

Technical Report no. 2008-68

The Synchronization Power of Coalesced Memory Accesses

Phuong Hoai Ha

Philippas Tsigas¹

Otto J. Anshus



Department of Computing Science
Faculty of Science
University of Tromsø
N-9037 Tromsø, Norway

Tromsø, February 2008.

¹Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden.

Technical Report in Computing Science at
University of Tromsø

Technical Report no. 2008-68
ISSN: XXXX-XXXX

Department of Computing Science
Faculty of Science
University of Tromsø
N-9037 Tromsø, Norway

Tromsø, Norway, February 2008.

Abstract

Multicore processor architectures have established themselves as the new generation of processor architectures. As part of the one core to many cores evolution, memory access mechanisms have advanced rapidly. Several new memory access mechanisms have been implemented in many modern commodity multicore processors. Memory access mechanisms, by devising how processors access the shared memory, directly influence the synchronization capability of the multicore processor. Therefore, it is crucial to investigate the synchronization power of the new memory access mechanisms.

This paper investigates the synchronization power of coalesced memory accesses, the new memory access mechanisms introduced in recent large multicore architectures like the CUDA graphics processors. We first design three memory access models to capture the fundamental features of the new memory access mechanisms. Subsequently, we prove the exact synchronization power of these models in terms of their consensus numbers. These tight results show that the coalesced memory access models can support strong synchronization capability to the threads of multicore processors, without the need of synchronization primitives other than reads and writes. In the case of the contemporary CUDA processors, our results imply that the coalesced memory access models have consensus numbers up to thirty two.

Keywords: memory access models, consensus numbers, multicore processors, interprocess synchronization.

1 Introduction

One of the fastest evolving multicore architectures is the graphics processor one. The computational power of graphics processors (GPUs) doubles every ten months, surpassing the Moore's Law for traditional microprocessors [12]. Consequently, GPUs are emerging as powerful computational co-processors for general-purpose computations. Along with their advances in computational power, their memory access mechanisms have also evolved rapidly. Several new memory access mechanisms have been implemented in current commodity graphics/media processors like the Compute Unified Device Architecture (CUDA) [2] and Cell BE architecture [1]. For instance, in CUDA, single-word write instructions can write to words of different sizes and their sizes (in bytes) are no longer restricted to be a power of two [2]. Another advanced memory access mechanism implemented in CUDA is the coalesced global memory access mechanism. The simultaneous global memory accesses by each thread of a multiprocessor during the execution of a single read/write instruction can be coalesced into a *single* aligned memory access [2]. Access coalescing takes place even if some of the threads do not actually access memory. It is well-known that memory access mechanisms, by devising how processors access the shared memory, directly influence the synchronization capability of the multicore processor. Therefore, it is crucial to investigate the synchronization power of the new memory access mechanisms.

Research on the synchronization power of memory access operations (or objects) in conventional architectures has received a great amount of attention in the literature. The synchronization power of memory access objects is conventionally determined by their consensus-solving ability, namely their consensus number [8]. The *consensus number* of an object type is either the maximum number of processes for which the consensus problem can be solved using only objects of this type and registers, or infinity if such a maximum does not exist. It has been proven that it is impossible to construct a wait-free implementation¹ of an object with consensus number n , shared by n processes, from objects with a lower consensus number [8, 9, 14, 6]. For hard real-time systems, it has been shown that any object with consensus number n is universal² for any numbers of processes running on n processors [13]. For systems that allow processes to simultaneously access several objects in one atomic operation (or multi-object operation), upper and lower bounds on the consensus number of the multi-object systems have been provided [4, 15]. Note that the aforementioned CUDA memory accesses are not such multi-objects due to the memory alignment constraint.

This paper investigates the consensus number of the new memory access mechanisms implemented in current graphics processor architectures. We first design three new memory access models to capture the fundamental features of the new memory access mechanisms. Subsequently we prove the exact synchronization power of these models in terms of their respective consensus number. These tight results show that the new memory access models can support strong synchronization capability to the threads of multicore processors, without the need of synchronization primitives other than reads and writes.

We first design a memory access model, the *svword* model, for the size-varying word access, the first of two aforementioned advanced memory access mechanisms implemented in CUDA. Unlike single-word assignments in conventional

¹A wait-free implementation of an object guarantees that every access by a non-faulty process will get a response, regardless of the execution of other processes [10].

²An object is *universal* in a system of n processes iff it has a consensus number not lower than n .

processor architectures, the new single-word assignments can write to words of size b (in bytes), where b can vary from 1 to an upper bound B and is no longer restricted to be the power of 2 [2]. By carefully choosing b for the single-word assignments, we can *partly* overlap the bytes written by two assignments, namely each assignment has some byte(s) that is not overwritten by the other overlapping assignment (cf. Figure 2(a) for an illustration). Note that words of different size must be aligned from the address base of the memory. This constraint on memory alignment prevents single-word assignments in conventional architectures from partly overlapping each other since the word-size is restricted to be a power of two. On the other hand, since the new single-word assignment can write to a byte of a *big* word (e.g. up to 16 bytes) and leave the other bytes of the word intact, the size of values to be written becomes a significant factor. The assignment can atomically write one value of size B or B values of size 1 to B consecutive memory locations. These facts have motivated us to develop the *svword* model. In this model, we present a wait-free consensus algorithm, *SVConsensus*, for three processes using only the single-word assignment. Subsequently we prove that there is no wait-free consensus algorithm for more than three processes using only the single-word assignment (cf. Section 3). In the *SVConsensus* algorithm, we also introduce a new technique to minimize the size of (proposal) values that must be written atomically. Unlike the seminal wait-free consensus algorithm using the m -word assignment by Herlihy [8], the *SVConsensus* algorithm stores proposal values in shared memory and uses only 2 bits to determine the preceding order between a pair of processes. This allows a single-word assignment to write many values atomically and handle the consensus problem for several processes.

Secondly, we design the *aiword* model for the coalesced memory accesses, the second of the aforementioned advanced memory access mechanisms. The mechanism coalesces simultaneous read/write instructions by each thread of a single-instruction multiple-data (SIMD) multiprocessor into a *single* aligned memory access even if some of the threads do not actually access memory [2]. This allows each processor/process to write to an arbitrary subset of the aligned memory units that can be written by the coalesced memory accesses. We generally model the mechanism as an aligned-inconsecutive-word access, *aiword*, in which the memory is aligned to A -unit words and a single-word assignment can write to an arbitrary non-empty subset of the A units of a word. Note that the single-*aiword* assignment is not the multiple assignment [8] due to the constraint on the memory alignment. We present a wait-free consensus algorithm for $N = \lfloor \frac{A+1}{2} \rfloor$ processes using only single-*aiword* assignment and subsequently prove that the single-*aiword* assignment have consensus number exactly $N = \lfloor \frac{A+1}{2} \rfloor$ (cf. Section 4).

Lastly, our third model, *asword*, is designed to capture the fundamental features of the combination of both the advanced memory access mechanisms described above. This model is inspired by the fact that the read/write instructions in different coalesced global memory accesses can access words of different sizes [2]. The third model is an extension of the second model *aiword* in which *aiword*'s A units are A *svwords* of the same size b . Subsequently, we prove the exact consensus number of the third model (cf. Section 5).

The contributions of this paper can be summarized as follows:

- We develop a general memory access model, the *svword* model, to capture the fundamental features of the size-varying word accesses. In this model, a single-word assignment can write to a word comprised of b *consecutive* memory units, where b can be any value between 1 and an upper bound B . We prove that the single-*svword* assignment has consensus number 3, $\forall B \geq 5$, and that consensus number 3 is also the upper bound of consensus numbers of the single-*svword* assignment $\forall B \geq 2$. We also introduce a new technique to minimize the size of (proposal) values in consensus algorithms that use only single-word assignments (cf. Section 3).
- We develop a general memory access model, the *aiword* model, to capture the fundamental features of the coalesced memory accesses. The second model is aligned-inconsecutive-word access in which the memory is aligned to A -unit words and a single-word assignment can write to an arbitrary non-empty subset of the A units of a word. We prove that the single-*aiword* assignment has consensus number exactly $N = \lfloor \frac{A+1}{2} \rfloor$ (cf. Section 4).
- We develop a general memory access model, *asword*, to capture the fundamental features of the *combination* of both the size-varying word accesses and the coalesced memory accesses. The third model is an extension of the second model *aiword* in which *aiword*'s A units are A *svwords* of the same size b , $b \in \{1, B\}$ (cf. Section 5). We prove that the consensus number of the single-*asword* assignment is exactly N , where

$$N = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B, t \in \mathbb{N}^* \\ \lfloor \frac{A+1}{2} \rfloor, & \text{if } B = tA, t \in \mathbb{N}^* \end{cases} \quad (1)$$

In the case of the contemporary CUDA processors in which $A = 16$ and $B = 4$, the consensus number of the *asword* model is thirty two.

