Sustainable Computing: Informatics and Systems xxx (2018) xxx–xxx

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment

Zhiyao Gong[a], Keni Qiu[a,*], Weiwen Chen[a], Yuanhui Ni[a], Yuanchao Xu[a], Jianlei Yang[b]

[a] Capital Normal University, Beijing, China
[b] Beihang University, Beijing, China

ARTICLE INFO

Article history:
Received 15 November 2017
Received in revised form 6 April 2018
Accepted 10 September 2018
Available online xxx

ABSTRACT

Electromagnetic radiation effects can cause several types of errors on traditional SRAM-based registers such as single event upset (SEU) and single event functional interrupt (SEFI). Especially in aerospace where radiation is quite intense, the stability and correctness of systems are greatly affected. By exploiting the beneficial features of high radiation resistance and non-volatility, spin-transfer torque RAM (STT-RAM), a kind of emerging nonvolatile memory (NVM), is promising to be used as registers to avoid errors caused by radiation. However, substituting SRAM with STT-RAM in registers will affect system performance because STT-RAM suffers from long write latency. The early write termination (EWT) method has been accepted as an effective technique to mitigate write problems by terminating redundant writes.

Based on the above background, this paper proposes to build registers by STT-RAM for embedded systems in rad-hard environment. Targeting the microarchitecture level of pipeline, the impact of architecting STT-RAM-based registers is discussed considering data hazard due to data dependencies. Furthermore, integrated with the EWT technique, Read Merging and Write Skipping methods are proposed to eliminate redundant normal reads and writes. As a result of carrying out these actions, the energy and performance can be improved greatly. The results report 70% (and 77%) and 39% (and 47%) improvements on performance (and energy) by the proposed Read Merging and Write Skipping compared to the cases where STT-RAM is naively used as registers and intelligently used by integrating EWT, respectively.

© 2018 Published by Elsevier Inc.

1. Introduction

The stability of electronic devices has been one of the main concerns of microelectronics industry and research for space applications. Radiation in the space environment may cause permanent errors as well as soft errors. As IC components keep reducing their feature size, along with the operating voltages and their power consumption, they become more sensitive to radiation effects. Single event upset (SEU) caused by radiation will change the state of register bits and may cause lasting problems to a system which cannot recover from such an error [1,2]. Hence, it is essential to make electronic components and systems resistant to damage or malfunctions caused by ionizing radiation.

To enhance the stability of systems in high radiation environment, spin-transfer torque RAM (STT-RAM) [3,4], a new generation of magnetic random access memory (MRAM), is considered to be used as registers to avoid errors caused by radiation [5–7]. Unlike traditional memory such as SRAM, STT-RAM exhibits excellent anti-radiation feature due to the fact that the data carrier of it is magnetic tunnel junction (MTJ) instead of electric charges. It has been experimentally demonstrated that the STT-MTJ materials are highly tolerant to radiations. Samples were exposed to 2 MeV and 220 MeV protons and showed no changes in bit-state or write performances [8]. Radiation testing results show that STT-RAM will not suffer SEUs when used in space [5]. Thanks to its easy integration with CMOS and infinite endurance, STT-RAM has been proposed to be widely used in order to overcome the power challenge of conventional CMOS circuits [6]. Therefore, in many harsh environments such as aerospace, STT-RAM is an ideal candidate to build registers. In fact, STT-RAM-based register file has been used in [9–11] to achieve lower dynamic and leakage energy consumption. Recently, IBM researchers in collaboration with Samsung researchers demonstrated 11nm STT-RAM junction, which is a significant achievement on the way to substitute DRAM with STT-RAM [12]. This work proposes to build STT-RAM-based registers for embedded systems in rad-hard environment.

* Corresponding author.
E-mail address: qiukn@cnu.edu.cn (K. Qiu).

https://doi.org/10.1016/j.suscom.2018.09.001
2210-5379/© 2018 Published by Elsevier Inc.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

However, replacing SRAM with STT-RAM will impact system performance due to the fact that STT-RAM suffers from longer latency and higher energy on write operation, resulting in asymmetric features in terms of read and write access latency, power consumption and endurance [13–16]. Generally, the latency of write operation is approximately three to six times longer than that of read operation. When STT-RAM is used as registers, the long latency and high energy on write operations will impose impact on performance and energy of architectural components.

In order to reduce write energy of STT-RAM, the technique Early Write Termination (EWT) has been proposed to significantly save write energy with no performance penalty [17]. The design of EWT is based on the observation that many bits are written with the same value that has been already stored. Those writes are therefore unnecessarily to be rewritten and should be removed to save energy. Another key support of EWT is the unique feature of STT-RAM that the change of MTJ resistance is not a gradual procedure during a write. Instead, resistance changes abruptly near the end of a write cycle [18]. A simple EWT circuit is designed to terminate such redundant bit writes at their early stages by comparing a to-be-written value with the old value obtained by a sensing circuit. EWT does not introduce any overhead to access latency and can easily be integrated with existing write circuit. Besides, it can be implemented with low complexity and low energy overhead.

Pipelining has been widely applied to exploit instruction level parallelism (ILP) [19,20]. The elements of a pipeline are often executed in parallel or in time-sliced fashion [21]. Pipeline is supposed to be full to achieve better efficiency, but when the same register is accessed at different time and pipeline stages, data hazard may happen. One solution is adding stalls to guarantee data correctness. As a result, pipeline will not be full anymore and its efficiency will decrease as well. In this paper, new impact on data hazard is observed when architecting STT-RAM-based registers. Data dependency will be different in STT-RAM-based registers because a write operation may take more cycles than a read operation. Comparison is made between SRAM-based and STT-RAM-based registers in terms of six different kinds of data dependencies: *forward-WAR*, *forward-RAW*, *forward-WAW*, *backward-WAR*, *backward-RAW* and *backward-WAW*.

Integrated with the EWT technique to mitigate unnecessary bit write, we propose that the normal read and the sensing read at the early stage of a write can be merged under RAW and WAR dependencies in pipeline. A normal read can be removed under a RAW dependency and a sensing read circuit can be shut off under a RAW dependency. Besides, redundant writes can be skipped for WAW dependencies. In this way, dynamic energy can be further improved for STT-RAM-based registers. By applying this Read Merging and Write Skipping to optimize pipeline integrated with EWT, the experimental results report 39% and 47% improvements on performance and energy compared to the pure EWT technique. In summary, the contributions of this paper are as follows.

- To the best of our knowledge, this is the first work that proposes to use STT-RAM as registers for embedded systems in hard radiation environment.
- In pipeline architecture, new definitions are made to classify data dependencies into six types. Impact of each data dependencies is analyzed in detail considering write with the EWT technique as well.
- A Read Merging method is proposed to further eliminate redundant normal reads under a RAW dependency or sensing reads along with a write under a WAR dependency. A Write Skipping method is proposed to skip redundant writes under a WAW dependency. In this way, access performance and dynamic energy of pipeline can be further improved.

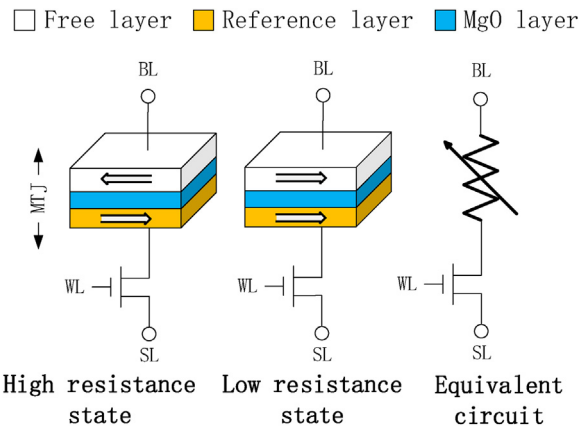

Fig. 1. Abstract STT-RAM structures.

- Experiments are conducted to quantitatively evaluate the impact on pipeline by building STT-RAM-based registers integrated with EWT, as well as the effectiveness of the proposed Read Merging and Write Skipping methods.

The rest of the paper is organized as follows. Section 2 introduces STT-RAM background on anti-electromagnetic radiation, the EWT technique and the motivation of this work as well. Section 3 makes definitions to classify data dependencies for STT-RAM based registers in pipeline. Section 4 discusses data dependency types in pipeline and presents impact on pipeline. Section 5 presents the proposed Read Merging and Write Skipping methods to further improve pipeline efficiency. Section 6 shows experimental results to evaluate the impact on pipeline, and improvements on access performance and dynamic energy with the proposed Read Merging method. Finally, this paper is concluded in Section 7.

2. Background and motivation

In this section, we first describe the STT-RAM preliminaries and its nature of anti-electromagnetic radiation. Then a technique, EWT, is introduced to explain how it is applied to mitigate the write problem. Finally, the two major motivations of this paper are presented.

