The Synchronization Power of Coalesced Memory Accesses

Phuong Hoai Ha, Philippas Tsigas, and Otto J. Anshus

***Abstract*—Multicore architectures have established themselves as the new generation of computer 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 architectures. By specifying how processing cores access shared memory, memory access mechanisms directly influence the synchronization capabilities of multicore architectures. Therefore, it is crucial to investigate the synchronization power of these new memory access mechanisms.**

This paper investigates the synchronization power of coalesced memory accesses, a family of memory access mechanisms introduced in recent large multicore architectures such as the Compute Unified Device Architecture (CUDA). We first define 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 mechanisms can facilitate strong synchronization between the threads of multicore architectures, 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 mechanisms have consensus numbers up to sixty four.

***Index Terms*—Memory access models, consensus, multicore architectures, inter-process synchronization.**

I. Introduction

One of the fastest evolving multicore architectures is the graphics processor. The computational power of graphics processors (GPUs) doubles every ten months, surpassing Moore's Law for traditional microprocessors [2]. Unlike previous GPU architectures, which are single-instruction multiple-data (SIMD), recent GPU architectures (e.g. Compute Unified Device Architecture (CUDA) [3]) are single-program multiple-data (SPMD). The latter consists of multiple SIMD multiprocessors of which each, at the same time, can execute a different instruction. This extends the set of applications on GPUs, which are no longer restricted to follow the SIMD-programming model. Consequently, GPUs are emerging as powerful computational co-processors for general-purpose computations.

Along with their advances in computational power, GPUs memory access mechanisms have also evolved rapidly. Several new memory access mechanisms have been implemented in current commodity graphics/media processors such as CUDA [3] and Cell BE architecture [4]. For instance, in CUDA, single-word write instructions can write to words of different size and their size (in bytes) is no longer restricted to be a power of two [3]. Another advanced memory access mechanism implemented in CUDA is the coalesced global memory access mechanism. The simultaneous global memory accesses by threads of a SIMD multiprocessor, during the execution of a single read/write instruction, are coalesced into a *single* aligned memory access if the simultaneous accesses follow the coalescence constraint [3]. It is well-known that by specifying how processing cores access shared memory, memory access mechanisms directly influence the synchronization capabilities of multicore architectures. 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/mechanisms is conventionally determined by their consensus-solving ability, namely their consensus number [5], [6]. The *consensus number* of an object type is either the maximum number of processes for which wait-free consensus can be solved using only objects of this type and registers, or infinity if such a maximum does not exist. An object is *universal* in a system of n processes if and only if it has consensus number n or higher. For hard real-time systems, it has been shown that any object with consensus number n is universal for an arbitrary number of processes running on n processors [7]. For systems that allow processes to simultaneously access m objects of type T in one atomic operation (or multi-object operation), where T has a *consensus number at least two*, upper and lower bounds on the consensus number of the multi-object type called T^m have been provided [8], [9], [10]. In the case of registers (which have consensus number one), the m-register assignment, which allows processes to write to m *arbitrary* registers atomically, has been proven to have consensus number $(2m-2)$, for $m > 1$ [5]. Using the m-register assignment, we can construct $(2m-3)$-resilient read-modify-write objects [11]. An object implementation is *t-resilient* if non-faulty processes can complete their operations on the object as long as no more than t processes fail [12], [13].

Note that the aforementioned CUDA coalesced memory accesses are neither the atomic m-register assignment [5] nor the multi-object types [8], [9], [10]. They are not the atomic m-register assignment since they do not allow processes

A preliminary version of this paper appeared in the Proceedings of the 22nd International Symposium on Distributed Computing (DISC) [1].

P. H. Ha and O. J. Anshus are with the Department of Computer Science, Faculty of Science, University of Tromsø, Norway. Email: {phuong,otto}@cs.uit.no.