The rest of this paper is organized as follows. Section 2 presents the three new memory access models. Sections 3, 4 and 5 prove the exact consensus numbers of the first, second and third models, respectively. Section 6 concludes this paper.

2 Models

Before describing the details of each of the three new memory access models, we present the common properties of all these three models. Like most of the previous research on the synchronization power of the conventional memory access models [8], the memory consistency model in the three new models is sequential consistency [11, 3]. Processes are completely asynchronous. The new models use the conventional 1-dimensional memory address space. The term “word” in the new models is defined generally as *aligned* memory portions. Words (of different sizes) are aligned from the address base of the memory. Words are accessed by atomic read and write operations.

The first model is the *size-varying-word* access model (*svword*) in which a single-word assignment can write to a word comprised of b *consecutive* memory units (i.e. bytes), where b can be any value between 1 and an upper bound B . This model is inspired by the CUDA graphics processor architecture in which single-word read/write operations can atomically read/write to words of different sizes (cf. types *float1*, *float2*, *float3* and *float4* in [2]). In this model, one *unit* is a *minimum* number of consecutive bytes/bits to which a single-word assignment can atomically write. The upper bound B is the maximum number of *consecutive* units to which a single-word assignment can atomically write. We denote b -*svword* to be a *svword* composed of b units and b -*svwrite* to be a b -*svword* assignment.

The second model is the *aligned-inconsecutive-word* access model (*aiword*) in which the memory is aligned to A -unit words and a single-word assignment can write to an arbitrary non-empty subset of the A units of a word. This model is inspired by the coalesced global memory accesses in the CUDA architecture [2]. In CUDA, the simultaneous global memory accesses by each thread of an SIMD multiprocessor during the execution of a single read/write instruction can be coalesced into a *single* aligned memory access. The coalescing happens even if some of the threads do not actually access memory. In the *aiword* model, the single-*aiword* assignment (or *aiwrite* for short) cannot *atomically* write to units located in different *aiwords* due to the memory alignment.

The third model is the coalesced memory access model (*asword*), an extension of the second model *aiword* in which *aiword*'s A units are A *svwords* of the same size b , $b \in [1, B]$. This model is inspired by the fact that in CUDA the read/write instructions in different coalesced global memory accesses can access words of different sizes [2]. Let $A \times b$ -*asword* be the *asword* that is composed of A *svwords* of which each is composed of b memory units. Due to the memory alignment, an $A \times b$ -*asword* assignment (or $A \times b$ -*aswrite* for short) cannot atomically write to b -*svwords* located in *different* $A \times b$ -*aswords*. Since in reality A and B are a power of 2, in this model we assume that either $B = k \cdot A$, $k \in \mathbb{N}^*$ (in the case of $B \geq A$) or $A = k \cdot B$, $k \in \mathbb{N}^*$ (in the case of $B < A$). For the sake of simplicity, we assume that $b \in \{1, B\}$ holds. Since both $A \times 1$ -*aswords* and $A \times B$ -*aswords* are aligned from the address base of the memory space, any $A \times B$ -*asword* can be aligned within B $A \times 1$ -*asword* as shown in Figure 1.

Figure 1 illustrates the *asword* model in which each dash-dotted rectangle represents a *svword* and each red/solid rectangle represents an *asword* composed of eight *svwords*. The two rows show the memory alignment corresponding to the size b of *svwords*, where b is 1 or 2. An *aswrite* operation can atomically write to some or all of the eight *svwords* of one *asword*. It can *atomically* write to 1 unit (write to one 1-*svword*) or up to 16 units (write to eight 2-*svwords*). For an 8×1 -*asword* on row $b = 1$, there are two methods to update it atomically using the *aswrite*: i) writing to all eight 1-*svwords* using one 8×1 -*aswrite* or ii) writing to four 2-*svwords* using one 8×2 -*aswrite*. However, if only one of eight units of an 8×1 -*asword* needs to be updated and the other units must remain untouched, the only possible method is to write to the 1-*svword* using 8×1 -*aswrites*. The other method, which writes to one 2-*svword* using 8×2 -*aswrites*, will have to overwrite another unit that is required to stay untouched.

Terminology This paper uses the conventional terminology from bivalency arguments [7, 8, 14]. The *configuration* of an algorithm at a moment in its execution consists of the state of every shared object and the internal state of every process. A configuration is *univalent* if all executions continuing from this configuration yield the same consensus value and *multivalent* otherwise. A configuration is *critical* if the next operation op_i by any process p_i will carry the algorithm from a *multivalent* to a *univalent* configuration. The operations op_i are called *critical operations*.

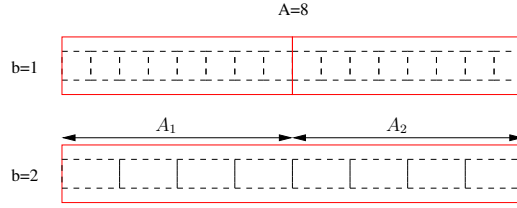


Figure 1. An illustration for the *asword* model.

3 Consensus number of the *svword* model

Before proving the consensus number of the single-*svword* assignment, we present the essential features of any wait-free consensus algorithm \mathcal{ALG} for N processes using only single-**word* assignments and registers, where **word* can be *svword*, *aiword* or *asword*. It has been proven that such a \mathcal{ALG} algorithm must have a critical configuration, C_0 , and the next assignment op_i (i.e. the critical operation) by each process p_i must write to the same object \mathcal{O} of type **word* [8]. Let *unit* be one of the components of \mathcal{O} 's state.

Lemma 1. *The critical assignment op_i by each process p_i must atomically write to*

- a “single-writer” unit (or 1W-unit for short) u_i written only by p_i and
- “two-writer” units (or 2W-unit for short) $u_{i,j}$ written only by two processes p_i and p_j , where p_j 's proposal value is different from p_i 's, $\forall j \neq i$.

Proof. The proof is similar to the bivalency argument of Theorem 13 in [8]. □

Definition 1. *The preceding process of a set of processes is the process p_i whose single-**word* assignment precedes those of the other processes p_j . In this case, we say that p_i precedes p_j .*

In this section, we first present a wait-free consensus algorithm for 3 processes using only the single-*svword* assignment with $B \geq 5$. Then, we prove that we cannot construct any wait-free consensus algorithms for more than 3 processes using only the single-*svword* assignment regardless of how large B is.

The new wait-free consensus algorithm SVW_CONSENSUS is presented in Algorithm 1. The main idea of the algorithm is to utilize the size-variation feature of the *svwrite* operation. Since the *b-svwrite* can atomically write either one value of size b units or b values of size 1 unit to b consecutive memory units, it is crucial to keep the values to be written atomically as small as possible. Unlike the seminal wait-free consensus algorithm using the *m-word* assignment by Herlihy [8], which requires the word size to be large enough to accommodate a proposal value, the new algorithm stores proposal values in shared memory and uses only two bits (or one unit) to determine the preceding order between two processes. This allows a single-*svword* assignment to write atomically up to B ordering-related values. The new algorithm utilizes process unique identifiers as in Herlihy's consensus model [5].

The SVW_CONSENSUS algorithm has two phases. In the first phase, two processes p_0 and p_1 will achieve an agreement on their proposal values (cf. Algorithm 2). The agreed value, $PROPOSAL[frist]$, is the proposal value of the preceding process (lines 6SF-11SF). Due to the memory alignment, in order to be able to find the variable WR_1 (cf. Algorithm 1) on which p_0 's and p_1 's SVWRITES can partly overlap, p_0 's and p_1 's SVWRITES are chosen as 2-*svwrite* and 3-*svwrite*, respectively. The WR_1 is located in a memory region consisting of 4 consecutive units $\{u_0, u_1, u_2, u_3\}$ of which the first two units $\{u_0, u_1\}$ can be written atomically by a 2-*svwrite* operation and the last 3 units $\{u_1, u_2, u_3\}$ by a 3-*svwrite* operation. The set WR_1 is $\{u_0, u_1, u_2\}$.