2.1. STT-RAM background

Spin-Transfer Torque RAM (STT-RAM) has emerged as a potential candidate for universal memory [18,22,15,23]. In a STT-RAM device, the spin of the electrons is flipped using a spin-polarized current. This effect is achieved in a Magnetic Tunnel Junction (MTJ). The magnetization direction of reference layer is fixed while that of free layer can be flipped by passing a spin-polarized current [15]. The MTJ resistance is determined by the relative magnetization direction of the FL relative to the RL: when their magnetization directions are parallel (anti-parallel), MTJ is in low (high) resistance state. In this way, "0" and "1" can be represented by the different MTJ resistances. Fig. 1 illustrates the abstract structures of STT-RAM.

It can be seen that STT-RAM cell does not fundamentally carry electric charge. This natural immunity to electromagnetic radiation in harsh space environment makes it as an ideal candidate to replace the traditional SRAM technology [6,8,5,24]. Addressing this issue, extensive researches have been done to verify its high resistance of radiation.

G. Tsiligiannis et al. evaluated the soft error resilience of a commercial toggle MRAM in [5]. Two kinds of test modes are used to evaluate the sensitivity of the Toggle MRAM under radiation test: the static and dynamic mode. Static mode provides that the mem-

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

Table 1
Parameters of SRAM, SLC STT-RAM and MLC STT-RAM.

Parameters	SRAM	SLC STT-RAM	MLC STT-RAM
Read latency (cycles)	7.43	9.08	S: 6.73, H: 9.80
Read dyn. eng. (nJ)	0.161	0.216	S: 0.22, H: 0.43
Write latency (cycles)	5.78	25.58	S: 25.31, H: 56.50
Write dyn. eng. (nJ)	0.156	0.839	S: 0.843, H: 2.502
Leakage power (mW)	295.58	18.39	7.02
Array area (mm^2)	7.28	1.86	1.01

ory will be written with a known bit pattern, dynamic testing mode requires that the memory is constantly accessed. No SEU has been observed neither in static mode nor in dynamic mode. Besides, there is no SEUs neither during the neutron and the alpha particle irradiation. This proves the high resistance of radiation of MRAM.

Y. Lakys et al. proposed novel hardening techniques to mitigate SEE in [6]. It is observed that logic circuits based on magneto-resistive storage cells are still vulnerable to radiations due to their CMOS peripheral circuits. To solve this problem, the authors present the Radhard version of the nonvolatile MRAM-based storage element. Besides, the authors present the implementation of triple modular redundancy (TMR) on the magnetic look-up table (MLUT). This technique proves efficiency in improving the overall reliability of the design against fabrication process variability.

D. Chabi et al. proposed a rad-hard design for STT-RAM sensing circuit in [8]. It is observed that the CMOS peripheral sensing circuits of STT-MTJ are vulnerable to radiation due to the stochastic behaviors of spin transfer torque mechanism. By using an accurate physics-based MTJ Spice model and a commercial 40 nm CMOS bulk design kit, transient and Monte-Carlo simulations have been performed to quantify the effect of charged particles strike and evaluate the performance of the proposed design. The results showed that the rad-hard STT-MTJ structure is quite robust against radiation effect with low area overhead and modest degradation in performance.

Although STT-RAM exhibits excellent immunity to electromagnetic radiation, on the other hand, STT-RAM suffers from longer latency and higher energy on write operation by replacing SRAM, resulting in asymmetric features in terms of read and write access latency, power consumption and endurance [13,16]. Table 1 shows the parameters of SRAM, SLC (Single-Level Cell) STT-RAM and MLC (Multi-Level Cell) STT-RAM [7]. It is known that registers are frequently written component in a system. When architecting STT-RAM for registers, the long write latency will impose great impact on both performance and energy of architectural components such as pipeline.

2.2. EWT technique

As discussed before, though STT-RAM is featured as radiation immunity, high density, low leakage power and long endurance, it suffers from long write latency and high dynamic energy on write operations. Several existing studies have shown that there is a high probability that a write does not change its content, and therefore can be removed. Such an observation has been used at the word level. Furthermore, the work [17] observed that the phenomenon is more significant at the bit level. Their experimental results for a 16MB STT-RAM L2 cache showed that 88% of bit-writes are redundant, which implies a significant amount of removable bit writes and a great potential of energy savings in STT-RAM.

Addressing the write problem of STT-RAM, the work [17] proposed the EWT technique, a novel write scheme with the capability of early termination in case of a redundant write. The basic idea is to sample the old value of resistance of the MTJ at early stage of a write operation, and throttle the write current if old value is the same as the new value. To achieve this goal, a sense amplifier *SA* is added for each column to sense the resistance of the MTJ during a write operation, and some additional logic to generate the control signals. The sense amplifier is lightweight but sufficient to sense the old resistance of MTJ. Different from the normal sense amplifier used for read operations, this separate special one works with write operations which generate much larger current than reads. For easier explanation, the procedure of EWT is restated here as follows.

Fig. 2. Comparison of write behavior between traditional STT-RAM and STT-RAM with EWT.

- When a write operation begins, write voltage is applied between BL (bit line) and SL (source line) to form a write current.
- When the signals are stabilized, sense amplifier *SA* is enabled to sense the old resistance value of the MTJ.
- After the old value is sensed, it is saved in a latch and *SA* is turned off to save power.
- If the old value is the same as the new value, a control signal *WCUT* (write cut) is generated to shut off the write circuit and terminate the operation on this cell. Otherwise, write operation on this cell continues normally.

As can be seen, the new kind of read R^{SA} by sensing the old value is done together with the write operation, with no overhead to write operations in terms of performance. In other words, the above process does not require an extra read to precede a write because sensing the old value is done together with the write operation. The content in the register can be read along with write operation by the extra R^{SA} by adopting EWT. In this way, write energy can be reduced since that some write requests can be completely throttled and terminated in their early stage. This scheme can be abstracted in Fig. 2 where R^{SA} is conducted at the early stage with a write operation. Note that the write operation have two possibilities: short write (SW) and long write (LW). They are corresponding to the situations with no change and with change on new value to be written respectively.

2.3. Motivation

This paper is motivated from the following two aspects.

First, STT-RAM is a good replacement of SRAM to be used as registers due to its natural advantage of immunity to electromagnetic radiation. The registers can be designed in a compact and light manner by removing redundant fault-tolerance and correction circuit against radiation.

Second, addressing the impact on pipeline by building STT-RAM-based registers, our work is inspired by the EWT technique to propose a Read Merging technique which improves performance and save energy by merging a normal read (R^{Norm}) and a sensing read R^{SA} along with a write. A Write Skipping method is proposed to skip redundant writes under a WAW dependency.

In the EWT technique, an extra read along with a write can be used to terminate redundant bit writes at their early stages. Different from a normal read, this extra read is completed by a specific circuit during the early stage of write operation. The original goal

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

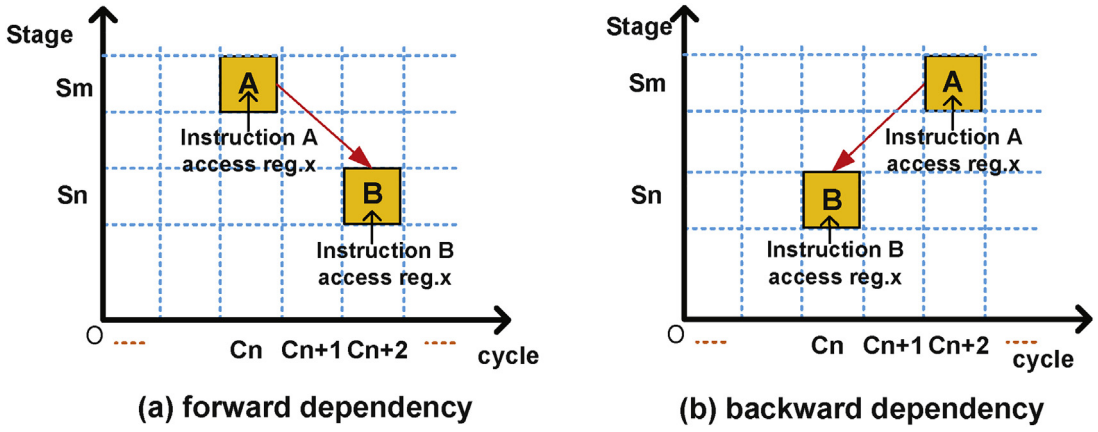

Fig. 3. Forward and backward dependency.

of the EWT is to greatly reduce the write energy. In this paper, it can be exploited to save normal read (R^{Norm}) if there is RAW dependency, or turn off *SA* if there is WAR dependency in pipeline. That is, R^{Norm} and R^{SA} can take use of each other by merging them to save energy and/or read latency in pipeline. For WAW dependency, it is observed that the first write operation can be seen as a redundant write and thus can be skipped to further reduce cycles of write.