P. Tsigas is with the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden. Email: tsigas@chalmers.se.

to atomically write to m *arbitrary* memory words; instead, processes can atomically write to m memory words only if the m memory words are located within an aligned size-bounded memory portion (i.e. memory alignment restriction) (cf. Section II). The CUDA coalesced memory accesses are not the multi-object type since their base object type T is the conventional memory word, which has *consensus number one*.

This paper investigates the consensus number of the new memory access mechanisms implemented in current graphics processor architectures. We first define 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 mechanisms can facilitate strong synchronization between the threads of multicore architectures, without the need of synchronization primitives other than reads and writes.

We first define a new memory access model, the *svword* model where *svword* stands for the *s*ize-*v*arying *word* access, the first of the two aforementioned advanced memory access mechanisms implemented in CUDA. Unlike single-word assignments in conventional 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 b is no longer restricted to be a power of 2 (e.g. built-in type *float3* in [3]). By carefully choosing b for the single-word assignments, we can *partly* overlap the bytes written by two assignments, namely each of the two assignments has some byte(s) that is not overwritten by the other overlapping assignment (cf. Figure 1(a) for an illustration). Note that words of size d must always start at addresses that are multiples of d, which is called *alignment restriction* as defined in the conventional computer architecture. The alignment restriction prevents single-word assignments in conventional architectures from partly overlapping each other since the word-size is restricted to be a power of two. This observation has motivated us to develop the *svword* model.

Inspired by the coalesced memory accesses, the second of the aforementioned advanced memory access mechanisms, we define two other models, the *aiword* and *asvword* models, to capture the fundamental features of the mechanism. In CUDA, the global shared memory is considered to be partitioned into segments of equal size and aligned to this size. Simultaneous memory accesses to the same segment by threads of a SIMD multiprocessor (or *half-warp* in CUDA terms [14]), during the execution of a single read/write instruction, can be coalesced into a *single* memory access. The coalescence happens even if some of the threads do not actually access memory (cf. Figure 5-1 in [3] or Figure 1(c)). This allows a SIMD multiprocessor (or a process) to atomically write to multiple memory locations (within a segment) that are not at consecutive addresses. Accesses to the same segment by different processes are executed sequentially.

We generally model this mechanism as an *a*ligned-*i*nconsecutive-*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 atomic m-register assignment [5] due to the memory alignment restriction[1]. Our third model, *asvword*, is an extension of the second model *aiword* in which *aiword*'s A memory units are now replaced by A *svword*s of the same size b. This model is inspired by the fact that the read/write instructions of different coalesced global memory accesses can access words of different size [3].

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 integer between 1 and an upper bound $B \geq 2$. We prove that the single-*svword* assignment has consensus number exactly 3 when $B \geq 5$, consensus number 2 when $B \in \{3, 4\}$, and consensus number 1 when $B = 2$. We also introduce a technique to minimize the size of (proposal) values in consensus algorithms, which allows a *single-word* assignment to write many values atomically and handle the consensus problem for several processes (cf. Section IV).
- We develop a general memory access model, the *aiword* model, to capture the fundamental features of the coalesced memory accesses. The second model is an aligned-inconsecutive-word access model 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 present a wait-free consensus algorithm for $N = \lfloor \frac{A+1}{2} \rfloor$ processes using only single-*aiword* assignments and subsequently prove that the single-*aiword* assignment has consensus number exactly $N = \lfloor \frac{A+1}{2} \rfloor$ (cf. Section V).
- We develop a general memory access model, *asvword*, to capture the fundamental features of the *combination* of 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 *svword*s of the same size $b, b \in \{1, B\}$ (cf. Section VI). We prove that the consensus number of the single-*asvword* assignment is exactly N, where

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

  In the case of the contemporary CUDA processors (with compute capability 1.2 and higher) in which $A = 32$ and $B = 4$, the consensus number of the *asvword* model is sixty four.

The rest of this paper is organized as follows. Section II presents the three new memory access models. Sections IV, V and VI present the exact consensus numbers of the first, second and third models, respectively. Finally, Section VII concludes this paper.