Subsequently, the agreed value will be used as the proposal value of both p_0 and p_1 in the second phase in order to achieve an agreement with the other process p_2 (cf. Algorithm 3). Let p_{frist} be the preceding process of p_0 and p_2 in the first phase. The second phase returns p_{frist} 's proposal value if either p_0 or p_1 precedes p_2 (line 9SS) and returns p_2 's proposal value otherwise.

Units written by processes' SVWRITE are illustrated in Figure 2(a). In order to be able to find the variable WR_2 , process p_0 's, p_1 's and p_2 's SVWRITES are chosen as 2-*svwrite*, 3-*svwrite* and 5-*svwrite*, respectively. The variable WR_2 is located in a memory region consisting of 7 consecutive units $\{u_0, \dots, u_7\}$ of which the first two units $\{u_0, u_1\}$ can be written

atomically by a 2-*svword*, the last three units $\{u_4, u_5, u_6\}$ by a 3-*svword* and five middle units $\{u_1, \dots, u_5\}$ by a 5-*svword*. The set WR_2 is $\{u_0, u_1, u_2, u_5, u_6\}$. Since 2, 3 and 5 are prime numbers, we always can find such a memory region. For instance, if the memory address space starts from the unit with index 0, the memory region from unit 14 to unit 20 can be used for WR_2 .

Lemma 2. *The SVW_FIRSTAGREEMENT procedure returns the index of the preceding process of p_0 and p_1 .*

Proof. Without loss of generality, we consider the value returned by the SVW_FIRSTAGREEMENT procedure that is invoked by process p_0 , i.e. $i = 0$.

If p_0 precedes p_1 , their 2W-unit $WR_1[1]$ is either *Lower* (when p_1 has not executed its SVWRITE yet) or *Higher* (when p_1 's SVWRITE has overwritten the value *Lower* written by p_0 's). In the former, $WR_1[2] = \perp$ holds (line 6SF), making the procedure return 0 (line 7SF). In the latter, predicate $(WR_1[1] = \textit{Higher} \text{ and } i = 0)$ holds, making the procedure return 0.

If p_1 precedes p_0 , their 2W-unit $WR_1[1]$ is either *Higher* (when p_0 has not executed its SVWRITE yet) or *Lower* (when p_0 's SVWRITE has overwritten the value *Higher* written by p_1 's). The former cannot happen since p_0 executes its SVWRITE at the beginning of the SVW_FIRSTAGREEMENT procedure and the procedure is assumed to be invoked by p_0 . In the latter, the procedure returns 1 (line 11SF) since both predicates at lines 6SF and 8SF fail. \square

Lemma 3. *The SVW_SECONFAGREEMENT procedure returns index 2 if p_2 precedes both p_0 and p_1 . Otherwise, it returns index *first*.*

Proof. If p_0 precedes p_2 , their 2W-unit $WR_2[1]$ is either *Lower* (when p_2 has not executed its SVWRITE yet) or *Higher* (when p_2 's SVWRITE has overwritten the value *Lower* written by p_0 's). In the former, predicate $(WR_2[0] \neq \perp \text{ and } WR_2[2] = \perp)$ holds (line 8SS), making the procedure return *first* (line 9SS). In the latter, predicate $(WR_2[0] \neq \perp \text{ and } WR_2[1] = \textit{Higher})$ holds (line 8SS), making the procedure return *first* (line 9SS).

Using similar argument, the procedure return *first* if p_1 precedes p_2 .

If p_2 precedes both p_0 and p_1 , then

- $WR_2[2] \neq \perp$ at line 8SS, and
- their 2W-unit $WR_2[1]$ is either *Higher* (when p_0 has not executed its SVWRITE yet) or *Lower* (when p_0 's SVWRITE has overwritten the value *Higher* written by p_2 's), and
- their 2W-unit $WR_2[3]$ is either *Higher* (when p_1 has not executed its SVWRITE yet) or *Lower* (when p_1 's SVWRITE has overwritten the value *Higher* written by p_2 's).

This makes the predicate at line 8SS false, causing the procedure to return 2 (line 11SS). \square

Lemma 4. *The SVW_CONSENSUS algorithm is wait-free and solves the consensus problem for 3 processes.*

Proof. It is obvious from the pseudocode in Algorithms 1, 2 and 3 that the SVW_CONSENSUS algorithm is wait-free.

From Lemmas 2 and 3, the SVW_CONSENSUS algorithm returns the same values for all invoking processes. The value is either $PROPOSAL[2]$ (if p_2 precedes both p_0 and p_1) or $PROPOSAL[\textit{first}]$, $\textit{first} \in \{0, 1\}$ (otherwise). \square

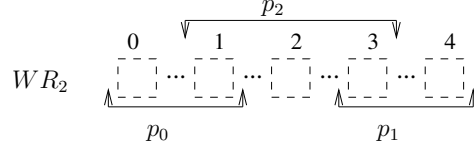
Lemma 5. *The single-*svword* assignment has consensus number at least 3, $\forall B \geq 5$.*

Proof. Since there is a wait-free consensus algorithm for 3 processes (cf. Algorithm 1) using only the single-*svword* assignment with $B \geq 5$, this lemma immediately follows. \square

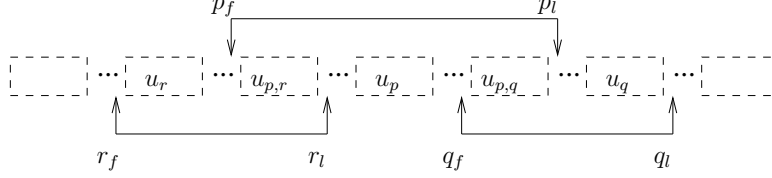
Lemma 6. *The single-*svword* assignment has consensus number at most 3, $\forall B \geq 2$.*

Proof. If $B = 2$, the single-*svword* assignment (or *svwrite* for short) becomes a *constraint* 2-unit assignment that can atomically write to 2 units only if the 2 units are *consecutive*. It has been proven that the 2-unit assignment has consensus number at most 2 [8]. Therefore, in the rest of the proof, we consider only the case of $B \geq 3$.

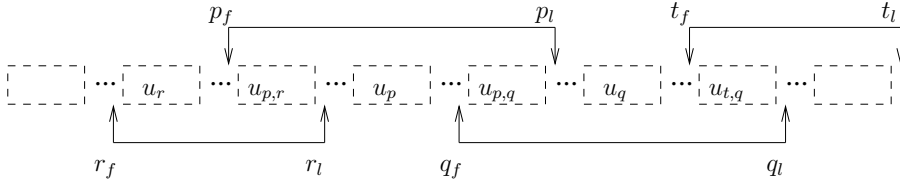
We prove the lemma by contradiction. Assume that there is a wait-free consensus algorithm \mathcal{ALG} for 4 processes p, q, r, t . At the critical configuration of the algorithm, we can always divide the set of the 4 processes into 2 non-empty subsets S and \bar{S} where S consists of at most 2 processes with the same proposal value called V and \bar{S} consists of processes with proposal values different from V . Since the *svwrite* operation writes to *consecutive* memory units in the conventional 1-*dimensional* memory address space, let $[k_f, k_l]$ be the range of consecutive units to which a process $k \in \{p, q, r, t\}$ atomically writes using its critical operation op_k (cf. Lemma 1). For any pair of processes $\{h, k\}$, where h and k belong to different subsets S and \bar{S} , $[h_f, h_l]$ and $[k_f, k_l]$ must partly overlap (due to the second requirement of Lemma 1) and none of them are completely covered by ranges $[v_f, v_l]$ of the other processes v (due to the first requirements of Lemma 1).



(a) SVW_SECONDAGREEMENT.



(b) $S = \{p\}$



(c) $S = \{p, t\}$

Figure 2. Illustrations for the SVW_SECONDAGREEMENT and the proof of Lemma 6.