In summary, targeting embedded systems in rad-hard environment, this paper proposes to build STT-RAM-based registers to defend against electromagnetic radiation. Addressing the write problem of STT-RAM, this paper provides insightful observations on its impact of pipeline. Furthermore, Read Merging and Write Skipping optimization methods are proposed to improve performance and save energy of pipeline integrated with the EWT technique.

3. Classification of data dependencies

In order to analyze the impact of building STT-RAM-based registers on pipeline considering data hazard, we first model data dependencies in pipeline. Conventionally, the data dependencies can be classified into three types according to their access pattern: data, anti, and output dependencies or read-after-write (RAW), write-after-read (WAR), and write-after-write (WAW) dependencies respectively. Though this classification provides helpful insight into the understanding of detection and resolution of hazards in some simple pipeline structures such as typical five-stage F-D-E-M-W (fetch, decode, execute, memory access, register write back) processors, it is not powerful enough for general pipeline structures. In generic pipeline structures such as pipelined architectures, register accesses may happen in many pipeline stages in order to support multiple-cycle instructions. Therefore, the work [25] makes additional definitions to classify dependencies further into three types: forward, backward and stationary dependencies. Since stationary dependency does not cause any pipeline hazard, here in this paper we concern forward and backward dependencies only. For convenience of further explanation, forward/backward dependencies referring to data flow direction are restated as below.

Definition 1 *(Forward dependency).* A forward dependency happens when a preceding instruction accesses a register at an earlier cycle and a succeeding instruction accesses the same register at a latter cycle. These two instructions access the same register at different pipeline stages.

Besides, we suppose instruction *A* is the preceding instruction of instruction *B* in this paper to make things easy to understand.

Definition 2 *(Backward dependency).* A backward dependency happens when a succeeding instruction accesses a register at an earlier cycle and preceding instruction accesses the same register at a latter cycle. These two instructions access the same register at different pipeline stages.

Similarly we suppose instruction *B* is followed by instruction *A* in this paper.

Fig. 3 illustrates this two kinds of dependencies. As shown in Fig. 3(a), instruction *A* accesses register *x* at cycle *Cn*, and instruction *B* accesses the same register at cycle *Cn* + 2. As instruction *A* accesses register *x* at an earlier cycle, it is denoted as a *forward dependency*. In Fig. 3(b), instruction *B* accesses register *x* at cycle *Cn*, and instruction *A* accesses the register *x* again at cycle *Cn* + 2. Instruction *B* accesses register *x* at an earlier cycle, it is denoted as a *backward dependency*.

On the basis of the above definitions, this paper classifies data dependencies in pipeline into six types: ***forward-WAR***, ***forward-RAW***, ***forward-WAW***, ***backward-WAR***, ***backward-RAW*** and ***backward-WAW*** dependencies. Compared with registers built of SRAM, several interesting impact is observed in pipeline where STT-RAM is used as registers because dependencies in pipeline will be very different due to its long write latency. In this paper we focus on the situation where the same register is accessed at different pipeline stages. Note that the stalls which are added into pipeline to eliminate data hazard are illustrated under the situation that LW happens in the worst cases. We give the details of each observation as follows:

4. Impact on pipeline

4.1. Forward-WAR dependency

According to Definition 1, Fig. 4(a) exhibits forward dependency due to the fact that a register is accessed by instruction *A* at an earlier cycle and the same register is accessed by instruction *B* later. What is more, a WAR dependency is exhibited because instruction *A* first reads a register and then instruction *B* writes the same register. Therefore, Fig. 4(a) and (b) both present a forward-WAR dependency. We use *WAR*→ to denote this kind of data dependency.

It can be seen that this kind of dependency will not cause any pipeline hazard in both STT-RAM-based and SRAM-based registers. The main difference is that in pipeline where registers are built of STT-RAM, the write operation takes more cycles due to its long latency compared with pipeline where registers are built of SRAM. Therefore, the write operation of STT-RAM costs extra cycles.

Observation 1 on pipeline. *For forward-WAR dependency, there are no data hazards for both SRAM-based and STT-RAM-based registers. But STT-RAM costs extra cycles on writes due to its long write latency.*

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

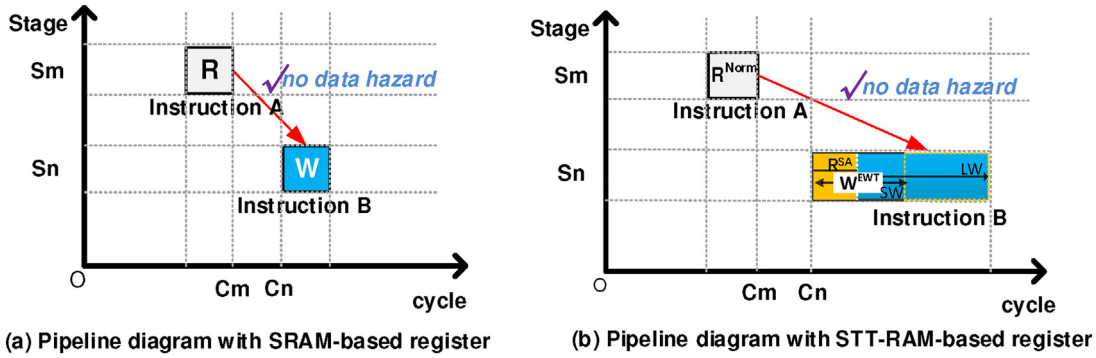

Fig. 4. Forward WAR dependency.

4.2. Forward-RAW dependency

Fig. 5(a) exhibits forward dependency according to Definition 1 due to the fact that a register is accessed by instruction *A* at an earlier cycle and the same register is accessed by instruction *B* later. What is more, a RAW dependency is exhibited because instruction *A* first writes a register and then instruction *B* reads the same register. Therefore, Fig. 5(a) and (b) both represent forward-RAW dependency. *RAW*→ is used to denote this kind of dependency in this paper.

In these cases, challenge occurs when we use STT-RAM as registers. While there are no pipeline hazards for SRAM-based registers as shown in Fig. 5(a), pipeline hazard is exhibited in STT-RAM-based registers because the write operation takes so long time that it completes after read operation finishes. Since instruction *A* is preceding of instruction *B*, instruction *B* needs to read the data written in register by instruction *A*, it is impossible to get the data while it is not yet written into the register. Therefore, data hazard occurs.

A commonly adopted solution to this problem is adding stalls before the read operation so that the correctness of data can be guaranteed. As illustrated in Fig. 5(c), two stalls are added before the read operation. Extra stalls in a pipeline will worsen its efficiency and performance. This impact will be calculated in the Experiments.

Observation 2 on pipeline. *For forward-RAW dependency, while there are no data hazards existing in SRAM-based registers, STT-RAM-based registers exhibit data hazard, therefore, stalls are needed to guarantee data correctness.*

4.3. Backward-WAR dependency

According to Definition 2, Fig. 6(a) exhibits backward dependency. Instruction *B* accesses a register at an earlier cycle and instruction *A* accesses the same register later. What is more, a WAR dependency is exhibited because instruction *A* first reads a register and then instruction *B* writes the same register. Putting them together, Fig. 6(a) represents a backward-WAR dependency. we use *WAR*⟵ to denote this kind of data dependency in this paper.

A pipeline hazard occurs in Fig. 6(a) because instruction *B* is followed by instruction *A*, and then write operation is supposed to be implemented after read operation finishes. A solution to resolve this hazard is adding stalls as depicted in Fig. 6(b). By adding three stalls into the pipeline, the correctness of data can be guaranteed for both cases of STT-RAM-based register and SRAM-based register.

Observation 3 on pipeline. *For backward-WAR dependency, same stalls are needed for both STT-RAM-based and SRAM-based registers to guarantee data correctness.*

4.4. Backward-RAW dependency

According to Definition 2, Fig. 7(a) exhibits backward dependency due to the fact that instruction *B* accesses a register at an earlier cycle and instruction *A* accesses the same register later. What is more, a RAW dependency is exhibited because instruction *A* first writes a register and then instruction *B* reads the same register. Putting them together, Fig. 7(a) and (c) both represent backward-RAW dependency. We denote this by the symbol *RAW*⟵.

This dependency will cause pipeline hazard because read operation of a register is supposed to be implemented after write operation of the same register. In other words, only after write operation to a register can we read the the data written into it. To resolve this data hazard, Fig. 7(b) adopts the traditional solution by adding 3 stalls into the pipeline. As a result, the read operation can be now executed after the write operation completes. And correctness of data is guaranteed in this way. As shown in Fig. 7(c), pipeline hazard is more severe for registers built of STT-RAM due to its long latency on write operation. This hazard can be eliminated by adding 6 stalls into the pipeline as depicted in Fig. 7(d), which exhibits a lower efficiency.