[1]In this paper, we use term "single" in *single-*word assignment* when we want to emphasize that the assignment is not the *multiple* assignment [5].

II. Models

A. General descriptions

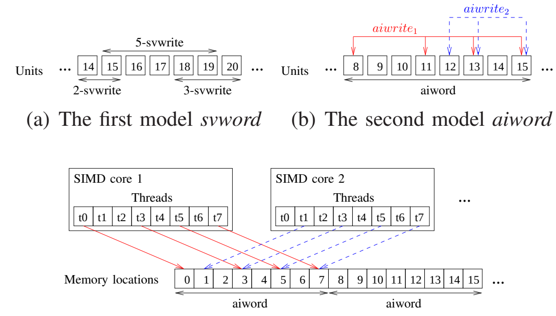
Before describing the details of each of the three new memory access models, we present the common properties of all these three models. The shared memory in the three new models is *sequentially consistent* [15], [16], which is weaker than the linearizable one [17] assumed in most of the previous research on the synchronization power of the conventional memory access models [5]. Processes are asynchronous. The new models use the conventional 1-dimensional memory address space. In these models, one memory *unit* is a *minimum* number of consecutive bytes/bits which a basic read/write operation can atomically read from/write to. These memory models address individual memory units. Memory is organized so that a group of d consecutive memory units called *word* can be stored or retrieved in a *single* basic write or read operation, respectively, and d is called *word size*. Words of size d must always start at addresses that are multiples of d, which is called *alignment restriction* as defined in the conventional computer architecture.

B. The first model svword

The first model is a *s*ize-*v*arying-*word* access model (*svword*) in which a single read/write operation can atomically read from/write to a word consisting of b *consecutive* memory units, where b, called *svword size*, can be any integer between 1 and an upper bound B. The upper bound B is the maximum number of *consecutive* units which a basic read/write operation can atomically read from/write to. *Svword*s of size b must always start at addresses that are multiples of b due to the memory alignment restriction. We denote b-*svword* to be an *svword* consisting of b units, b-*svwrite* to be a b-*svword* assignment and b-*svread* to be a b-*svword* read operation. Reading a unit U is denoted by 1-*svread*(U) or just by U for short. This model is inspired by the CUDA graphics processor architecture in which basic read/write operations can atomically read from/write to words of different size (cf. built-in types *float1, float2, float3* and *float4* in [3], Section 4.3.1.1). Figure 1(a) illustrates how 2-*svwrite*, 3-*svwrite* and 5-*svwrite* can partly overlap their units with addresses from 14 to 20, with respect to the memory alignment restriction.

C. The second model aiword

The second model is an *a*ligned-*i*nconsecutive-*word* access model (*aiword*) in which the memory is aligned to A-unit words and a single read/write operation can atomically read from/write to an arbitrary non-empty subset of the A units of a word, where A is a constant. *Aiword*s must always start at addresses that are multiples of A due to the memory alignment restriction. We denote A-*aiword* to be an *aiword* consisting of A units, A-*aiwrite* to be an A-*aiword* assignment and A-*airead* to be an A-*aiword* read operation. Reading only one unit U (using *airead*) is denoted by U for short. In the *aiword* model, an *aiwrite* operation executed by a process cannot *atomically* write to units located in different *aiword*s due to the memory alignment restriction.



(a) The first model *svword* (b) The second model *aiword*

(c) The coalesced memory access

Fig. 1. Illustrations for the first model, *s*ize-*v*arying-*word* access (*svword*), the second model, *a*ligned-*i*nconsecutive-*word* access (*aiword*) and the coalesced memory access.

Figure 1(b) illustrates the *aiword* model with $A = 8$ in which the *aiword* consists of eight consecutive units with addresses from 8 to 15. Unlike in the *svword* model, the assignment in the *aiword* model can atomically write to *inconsecutive* units of the eight units: $aiwrite_1$ atomically writes to four units $8, 11, 13$ and 15; $aiwrite_2$ writes to three units $12, 13$ and 15.