- If S consists of 1 process, let $S = \{p\}$. Since p 's proposal value is different from those of the three other processes q , r and t , process p 's critical assignment must atomically write to 4 units $u_{p,q}$, $u_{p,r}$, $u_{p,t}$ and u_p (cf. Lemma 1). The atomic assignment determines the relative ordering between p and the three other processes with proposal values different from p 's. If p 's assignment precedes q 's, p is considered preceding q (cf. Definition 1).

Without loss of generality, assume $p_f < q_f \leq p_l < q_l$ where the 2W-unit $u_{p,q}$ of p and q is between q_f and p_l , $q_f \leq u_{p,q} \leq p_l$, and the 1W-units $u_p < q_f$ and $u_q > p_l$ (cf. Figure 2(b)).

We prove that $r_f < p_f \leq r_l < p_l$. Since ranges $[r_f, r_l]$ and $[p_f, p_l]$ must partly overlap, either $r_f < p_f \leq r_l < p_l$ or $p_f < r_f \leq p_l < r_l$ must hold. If the latter holds, $q_l < r_l$ must hold due to the first requirement of Lemma 1 for the process r . That means $p_f < q_f < q_l < r_l$, or q 's range $[q_f, q_l]$ is covered completely by ranges $[p_f, p_l]$ and $[r_f, r_l]$, violating the first requirement of Lemma 1 for the process q .

Arguing similarly, we have $t_f < p_f \leq t_l < p_l$. If $t_f < r_f$, r 's range $[r_f, r_l]$ is covered completely by ranges $[t_f, t_l]$ and $[p_f, p_l]$. Therefore, $r_f < t_f$ must hold, leading to t 's range $[t_f, t_l]$ covered completely by ranges $[r_f, r_l]$ and $[p_f, p_l]$, a contradiction to the first requirement of Lemma 1 for the process t .

- If S consists of 2 processes, let $S = \{p, t\}$. Since p 's and t 's proposal values are different from those of the two other processes q and r , processes p and t must atomically write to units $\{u_{p,q}, u_{p,r}, u_p\}$ and $\{u_{t,q}, u_{t,r}, u_t\}$, respectively (cf. Lemma 1). Similarly, q and r must atomically write to units $\{u_{p,q}, u_{t,q}, u_q\}$ and $\{u_{p,r}, u_{t,r}, u_r\}$, respectively.

Since p must atomically write to units $\{u_{p,q}, u_{p,r}, u_p\}$, arguing similarly to the above case $S = \{p\}$, we have either $r_f < p_f \leq r_l < q_f \leq p_l < q_l$ (cf. Figure 2(b)) or $q_f < p_f \leq q_l < r_f \leq p_l < r_l$. Without loss of generality, assume that the former holds.

Similarly, since i) q must atomically write to units $\{u_{p,q}, u_{t,q}, u_q\}$ and ii) $p_f < q_f \leq p_l < q_l$, we have $p_f < q_f \leq p_l <$

Algorithm 1 SVW_CONSENSUS(buf_i : proposal) invoked by process $p_i, i \in \{0, 1, 2\}$

$PROPOSAL[0, 1, 2]$: contains proposals of 3 processes. $PROPOSAL[i]$ is only written by process p_i but can be read by all processes.

$WR_1 = \text{set } \{u_0, u_1, u_2\}$ of *units*: initialized to *Init* and used in the first phase. $WR_1[0]$ and $WR_1[2]$ are 1W-units written only by p_0 and p_1 , respectively. $WR_1[1]$ is a 2W-unit written by both processes

$WR_2 = \text{set } \{v_0, \dots, v_4\}$ of *units*: initialized to *Init* and used in the second phase. $WR_2[0]$, $WR_2[2]$ and $WR_2[4]$ are 1W-units written only by p_0 , p_2 and p_1 , respectively. $WR_2[1]$ and $WR_2[3]$ are 2W-units written by pairs $\{p_0, p_2\}$ and $\{p_2, p_1\}$, respectively.

Input: process p_i 's proposal value, buf_i .

Output: the value upon which all 3 processes (will) agree.

1V: $PROPOSAL[i] \leftarrow buf_i$; // Declare p_i 's proposal

// **Phase I:** Achieve an agreement between p_0 and p_1 .

2V: **if** $i = 0$ or $i = 1$ **then**

3V: $first \leftarrow SVW_FIRSTAGREEMENT(i)$;

4V: **end if**

// **Phase II:** Achieve an agreement between all three processes.

5V: $winner \leftarrow SVW_SECONDAGREEMENT(i, first)$;

6V: **return** $PROPOSAL[winner]$

Algorithm 2 SVW_FIRSTAGREEMENT(i : bit) invoked by process $p_i, i \in \{0, 1\}$

Output: the preceding process of $\{p_0, p_1\}$

1SF: **if** $i = 0$ **then**

2SF: $SVWRITE(\{WR_1[0], WR_1[1]\}, \{Lower, Lower\})$; // atomically write to 2 units

3SF: **else**

4SF: $SVWRITE(\{WR_1[1], WR_1[2]\}, \{Higher, Higher\})$; // $i = 1$

5SF: **end if**

6SF: **if** $WR_1[(\neg i) * 2] = \perp$ **then**

7SF: **return** i ; // The other process hasn't written its value

8SF: **else if** $(WR_1[1] = Higher \text{ and } i = 0)$ or $(WR_1[1] = Lower \text{ and } i = 1)$ **then**

9SF: **return** i ; // The other process comes later and overwrites p_i 's value in $WR_1[1]$

10SF: **else**

11SF: **return** $(\neg i)$;

12SF: **end if**

$t_f \leq q_l < t_l$ (cf. Figure 2(c)). On the other hand, since $q_f < t_f \leq q_l < t_l$ and t must atomically write to units $\{u_{t,q}, u_{t,r}, u_t\}$, we have $q_f < t_f \leq q_l < r_f \leq t_l < r_l$. This contradicts the assumption $r_f < p_f \leq r_l < q_f \leq p_l < q_l$. □

From Lemmas 5 and 6, we have the following Theorem:

Theorem 1. *The single-sword assignment has consensus number 3 when $B \geq 5$ and three is the upper bound of consensus numbers of single-sword assignments $\forall B \geq 2$.*

Algorithm 3 SVW_SECONDAGREEMENT($i, first$: index) invoked by process $p_i, i \in \{0, 1, 2\}$

1SS: **if** $i = 0$ **then**

2SS: $SVWRITE(\{WR_2[0], WR_2[1]\}, \{Lower, Lower\})$;

3SS: **else if** $i = 1$ **then**

4SS: $SVWRITE(\{WR_2[3], WR_2[4]\}, \{Lower, Lower\})$;

5SS: **else**

6SS: $SVWRITE(\{WR_2[1], WR_2[2], WR_2[3]\}, \{Higher, Higher, Higher\})$;

7SS: **end if**

8SS: **if** $((WR_2[0] \neq \perp \text{ or } WR_2[4] \neq \perp) \text{ and } WR_2[2] = \perp)$ or $(WR_2[0] \neq \perp \text{ and } WR_2[1] = Higher)$ or $(WR_2[4] \neq \perp \text{ and } WR_2[3] = Higher)$ **then**

9SS: **return** $first$; // p_2 is preceded by either p_0 or p_1 .

10SS: **else**

11SS: **return** 2;

12SS: **end if**

4 Consensus number of the *aiword* model

In this section, we prove that the single-*aiword* assignment (or *aiwrite* for short) has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$. First, we prove that the *aiwrite* operation has consensus number at least $\lfloor \frac{A+1}{2} \rfloor$. We prove this by presenting a wait-free consensus algorithm AIW_CONSENSUS for $N = \lfloor \frac{A+1}{2} \rfloor$ processes (cf. Algorithm 4) using only *aiwrite* operations. Subsequently, we prove that there is no wait-consensus algorithm for $N + 1$ processes using only *aiwrite* operations and registers.

The main idea of the AIW_CONSENSUS algorithm is to gradually extend the set S of processes agreeing on the same value by one at a time. This is to minimize the number of 1W- and 2W-units that must be written atomically by *aiword* operations (cf. Lemma 10). The algorithm consists of N rounds and a process $p_i, i \in [1, N]$, participates from round r_i to round r_N . A process p_i leaves a round $r_j, j \geq i$, and enters the next round r_{j+1} when it reads the value upon which all processes in the round r_j (will) agree. A round r_j starts with the first process that enters the round, and ends when all j processes $p_i, 1 \leq i \leq j$, have left the round. At the end of a round r_j , the set S consists of j processes $p_i, 1 \leq i \leq j$.

