decreased. That is, memory latency could be reduced significantly since the highly accessed cache lines now either are stored in the memory controller or retained in the LLC.

To get an insight about the possible performance improvement that can be achieved using ARMoR, the performance improvement obtainable by the second of these two solutions is investigated below.

5.2.1. ARMoR – Notifying a Last Level Cache of the hot rows

As would be appreciated by a skilled person, a typical computer system includes the following memory hierarchy: CPU → L1-Cache → L2-Cache → Last-Level-Cache (LLC) → DRAM(s).

The terms "processor" and "CPU" may be used interchangeably herein.

Accessing to different levels of the memory hierarchy has different latencies. For instance, accessing to L1-Cache may for some example processors require around 5 clock cycles whereas accessing DRAMs typically requires 200~300 clock cycles. These numbers will of course vary from processor to processor.

The multi-level cache hierarchy is therefore designed to keep frequently accessed data close to processor to reduce the access latency imposed by accessing DRAMs.

In most memory architectures, every time the processor issues a read/write request, all the cache levels are searched to see if the requested data can be found otherwise the request is sent to the next level cache until it reaches the DRAM.

Caches have different size in different levels. For instance:

L1-Cache = 32KB
L2-Cache = 256KB
LLC = 8MB
DRAMs = 4GB~256GB

Thus, CPU caches (L1-Cache, L2-Cache, LLC) are usually significantly smaller than a DRAM. Therefore the CPU caches cannot cache all the frequently accessed data. As a result, caches use different eviction policies (like Last Recently Used (LRU)) to evict a cache line and replace it with a new data.

Thus, if the LLC evicts a cache line, the data contained in that cache line will be written back to the DRAM and will be removed from the LLC. Now, if the CPU requests the same data, the request will be sent to the DRAMs. This requires that one of the DRAM rows (one that holds the requested data) become ACTIVATED to service the request. Now if this happens a lot, several activation commands can be issued to the same row and the target row may become a hot row that might cause a row hammer error.

Therefore, it can be seen that if a memory controller notifies the LLC of a detected hot row, e.g. by sharing information about the detected hot row with the LLC, then the LLC can decide not evict any cache line(s) that correspond to the hot row. This could

help to reduce the activation commands to the hot row so that, over time, the hot row will avoid being frequently activated, thereby reducing the probability of row hammer error occurrence.

This could of course be implemented in various ways. For example, whenever LLC decided to evict a cache line (e.g. according to an eviction policy), it could send a request to the memory controller to see if the eviction candidate is in the hot row table or not. If it is in the hot row then LLC could decide not to evict that cache line and will instead looking for another cache line for eviction. Note that this process would avoid activating the hot row in the DRAM.

The performance improvement obtainable by this solution is not evaluated further in this document, though such an evaluation could if wanted be evaluated using an integrated, detailed memory controller with a full architecture simulator.

Note that although the process of notifying an LLC cache has been described with reference to hot rows, the same principles and memory buffer solution could equally be applied to any frequency accessed data item, e.g. as detected from a stream of read requests or write requests.

5.2.2. ARMoR – Dedicated Memory Buffer

The experimental and analytical results discussed below show that the number of hot rows are limited to a small number within specific time intervals (RI). This suggests that a small dedicated memory buffer with a suitable eviction algorithm would be enough to accommodate all the hot rows within different time intervals.

To have an insight about the required size of such a buffer we recap the basic principals described so far. A key point is that in the worst case scenario there are only few rows that can reach a certain threshold (ACT_{th}) within a specific period of time (RI). Therefore, the maximum buffer size could be defined by the maximum number of possible row aggressors per RI times the size of row (in the case that entire row is to be buffered). Moreover, reducing the size of this buffer to less than the maximum required size should not affect the accuracy of the ARMoR, so the size of the buffer will in general just be a trade of between area overhead and the desired performance improvement.

6. Experimental Methodology

To investigate the performance of the ARMoR proposal, we compare it against the alternative option of having one counter per row in DRAM to keep track of the exact number of activations within RI (referred to herein as the "ground truth") as well as the proposed solution by Kim et al. [19] ("PARA"). We monitor all the counters and check whether they reach the threshold which would corrupt data, i.e. ACT_{th}. As soon as one counter reaches ACT_{th} (i.e. hammered rows are detected) we check the ARMoR's hot rows table to see if the detected row exists in the table and if so what is the predicted number of activations.

We also evaluate PARA using a similar methodology. This means that every time that an actual hot row is detected using the embedded counters, for each row we check if PARA has issued any refreshes for detected hot rows or not. If it does we

assume that PARA has issued the correct refresh to the victim rows (despite that it has only 50% chance to issue the refresh to the possible row victim due to its nature).

We use the ACT_{th} parameters measured by Kim et al. [19]. We use RI = 64ms and ACT_{th} of 139K. Simulator: USIMM, a detailed DRAM simulator [7, 6], is the main simulation platform in our experiments. Table 2 contains the parameters used to configure USIMM. We implement ARMoR and PARA in USIMM to support direct comparison.

Model	**Description**	**Value**
Processor	Clock Speed	3.2GHz
	Pipeline Depth	10
	ROB Size	128
Memory system	Bus Speed	800 MHz
	Number of Channels	1
	Ranks Per Channel	1
	Bank Per Rank	8
	Row Per Rank	65,536
	Cache Line Per Row	128
	Cache Line Size	64 Byte

Table 2: USIMM Configuration Parameters Used in the Experiments.

Workloads: To have an extensive evaluation we used several workloads from different benchmark suites presented in Table 3:

Suites	Workload	Suites	Workload
PARASEC	black	SPEC	leslie
	face		libq
	ferret	BIOBENCH	mummer
	fluid		tigr
	freq	COMMERCIAL	Com1
	stream		Comm2
	swapt		Comm3
	MT-canneal		Comm4
	MT-fluid		Comm5

Table 3: Workloads and Corresponding Benchmarks Suites Used for Experiments

- Standard Benchmarks: To avoid bias in how we generate/execute the benchmark we use the memory traces generated for the Memory Scheduling Championship (MSC [6]) covering different benchmarks {PARSEC [4], SPEC [10], BIOBENCH [3]} and some anonymous commercial benchmarks {COMMERCIAL}. Most of the MSC traces are captured from a single-thread run of each workload except for MT-canneal and MT-fluid which are collected on a four-thread run of the corresponding PARSEC's benchmark. Also, we produce an extra 18 workloads, combinations of 8-thread standard benchmarks, to increase the randomness of access patterns in our experiments.

- Synthetic Kernels: Using the standard benchmarks we found that only one case exposed row hammer scenarios. Thus, to explore scenarios with rows hammered we use synthetic kernels. We simulated 500M instructions of different memory intensive synthetic kernels and captured the memory traces. Using these kernels we have full control on modelling row hammer faults (e.g. by producing hot rows) on a specific number of rows with specific access patterns. In each kernel, we targeted a specific number of rows (e.g. up to 20 rows) and repeatedly opened them during execution time. Also, we used three different random distributions, Uniform, Gaussian and Poisson, to access to these targeted (hot) rows. The Uniform distribution would represent a malignant program trying to damage other programs.

7. Experimental Results and Discussions

First we evaluated ARMoR using standard benchmarks to investigate its performance in average case scenarios.

Standard Benchmark Profiling: considering the nature of the row hammer, it has a strong correlation with the activation interval in DRAMs. Therefore, we profiled our benchmarks to investigate the activation patterns for different workloads. Fig. 8 presents the Average Activation Intervals (AAI) for individual banks in the system. The AAI distribution of *MT-Fluid* (Fig. 8r) shows that there is an intensive activation stream to Bank 3 (AAI =67:3) which suggests that, most likely, this workload is going to suffer row hammer. The X-axis of all the graphs in Fig. 8 shows the AAI in nanosecond (ns) and Y-axis presents the bank IDs.

Synthetic Kernels Profiling: In this section we profile our 36 synthetic kernels (12 kernels with Uniform distribution, 12 kernels with Gaussian distribution and 12 kernels with Poisson distribution) that were discussed in Section 6. Fig. 9 presents the number of unique row aggressors (some of them might be flagged as row aggressors several times within a RI) when increasing the number of targeted rows. This figure shows that for uniform access pattern as the number of targeted row increases the induced row aggressors also increases linearly. However, when reaching a certain point (in this example, 10 targeted rows and assuming ACT_{th} = 139K) having a uniform access pattern cause that none of the targeted rows reach the ACT_{th}. On the other hand, since Gaussian and Poisson distributions favour some targeted rows more than others then there are still induced row aggressors in the system even as we increase the number of targeted rows.

It is briefly mentioned that each row aggressors can reach ACT_{th} multiple times within one RI period and as results produce multiple row hammer problems. Fig. 10 presents the total row hammer occurrences during entire simulation time (i.e. two consecutive RI) using different random distributions.

Maximum ACT within RI: To investigate the vulnerability of our standard benchmarks to the row hammer phenomenon we monitor the maximum number of activations that happen within RI for all the workloads. Our experimental results show that only one workload (MT Fluid can reach up to 412k activations per RI) out of the 18 standard benchmarks suffers from row hammer phenomenon no matter which one of the three evaluated ACT_{th} in [19] is used. Fig. 11 plots the maximum ACT value for all other benchmarks.

Performance Analysis: Fig. 12 depicts the performance overhead of ARMoR and PARA (with probability values of 0.001 and 0.005 as proposed in [19]) for standard benchmarks. Since ARMoR only refreshes the victim rows that are adjacent to detected hot rows then the performance overhead is fairly negligible. Averaged across all 18 standard benchmarks ARMoR degrades the performance by $9.5x10^{-6}$%. This number is around 37,000x smaller than the performance overhead of PARA, that is 0.07% and 0.35% for probability values of 0.001 and 0.005 respectively.

To increase the randomness of memory access patterns we run multi-threads of standard workloads and modified the USIMM simulator to map all the workloads to

the same memory space. This increases the randomness of memory behaviour and it might also increase the vulnerability of system to row hammer phenomenon if multiple workloads try to access to different rows within the same bank. We selected these workloads in such a way to produce more row aggressors in the system when running together. Fig. 13 depicts the performance overhead of ARMoR and PARA while running on 8-thread workload mixes of our standard benchmarks. This time from the existing 18 workload mixes, 13 workloads mixes suffer from row hammer phenomenon.

Similarly to previous experiment on single thread benchmarks, for multi-thread applications ARMoR detects all the row aggressors and refreshes all the possible row victims with performance overhead of $3.35x10^{-6}$ %. The performance overhead imposed by ARMoR has a linear relation with number of row aggressors (and, as a result, row victims) in the memory systems. Thus, since the number of row aggressors has increased for multi-thread applications then the performance overhead induced by ARMoR also has increased. On the other hand, since PARA is a stateless technique (i.e. it does not record any previous state of the memory to do its predictions, so each time it predicts row victims by some independent probabilistic chance), its performance overhead is independent of the number of row aggressors in the system. Our experimental results show a 0.018% and 0.090% performance overhead for PARA with 0.001 and 0.005 probability value respectively. Again, ARMoR has 2,700x less performance overhead than PARA. Although PARA delivers an acceptable performance overhead, due to its probabilistic nature it cannot guarantee to prevent row hammer with absolute certainty. On the other hand, ARMoR offers a more robust solution to prevent this phenomenon, with much less performance overhead; it also imposes small area overhead to the system. To investigate this more closely we have evaluated both ARMoR and PARA against our synthetic kernels as well. Fig. 14 shows the performance overhead of ARMoR when it detect all the possible row aggressors in the system and refresh both adjacent rows to this row aggressors. According to this figure ARMoR delivers a higher performance overhead for workloads with Poisson distribution when number of targeted row is equal to 2 and 5. The reason is that performance overhead of ARMoR depends on existing number of row hammer problems in the systems and according to Fig. 10, kernels with Poisson distribution induce more row hammer problems than other situation (i.e. 20 row hammer errors) for both when number of targeted rows are 2 and 5.

We ran similar experiments to evaluate PARA's performance for our synthetic kernels. We used the recommended probability value by [19] (i.e. P = 0.001 and P = 0.005) and increase this value until PARA detect (refresh by probability) all the possible row aggressors in the system. Fig. 15 presents the performance overhead of PARA for different probability value. For P=0.2 PARA has a miss-rate of 3.2%, 0.6% and 0.6% for synthetic kernels with Uniform, Gaussian and Poisson distributions respectively. Fig. 16 depicts the PARA's miss-rate for different probability values for synthetic kernels.

Hot-Page Table Size vs Accuracy: In Section 4 it was discussed that, theoretically, the maximum number of hot row table entries should be equal to the maximum possible row aggressors that can be exist within RI to guarantee that ARMoR detects all the row aggressors. However, in practice, such a situation (i.e. maximum possible

row aggressors occurring within RI) happening has a very low probability. To investigate this for standard benchmarks we chose MT-Fluid, since this is the only workload that suffers from row hammer, and ran experiments with different hot row table sizes. The presented results in Fig. 17 results show that, although the maximum required entries to cover a memory organisation with 8 banks (and assuming the ACT_{th} = 139K) is 80, with only 3 entries ARMoR delivers 100% detection of row aggressors.

Also, we ran similar experiments for our 36 synthetic kernels and the results show that overall 81% of kernels require less than 5 entries, 14% of kernels require 5 to 7 entries and 5% of kernels require 8 to 9 entries of hot row table per bank for ARMoR to detect all the possible row hammer errors in the system (Fig. 18).