This model is inspired by the coalesced global memory accesses in the CUDA architecture [3]. The CUDA architecture can be generalized to an abstract model of a MIMD[2] chip with multiple SIMD cores sharing memory. Each core (or streaming multiprocessor SM in CUDA terms) executes A *identical* instructions (on different data) simultaneously, but different cores can simultaneously execute different instructions. The sequence of instructions that are being executed by one SIMD core is called a *process*. Namely, each process consists of A parallel threads that are running in a SIMD manner. The process accesses the shared memory using the CUDA memory access mechanism. In CUDA, the global shared memory is considered to be partitioned into segments of equal size and aligned to this size. Simultaneous memory accesses to the same segment by threads of a SIMD core during the execution of a single read/write instruction can be coalesced into a *single* memory access. The coalescence happens even if some of the threads do not actually access memory (cf. Figure 5-1 in [3] or Figure 1(c)). This allows a SIMD core (or a process consisting of A parallel threads running in a SIMD manner) to atomically access multiple memory locations (within a segment) that are not at consecutive addresses. Accesses to the same segment by different processes are executed sequentially.

Figure 1(c) illustrates the coalesced memory access, where $A = 8$. The left SIMD core can write atomically to four memory locations 0, 3, 5 and 7 by letting only four of its eight threads, $t0, t3, t5$ and $t7$, simultaneously execute a write operation (i.e. divergent threads). The right SIMD core can write atomically to its own memory location 1 and shared memory locations 3, 5 and 7 by letting only four threads $t1, t3, t5$ and $t7$ simultaneously execute a write operation. Note that the CUDA architecture allows threads from different SIMD cores to communicate through the global shared memory [18].

[2]MIMD: Multiple-Instruction-Multiple-Data

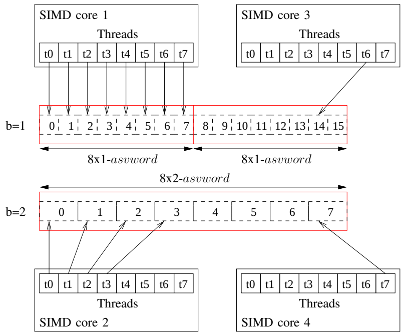


Fig. 2. An illustration for the *asvword* model.

D. *The third model* asvword

The third model is a coalesced memory access model (*asvword*), an extension of the second model *aiword*, in which *aiword*'s A units are now replaced by A *svword*s of the same size $b, b \in [1, B]$. Namely, the second model *aiword* can be considered a special case of the third model *asvword* where $B = 1$. This model is inspired by the fact that in CUDA the read/write instructions of different coalesced global memory accesses can access words of different size. For instance, CUDA with compute capability 1.0 or 1.1 supports the *atomic* coalesced memory access to a segment of 16 words of size 4 bytes (resulting in a 64-byte segment) or to a segment of 16 words of size 8 bytes (resulting in an 128-byte segment) (cf. Section 5.1.2.1 in [3]). This coalesced memory access can be represented by the *asvword* model, where $A = 16$ and $b \in \{1, 2\}$. CUDA with compute capability 1.2 or higher supports the atomic coalesced memory access to a segment of 32 words of size 1 byte (resulting in a 32-byte segment), to a segment of 32 words of size 2 bytes (resulting in a 64-byte segment), or to a segment of 32 words of size 4 bytes (resulting in an 128-byte segment). This coalesced memory access can be represented by the *asvword* model, where $A = 32$ and $b \in \{1, 2, 4\}$.