Lemma 7. *All correct processes³ p_i agree on the same value in round r_j , where $1 \leq i \leq j \leq N$.*

Proof. We will prove this lemma by induction on j , the round index. The lemma is true for $j = 1$ since there is only one process p_1 in round r_1 . Assume that the lemma is true for $(j - 1)$, we need to prove that the lemma is true for j . That means we need to prove that if all correct processes $p_i, 1 \leq i \leq j - 1$, agree on the same value in round r_{j-1} , then all correct processes $p_i, 1 \leq i \leq j$, will agree on the same value in round r_j .

Indeed, since all correct processes $p_i, 1 \leq i \leq j - 1$, agree on the same value in round r_{j-1} , their proposal values in round r_i are the same (line 10I) called A_S^j . Let A_j^j be p_j 's proposal value in round j , its original proposal value (line 1I). The agreed value in round r_i will be either A_S^j or A_j^j . At this moment, we assume that AIWRITE can atomically write to p_j 's units at line 2I and p_i 's units at line 1I. The condition for the assumption to be true is presented in Lemma 8. We will prove that the agreed value in round r_i will be A_j^j if p_j precedes *all* the other processes $p_i, 1 \leq i < j$, and A_S^j otherwise.

- If p_j precedes *all* the other processes $p_i, 1 \leq i < j$, all processes will see $U_j^j \neq \perp$ after their AIWRITE (line 2I for p_j and line 1I for $p_i, i < j$). Let $p_l, l \in [1, j]$ be the process that is executing the AIW_CONSENSUS procedure.

If $l = j$, p_j determines its relative ordering with processes $p_i, i < j$ using the 2W-unit $U_{j,i}^j$, which is only written by p_j 's and p_i 's AIWRITES (lines 3I-8I). Since p_j precedes all processes p_i , predicate $U_{j,i}^j = Higher$ holds only if p_i has not executed its AIWRITE yet, which will overwrite the value *Higher* written by p_j with the value *Lower*. This leads to $U_i^j = \perp$, making predicate $(U_i^j \neq \perp \text{ and } U_{j,i}^j = Higher)$ at line 4I false, $\forall i < j$. Consequently, p_j keeps its proposal value A_j^j as the agreed value.

If $l = i, i \neq j$, process p_i determines its relative ordering with process p_j using their 2W-unit $U_{j,i}^j$ (line 12I). Since p_j precedes all p_i , predicate $(U_{j,i}^j \neq \perp \text{ and } U_{j,i}^j = Lower)$ holds (line 12I) but predicate $(U_i^j \neq \perp \text{ and } U_{j,i}^j = Higher)$ does not hold (line 15I). This makes p_i agree on A_j^j (line 21I).

- If there is a process $p_I, I < j$, that precedes p_j , predicate $(U_I^j \neq \perp \text{ and } U_{j,I}^j = Higher)$ at line 4I holds, making p_j agree on A_S^j (line 5I). For a process $p_i, i < j$, that is executing the AIW_CONSENSUS procedure, if it is preceded by p_j , it will find the predicate $(U_I^j \neq \perp \text{ and } U_{j,I}^j = Higher)$ at line 15I held and consequently keep its proposal value A_S^j as the agreed value.

□

With the assumption that AIWRITE can atomically write to p_j 's units at line 2I and p_i 's units at line 1I, it follows directly from Lemma 7 that all the N processes will achieve an agreement in round r_N .

Lemma 8. *The AIW_CONSENSUS algorithm is wait-free and can solve the consensus problem for $N = \lfloor \frac{A+1}{2} \rfloor$ processes.*

Proof. The time complexity for a process using AIW_CONSENSUS to achieve an agreement among N processes is $O(N^2)$ due to the for-loops at lines 9I and 14I. Therefore, the AIW_CONSENSUS algorithm is wait-free.

From Lemma 7, the AIW_CONSENSUS algorithm can solve the consensus problem for $N = \lfloor \frac{A+1}{2} \rfloor$ processes if i) p_j 's AIWRITE can atomically write to j units $\{U_j^j, U_{j,1}^j, \dots, U_{j,j-1}^j\}$, where $1 \leq j \leq N$ (line 2I), ii) p_i 's AIWRITE can

³A correct process is the process that does not crash.

Algorithm 4 AIW_CONSENSUS(buf_i : proposal) invoked by process $p_i, i \in [1, N]$

$A^r[i]$: p_i 's agreed value in round r ;
 $U_{i,j}^r$: the 2W-unit of processes p_i and p_j in round r . U_i^r : the 1W-unit of process p_i in round r (cf. Lemma 1);
Input: process p_i 's proposal value, buf_i .
Output: the value upon which all N processes (will) agree.

```

//  $p_i$  starts from round  $i$ 
1I:  $A^i[i] \leftarrow buf_i$ ; // Initialized  $p_i$ 's agreed value for round  $i$ 
2I: AIWRITE( $\{U_i^i, U_{i,1}^i, \dots, U_{i,i-1}^i\}, \{Higher, Higher, \dots, Higher\}$ ) // Atomic assignment
3I: for  $k = 1$  to  $(i - 1)$  do
4I:   if  $U_k^i \neq \perp$  and  $U_{i,k}^i = Higher$  then
5I:      $A^i[i] \leftarrow A^i[k]$ ; // Update  $p_i$ 's agreed value to the set  $S$ 's agreed value
6I:     break;
7I:   end if
8I: end for
// Participate rounds from  $(i + 1)$  to  $N$ 
9I: for  $j = i + 1$  to  $N$  do
10I:   $A^j[i] \leftarrow A^{j-1}[i]$ ; // Initialized  $p_i$ 's agreed value for round  $j$ 
11I:  AIWRITE( $\{U_i^j, U_{j,i}^j\}, \{Lower, Lower\}$ ) // Atomic assignment
12I:  if  $U_j^j \neq \perp$  and  $U_{j,i}^j = Lower$  then
13I:     $WinnerIsJ \leftarrow \text{true}$ ; // Check if  $p_j$  precedes  $p_k, \forall k < j$ .
14I:    for  $k = 1$  to  $j - 1$  do
15I:      if  $U_k^j \neq \perp$  and  $U_{j,k}^j = Higher$  then
16I:         $WinnerIsJ \leftarrow \text{false}$ ; //  $p_k$  precedes  $p_j$ ;
17I:        break;
18I:      end if
19I:    end for
20I:    if  $WinnerIsJ = \text{true}$  then
21I:       $A^j[i] \leftarrow A^j[j]$ ; //  $p_j$  precedes  $p_k, \forall k < j, \Rightarrow p_j$ 's value is the agreed value in round  $j$ .
22I:    end if
23I:  end if
24I: end for
25I: return  $A^N[i]$ ;
  
```

atomically write to 2 units $\{U_i^j, U_{j,i}^j\}$, where $1 \leq i < j$. Since AIWRITE can write to an *arbitrary* subsets of A units of an *aiword* AI , if AIWRITE can atomically write to a units of AI , $a \leq A$, it can atomically write to b units of AI where $b \leq a$. Therefore, we only need to prove that these two requirements are satisfied for the case $j = N$.

Since $N = \lfloor \frac{A+1}{2} \rfloor$, an A -unit *aiword* (or A -*aiword* for short) can accommodate both $(N - 1)$ 2W-units $U_{N,i}^N, 1 \leq i < N$, and N 1W-units $U_k^N, 1 \leq k \leq N$, used in round r_N . Figure 3 illustrates the 2-dimensional layout of the $(2N - 1)$ units to be mapped on A units of an *aiword*. Since the single-*aiword* assignment AIWRITE can atomically write to an arbitrary subset of the A units of an *aiword* and leave the other units untouched, each process $p_k, 1 \leq k \leq N$ can atomically write to *only* its 1W and 2W units. This guarantees that a 1W-unit U_i^N is written only by p_i and a 2W-unit $U_{N,i}^N$ is written only p_N and p_i (cf. Lemma 1). \square

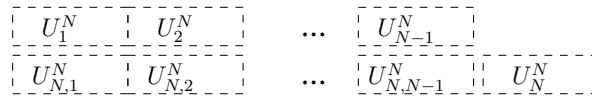


Figure 3. The layout of units U_i^N and $U_{N,i}^N$ on a A -*aiword*.