Observation 4 on pipeline. *For backward-RAW dependency, both SRAM-based and STT-RAM-based registers exhibit data hazard. The main difference is that STT-RAM-based registers need more stalls than SRAM-based registers to eliminate the hazard because of longer write latency for STT-RAM.*

4.5. Forward-WAW dependency

As shown in Fig. 8(a), a register is accessed by instruction *A* at an earlier cycle and the same register is accessed by instruction *B* later. This figure exhibits forward dependency according to Definition 1. Besides, a WAW dependency is exhibited due to the fact that instruction *A* first writes a register and then instruction *B* writes the same register. Therefore, Fig. 8(a) and (c) both represent forward-WAW dependency. We use *WAW*→ to denote this kind of dependency in this paper.

This kind of dependency will not cause data hazard for SRAM-based registers since the write operation of the second instruction is done after that of the first instruction as shown in Fig. 8(a). However, for registers built of STT-RAM, pipeline hazard occurs. This is because the write latency of STT-RAM is so long that write operation of the second instruction starts while the first instruction hasn't finished its write operation. As shown in Fig. 8(b), the two write operations overlapped, which causes problems of data correctness. To resolve this data hazard, by adding 3 stalls into the pipeline as shown in Fig. 8(c), the write operation of the second instruction can be now executed after conducting the write operation of the first instruction. Data correctness can be guaranteed in this way.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

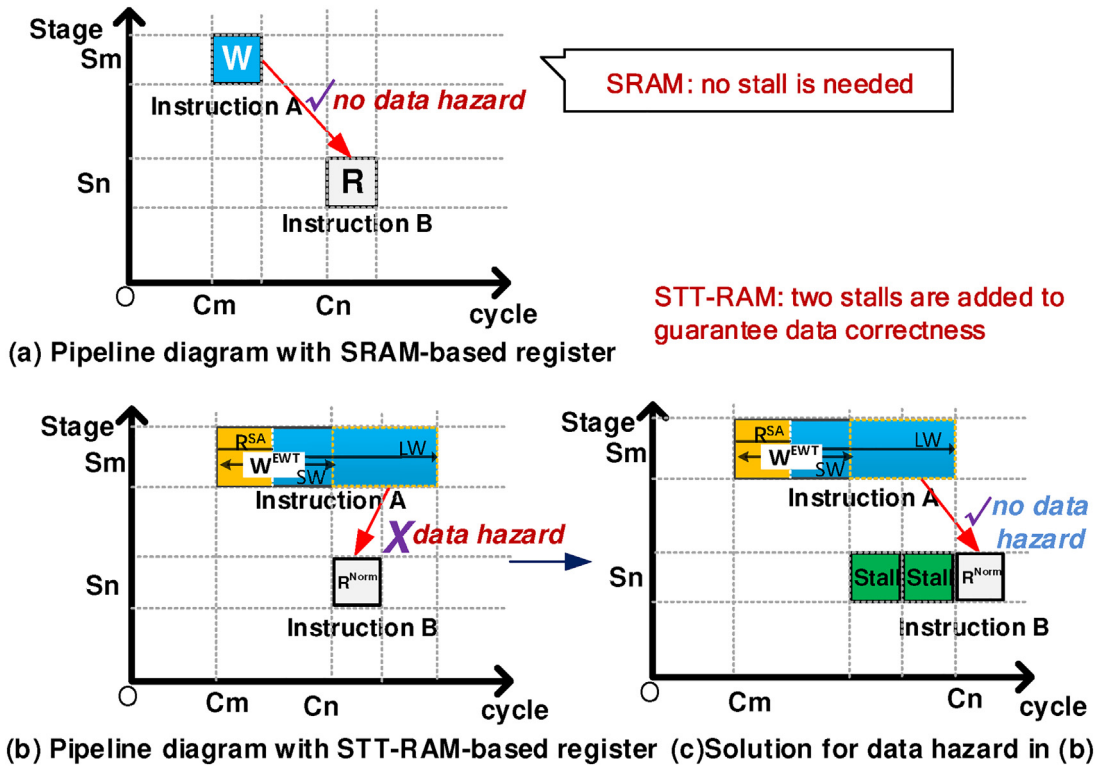

Fig. 5. Forward RAW dependency.

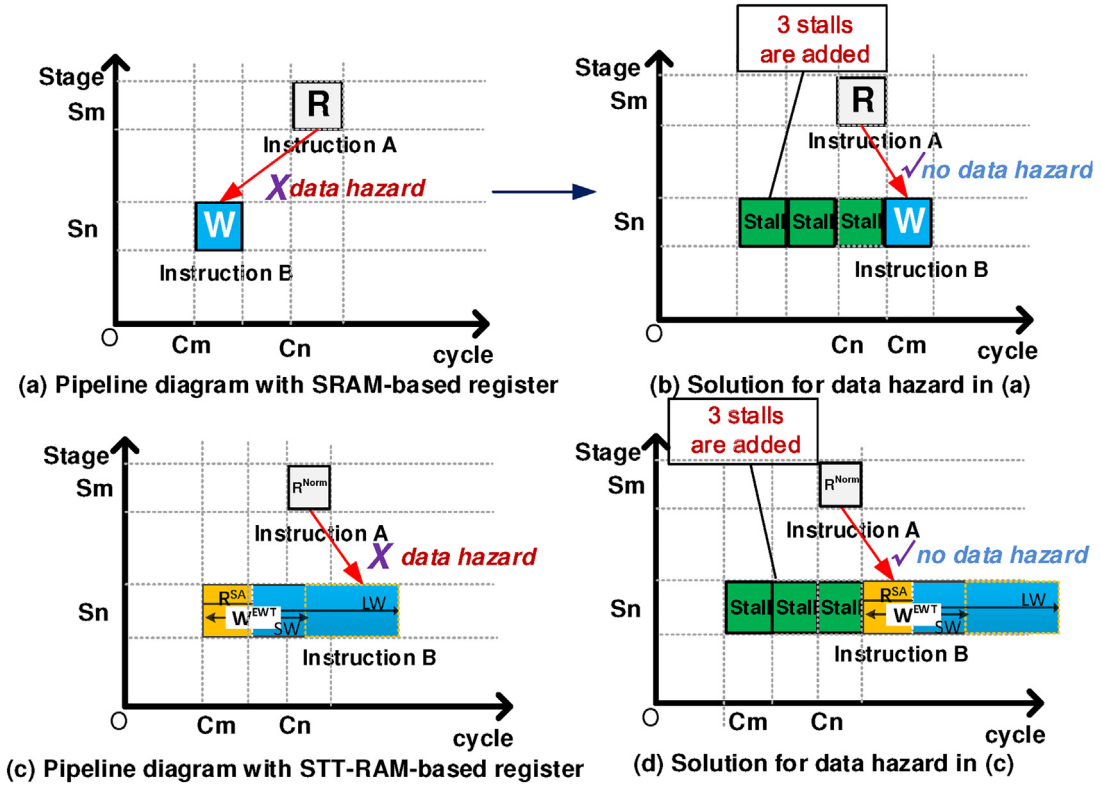

Fig. 6. Backward WAR dependency.

Observation 5 on pipeline. *For forward-WAW dependency, while there are no data hazards existing in SRAM-based registers, STT-RAM-based registers exhibit data hazard, therefore, stalls are needed to guarantee data correctness.*

4.6. Backward-WAW dependency

As shown in Fig. 9(a), a register is accessed by instruction *B* at an earlier cycle and the same register is accessed by instruction *A* later.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