Let Axb-*asvword* be the *asvword* that is composed of A *svword*s each of which consists of b memory units. Axb-*asvword*s whose size is $A \cdot b$ must always start at addresses that are multiples of $A \cdot b$ due to the memory alignment restriction. We denote Axb-*asvwrite* to be an Axb-*asvword* assignment and Axb-*asvread* to be an Axb-*asvword* read operation. Reading only one unit U (using Ax1-*asvread*) is denoted by U for short. Due to the memory alignment restriction, an Axb-*asvwrite* operation cannot atomically write to b-*svword*s located in *different* Axb-*asvword*s. 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. A variant of the model in which $b = 2^c, c = 0, 1, \ldots, \log_2 B$, and A, B are powers of 2, can be established from this model (cf. Section VI). Since both Ax1-*asvword*s and AxB-*asvword*s are aligned from the address base of the memory space, any AxB-*asvword* can be aligned with B Ax1-*asvword*s as shown in Figure 2.

Figure 2 illustrates the *asvword* model in which each dash-dotted rectangle/square represents an *svword* and each red/solid rectangle represents an *asvword* composed of eight *svword*s (i.e. $A = 8$). The two rows show the memory alignment corresponding to the size b of *svword*s, where b is 1 or 2 (i.e. $B = 2$), on the same sixteen consecutive memory units with addresses from 0 to 15. An *asvwrite* operation can atomically write to some or all of the eight *svword*s of an *asvword*. Unlike the *aiwrite* operation in the second model, which can atomically write to at most 8 units (or A units), the *asvwrite* operation in the third model can atomically write to 16 units (or $A \cdot B$ units) using a single 8x2-*asvwrite* operation (i.e. write to the whole set of eight 2-*svword*s, cf. row $b = 2$). For an 8x1-*asvword* on row $b = 1$, there are two methods to update it atomically using the *asvwrite* operation: i) writing to the whole set of eight 1-*svword*s using a single 8x1-*asvwrite* (cf. SIMD core 1) or ii) writing to a subset consisting of four 2-*svword*s using a single 8x2-*asvwrite* (cf. SIMD core 2). However, if only one of the eight units of an 8x1-*asvword* (e.g. unit 14) needs to be updated and the other units (e.g. unit 15) must remain untouched, the only possible method is to write to the unit using a single 8x1-*asvwrite* (cf. SIMD core 3). The other method, which writes to one 2-*svword* using a single 8x2-*asvwrite*, will have to overwrite another unit that is required to stay untouched (cf. SIMD core 4).

III. Preliminary results on wait-free consensus

This paper uses the conventional terminology from bivalency arguments [13], [5], [19]. The *configuration* of an algorithm at a moment in its execution consists of the value 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*. The *critical value* of a process is the value that would get decided if that process takes the next step after the critical configuration.

Definition 3.1 (Wait-free consensus): Wait-free consensus is a problem in which each process starts with an input value from some set S, $|S| \geq 2$, and must eventually produce an output value so that the following properties are satisfied in *every* execution:

- *Agreement*: the output values of all processes are identical;
- *Validity*: the output value of each process is the input value of some process;
- *Wait-freedom*: each process produces an output value after a finite number of steps.

Definition 3.2 (Consensus number): The *consensus number* of an object type is either the maximum number of processes for which wait-free consensus can be solved using only objects of this type and registers [3], or infinity if such a maximum does not exist.

Before proving the consensus number of single-**word* assignments, we present the essential features of any wait-free

[3] A *register* supports only read and write operations.

consensus algorithm $\mathcal{ALG}$ for $N \geq 2$ processes using only single-**word* assignments and registers, where **word* can be *svword, aiword* or *asvword*.

Lemma 3.1: Algorithm $\mathcal{ALG}$ must have a critical configuration C^*, and the critical operations op_i of processes p_i with different critical values must write to the same object $\mathcal{O}$, which consists of *memory units*.