Lemma 9. The single-*aiword* assignment has consensus number at least $\lfloor \frac{A+1}{2} \rfloor$.

Proof. Since there is a wait-free consensus algorithm for $N = \lfloor \frac{A+1}{2} \rfloor$ processes (cf. Lemma 8) using only the single-*aiword* assignment, this lemma immediately follows. \square

Lemma 10. The single-*aiword* assignment has consensus number at most $\lfloor \frac{A+1}{2} \rfloor$.

Proof. We prove this lemma by contradiction. Assume that there is a wait-free consensus algorithm \mathcal{ALG} for N processes where $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$. At the critical configuration of the \mathcal{ALG} algorithm, we divide N processes into $k \geq 2$ subsets s_1, \dots, s_k which each consists of processes with the same proposal value. Let n_1, \dots, n_k to be the size of the subsets, we have $\sum_{l=1}^k n_l = N$. Let p_j^i be a process in s_i , $1 \leq j \leq n_i$. Since p_j^i 's proposal value is different from that of $(\sum_{l \neq i} n_l)$ processes in the other $(k-1)$ subsets, p_j^i 's critical assignment must atomically write to its 1W-unit and $(\sum_{l \neq i} n_l)$ 2W-units (cf. Lemma 1). The assignment determines the relative ordering between p_j^i and the $(\sum_{l \neq i} n_l)$ other processes with proposal values different from p_j^i 's.

First, we prove that for any pair of processes in different subsets p_j^i and $p_{j'}^{i'}$, $i \neq i'$, their 1W-units $u_j^i, u_{j'}^{i'}$, and 2W-unit $u_{j,j'}^{i,i'}$ must be located in the same *A-aiword*. Indeed, since p_j^i and $p_{j'}^{i'}$ belong to different subsets, they have different proposal values. Therefore, p_j^i (resp. $p_{j'}^{i'}$) must *atomically* write to its 1W-/2W-units $\{u_j^i, u_{j,j'}^{i,i'}\}$ (resp. $\{u_{j'}^{i'}, u_{j,j'}^{i,i'}\}$). Since the *aiwrite* operation cannot atomically write to units located in different *aiwords* due to the memory alignment, units u_j^i and $u_{j,j'}^{i,i'}$ must be located in the same *aiword* due to p_j^i . Similarly, units $u_{j'}^{i'}$ and $u_{j,j'}^{i,i'}$ must be located in the same *aiword* due to $p_{j'}^{i'}$. It follows that all three units $u_j^i, u_{j,j'}^{i,i'}$, and $u_{j'}^{i'}$ must be located in the same *aiword*.

Therefore, that all the 1W-units $u_j^i, 1 \leq i \leq k, 1 \leq j \leq n_i$, and 2W-units $u_{j,j'}^{i,i'}, i \neq i'$, used in the \mathcal{ALG} algorithm to solve the consensus problem for the N processes must be located in the same *A-aiword* called *AI*. Let M be the number of 1W-/2W-units must be located in the *A-aiword AI*, we have $M \leq A$.

Second, we prove that the \mathcal{ALG} algorithm maximizes N when k is 2, the minimum. In order to maximize the number N of processes, we need to minimize the number M of the processes' 1W-/2W-units that must be located in the *A-aiword AI*, where A is a constant. The number M in the \mathcal{ALG} algorithm is:

$$\begin{aligned} M &= N + n_1(n_2 + \dots + n_k) + n_2(n_3 + \dots + n_k) + \dots + n_{k-1}n_k \quad /*N: the number of 1W-units*/ \\ &\geq N + n_1(n_i + \dots + n_k) + n_2(n_i + \dots + n_k) + \dots + n_{i-1}(n_i + \dots + n_k), \text{ where } 2 \leq i \leq k \\ &= N + (n_1 + n_2 + \dots + n_{i-1})(n_i + \dots + n_k) \end{aligned} \quad (2)$$

That means M will be less if there are only two subsets s_I, s_{II} of processes with the same proposal value, where $n_I = \sum_{l=1}^{i-1} n_l$ and $n_{II} = \sum_{l=i}^k n_l$. In this case, we have

$$M = N + n_I.n_{II} = N + n_I(N - n_I) = -n_I^2 + N.n_I + N, \text{ where } 1 \leq n_I \leq N - 1$$

It follows that M achieves the minimum $(2N - 1)$ when $n_I = 1$ or $n_I = N - 1$.

Since $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$ due to the hypothesis, $M \geq (A + 1)$ must hold. This contradicts the requirement $M \leq A$. \square

From Lemmas 9 and 10, we have the following theorem

Theorem 2. *The single-aiword assignment has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$.*

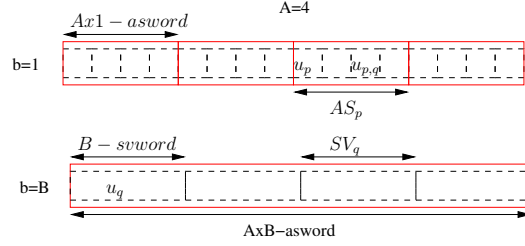
5 Consensus number of the *asword* model

We will prove that the single-*asword* assignment has consensus number exactly N , where

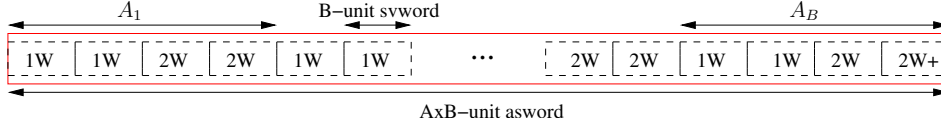
$$N = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B, t \in \mathbb{N}^* \\ \lfloor \frac{A+1}{2} \rfloor, & \text{if } B = tA, t \in \mathbb{N}^* \end{cases} \quad (3)$$

First, we prove that the *aswrite* operation has consensus number at least N . We prove this by presenting a wait-free consensus algorithm *ASW_CONSENSUS* for N processes using only *aswrite* operations. Subsequently, we prove that there is no wait-free consensus algorithm for $N + 1$ processes using only *aswrite* operations and registers.

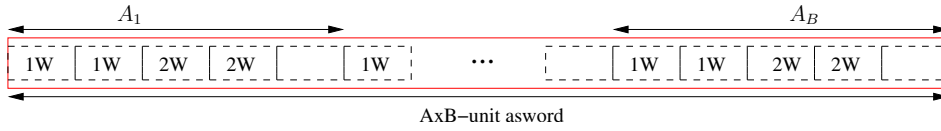
Lemma 11. *The single-*asword* assignment has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$ for $B = tA, t \in \mathbb{N}^*$.*



(a) $A = B$.



(b) $A = 2tB$



(c) $A = (2t + 1)B$

Figure 4. Illustrations for the proof of Lemmas 11 and 12.

Proof. Since the *asword* model is an extension of the *aiword* model, the *aswrite* operation has consensus number at least $N = \lfloor \frac{A+1}{2} \rfloor$ (cf. Theorem 2). When the size b of *svwords* is the same for all *aswrites*, the *asword* model degenerates to the *aiword* model. The *aswrite* operation can achieve a higher consensus number when the size b of *svwords* is allowed to be different between *aswrites*, namely utilizing both *Ax1-aswrites* and *AxB-aswrites*.