Scalability: To investigate the scalability of ARMoR we profiled the maximum imposed storage overhead of our proposal against various memory organisation populated with different memory sizes. We used the Micron's standard DDR3 memory modules with available capacities in the market [24, 25]. Fig. 19 presents the maximum required storage overhead for ARMoR to detect the maximum row aggressors, considering the evaluated ACT_{th} of 139K, 155K and 284K [19], for wide range of memory capacities. However, as it discussed, this is the maximum theoretical requirement and according to our experiment in the last Section, 5 entries per bank should be enough to cover possible row hammer errors in the system. The corresponding storage overhead in this case can be modelled as presented results for Module A in Fig. 19.

ARMoR Buffer's Performance: To investigate the possible performance improvement that ARMoR can offer using a simple buffer structure, we considered two different scenarios in a high level of abstraction. In the first scenario, we assume that as soon as a row is flagged as a hot row, the entire row will be moved to an embedded buffer in the memory controller. Although, there will be a performance overhead to move the entire row to the memory controller, this scenario can still present an insight about the upper-bound improvement that can be achieved using this buffer. Fig. 20 presents the achieved improvement in execution time for all the synthetic kernels.

In the second scenario, we assumed that as soon as a row is flagged as a hot row, then only the cache-lines located to this hot row will be moved to an embedded buffer in the memory controller after they have been accessed. Fig. 21 depicts the gained improvement in execution time for our synthetic kernels.

In both scenarios, the performance improvement that can be achieved has a strong correlation with number of memory requests that can be serviced using the ARMoR's Buffer. The evaluated kernels are simulated only for two consecutive refresh intervals which means that only few hot rows that are flagged in the first refresh interval are effective for improving the execution time.

Our experimental results for MT-Fluid from standard benchmarks show 66% and 61% improvement in execution time in the first and second scenarios respectively. This significant improvement is due to that MT-Fluid has only three row aggressors that are repeatedly induce 24 row hammer errors during 5 consecutive refresh intervals. Therefore, by detecting these hot rows in the first RI, future memory

request in the remaining four RIs will take advantage of buffered data in the memory controller.

8. Related Work

In general, the DRAM fault and reliability problem has been studied extensible [8, 12, 18, 34]. However, more specifically, the row hammer problem has not been widely understood and covered by literature except by two recent works.

Initially, the Row hammer phenomenon was highlighted by a test equipment company called Teledyne LeCroy [26] at MemCon 2013 [21] specifically in the context of DDR4 DRAMs. They designed and developed a monitoring platform that could profile the internal operation of DRAMs (e.g. such as the number of activations per RI) at run time. This equipment can investigate the existing of row hammer issue base on programmable parameters (e.g. ACT_{th}).

To our knowledge, the most comprehensive work that investigated the existence of row hammer issue in commodity DRAM devices is the recent paper by Kim et al. [19]. In this work, 129 DRAM modules from three different major DRAM manufacturers have been intensively evaluated for the row hammer phenomenon using eight FPGA platforms. They showed that 110 out of 129 DRAM modules suffer from disturbance error (i.e. "row hammer"). Several possible solutions have been investigated in this work and they proposed PARA (Probabilistic Adjacent Row Activation) to overcome row hammer. The key idea behind PARA is that every time a row is opened and closed one of its adjacent rows is refreshed (e.g. opened) with some low probability. Thus, if a particular row is opened and closed repeatedly then, statistically, the adjacent rows will be refreshed.

The main advantages of ARMoR over PARA are as follows:

- First of all, PARA cannot guarantee to detect all the possible row victims since it works based on probabilistic estimation. Instead it reduces the probability of row hammer significantly. However, since ARMoR can detect the exact row aggressor the final result is much more accurate.

- Although, PARA has a very low overhead it still imposes some extra penalty by unnecessarily refreshing rows. Again, since ARMoR predicts the exact row aggressor only the possible row victims will be refreshed.

- Finally, having knowledge of exact row aggressors' IDs offered by ARMoR provides an opportunity to improve the DRAM's performance as well as improve the reliability by overcoming the row hammer phenomenon.

9. Final Remarks

The examples described above as ARMoR are capable of preventing data corruption in DRAMs due to the "row hammer" effect. Uniquely ARMoR is able to guarantee that the row hammer effect cannot modify the contents of DRAM with modest hardware requirements. Given the known datasheet parameters of DDR3 and DD4 together with the minimum number of activations within a refresh interval required to trigger a row hammer effect, it has been shown how to derive mathematically the maximum

number of rows that could be affected. This number is in practice small; no more than 10 per bank. ARMoR detects hammered rows (rows with aggressive activation) at run time and either refreshes their adjacent rows or moves the hot rows to a local buffer in the memory controller to improve the performance of memory system.

We have evaluated ARMoR and compared it directly against PARA [19] over 18 standard workloads and 36 synthetic memory intensive kernels with three different random access distributions (i.e. Uniform, Gaussian and Poisson). Out of the 18 standard workloads, only one exhibits the row hammer pattern that would lead to data corruption. In this case, ARMoR has a smaller execution time overhead than PARA. Using the synthetics kernels, we have shown how PARA would miss row hummer episodes. Our experimental results showed that ARMoR does not affect the execution time of the workloads; cf. $9.5x10^{-6}$% and $3.5x10^{-3}$% for standard and synthetic kernels respectively while detects all the possible row hammer errors in the memory system. Moreover, we show that by using a small buffer in the memory controller, ARMoR provides an opportunity to improve the overall execution time of standard workloads and synthetic kernels by up to 66% and 7% respectively.

When used in this specification and claims, the terms "comprises" and "comprising", "including" and variations thereof mean that the specified features, steps or integers are included. The terms are not to be interpreted to exclude the possibility of other features, steps or integers being present.

The features disclosed in the foregoing description, or in the following claims, or in the accompanying drawings, expressed in their specific forms or in terms of a means for performing the disclosed function, or a method or process for obtaining the disclosed results, as appropriate, may, separately, or in any combination of such features, be utilised for realising the invention in diverse forms thereof.

While the invention has been described in conjunction with the exemplary embodiments described above, many equivalent modifications and variations will be apparent to those skilled in the art when given this disclosure. Accordingly, the exemplary embodiments of the invention set forth above are considered to be illustrative and not limiting. Various changes to the described embodiments may be made without departing from the spirit and scope of the invention.

For the avoidance of any doubt, any theoretical explanations provided herein are provided for the purposes of improving the understanding of a reader. The inventors do not wish to be bound by any of these theoretical explanations.

All references referred to above are hereby incorporated by reference.

References

[1] John Ousterhout, "Ramcloud." [Online]. Available: https://ramcloud.stanford.edu/wiki/display/ramcloud/RAMCloud

[2] Z. Al-Ars, "Dram fault analysis and test generation," 2005.

[3] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung, "Biobench: A benchmark suite of bioinformatics applications," in Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on. IEEE, 2005, pp. 2–9.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in Proceedings of the 17th international conference on Parallel architectures and compilation techniques. ACM, 2008, pp. 72–81.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.

[6] N. Chatterjee, R. Balasubramonian, and Z. Chishti, "Msc-sim: The memory scheduling championship simulator," 2012.

[7] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "Usimm: the Utah simulated memory module," University of Utah, Tech. Rep, 2012.

[8] Q. Chen, H. Mahmoodi, S. Bhunia, and K. Roy, "Modeling and testing of sram for new failure mechanisms due to process variations in nanoscale cmos," in VLSI Test Symposium, 2005. Proceedings. 23rd IEEE. IEEE, 2005, pp. 292–297.

[9] S. Cohen and Y. Matias, "Spectral bloom filters," in Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 2003, pp. 241–252.

[10] K. M. Dixit, "The spec benchmarks," Parallel computing, vol. 17, no. 10, pp. 1195–1209, 1991.

[11] J. A. Fifield and H. L. Kalter, "Crosstalk-shielded-bit-line dram," Apr. 23 1991, uS Patent 5,010,524.

[12] Z. Guo, A. Carlson, L.-T. Pang, K. T. Duong, T.-J. K. Liu, and B. Nikolic, "Large-scale sram variability characterization in 45 nm cmos," Solid-State Circuits, IEEE Journal of, vol. 44, no. 11, pp. 3174– 3192, 2009.

[13] K. Itoh, VLSI memory chip design. Springer New York, 2001, vol. 5.

[14] B. Jacob, S. Ng, and D. Wang, Memory systems: cache, DRAM, disk. Morgan Kaufmann, 2010.

[15] Y. H. Jung, H. C. Hillery, and T. A. Gary, "Flash and dram si scaling challenges, emerging non-volatile memory technology enablement," in In Flash Memory Summit, 2013.

[16] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," ACM Transactions on Database Systems (TODS), vol. 28, no. 1, pp. 51–55, 2003.

[17] B. Keeth, DRAM circuit design: fundamental and high-speed topics. Wiley. com, 2008, vol. 13.

[18] D. Kim, V. Chandra, R. Aitken, D. Blaauw, and D. Sylvester, "Variation-aware static and dynamic writability analysis for voltagescaled bit-interleaved 8-t srams," in Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design. IEEE Press, 2011, pp. 145–150.

[19] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on. IEEE, 2014, pp. 361–372.

[20] J. A. Mandelman, R. H. Dennard, G. B. Bronner, J. K. DeBrosse, R. Divakaruni, Y. Li, and C. J. Radens, "Challenges and future directions for the scaling of dynamic random-access memory (dram)," IBM Journal of Research and Development, vol. 46, no. 2.3, pp. 187–212, 2002.

[21] M. Micheletti, "Tuning ddr4 for power and performance," in In Mem- Con, 2013.

[22] Micron Technology Inc., "Ddr4 sdram." [Online]. Available: http://www.micron.com/products/dram/ddr4-sdram

[23] ——, "Hybrid memory cube." [Online]. Available: http://www.micron.com/products/hybrid-memory-cube

[24] ——, "Rdimm." [Online]. Available: http://www.micron.com/products/dram-modules/rdimm

[25] ——, "Rdimm: Registered memory modules." [Online]. Available: http://www.micron.com/-/media/documents/products/data%20sheet/modules/parity_rdimm/jszs72c2g_4gx72p.pdf

[26] Mike Micheletti, "Teledyne lecroy upgraded ddr protocol analyser adds ?row hammer? reporting and system memory mapping." [Online]. Available: http://cdn.teledynelecroy.com/files/pressreleases/09042013.pdf

[27] D.-S. Min, D.-I. Seo, J. You, S. Cho, D. Chin et al., "Wordline coupling noise reduction techniques for scaled drams," in Digest of Technical Papers., 1990 Symposium on VLSI Circuits, 1990, pp. 81–82.

[28] R. Morris, "Counting large numbers of events in small registers," Communications of the ACM, vol. 21, no. 10, pp. 840–842, 1978.

[29] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in ramcloud," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 29–41.

[30] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum et al., "The case for ramclouds: scalable high-performance storage entirely in dram," ACM SIGOPS Operating Systems Review, vol. 43, no. 4, pp. 92–105, 2010.

[31] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deepsubmicrometer cmos circuits," Proceedings of the IEEE, vol. 91, no. 2, pp. 305–327, 2003.

[32] K. Saino, S. Horiba, S. Uchiyama, Y. Takaishi, M. Takenaka, T. Uchida, Y. Takada, K. Koyama, H. Miyake, and C. Hu, "Impact of gate-induced drain leakage current on the tail distribution of dram data retention time," in Electron Devices Meeting, 2000. IEDM'00. Technical Digest. International. IEEE, 2000, pp. 837–840.

[33] SPECIFICATION, DDR3 SDRAM, "JEDEC STANDARD," 2009.

[34] V. Sridharan and D. Liberty, "A field study of dram errors," studies, vol. 3, no. 5, p. 10, 2012.

[35] A. Tanabe, T. Takeshima, H. Koike, Y. Aimoto, M. Takada, T. Ishijima, N. Kasai, H. Hada, K. Shibahara, T. Kunio et al., "A 30-ns 64-mb dram with built-in self-test and self-repair function," Solid-State Circuits, IEEE Journal of, vol. 27, no. 11, pp. 1525–1533, 1992.

[36] A. J. van de Goor and J. De Neef, "Industrial evaluation of dram tests," in Proceedings of the conference on Design, automation and test in Europe. ACM, 1999, p. 123.

[37] http://gigglehd.com/zbxe/newsreport/12225653

[38] http://www.passmark.com/press/index.htm

[39] http://teledynelecroy.com/pressreleases/document.aspx?news_id=1805&capid=107&mid=554

[40] http://www.ddrdetective.com/row hammer

[41] https://www.youtube.com/watch?v=7wIUQ04Vkes

[42] http://ddrdetective.com/files/6414/1036/5710/The_Known_Failure_Mechanism_in_DDR3_memory_referred_to_as_Row_Hammer.pdf

[43] http://electronicdesign.com/embedded/achieve-reliability-availability-and-serviceability-memoryinterfaces

[44] http://forums.xilinx.com/t5/Xcell-Daily-Blog/Unexplained-memory-errors-in-your-DDR3-design-Maybe-it-s-Row/ba-p/497600

[45] http://www.memcon.com/pdfs/proceedings2014/NET102.pdf

[46] http://www.arm.com/products/system-ip/memory-controllers/corelink-dmc-520.php

[47] http://www.micron.com/products/datasheets/3d323c4d-6bc7-4193-908d-e99ad746aa4e?page=13

[48] http://www.samsunginvestorsforum2014.com/

[49] http://www.cs.utah.edu/events/thememoryforum/kang_slides.pdf

[50] http://www.cs.utah.edu/events/thememoryforum/kang.pdf

[51] http://users.ece.gatech.edu/~pnair6/rowhammer/rowhammer.pdf

[52] http://www-947.ibm.com/support/entry/portal/docdisplay?Indocid=migr-5092492 ("UEFI enhancements to reduce DDR3 Row Hammer issue exposure")

[53] http://h20564.www2.hp.com/hpsc/doc/public/display?docId=emr_na-c04274536 ("Added support for Pseudo Target Row Refresh (pTRR) DIMM functionality")

[54] Kyungbae Park, "Active-Precharge Hammering on a Row Induced Failure in DDR3 SDRAMs under 3x nm Technology", presented to the 2014 IEEE International Integrated Reliability Workshop on 16 October 2014

CLAIMS:

1. A monitoring device for monitoring the accessing of data items in a memory, the monitoring device including:

a first monitoring unit configured to:

monitor a stream of access requests/commands, wherein each access request/command is directed to a data item in the memory;

determine whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access; and

if it is determined that a data item meets the condition indicative of frequent access, notify a second monitoring unit of the data item; and

the second monitoring unit, the second monitoring unit being configured to:

monitor the accessing of data items notified to the second monitoring unit by the first monitoring unit.