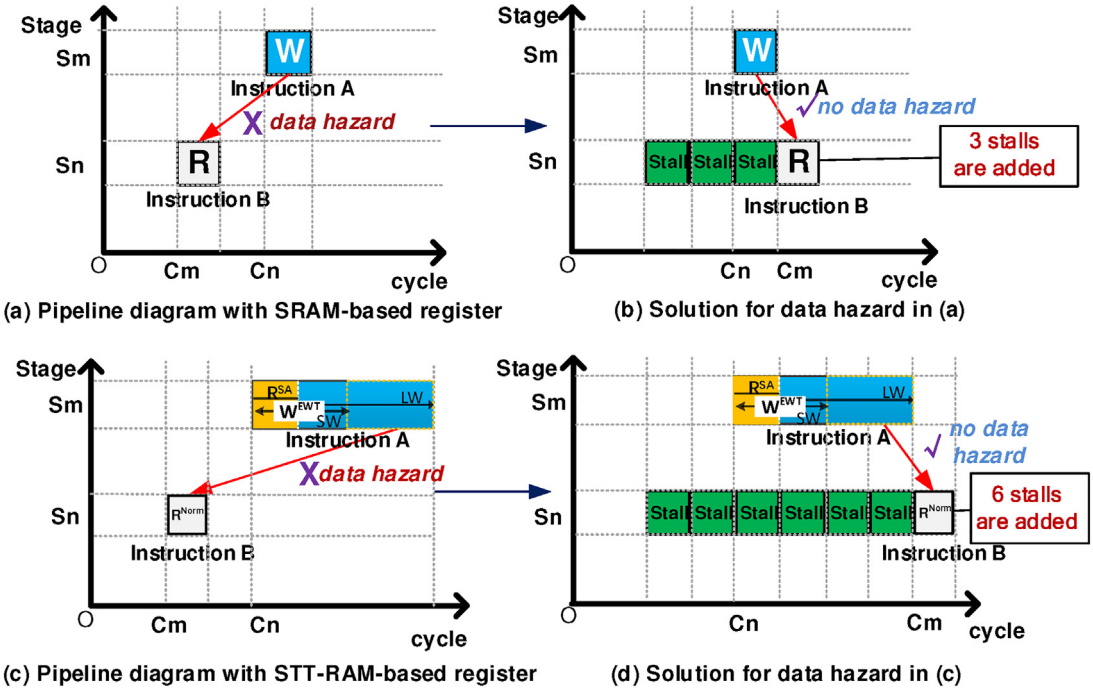

Fig. 7. Backward RAW dependency.

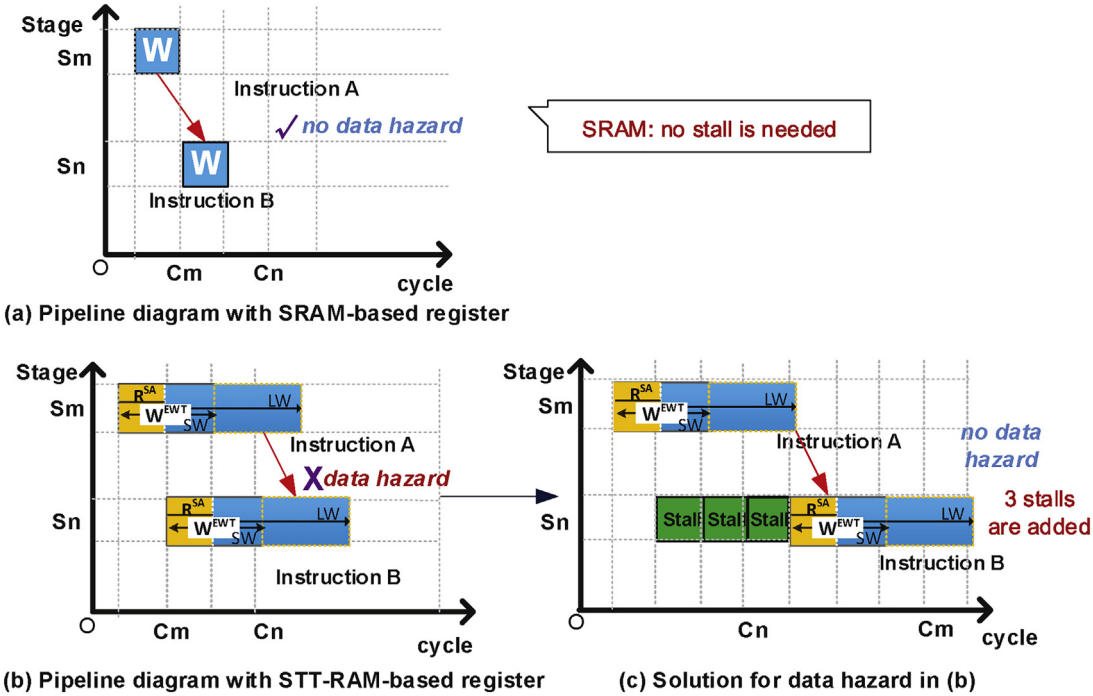

Fig. 8. Forward WAW dependency.

This figure exhibits backward dependency according to Definition 2. Besides, a WAW dependency is exhibited due to the fact that the same register is written by two instructions in succession. Therefore, Fig. 9(a) and (c) both represent backward-WAW dependency. We use *WAW*⟵ to denote this kind of dependency in this paper.

This kind of dependency will cause data hazard in pipeline. This is because write operation of instruction *A* is supposed to be implemented before write operation of instruction *B*. To resolve this data hazard, Fig. 9(b) adopts the traditional solution by adding 2 stalls into the pipeline. As a result, the write operation of the first instruction can be now executed after that of the second instruction. In this way, correctness of data is guaranteed. As shown in Fig. 9(c), pipeline hazard is more severe for registers built of STT-RAM due to its long latency on write operation. This hazard can be eliminated by adding 5 stalls into the pipeline as depicted in Fig. 9(d), which exhibits a lower efficiency.

Observation 6 on pipeline. *For backward-WAW dependency, both SRAM-based and STT-RAM-based registers exhibit data hazard. The main difference is that STT-RAM-based registers need more stalls than*

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

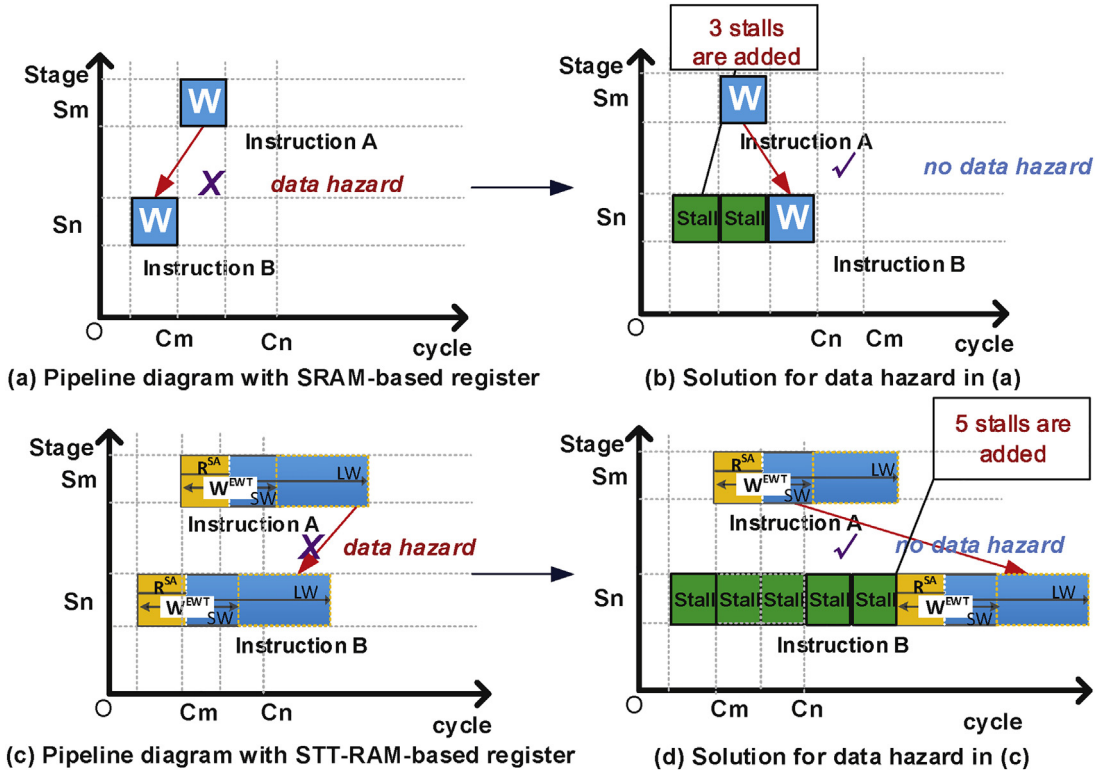

Fig. 9. Backward WAW dependency.

SRAM-based registers to eliminate the hazard because of longer write latency for STT-RAM.

5. Pipeline optimization strategy

5.1. Read Merging

5.1.1. Overview of Read Merging

In pipeline architecture, it is known that data dependency may cause data hazard. When there are dependencies concerning a write operation and a read operation, two types of reads are implemented in fact. The first type of read is the normal read of STT-RAM (R^{Norm}). The second type of read is conducted in the early phase of write operation by adopting EWT technique (R^{SA}). When the same register is frequently read/written alternatively, the two kinds of read operations are conducted. It can be seen that for WAR/RAW dependencies, the same register needs to be read twice. Although the two reads are different, it is observed that they can make use of each other and redundant read operation can be avoided. The detailed Read Merging solutions are categorized into two types for RAW and WAR dependencies respectively.