However, we will prove that when $B = tA$, the combination of *Ax1-aswrites* and *AxB-aswrites* does not provide any additional strength. Indeed, assume that there are two processes p and q that use the *Ax1-aswrite* and *AxB-aswrite* to write to their $2W$ -unit $u_{p,q}$ (cf. Lemma 1), respectively. Since p must atomically write to its $1W$ -unit u_p and $2W$ -unit $u_{p,q}$ using *Ax1-aswrites*, the two units (or two 1 -*svwords*) must be located in the same *Ax1-asword* called AS_p (cf. Figure 4(a)). Since *AxB-aswrite* uses B -*svwords* as its working units, q whose write-operation is *AxB-aswrite* must write to the B -*svword* called SV_q that overlaps the 1 -*svword* $u_{p,q}$. Due to the memory alignment and $B = tA$, the SV_q overlaps the whole AS_p . That means q overwrites the whole AS_p including p 's $1W$ -unit u_p , a contradiction to the first requirement of Lemma 1 for process p . \square

Lemma 12. *The single-asword assignment has consensus number at least*

$$m = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t + 1)B, t \in \mathbb{N}^* \end{cases} \quad (4)$$

Proof. We prove this lemma by presenting a wait-free consensus algorithm $ASW_CONSENSUS$ for m processes using only single-*asword* assignments $ASWRITE$. The $ASW_CONSENSUS$ algorithm is similar to the $AIW_CONSENSUS$ algorithm (Algorithm 4) except that the $AIWRITE$ operations used at lines 2I and 11I are replaced by the $ASWRITE$ operations.

Similar to the proof of Lemma 8, we will prove that: i) p_m 's $ASWRITE$ can atomically write to m units $\{U_m^m, U_{m,1}^m, \dots, U_{m,m-1}^m\}$ (line 2I), ii) p_i 's $ASWRITE$ can atomically write to 2 units $\{U_i^m, U_{m,i}^m\}$, where $1 \leq i < m$, and iii) an $1W$ -unit U_i^m is written only by p_i and a $2W$ -unit $U_{m,i}^m$ is written only p_m and p_i (cf. Lemma 1). We will present an assignment of the units on an *AxB-asword* that satisfies all the three requirements. Figures 4(b) and 4(c) illustrate this proof.

- $A = 2tB$: Due to the memory alignment, an AxB -asword AS is always aligned with B $Ax1$ -aswords called A_1, \dots, A_B , of which each can be atomically written by an $Ax1$ -aswrite (cf. Figure 4(b)).

The 1W-unit U_i^m and 2W-unit $U_{m,i}^m$ of each process $p_i, i \neq m$, are located in the the same $A_l, 1 \leq l \leq B$, so that p_i can atomically write to *only* its two units using an $Ax1$ -aswrite. This makes the assignment satisfy the requirements ii) and iii) for p_i . Each B -svword located in A_l contains either 1W-units or 2W-units but not both, which allows p_m to modify only B -svwords with 2W-units $U_{m,i}^m$ and keep 1W-units U_i^m untouched by using one AxB -aswrite. This makes the assignment satisfy the requirement iii) for p_m . The distribution of 1W-units and 2W-units on A_l is shown in Figure 4(b), where t B -svwords labeled “1W” contains only 1W-units and t B -svwords labeled “2W” contains only 2W-units. Since each process p_i requires one 1W-unit U_i^m and one 2W-unit $U_{m,i}^m$, the number of processes that A_l can support is $n_l = tB$. Consequently, the number of 2W-units in A_l that can be written atomically by p_m 's ASWRITE is $n_l = tB$.

The last svword labeled “2W+” in Figure 4(b) contains also p_m 's 1W-unit U_m^m , which, together with 2W-units $U_{m,i}^m$, will be written atomically by p_m 's AxB -aswrite. Therefore, the number of 2W-units $U_{m,i}^m$ located in A_B is $n_B = tB - 1$. The total number of 2W-units to which p_m can write atomically together with its 1W-unit U_m^m is

$$s = \sum_{l=1}^{B-1} tB + (tB - 1) = tBB - 1 = \frac{AB}{2} - 1 = m - 1$$

That means p_m can atomically write to its 1W-unit U_m^m and $(m - 1)$ 2W-units $U_{m,i}^m, 1 \leq i \leq (m - 1)$, using ASWRITE. This makes the assignment satisfy the requirement i) for p_m .

- $A = (2t + 1)B$: Similarly, for each A_l with $(2t + 1)$ B -svwords, t B -svwords labeled “1W” contains only 1W-units, t B -svwords labeled “2W” contains only 2W-units and the other B -svword without label is not used (cf. Figure 4(c)). This makes the assignment satisfy the requirements ii) and iii). Therefore, the number of processes that A_l can support is $n_l = tB$. It follows that the number of 2W-units in A_l that can be written atomically by p_m 's ASWRITE is $n_l = tB$.

Unlike in the case of $A = 2tB$, in this case p_m 's 1W-unit U_m^m can be located in the unused B -svword of A_B and thus the number of 2W-units $U_{m,i}^m$ located in A_B is $n_B = tB$, the same as those in other $A_l, 1 \leq l \leq B$. The total number of 2W-units to which p_m can write atomically together with its 1W-unit U_m^m is

$$s = \sum_{l=1}^B tB = tBB = \frac{(A - B)B}{2} = m - 1$$

That means p_m can atomically write to its 1W-unit U_m^m and $(m - 1)$ 2W-units $U_{m,i}^m, 1 \leq i \leq (m - 1)$, using an ASWRITE. This makes the assignment satisfy the requirement i) for p_m . □

Lemma 13. *The single-asword assignment has consensus number at most*

$$M = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t + 1)B, t \in \mathbb{N}^* \end{cases} \quad (5)$$

Proof. We prove this lemma by contradiction. Assume that there is a wait-free consensus algorithm \mathcal{ALG} for N processes where $N \geq M$. At the critical configuration of the \mathcal{ALG} algorithm, we divide N processes into $k \geq 2$ subsets s_1, \dots, s_k which each consists of processes with the same proposal value. Let n_1, \dots, n_k to be the size of the subsets, we have $\sum_{l=1}^k n_l = N$. Let p_j^i be a process in $s_i, 1 \leq j \leq n_i$. Since p_j^i 's proposal value is different from that of $(\sum_{l \neq i} n_l)$ processes in the other $(k - 1)$ subsets, p_j^i 's critical assignment must atomically write to its 1W-unit and $\sum_{l \neq i} n_l$ 2W-units (cf. Lemma 1). The assignment determines the relative ordering between p_j^i and the $\sum_{l \neq i} n_l$ other processes with proposal values different from p_j^i 's.

Since N is larger than $\lfloor \frac{A+1}{2} \rfloor$, the consensus number of single-aiword assignments (or A -aiwrites), processes in the \mathcal{ALG} algorithm must use both AxB -aswrites and $Ax1$ -aswrites. Note that if all processes use only either AxB -aswrite or $Ax1$ -aswrite, the asword model degenerates into the aiword model. Let p_b^a and $p_{b'}^{a'}, a \neq a'$, be the processes that use an AxB -aswrite and an $Ax1$ -aswrite to modify their 2W-unit $u_{b,b'}^{a,a'}$, respectively. Let AS be the AxB -asword written by p_b^a 's AxB -aswrite. AS contains p_b^a 's 1W-unit u_b^a .

First, we will prove that for any pair of processes in different subsets p_j^i and $p_{j'}^{i'}$, $i \neq i'$, if p_j^i 's 1W-unit u_j^i is located in AS , then their 2W-unit $u_{j,j'}^{i,i'}$ and $p_{j'}^{i'}$'s 1W-unit $u_{j'}^{i'}$ must be located in AS . Indeed, since the 1W-unit u_j^i is located in AS , there is one of AS 's B Ax1-*aswords* that contains this unit. Let A_c be this Ax1-*asword*. Since p_j^i and $p_{j'}^{i'}$ belong to different subsets, they have different proposal values. Therefore, p_j^i (resp. $p_{j'}^{i'}$) must *atomically* write to its 1W-/2W-units $\{u_j^i, u_{j,j'}^{i,i'}\}$ (resp. $\{u_{j'}^{i'}, u_{j,j'}^{i,i'}\}$). If p_j^i uses AxB -*aswrite*, its 2W-unit $u_{j,j'}^{i,i'}$ must be located in AS since AxB -*aswrite* cannot atomically write to two units belonging to two different AxB -*aswords*. If p_j^i uses $Ax1$ -*aswrite*, its 2W-unit $u_{j,j'}^{i,i'}$ must be located in $A_c \in AS$ since $Ax1$ -*aswrite* cannot atomically write to two units belonging to two different Ax1-*aswords*. Therefore, their 2W-unit must be located in AS . Since $p_{j'}^{i'}$ must atomically write to both the 2W-unit $u_{j,j'}^{i,i'}$ and its 1W-unit $u_{j'}^{i'}$, using a similar argument we can conclude that its 1W-unit $u_{j'}^{i'}$ must be located in AS .