2. A monitoring device according to claim 1, wherein the stream of access requests/commands is a stream of activation commands, wherein each activation command is for activating an address line in the memory.

3. A monitoring device according to claim 1 or 2, wherein the condition indicative of frequent access is met by a data item to which an access request/command is directed, if a portion of the stream of access requests/commands corresponding to a recent time window includes more than a predetermined number of access requests/commands directed to that data item.

4. A monitoring device according to any previous claim, wherein:

the first monitoring unit includes a register with N memory elements configured to store a portion of the stream of access requests/commands corresponding to a recent time window divided into N time intervals of predetermined length, where N is an integer, wherein each memory element in the register corresponds to a respective one of the N time intervals so that if the respective time interval includes an access request/command directed to a data item, then an identifier of that data item is stored as an entry in the memory element.

5. A monitoring device according to claim 4, wherein the register is a shift register.

6. A monitoring device according to any previous claim, wherein the second monitoring unit is configured to identify one or more frequently accessed data items from the data items notified to it by the first monitoring unit.

7. A monitoring device according to any previous claim, wherein the second monitoring unit is configured to:

for each data item notified to the second monitoring unit, count the number of times the stream of access requests/commands includes an access request/command directed to that data item, and identify the data item as a frequently accessed data item if the counted number of times the stream of access

requests/commands includes an access request/command directed to that data item reaches a predetermined threshold.

8. A monitoring device according to claim 6 or 7, wherein the second monitoring unit is configured to, if a data item is identified by the second monitoring unit as a frequently accessed data item, flag the data item as a frequently accessed data item for use by a memory buffer configured to temporarily store data from the memory.

9. A monitoring device according to any previous claim, wherein the memory buffer is configured to determine its content based on a data item flagged as a frequently accessed data item by:

storing the frequently accessed data item or a block of data including at least part of the frequently accessed data item in the memory buffer; and/or

retaining the frequently accessed data item or a block of data including at least part of the frequently accessed data item in the memory buffer.

10. A monitoring device according to claim 9, wherein the memory buffer is a CPU cache.

11. A monitoring device according to claim 9, wherein the memory buffer is a dedicated memory buffer configured to store data items identified by the second monitoring unit as frequently accessed data items.

12. A monitoring device according to any previous claim, wherein the second monitoring unit is configured to:

for each data item notified to the second monitoring unit, if the data item meets a condition indicative of non-frequent access, flag the data item as a candidate for eviction from the second monitoring unit or evict the data item from the second monitoring unit.

13. A monitoring device according to any previous claim, wherein second monitoring unit is configured to:

for each data item notified to the second monitoring unit, and for each of a plurality of periods of time of predetermined length: count in a respective credit counter corresponding to the data item the number of times the address line is accessed more than a predetermined number of times in the period of time of predetermined length.

14. A monitoring device according to any previous claim, wherein the monitoring device is or forms part of a memory controller for controlling the memory.

15. A monitoring device according to any previous claim, wherein the memory is a DRAM.

16. A monitoring device for monitoring the activation of address lines in a memory, the monitoring device including:

a first monitoring unit configured to:

monitor a stream of activation commands, wherein each activation command is directed to an address line in the memory;

determine whether any of the address lines to which activation

commands are directed meet a condition indicative of frequent activation; and

if it is determined that an address line meets the condition indicative of frequent activation, notify a second monitoring unit of the address line; and

the second monitoring unit, the second monitoring unit being configured to:

monitor the activation of address lines notified to the second monitoring unit by the first monitoring unit.

17. A monitoring device according to any previous claim, wherein the monitoring device is configured to:

if an address line is identified as a frequently activated address line by the second monitoring unit, initiate the refreshing of one or more address lines that are adjacent to the frequently activated address line.

18. A monitoring method performed by a monitoring device according to any previous claim, the method including:

at the first monitoring unit:

monitoring a stream of access requests/commands, wherein each access request/command is directed to a data item in the memory;

determining whether any of the data items to which access requests/commands are directed meet a condition indicative of frequent access; and

if it is determined that a data item meets the condition indicative of frequent access, notifying a second monitoring unit of the data item;

at the second monitoring unit:

monitoring the accessing of data items notified to the second monitoring unit by the first monitoring unit:

19. A monitoring device substantially as any one embodiment herein described with reference to and as shown in the accompanying drawings.

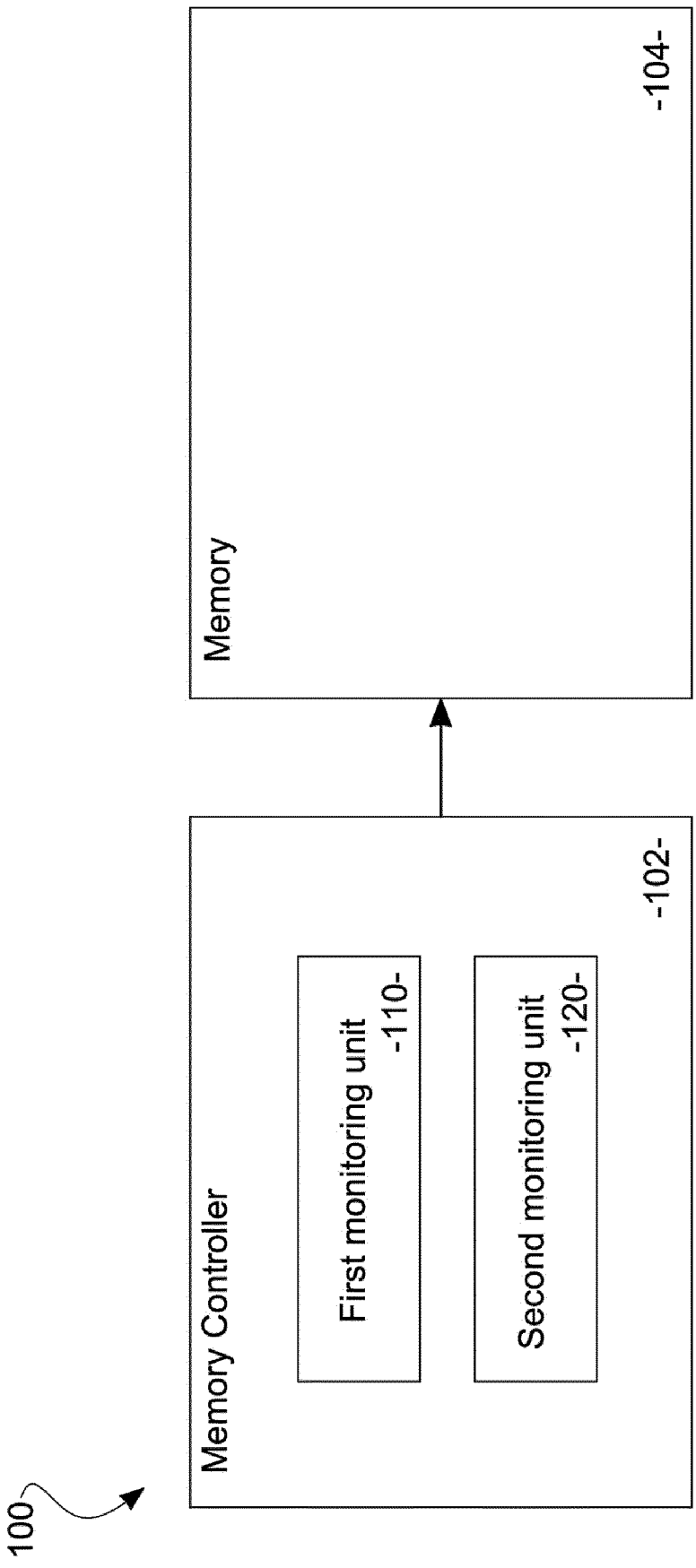
Memory
-104-
Memory Controller
-102-
First monitoring unit
-110-
Second monitoring unit
-120-
100

Figure 1

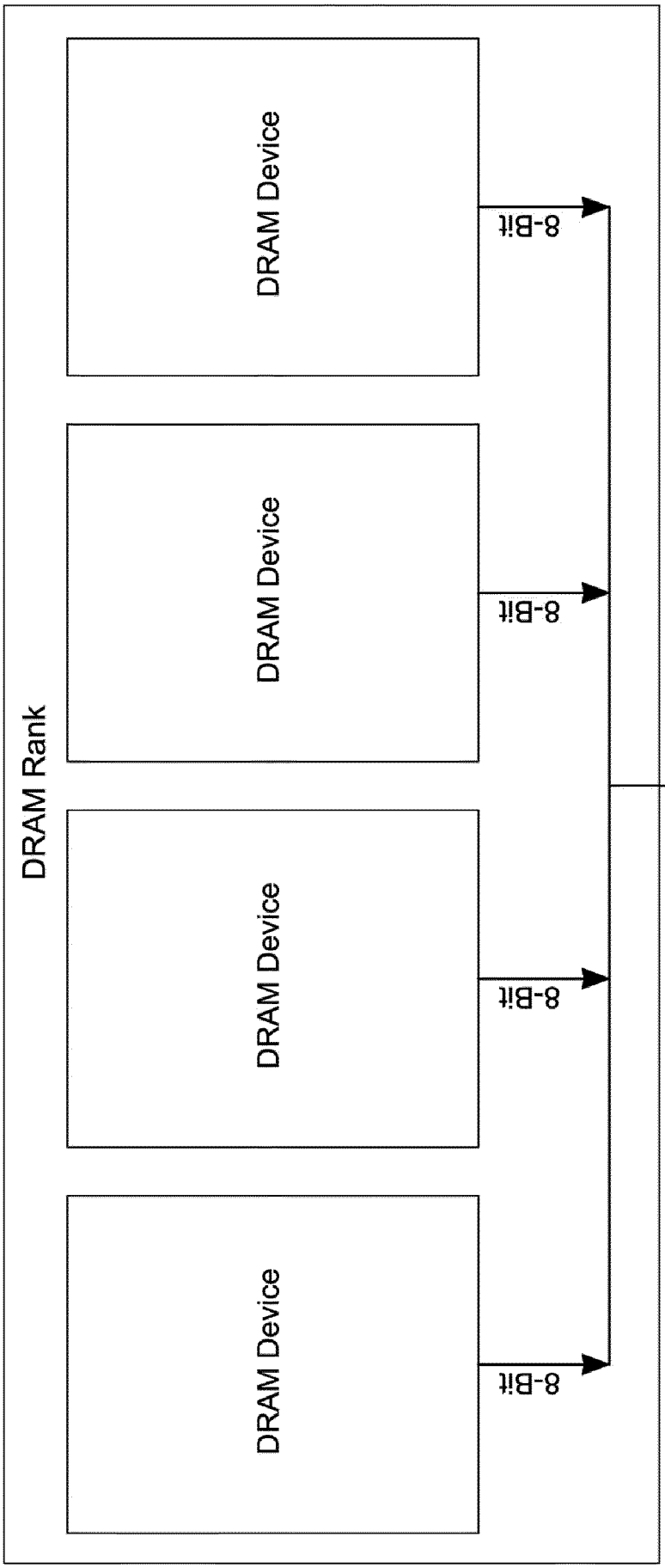
DRAM Rank
DRAM Device
DRAM Device
DRAM Device
DRAM Device
8-Bit
8-Bit
8-Bit
8-Bit
32-Bit
DRAM Rank

Figure 2(a)