5.1.2. Read Merging for RAW dependencies

For the cases of forward/backward-RAW dependencies corresponding to Observations 2 and 4, the scenario after data hazard is depicted in Fig. 10. The read operation of the second instruction can be now conducted after the write operation of the first instruction. It can be seen that a special read R^{SA} is conducted and then a normal read R^{Norm} is conducted. In this case, we can forward the value of R^{SA} to the start stage of R^{Norm} such that the latter can be throttled. As known that the old value of MTJ is saved in a latch after being sensed, the value forwarding can be feasibly conducted in pipeline. Using this method, both the read delay and energy of R^{Norm} can be saved.

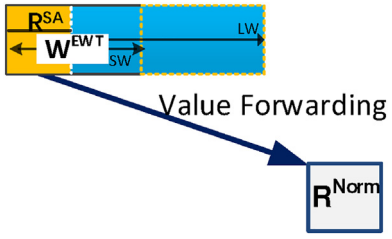

Fig. 10. Read Merging under RAW dependencies.

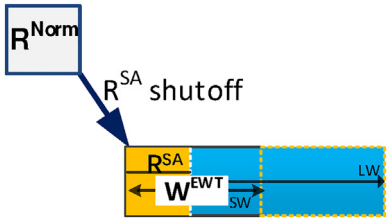

Fig. 11. Read Merging under WAR dependencies.

5.1.3. Read Merging for WAR dependencies

For the cases of forward/backward-WAR dependencies corresponding to Observations 1 and 3, the scenario after data hazard is depicted in Fig. 11. The write operation of the second instruction can be now conducted after the read operation of the first instruction. We know that at the early stage of write, a R^{SA} will be conducted. It can be seen that a normal read R^{Norm} is conducted and then a special read R^{SA} is conducted. In this case, we can forward the value of R^{Norm} in instruction *A* into the latch of R^{SA} circuit directly. The sense amplifier of R^{SA} can be immediately turned off to save energy. In order to realize this, the circuit design of the early stage of a write in the EWT should receive small modifications as shown in Fig. 12. In the modified circuit highlighted in the dash box,

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

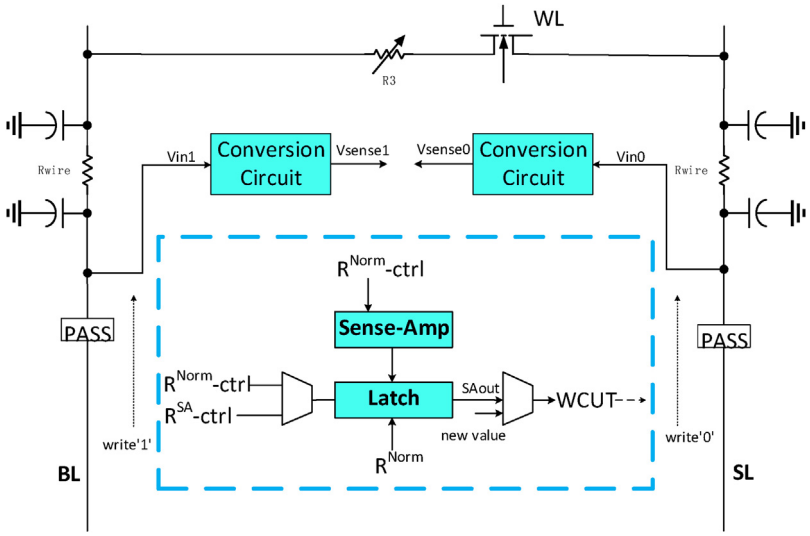

Fig. 12. Modified EWT circuit to support Read Merging.

an input from R^{Norm} is directed to *Latch* and meanwhile the signal R^{Norm}*-ctrl* is used to disable the sense amplifier circuit for the case that R^{SA} can be throttled under WAR dependencies.

5.2. Write Skipping

5.2.1. Overview of Write Skipping

As mentioned before, WAW dependency will result in data hazards in pipeline. WAW dependency includes twice write operations. It is known that the write latency is far longer than that of read operation. Especially for loops consisting intensive write operations, the long write latency makes this dependency need more cycles than other dependency, which worsens the overall performance of pipeline. To improve its performance, a Write Skipping strategy is proposed to avoid unnecessary write operations to further reduce cycles for WAW dependency.

5.2.2. Methodology of Write Skipping

Program locality has been studied in [26]. The locality phase prediction is well suited to identify recurring phases in programs. In pipeline, there will be intensive WAW dependencies in a specific phase while the number of other dependencies is very small. Write Skipping strategy is adopted in this phase to improve write performance.

For WAW dependency, the variable in a register is written by the first write operation and then the second write operation changes its value. Due to the fact that the value written by the first operation is never used, it is observed that the first write can be avoided. In other words, for WAW dependency, the first write operation can be seen as a redundant write. Thus we are motivated to skip the first write and only conduct the second write.

The challenge lies in that how to find WAW dependency from all dependencies in this specific phase. The proposed Write Skipping detects the first operation. When a write operation is detected, we skip this write operation. Then we detect the second operation. If the second operation is a write, then it is a WAW dependency, the first write can be avoided and there is no error, as depicted in Fig. 13(a) Circumstance 1.

Definition 3 *(Source register).* Source Register is the register which provides the value of the first write operation in WAW dependency.

However, if the second operation is a read, data correctness will be destroyed since that we have skipped the first write while the second read needs its value. To solve this problem, the second read must read the value from Source Register. As shown in Fig. 13(b) Circumstance 2, the first operation is detected as a write, the second operation is a read. Since that the first write is skipped, the second operation cannot read its value. So the second operation reads value from Source Register *x* which provides the value of the first write to guarantee data correctness.

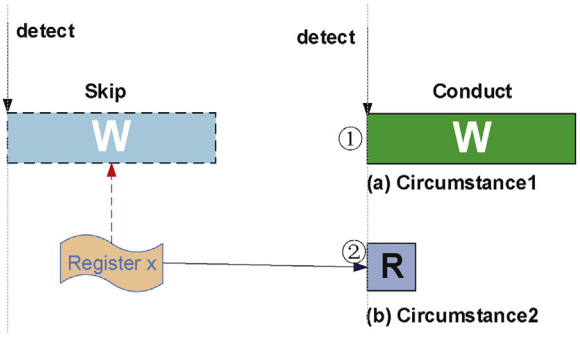

Fig. 13. Write Skipping.

6. Experimental evaluation

In this section, latency and energy in pipeline considering the EWT and Read Merging and Write Skipping techniques are first modeled. Then the experimental setup of this work is introduced. Finally, the evaluation results on performance and energy are presented respectively.

6.1. Modeling

1) *Latency* We adopt the latency model in the work [17] since we also integrate the EWT technique into our method. The delay of the peripheral circuit is considered. It is assumed the concerned register is in length of 32 bits. The latencies for different read and write accesses are listed in Table 2.
2) *Energy* When EWT is enabled, the write energy can be expressed by the sum of three parts in Eq. (1).

$$R_{EWT} = E_{peri} + E_{oh} + 2.767\dot{N}_{changed} + 0.148\dot{N}_{unchanged} \quad (1)$$

E_{peri} is the energy consumed by the peripheral logic. This is 0.203 nJ, same as in [17]. E_{oh} is the energy consumed by the

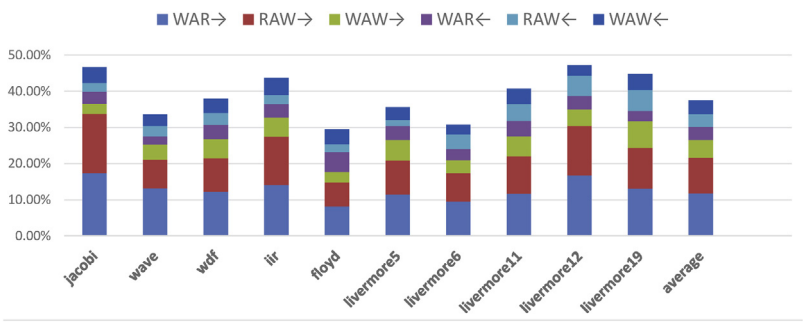

Fig. 14. Percentage of each data dependency.

Table 2
Latency model.

Access	Latency
Read	6.232ns
Write with no EWT	12.544 ns
Write with EWT	12.544 ns/3.090 ns

Table 3
Energy model.

Access	Energy
Read	0.205 nJ
Write with no EWT	1.620 nJ
Write with EWT	0.2487 nJ+2.767$\dot{N}_{changed}$ + 0.148$\dot{N}_{unchanged}$

EWT circuits. This part is 0.0457 nJ per write access, calculated as per cell overhead multiplied by the number of cells in a 32-bit register since there is one set of EWT circuit per column. This latter two parts in Eq. (1) depend on how many cells are actually changed during a write operation. According to the simulation in [17], the energy used to change one cell is 2.767 pJ. Write operations on unchanged cells are terminated which amounts to 0.148 pJ per cell.