From the above, it follows that all 1W-units u_j^i , $1 \leq i \leq k$, $1 \leq j \leq n_i$, and 2W-units $u_{j,j'}^{i,i'}$, $i \neq i'$ must be located in AS . Namely, since p_b^a 's 1W-unit u_b^a is located in AS , all $\sum_{l \neq a} n_l$ processes p_j^l in the other $(k-1)$ subsets must have their 1W-unit u_j^l located in AS . Similarly, since p_j^l 's 1W-unit u_j^l is located in AS , where $l \neq a$, all n_a processes of s_a must have their 1W-unit u_j^a located in AS . That means all processes must have their 1W-unit located in AS . This infers that all 2W-units used by a pair of processes in different subsets must be located in AS .

Second, we prove that the \mathcal{ALG} algorithm maximizes N when k is 2, the minimum. Since all the 1W-units and 2W-units of N processes must be located in AS of a fixed size, in order to maximize N we need to minimize the number M of the 1W- and 2W-units used by the N processes. Using similar argument to the proof of Lemma 10, it follows that M achieves the minimum $(2N-1)$ when there are only two subsets: one containing $(N-1)$ processes p with the same proposal value and the other containing only 1 process q .

Lastly, we prove that N cannot be larger than M as defined in Equation 5.

If q uses an $Ax1$ -*aswrite*, let A_q be the $Ax1$ -*asword* written by q . A_q contains p 's 1W-unit u_q and all $(N-1)$ 2W-units $u_{p,q}$. We prove that the number of A_q 's 1-*svwords* required by a process p is at least 2. Indeed, if p uses $Ax1$ -*aswrite*, both its 1W-unit u_p and 2W-unit $u_{p,q}$ must be located in A_q since $Ax1$ -*aswrites* cannot atomically write to two 1-*svwords* located in different $Ax1$ -*aswords*. Therefore, p requires two 1-*svwords* of A_q . If p uses an AxB -*aswrite*, its 2W-unit $u_{p,q}$ must be B -*svword* so that p 's AxB -*aswrite* does not overwrite other 1W-units nor 2W-units that belong to other processes. Since $B \geq 2$, p 's 2W-unit $u_{p,q}$ requires at least two 1-*svwords* of A_q . That means the number of A_q 's 1-*svwords* required by N processes including q is at least $2(N-1) + 1 = 2N-1$. Since the $Ax1$ -*asword* A_q has A 1-*svwords*, it follows that $N = \lfloor \frac{A+1}{2} \rfloor$, a contradiction to the assumption that $N > M$.

If q uses an AxB -*aswrite*, let AS be the AxB -*asword* written by q . Due to the memory alignment, the AxB -*asword* AS is aligned with B $Ax1$ -*aswords* called A_l , $1 \leq l \leq B$ (cf. Figure 4(b)). It follows from the above argument that all N 1W-units u_q, u_p and $(N-1)$ 2W-units $u_{p,q}$ must be located in AS of a fixed size. In order to maximize N , these 1W-units and 2W-units must be 1-*svwords* (instead of B -*svwords*). This infers that all $(N-1)$ processes $p \neq q$ must use $Ax1$ -*aswrite*. Each process p must have its 1W-unit u_p and 2W-unit $u_{p,q}$ located in the same $Ax1$ -*asword* in order to be able to write to them atomically.

- If $A = 2tB$, the maximum number of processes $p, p \neq q$ that an $Ax1$ -*asword* A_l can accommodate their 1W- and 2W-units is $n_l = tB$. For the $Ax1$ -*asword* A_q that contains q 's 1W-unit u_q , $n_q = \lfloor \frac{2tB-1}{2} \rfloor = tB-1$. Therefore, the maximum number of processes (including q) that the AxB -*asword* AS can support is $N = (B-1)tB + (tB-1) + 1 = tBB = \frac{AB}{2} = M$, a contradiction to the assumption that $N > M$.
- If $A = (2t+1)B$, we prove that each $Ax1$ -*asword* A_l , $1 \leq l \leq B$, cannot accommodate more than tB processes $p, p \neq q$, by contradiction. Assume that there is an A_l that can accommodate $(tB+1)$ processes p . Since each process p has one 1W-unit and one 2W-unit, A_l contains $(tB+1)$ 1W-units and $(tB+1)$ 2W-units. This infers that there is at least one B -*svword* that contains *both* 1W-units $u_r, r \neq q$, and 2W-units $u_{r',q}$, where r' can be r (cf. Figure 4(c)). Since q writes to the 2W-units $u_{r',q}$ using an AxB -*aswrite*, which uses B -*svwords* as its working units, q will overwrite $u_r, r \neq q$, a contradiction to the first requirement of Lemma 1 for process r .

Therefore, the maximum number of processes (including q) that the AxB -*asword* AS can support is $N = tBB + 1 = \frac{(A-B)B}{2} + 1 = M$, a contradiction to the assumption that $N > M$. □

6 Conclusions

This paper has introduced three new memory access models to capture the fundamental features of new memory access mechanisms implemented in current graphics processor architectures. The first model is the size-varying word access model (*svword*) in which a single-word assignment can write to a word comprised of b consecutive memory units, where b can be any value between 1 and an upper bound B . The second model is the coalesced memory access model (*aiword*) in which the memory is aligned to A -unit words and a single-word assignment can write to an arbitrary non-empty subset of the A units of a word. The third model is the combination of both the size-varying word access and coalesced memory access. It is an extension of the second model in which *aiword*'s A units are A *svwords* of the same size b , $b \in \{1, B\}$.

After introducing these new models, this paper has proven the exact synchronization power of all these models in terms of their consensus number. More precisely, it has shown that the single-*svword* assignment has consensus number exactly 3, $\forall B \geq 5$, the single-*aiword* assignment has consensus number exactly $N = \lfloor \frac{A+1}{2} \rfloor$ and the single-*asword* assignment has consensus number exactly N , where

$$N = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B, t \in \mathbb{N}^* \\ \lfloor \frac{A+1}{2} \rfloor, & \text{if } B = tA, t \in \mathbb{N}^* \end{cases} \quad (6)$$

The results presented in this paper provide a starting point to investigate more comprehensively the synchronization capability of new memory access mechanisms that are advancing rapidly as part of the one core to many cores processor evolution. The results show that the coalesced memory access models, which have been implemented in the CUDA graphics processors, can provide strong synchronization capability to the threads of multicore processors without the need of synchronization primitives other than reads and writes. In the case of the current CUDA processors, the results imply that the coalesced memory access models have consensus numbers up to thirty two.

References

- [1] *Cell Broadband Engine Architecture, version 1.01*. IBM, Sony and Toshiba Corporations, 2006.
- [2] *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, version 1.1*. NVIDIA Corporation, 2007.
- [3] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 29(12):66–76, 1996.
- [4] Y. Afek, M. Merritt, and G. Taubenfeld. The power of multi-objects (extended abstract). In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 213–222, 1996.
- [5] H. Buhrman, A. Panconesi, R. Silvestri, and P. Vitanyi. On the importance of having an identity or, is consensus really universal? *Distrib. Comput.*, 18(3):167–176, 2006.
- [6] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg. Generalized irreducibility of consensus and the equivalence of t -resilient and wait-free implementations of consensus. *SIAM Journal on Computing*, 34(2):333–357, 2005.
- [7] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
- [8] M. Herlihy. Wait-free synchronization. *ACM Transaction on Programming and Systems*, 11(1):124–149, Jan. 1991.
- [9] P. Jayanti and S. Toueg. Some results on the impossibility, universality, and decidability of consensus. In *WDAG '92: Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 69–84, 1992.
- [10] L. Lamport. Concurrent reading and writing. *Commun. ACM*, 20(11):806–811, 1977.
- [11] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess program. *IEEE Trans. Comput.*, 28(9):690–691, 1979.
- [12] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.
- [13] S. Ramamurthy, M. Moir, and J. H. Anderson. Real-time object sharing with minimal system support. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, pages 233–242, 1996.
- [14] E. Ruppert. Determining consensus numbers. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, pages 93–99, 1997.
- [15] E. Ruppert. Consensus numbers of multi-objects. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, pages 211–217, 1998.