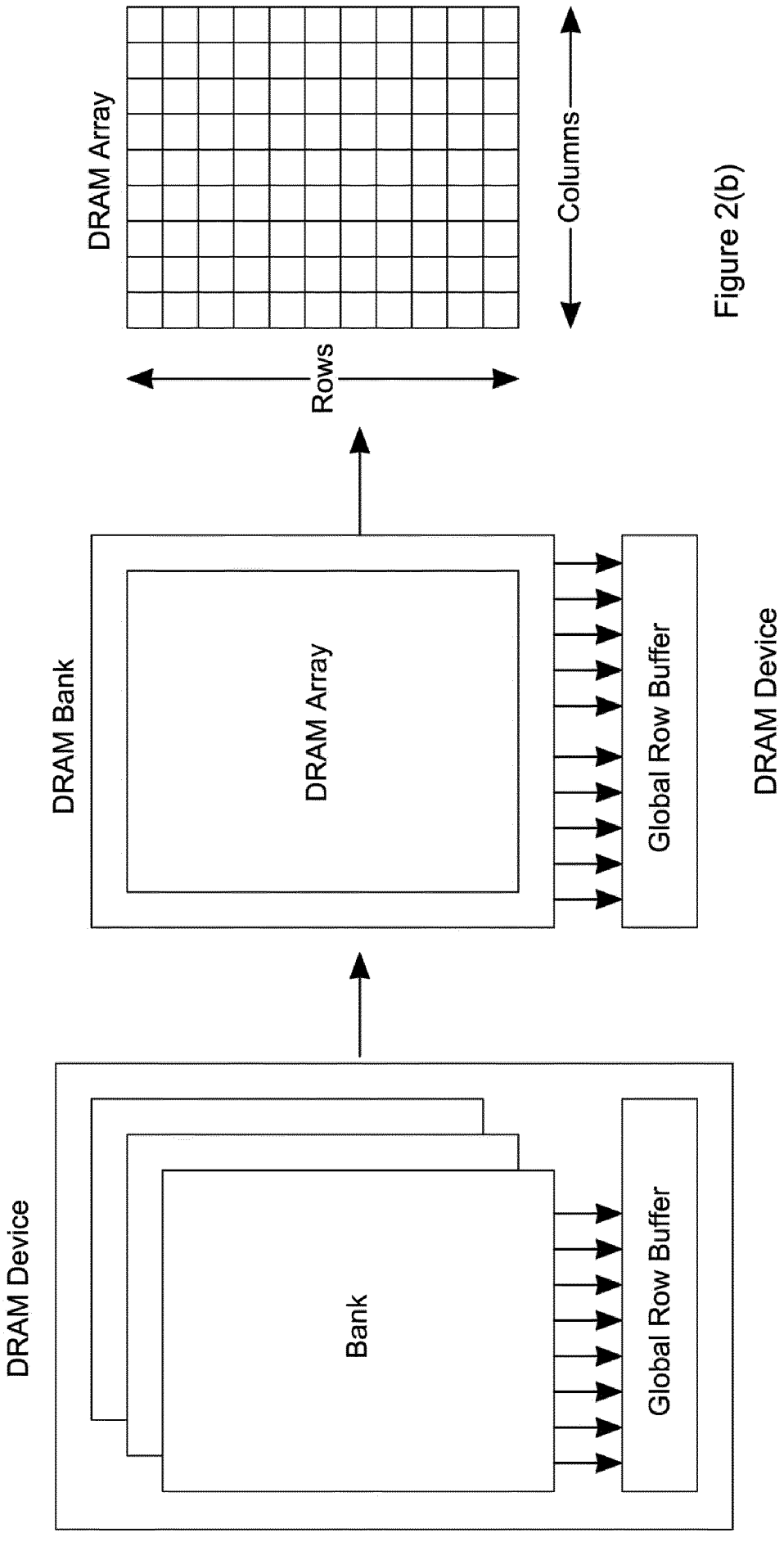
DRAM Array
Columns
Rows
DRAM Bank
DRAM Array
Global Row Buffer
DRAM Device
DRAM Device
Bank
Global Row Buffer

Figure 2(b)

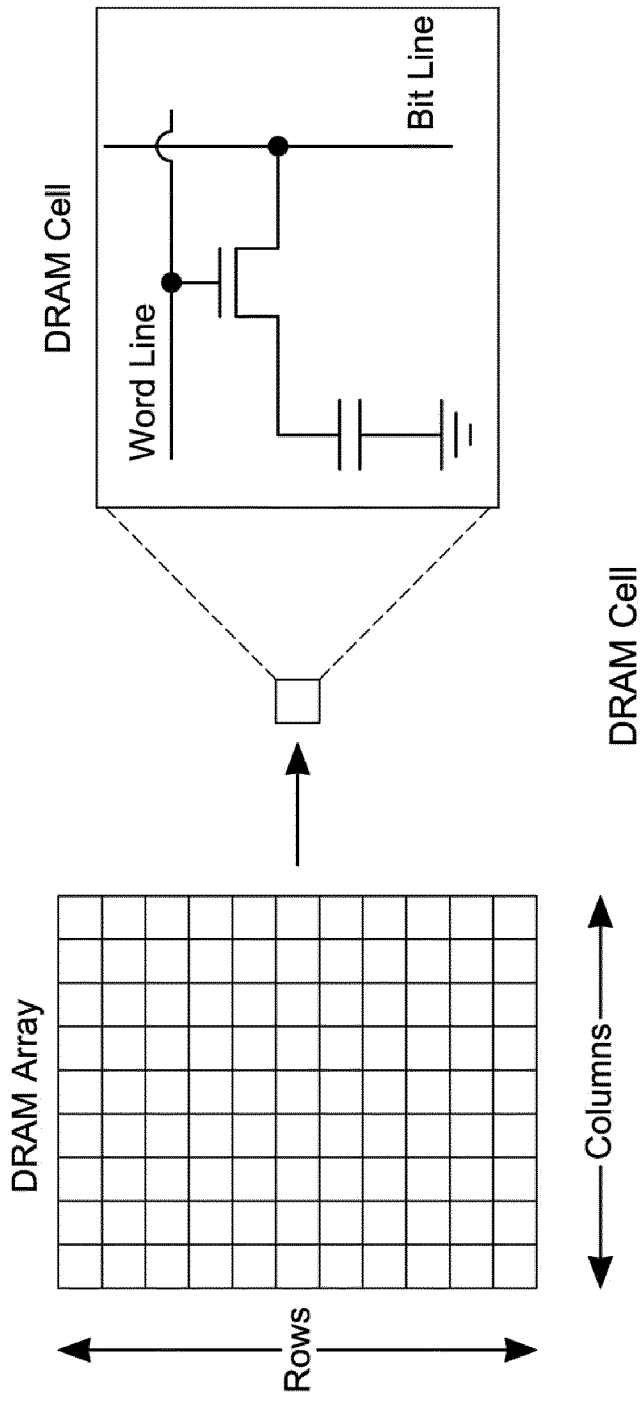
DRAM Cell
Bit Line
Word Line
DRAM Cell
DRAM Array
Columns
Rows

Figure 2(c)