Put all together, the dynamic energy for different accesses are listed in Table 3.

6.2. Experimental setup

Our evaluation on pipeline is based on *SimpleScalar* [27]. The module *sim-outorder* is used to implement pipeline simulations where the *inorder* issuing mode is set true. The EWT and Read Merging and Write Skipping techniques are both simulated. The benchmarks are selected from DSP programs and Livermore benchmarks [28] where there are many read and write dependencies in loops.

First, we compare the performance of pipeline with traditional STT-RAM-based registers to that with SRAM-based registers with respect to the number of stalls and execution time overhead due to the stalls. Then, we evaluate the access performance and dynamic energy under three circumstances: traditional STT-RAM-based registers (*naive*), STT-RAM-based registers with EWT (*EWT*) and Read Merging and Write Skipping integrated with EWT (*EWT+RM+WS*).

6.3. Percentage of dependency

Fig. 14 presents the percentage of each dependency for all benchmarks. As we can see, the percentage of each dependency varies for different benchmarks. The data dependency takes up 37.5% for all benchmarks on average. Although the ratio of data dependency is high, note that some kinds of dependency will not result in data hazard. As can be seen, for all the benchmarks, the proportion of forward-WAR dependency is the largest, while that of other dependencies varies a lot. The proportion of forward dependencies is larger than that of backward dependencies. The effect of data dependency will be discussed in following sections.

6.4. Comparison to SRAM

1) *Stall number*:

Fig. 15 shows the results of the percentage of stall increasing under different dependency types between that with STT-RAM-based registers and SRAM-based registers. The experimental results comply with our analysis. It is observed that for all benchmarks, the increased number of stalls under forward-WAR, forward-RAW, forward-WAW, backward-WAR, backward-RAW and backward-WAW dependency over that of SRAM is 0%, 12.4%, 6.6%, 0%, 7.3% and 5.5% respectively on average. We have the following observations.

First, for all benchmarks, the stall number over that of SRAM under forward/backward-WAR dependency is zero. The reason lies in that there are no data hazards for STT-RAM-based

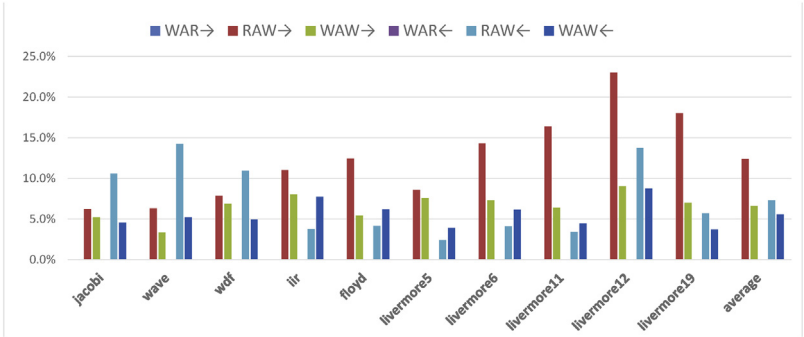

Fig. 15. Percentage of additional stalls to eliminate data hazard in pipeline with STT-RAM-based registers compared to that with SRAM-based registers.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

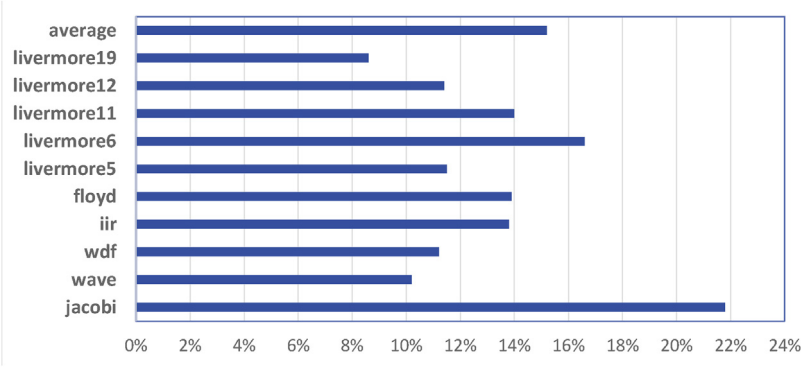

Fig. 16. Normalized execution cycles resulting from stalls to eliminate data hazard in pipeline with STT-RAM-based registers compared to that with SRAM-based registers.

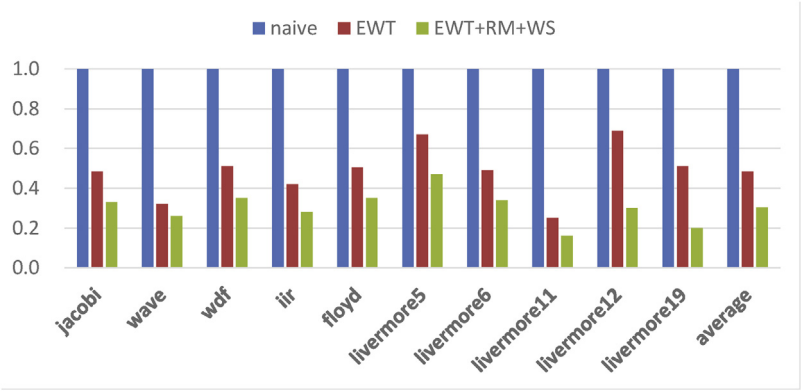

Fig. 17. Normalized access performance under *EWT* and *EWT + RM + WS*.

registers under forward-WAR dependency, and the same stalls as that of SRAM-based registers under backward-WAR dependency. Therefore, no additional stalls need to be added into the pipeline.

Second, the benchmarks under different data dependency types need different stall number. For benchmark *jacobi*, the stall number under forward-RAW dependency over that of SRAM is the smallest. This is mainly because this benchmark has few instructions with forward-RAW dependency, therefore, the number of stalls used to eliminate data hazard is small. In benchmark *wave*, the stall number under backward-RAW dependency over that of SRAM is the largest. Therefore, more data hazards occur in this situation, and more stalls are needed to guarantee data correctness. The stalls added for WAW dependencies vary for different benchmarks. The reason may lie in that the proportion of WAW dependencies is different and the stalls to resolve once data hazard vary for different benchmarks.

2) *Execution time*:

Fig. 16 presents the normalized execution time resulting from stalls in order to eliminate data hazard in pipeline with STT-RAM-based registers compared to that with SRAM-based registers. Both the read and write latencies are set as the same as that of read STT-RAM. The writes of STT-RAM are conducted under the EWT scheme. On average, 15.2% extra time is observed in the experiments by using STT-RAM as registers. The underlying reasons lie in two aspects. First, as discussed above, pipeline stalls can be increased under forward-RAW and backward-RAW dependencies with long write to STT-RAM. Second, the long write latency of STT-RAM itself is large, which results in more execution cycles due to stalls. Since the ratio of changed bits to all to-be-written bits is small, the execution performance does not degrade much.

6.5. Access performance

Fig. 17 presents the evaluation of access performance by counting register access cycles under the three circumstances of *naive*, *EWT* and *EWT+RM+WS*. The results are normalized to the baseline of *naive*. Two key observations are made by analyzing the results.

First, 51% improvement on average is observed from the comparison between *naive* and *EWT* due to the fact that some unnecessary bit write is eliminated by integrating EWT, therefore the latency consumed by writes is reduced.

Second, it is observed that by adopting the proposed *EWT+RM+WS* technology, 39% performance improvement on average is achieved compared to *EWT*. The reason lies in that Read Merging eliminates redundant normal reads under RAW dependency and Write Skipping eliminates redundant writes under WAW dependency, as a result, the access latency consumed by reads and writes are reduced.

In all, the access performance is improved through both write and read optimizations.

6.6. Dynamic energy

To evaluate the energy efficiency, we only consider the dynamic energy consumed by reads and writes of registers. As shown in Fig. 18, the total dynamic energy of *EWT+RM+WS* is reduced up to 77% over the baseline of *naive*. The improvement results from two-fold optimizations. First, the bit write number is largely reduced by *EWT*. Second, *EWT+RM+WS* can reduce R^{Norm} and R^{SA} and redundant writes under RAW and WAR and WAW dependencies respectively, which contributes a lot to energy reduction.