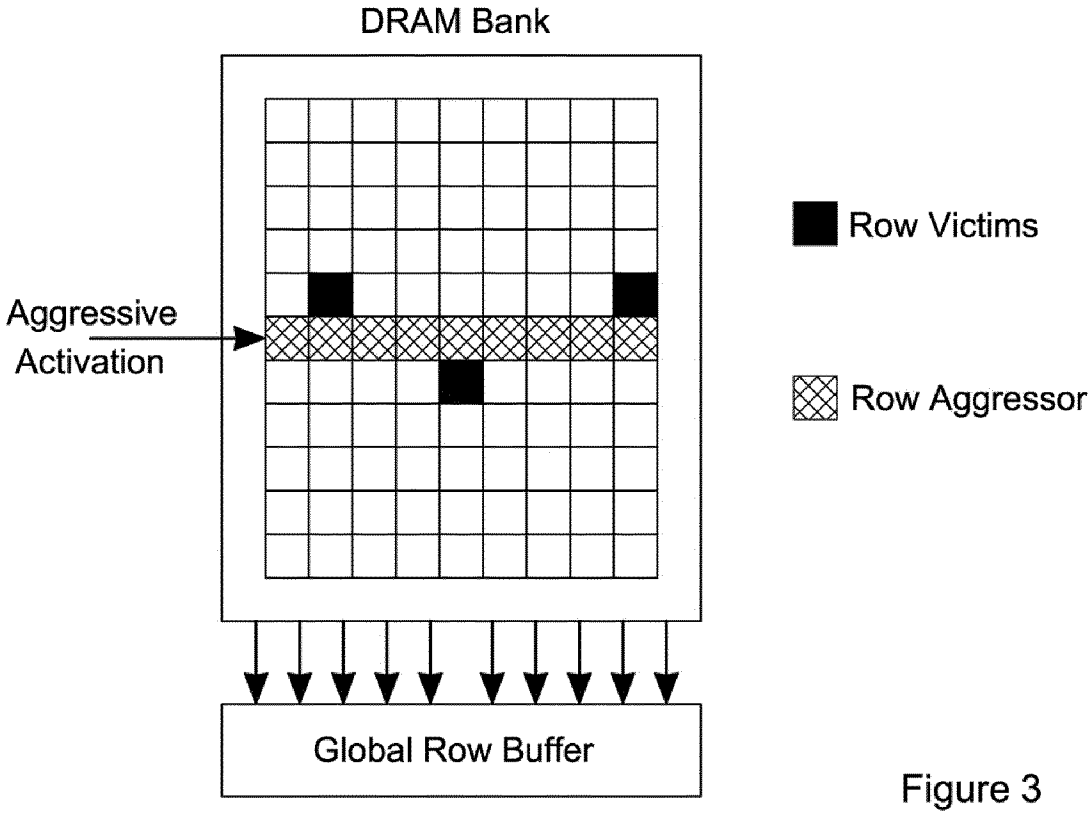

Figure 3

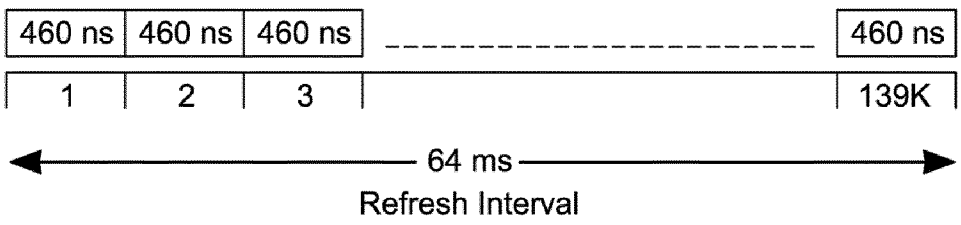

Figure 4

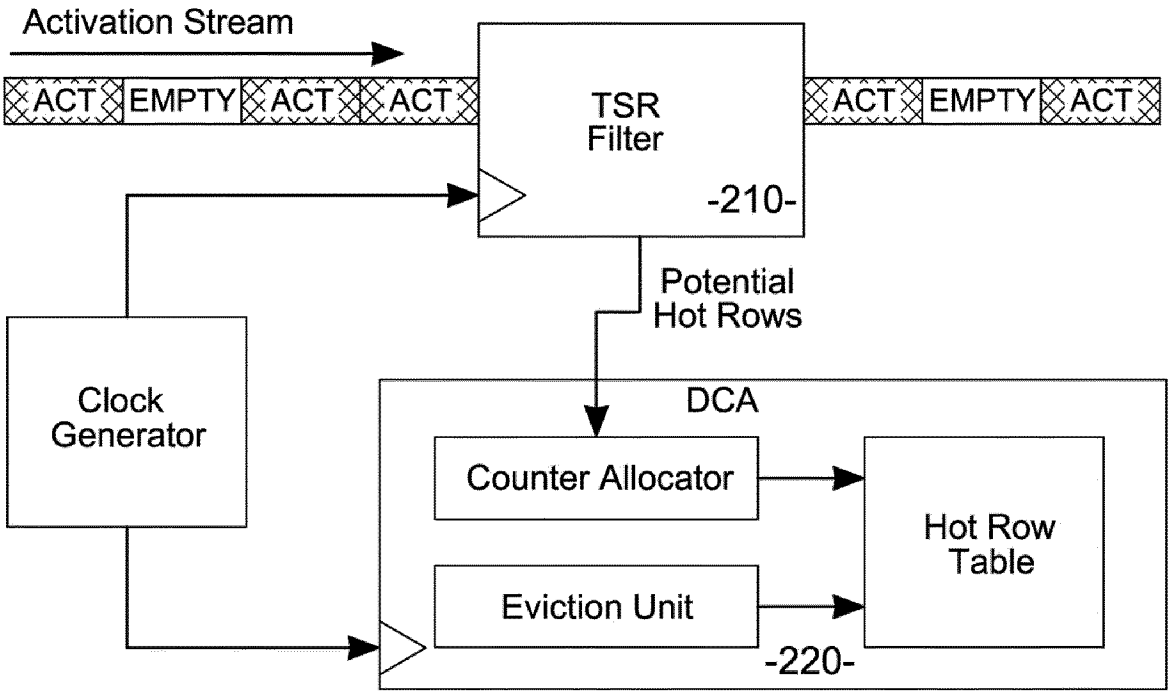

Figure 5

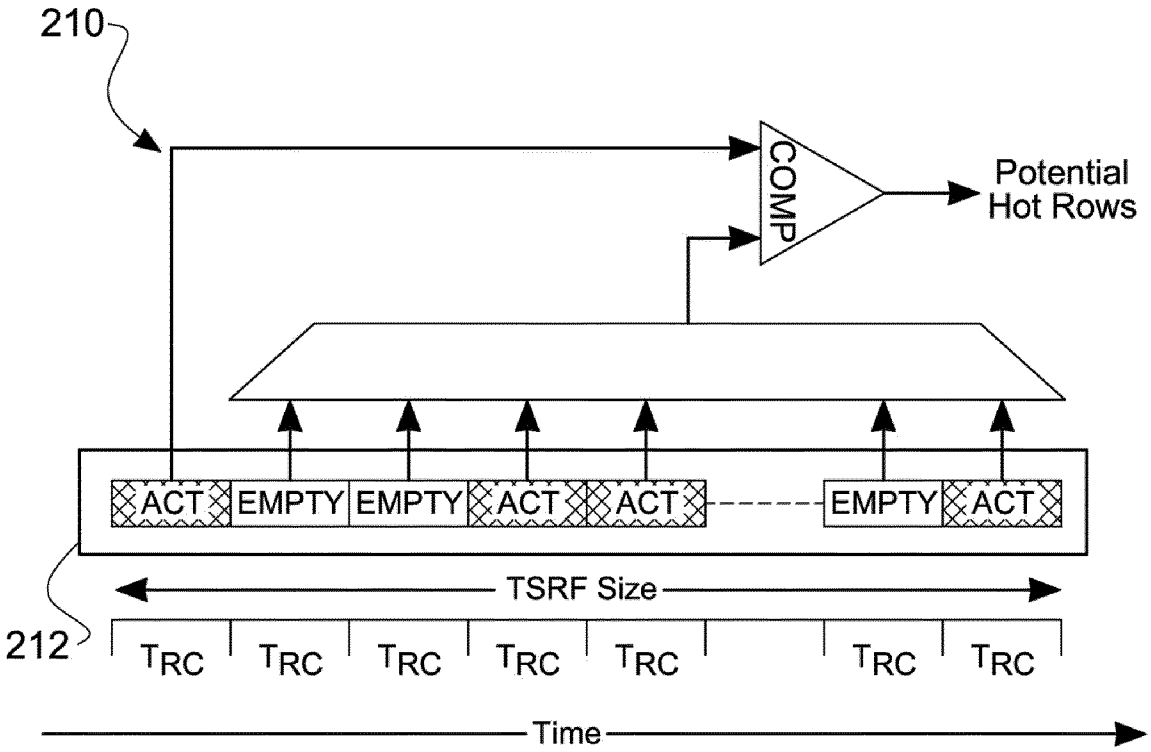

Figure 6

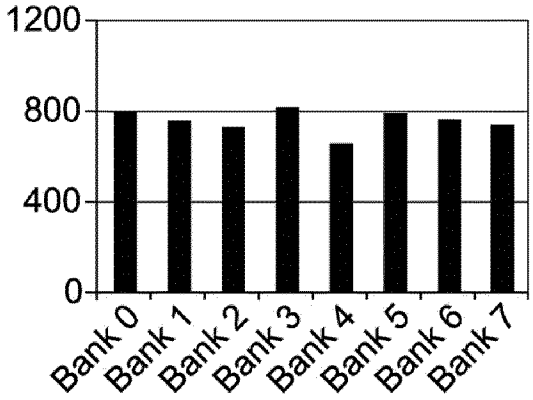

Figure 8(a) Black

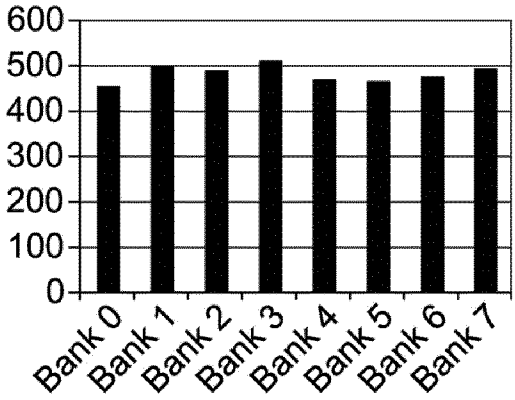

Figure 8(b) Comm1

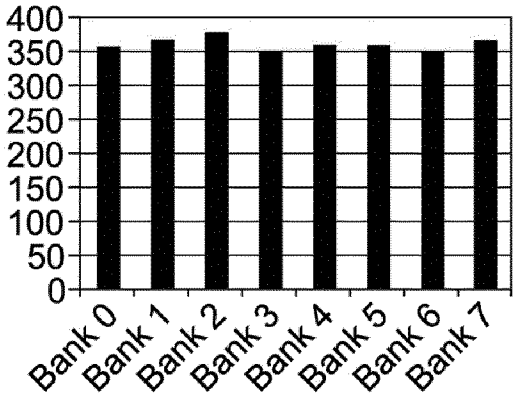

Figure 8(c) Comm2

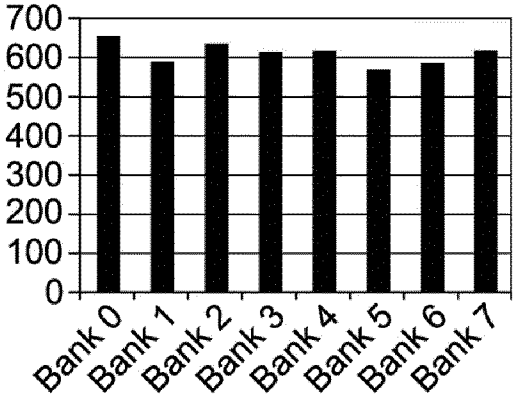

Figure 8(d) Comm3

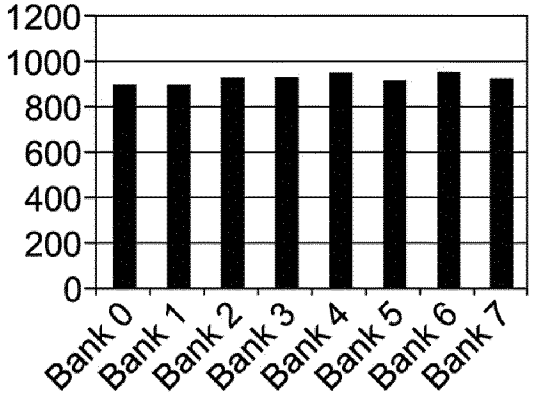

Figure 8(e) Comm4

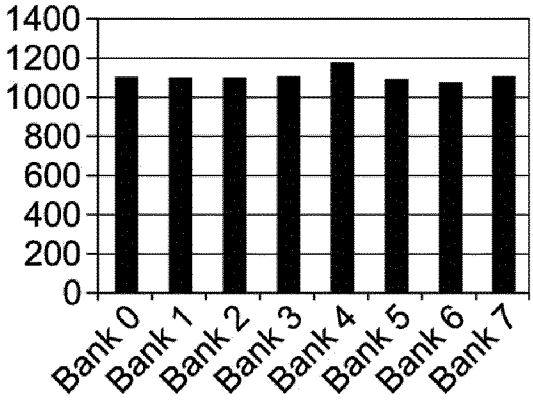

Figure 8(f) Comm5

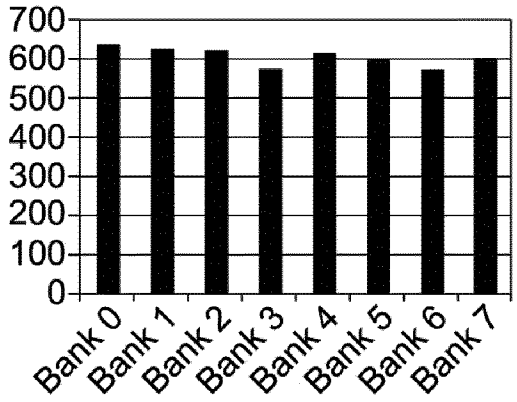

Figure 8(g) Face

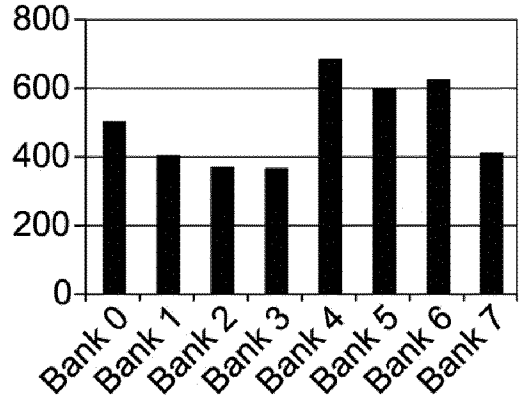

Figure 8(h) Ferret

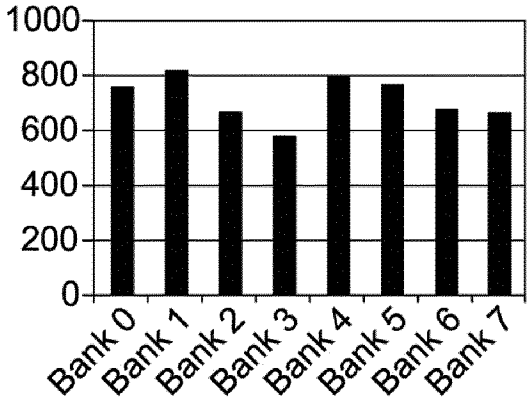

Figure 8(i) Fluid

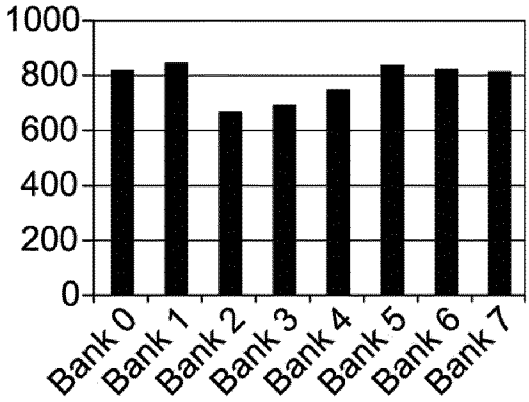

Figure 8(j) Freq

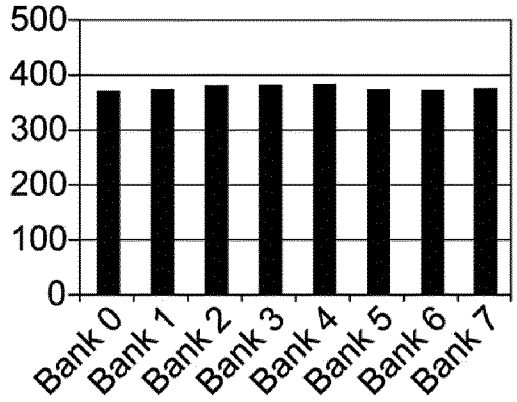

Figure 8(k) Leslie

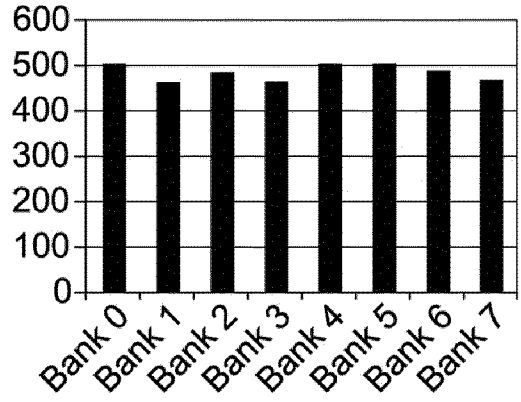

Figure 8(l) Libq

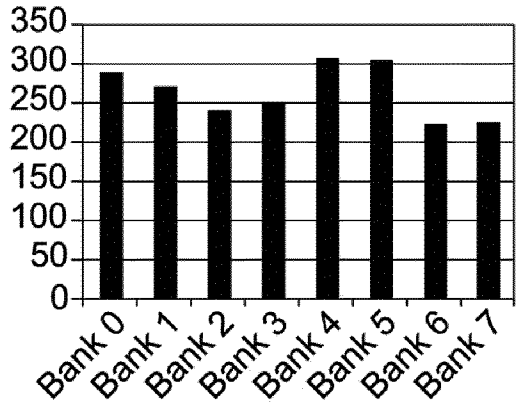

Figure 8(m) Mummer

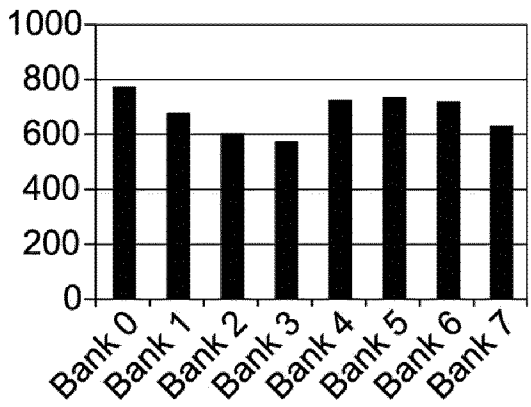

Figure 8(n) Stream

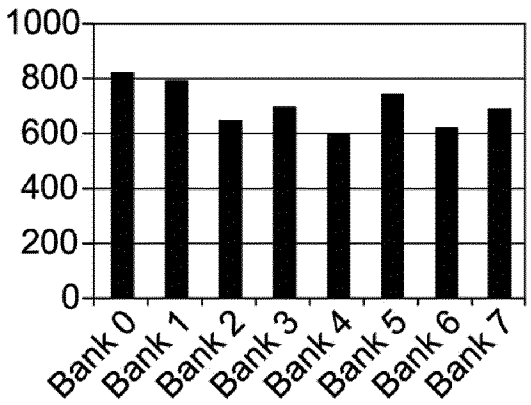

Figure 8(o) Swapt

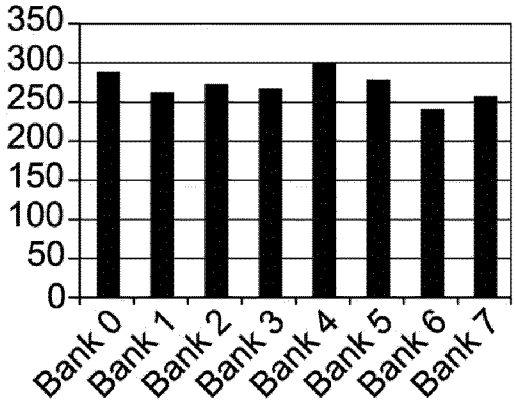

Figure 8(p) Tigr

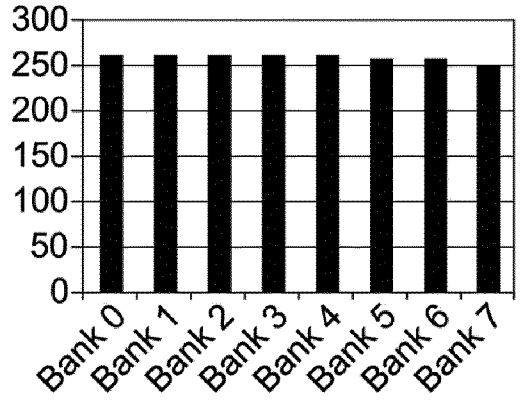

Figure 8(q) MT-Canneal

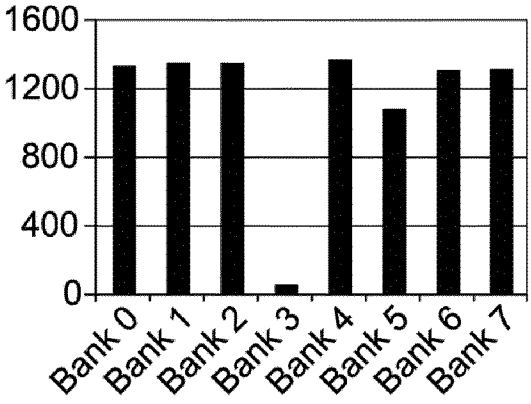

Figure 8(r) MT-Fluid

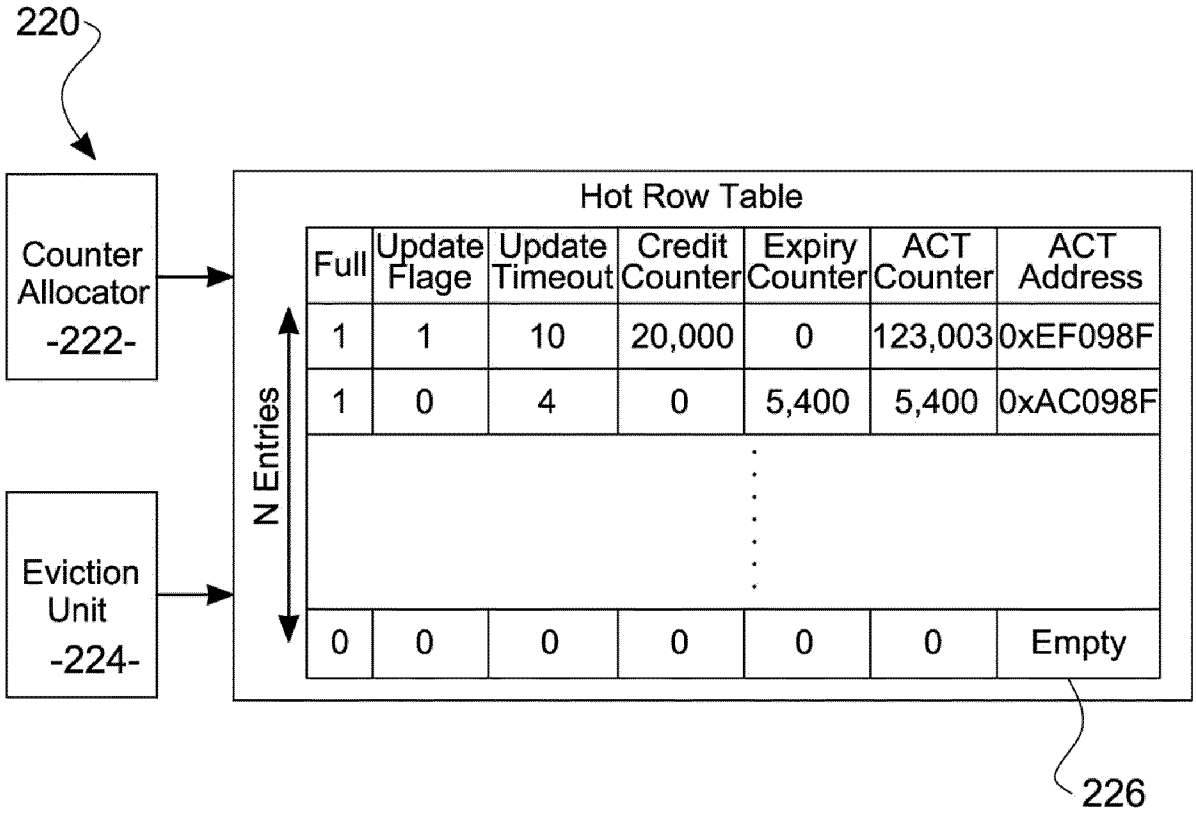

Figure 7

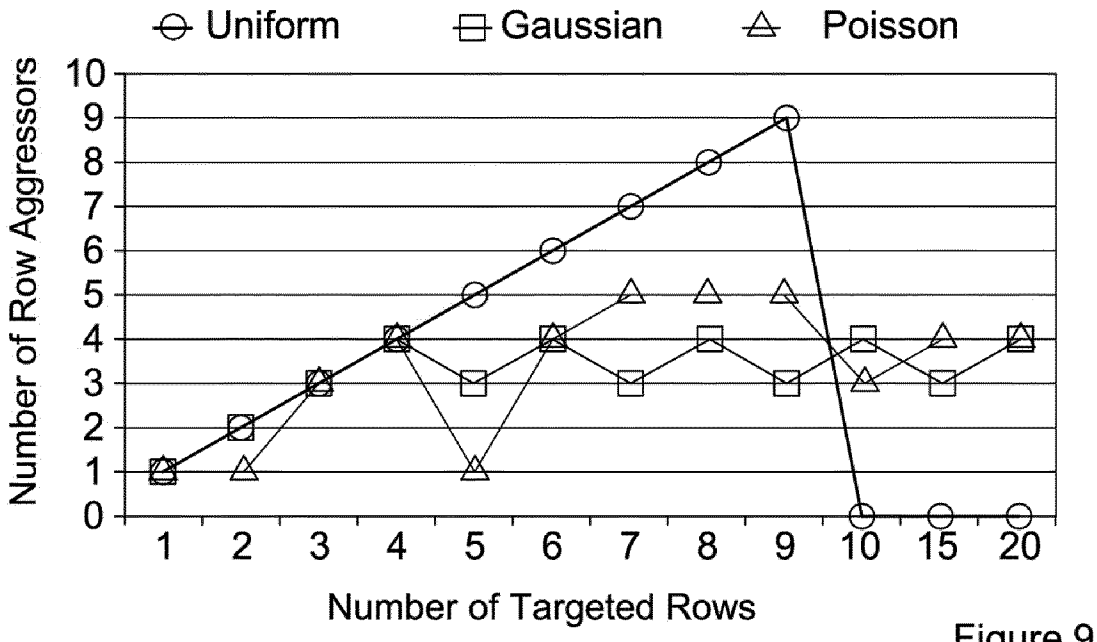

Figure 9

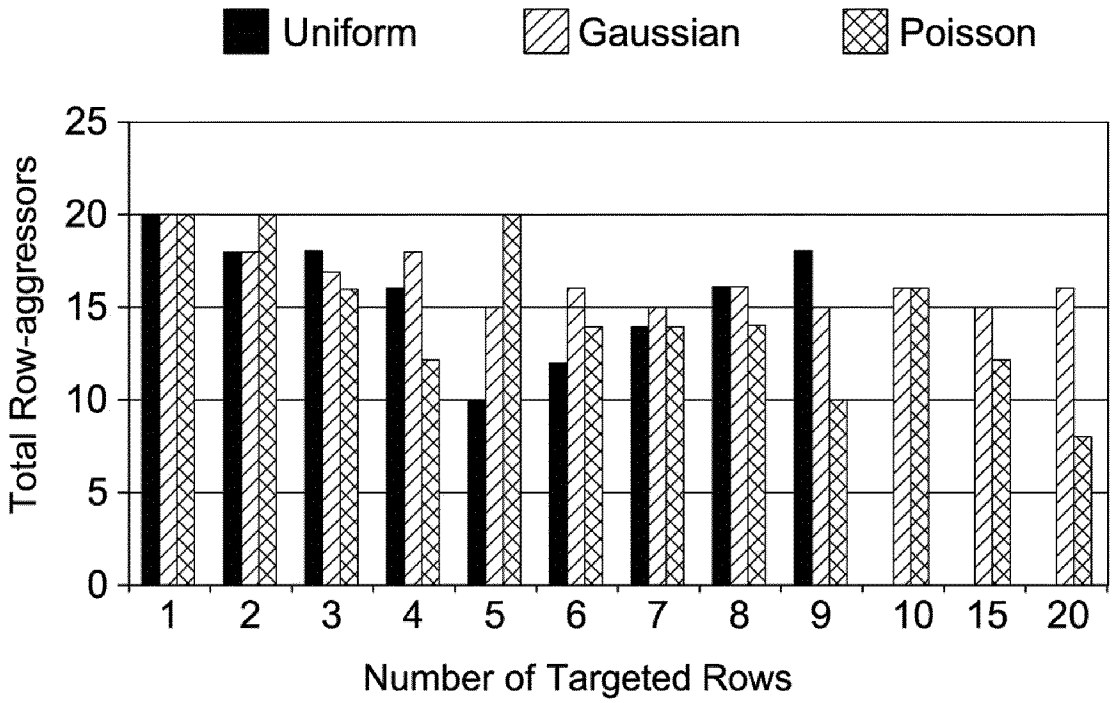

Figure 10

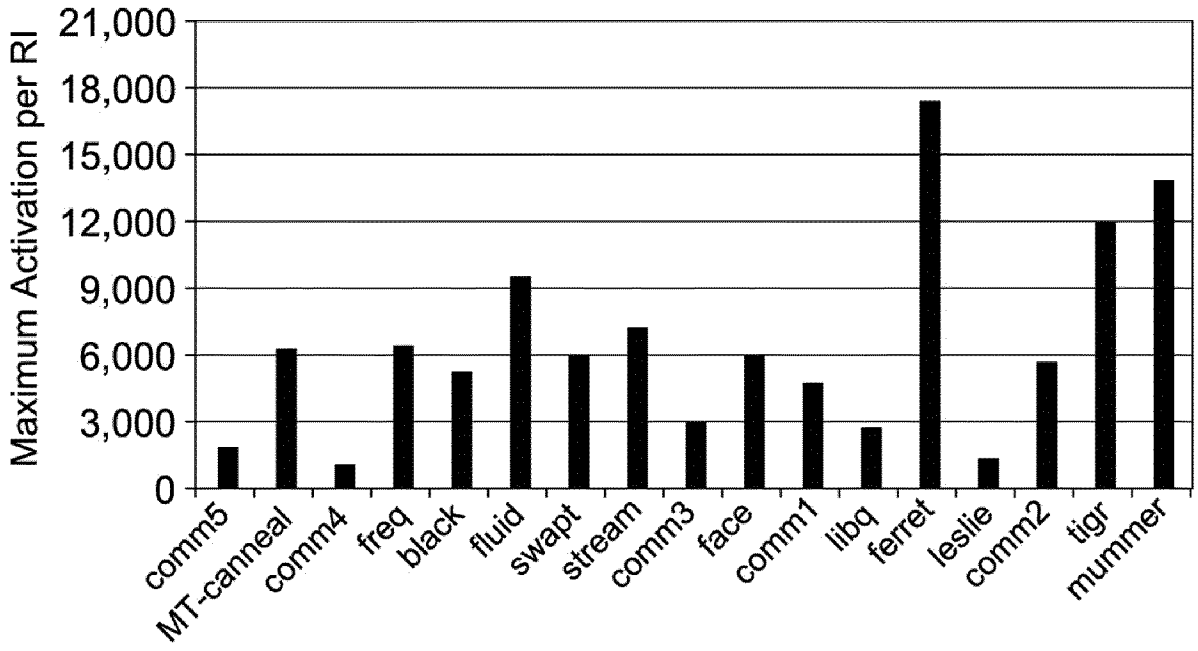

Figure 11

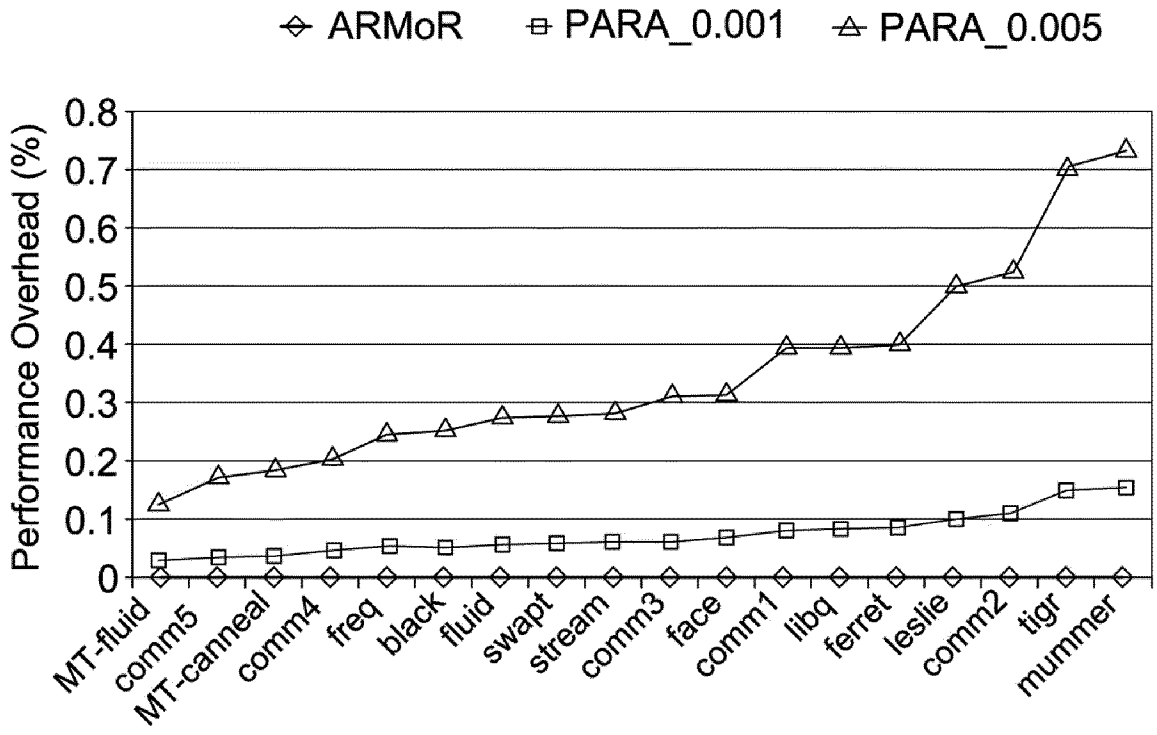

Figure 12

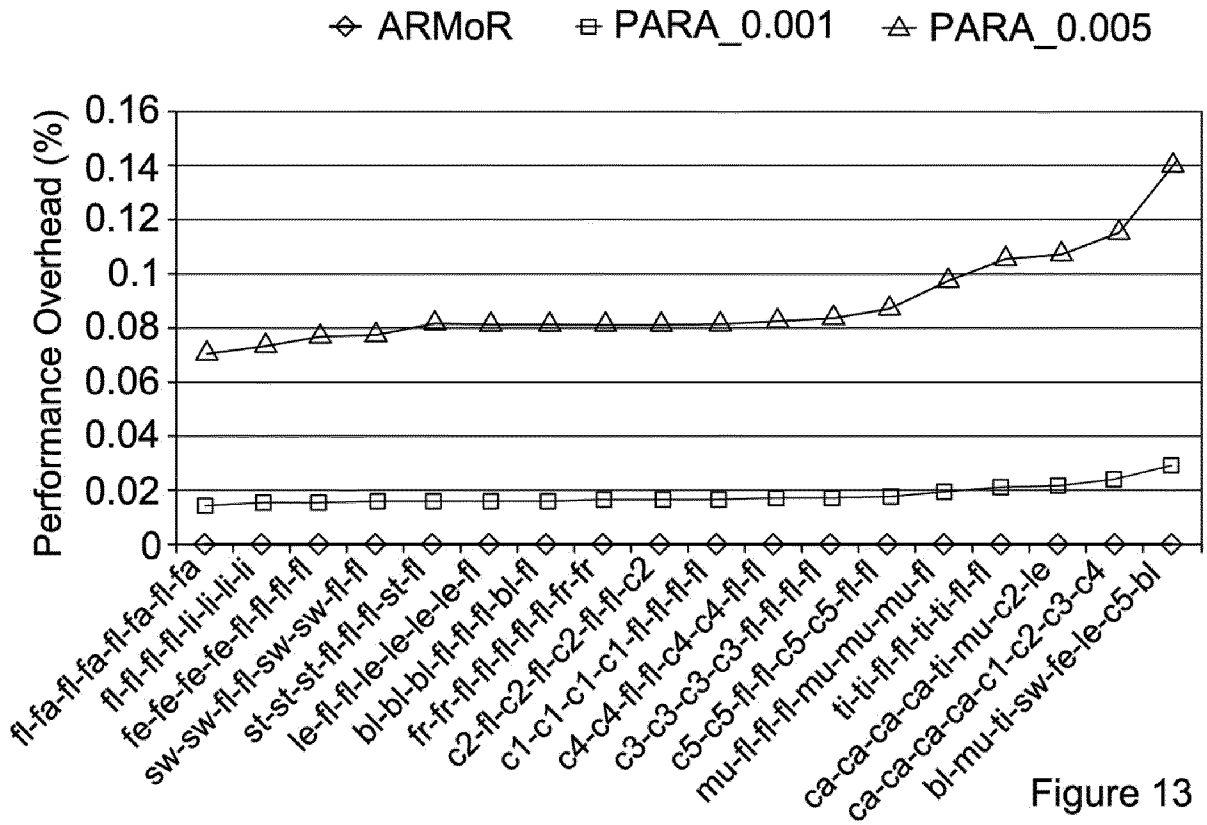

Figure 13

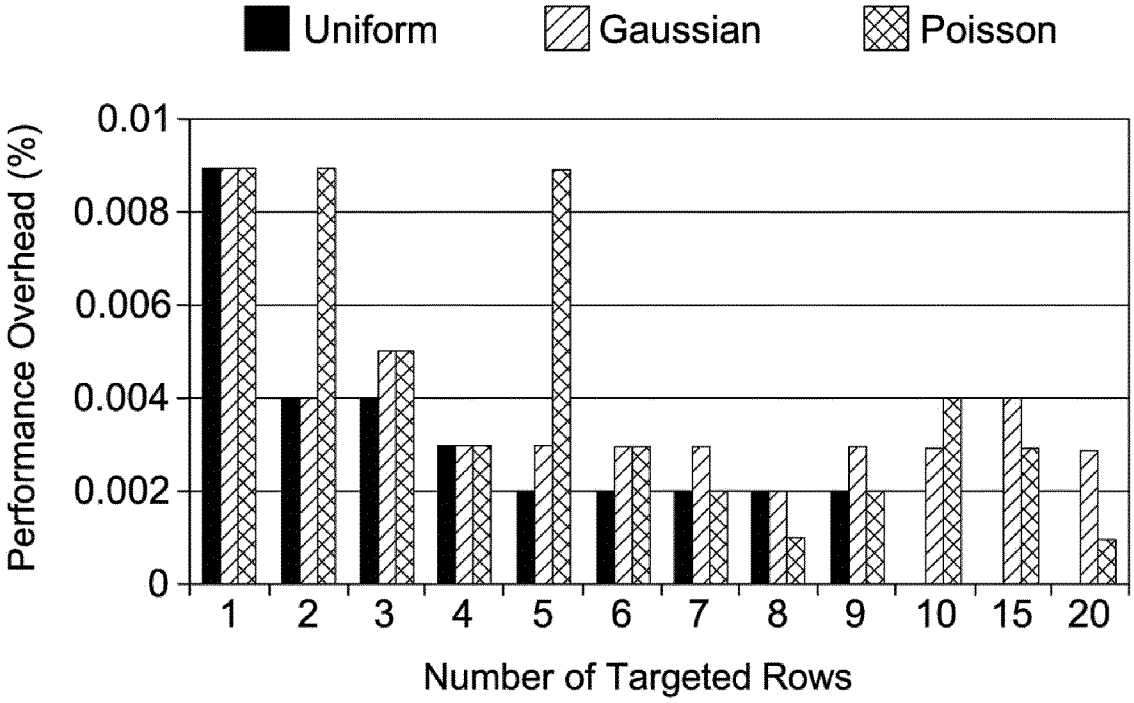

Figure 14

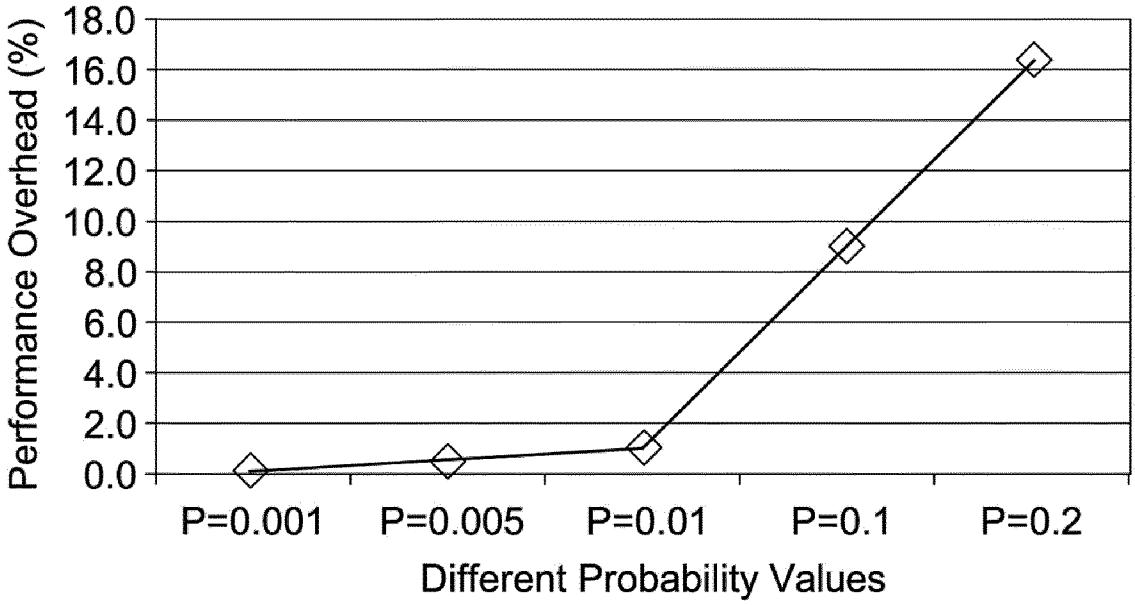

Figure 15

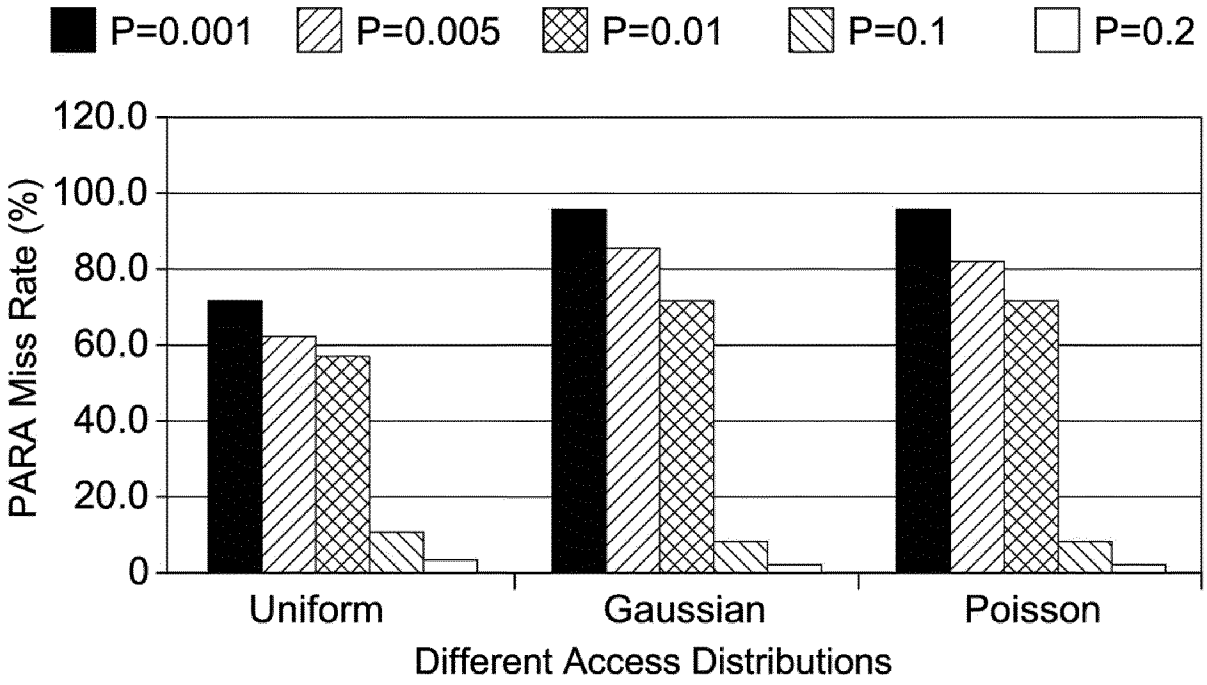

Figure 16

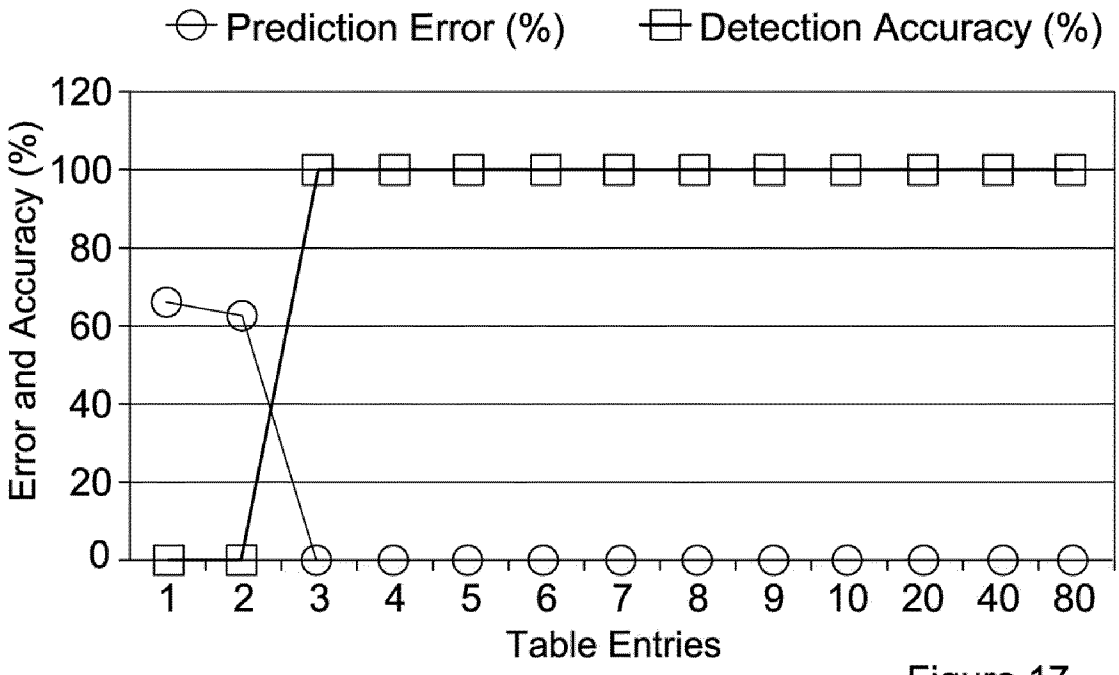

Figure 17

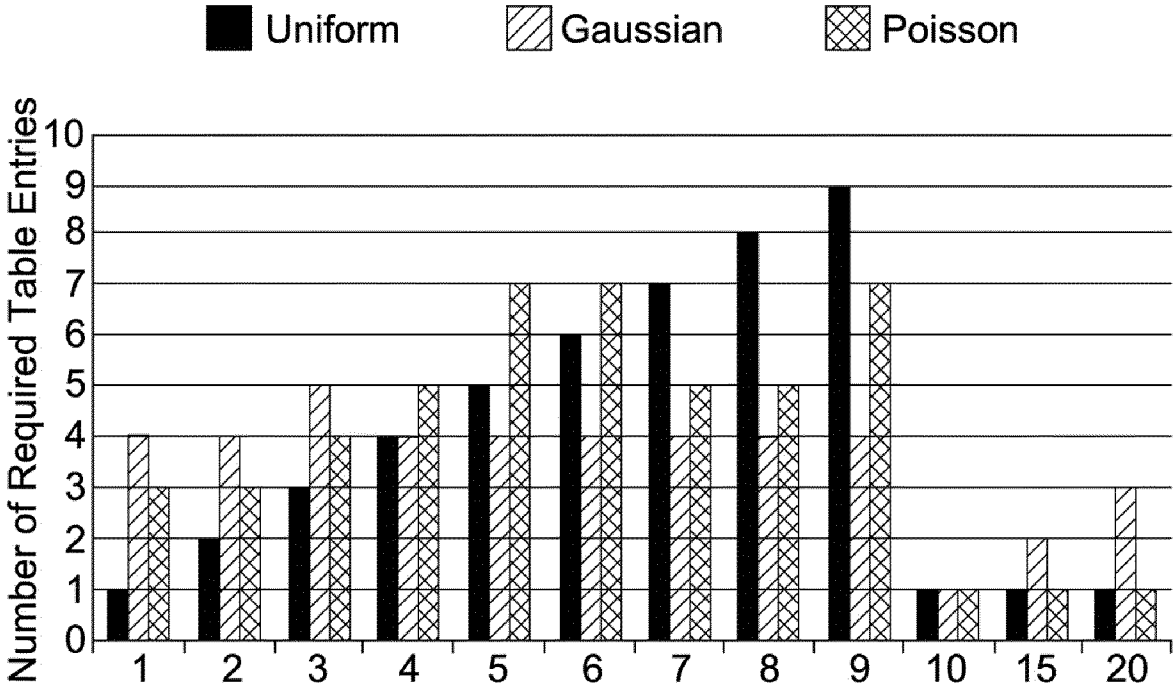

Figure 18

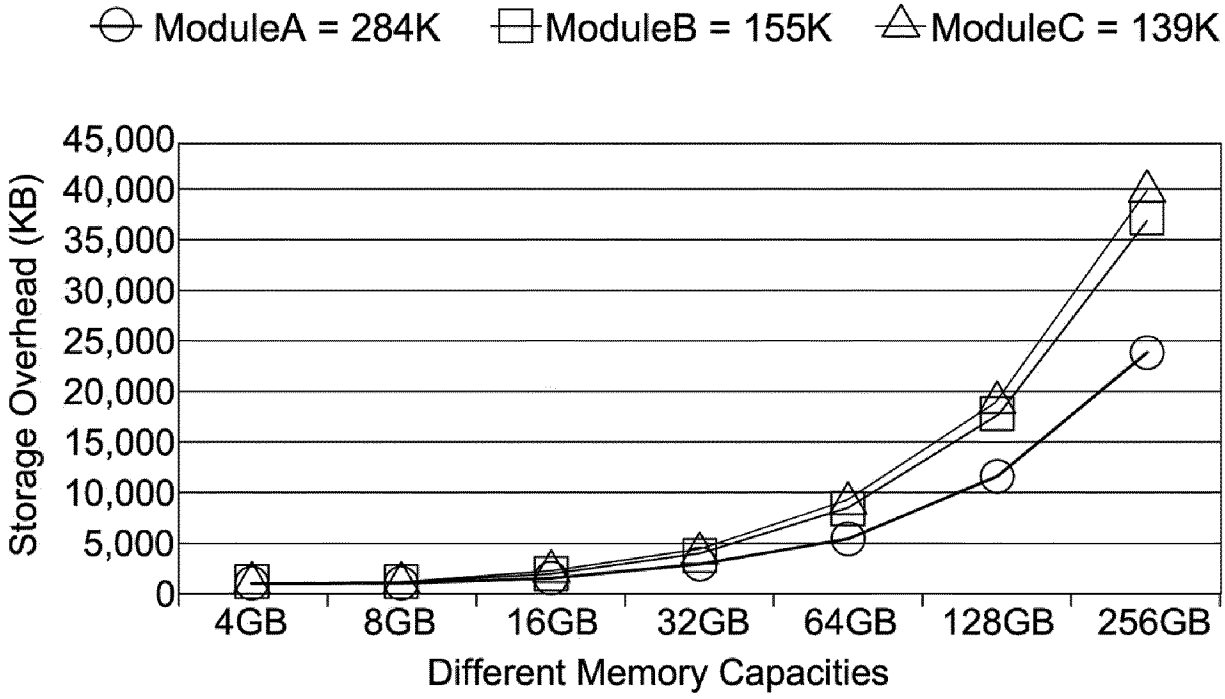

Figure 19

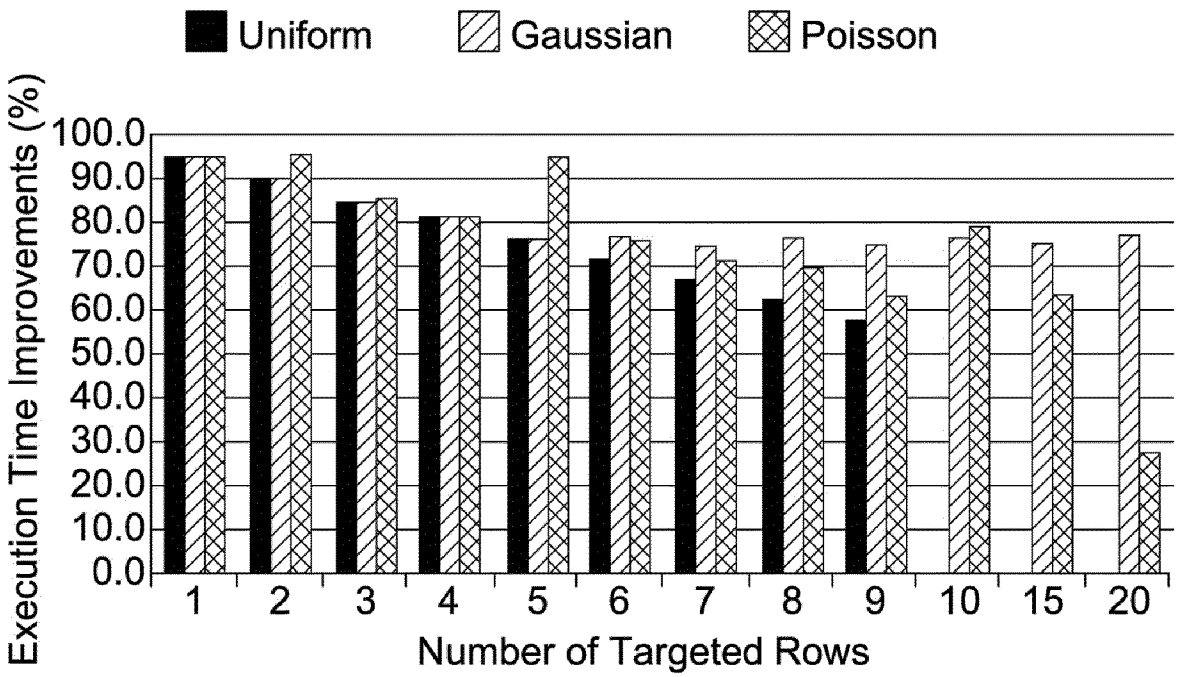