For the benchmark *jacobi*, its dynamic energy reduction in terms comparison between *EWT* and *EWT+RM+WS* is the largest

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

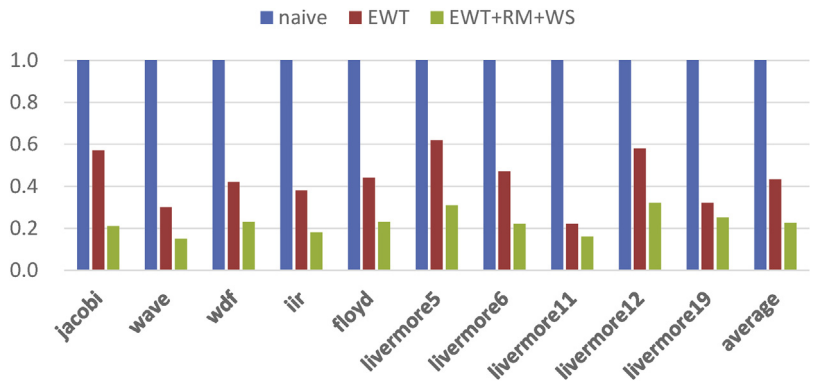

Fig. 18. Normalized dynamic energy under *EWT* and *EWT+RM+WS*.

among all the benchmarks. The reason is possibly that the proposed *EWT+RM+WS* effectively reduces unnecessary writes under WAW dependencies.

6.7. Overhead analysis

The energy overhead is mainly from the EWT circuit. Paper [17] implemented the EWT circuit and simulated them in HSPICE using 45 nm technology. The additional energy introduced by EWT circuits are measured, including pass gates, conversion circuit, muxes, sense amplifier, latch, and inverters. These components are added on per column basis. That is, all cells in one column share one set of the EWT circuit. The results show that the average energy overhead per cell per write is 89.29 fJ. Comparing to the 2.767 pJ cell write energy, the energy overhead introduced by EWT is 3.23%.

Since EWT is carried within a write operation, there is no performance overhead to the write latency. On the contrary, some write requests can even finish earlier if all the bits are the same as what are already stored in the cache which leads to a slight performance gain.

7. Conclusion

STT-RAM has the advantage of immunity to electromagnetic radiation. This paper proposed to build STT-RAM as registers for embedded systems in rad-hard environment. However, long latency and high energy on write operations affect system performance and energy consumption. EWT is an efficient technique to eliminate redundant bit writes. In this paper, impact of architecting STT-RAM as registers in pipeline is discussed where the EWT is embedded, followed by Read Merging and Write Skipping methods to further improve the performance and energy. Experiments are conducted to analyze the impact of write of STT-RAM on pipeline and evaluate the effectiveness of the proposed Read Merging and Writ Skipping methods. Results show that 70% (and 77%) and 39% (and 47%) improvements on performance (and energy) can be obtained by the proposed Read Merging and Write Skipping methods compared to the cases where STT-RAM is naively used as registers and intelligently used with EWT, respectively.

Acknowledgment

This work is supported by the grants from Beijing Advanced Innovation Center for Imaging Technology, Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), National Natural Science Foundation of China [Project No. 61502321, 61602022] and the Project of Beijing Municipal Education Commission [Project No. KM201710028016].

References

[1] A. Snchez-Macin, P. Reviriego, J.A. Maestro, Combined SEU and SEFI protection for memories using orthogonal Latin square codes, IEEE Trans. Circ. Syst. 63 (11) (2016) 1933–1943.
[2] Heather Quinn, Diane Roussel-Dupre, Mike Caffrey, Paul Graham, Michael Wirthlin, Keith Morgan, Anthony Salazar, Tony Nelson, Will Howes, Eric Johnson, Jon Johnson, Brian Pratt, Nathan Rollins, Jim Krone, The cibola flight experiment, ACM Trans. Reconfig. Technol. Syst. 8 (1) (2015 March).
[3] Chun Jason Xue, Youtao Zhang, Yiran Chen, Guangyu Sun, J. Jianhua Yang, Hai Li, Emerging non-volatile memories: opportunities and challenges, 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2011) 325–334.
[4] S. Wang, H. Lee, et al., Comparative evaluation of spin-transfer-torque and magnetoelectric random access memory, IEEE J. Emerging Selected Top. Circuits Syst. 6 (2) (2016) 134–145.
[5] G. Tsiligiannis, L. Dilillo, et al., Testing a commercial MRAM under neutron and alpha radiation in dynamic mode, IEEE Trans. Nucl. Sci. 60 (4) (2013) 2617–2622.
[6] Y. Lakys, W.S. Zhao, J.O. Klein, C. Chappert, Hardening techniques for MRAM-Based nonvolatile latches and logic, IEEE Trans. Nucl. Sci. 59 (4) (2012) 1136–1141.
[7] X. Chen, N. Khoshavi, J. Zhou, D. Huang, R.F. DeMara, J. Wang, W. Wen, Y. Chen, AOS: Adaptive overwrite scheme for energy-efficient MLC STT-RAM cache, 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC) (2016) 1–6.
[8] D. Chabi, W. Zhao, J.O. Klein, C. Chappert, Design and analysis of radiation hardened sensing circuits for spin transfer torque magnetic memory and logic, IEEE Trans. Nucl. Sci. 61 (6) (2014) 3258–3264.
[9] N. Goswami, B. Cao, T. Li, Power-performance co-optimization of throughput core architecture using resistive memory, 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA) (2013) 342–353.
[10] Jue Wang, Yuan Xie, A write-aware STTRAM-Based register file architecture for GPGPU, ACM J. Emerging Technol. Comput. Syst. 12 (1) (2015 August).
[11] Hang Zhang, Xuhao Chen, Nong Xiao, Fang Liu, Architecting energy-efficient STT-RAM based register file on GPGPUs via delta compression, 53rd Annual Design Automation Conference (DAC) (2016).
[12] http://www.mram-info.com/tags/companies/ibm.
[13] Yong Li, Yiran Chen, Alex K. Jones, A software approach for combating asymmetries of non-volatile memories, 2012 International Symposium on Low Power Electronics and Design (ISLPED) (2012) 191–196.
[14] Dmytro Apalkov, Khvalkovskiy, et al., Spin-transfer torque magnetic random access memory STT-MRAM, ACM J. Emerging Technol. Comput. Syst. 9 (2) (2013 May), 13:1-13:35.
[15] M. Hosomi, H. Yamagishi, et al., A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram, 2005 IEEE International Electron Devices Meeting (IEDM) (2005) 459–462.
[16] P. Chi, S. Li, et al., Architecture design with STT-RAM: Opportunities and challenges, 2016 Asia and South Pacific Design Automation Conference (ASP-DAC) (2016) 109–114.
[17] P. Zhou, B. Zhao, J. Yang, Y. Zhang, Energy reduction for STT-RAM using early write termination, 2009 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2009) 264–268.
[18] Y. Chen, X. Wang, H. Li, H. Liu, D.V. Dimitrov, Design margin exploration of spin-torque transfer RAM (SPRAM), 9th International Symposium on Quality Electronic Design (ISQED) (2008) 684–690.
[19] R. Canal, A. Gonzalez, J.E. Smith, Very low power pipelines using significance compression, 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (2000) 181–190.
[20] X. Lou, P.K. Meher, Y.J. Yu, Fine-grained pipelining for multiple constant multiplications, 2015 IEEE International Symposium on Circuits and Systems (ISCAS) (2015) 966–969.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001

[21] A. Prihozhy, E. Bezati, A.A.H. Ab Rahman, M. Mattavelli, Synthesis and optimization of pipelines for HW implementations of dataflow programs, IEEE Trans. Comput.-Aided Design Integrated Circuits Syst. 34 (10) (2015) 1613–1626.
[22] http://www.embedded.com/design/real-time-and-performance/4026000/the-future-of-scalable-stt-ram-as-a-universal-embedded-memory.
[23] Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, H. Li, Yiran Chen, Circuit and microarchitecture evaluation of 3d stacking magnetic RAM (MRAM) as a universal memory replacement, 2008 45th ACM/IEEE Design Automation Conference (DAC) (2008) 554–559.
[24] H. Hughes, K. Bussmann, P.J. McMarr, S.F. Cheng, R. Shull, A.P. Chen, S. Schafer, T. Mewes, A. Ong, E. Chen, M.H. Mendenhall, R.A. Reed, Radiation studies of spin-transfer torque materials and devices, IEEE Trans. Nucl. Sci. 59 (6) (2012) 3027–3033.
[25] Ing-Jer Huang, A.M. Despain, An extended classification of inter-instruction dependency and its application in automatic synthesis of pipelined processors, 26th Annual International Symposium on Microarchitecture (MICRO) (1993) 236–246.
[26] Xipeng Shen, Yutao Zhong, Chen Ding, Locality phase prediction, Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2004) 165–176.
[27] Doug Burger, Todd M. Austin, The simplescalar tool set, version 2.0, ACM SIGARCH Computer Architecture News 25 (1997) 13–25.
[28] http://www.netlib.org/benchmark/livermore/.

Please cite this article in press as: Z. Gong, et al., Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment, Sustain. Comput.: Inform. Syst. (2018), https://doi.org/10.1016/j.suscom.2018.09.001