Figure 20

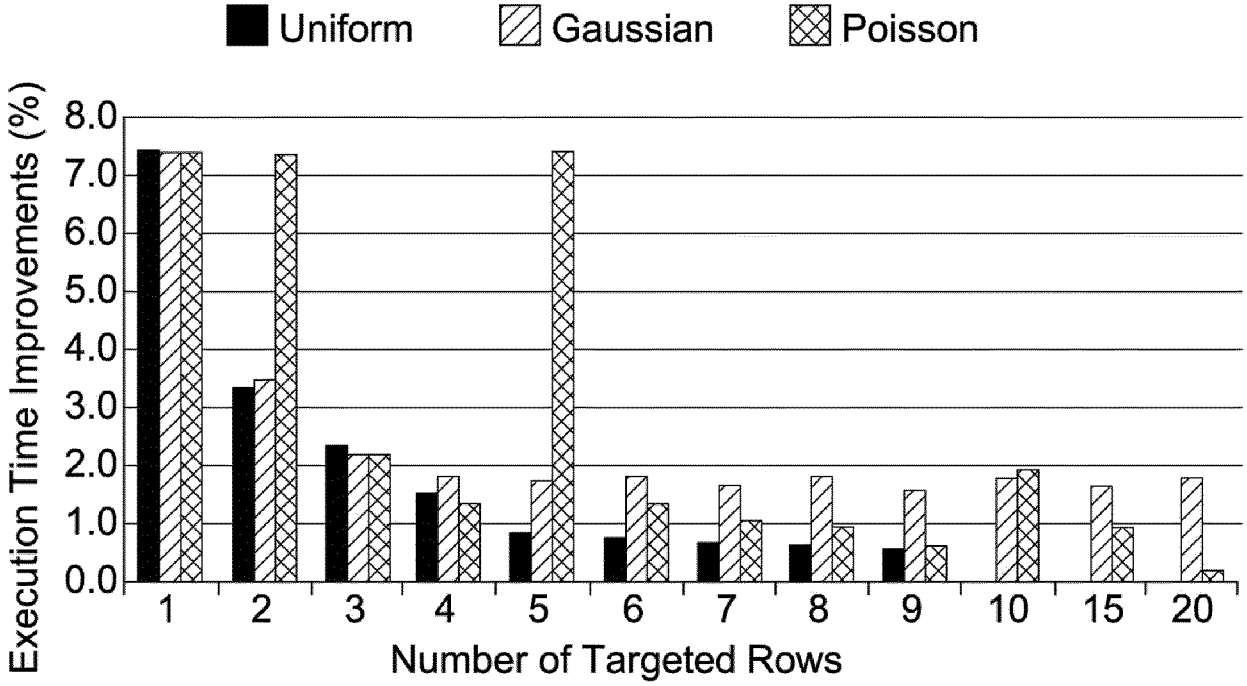

Figure 21

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2016/050385

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F11/30
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data, INSPEC, COMPENDEX, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2014/120215 A1 (HEWLETT PACKARD DEVELOPMENT CO [US]) 7 August 2014 (2014-08-07)	1-16,18, 19
Y	paragraphs [0012] - [0016], [0020] - [0030], [0033] - [0040] figures 2, 3, 5, 6	17

Y	US 2014/006703 A1 (BAINS KULJIT S [US] ET AL) 2 January 2014 (2014-01-02) paragraphs [0004] - [0006], [0018] - [0060], [0084] - [0093] figures 1, 2A, 2B, 3, 6, 7	17

X	US 2002/073405 A1 (CHILIMBI TRISHUL [US]) 13 June 2002 (2002-06-13)	1-16,18, 19
A	paragraphs [0003] - [0006], [0018] - [0038], [0043] - [0048], [0060] - [0097] figures 1-5, 8	17

	-/--	

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 April 2016	29/04/2016

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Johansson, Ulf

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2016/050385

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 2014/006704 A1 (GREENFIELD ZVIKA [US] ET AL) 2 January 2014 (2014-01-02) paragraphs [0004], [0013] - [0041], [0053] - [0057], [0076] - [0085] figures 1-5 -----	1-7,13, 14,16-19 8-12,15

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2016/050385

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2014120215 A1	07-08-2014	CN 105027211 A	04-11-2015
		KR 20150115752 A	14-10-2015
		TW 201439897 A	16-10-2014
		US 2015371689 A1	24-12-2015
		WO 2014120215 A1	07-08-2014
US 2014006703 A1	02-01-2014	CN 104350546 A	11-02-2015
		KR 20150002783 A	07-01-2015
		US 2014006703 A1	02-01-2014
		WO 2014004748 A1	03-01-2014
US 2002073405 A1	13-06-2002	US 2002073405 A1	13-06-2002
		US 2004255083 A1	16-12-2004
		US 2004261062 A1	23-12-2004
US 2014006704 A1	02-01-2014	DE 112013003312 T5	07-05-2015
		US 2014006704 A1	02-01-2014
		WO 2014004111 A1	03-01-2014