Proof: We first prove that $\mathcal{ALG}$ must have a critical configuration C^* by contradiction. Suppose that $\mathcal{ALG}$ has no critical configuration. Since $\mathcal{ALG}$ solves wait-free consensus for $N \geq 2$ processes with different input values, $\mathcal{ALG}$'s initial configuration C_0 is multivalent due to $\mathcal{ALG}$'s validity property (cf. Definition 3.1). Since $\mathcal{ALG}$ has no critical configuration due to the hypothesis, in *any* multivalent configuration C_i (e.g. C_0), there always exists an operation that carries $\mathcal{ALG}$ from C_i to another multivalent configuration C_{i+1}. That means there must exist a non-terminating execution, a contradiction to $\mathcal{ALG}$'s wait-freedom property (cf. Definition 3.1).

We now prove that the critical operations op_i of processes p_i with different critical values must *write* to the *same* object $\mathcal{O}$ by contradiction.

- Suppose that the critical operation op_i of a process p_i is to *read* an object $\mathcal{O}$ and carries $\mathcal{ALG}$ from a critical configuration C^* to an x-valent configuration. Since configuration C^* is critical, there must be a process p_j whose critical operation op_j carries $\mathcal{ALG}$ from C^* to a y-valent configuration, $y \neq x$. The configuration C_1 that immediately follows the execution $e_1 = op_i, op_j$ continuing from C^*, is x-valent since p_i executes its critical operation op_i first. Similarly, the configuration C_2 that immediately follows the execution $e_2 = op_j$ continuing from C^*, is y-valent. Due to the hypothesis that op_i only reads object $\mathcal{O}$, configurations C_1 and C_2 are indistinguishable to process p_j [4], a contradiction since C_1 is x-valent and C_2 is y-valent. Therefore, the critical operations of processes with different critical values must be write-operations.
- Suppose that in a critical configuration C^* there are two processes p_i and p_j whose critical operations op_i and op_j are to write x and y, $x \neq y$, to *different* objects $\mathcal{O}_i$ and $\mathcal{O}_j$, respectively. The configuration C_1 that immediately follows the execution $e_1 = op_i, op_j$ continuing from C^*, is x-valent since p_i executes its critical operation op_i first. Similarly, the configuration C_2 that immediately follows the execution $e_2 = op_j, op_i$ (i.e. reversing the order of op_i and op_j in e_1), is y-valent. Due to the hypothesis that op_i and op_j write to different objects $\mathcal{O}_i$ and $\mathcal{O}_j$, configurations C_1 and C_2 are indistinguishable to processes p_i and p_j, a contradiction since C_1 is x-valent and C_2 is y-valent.

■

Definition 3.3: One-writer (resp. *two-writer*) unit, or 1W-unit (resp. 2W-unit) for short, is a memory unit that is written by only one critical operation (resp. two critical operations) in a critical configuration.

[4]Two configurations c and c' are *indistinguishable* to a process p_j if the internal state of process p_j and the value of every shared object are the same in c and c' [20].

Lemma 3.2: In a critical configuration C^* of $\mathcal{ALG}$, critical operation op_i by each process p_i must atomically write to

1) an one-writer unit u_i written by p_i and
2) two-writer units $u_{i,j}$ written by two processes p_i and p_j, where p_j's critical value is different from p_i's, $\forall j \neq i$.

Proof: The proof is similar to the bivalency argument of Theorem 13 in [5]. Due to Lemma 3.1, $\mathcal{ALG}$ must have a critical configuration C^* and critical operations op_i of processes p_i with different critical values must be write-operations. Let x be p_i's critical value in the critical configuration C^*. Since configuration C^* is critical, there must be another process p_j whose critical value y is different from x. Let op_j be p_j's critical write-operation.

- We first prove that op_i must write to an one-writer unit by contradiction. Suppose that all op_i's units are overwritten by p_j's and other processes operations. The configuration C_1 that immediately follows an execution $e_1 = op_i, op_j, op'_1, \ldots, op'_k$ continuing from C^*, where all op_i's units are overwritten by (some of) other processes operations $op_j, op'_1, \ldots, op'_k$, is x-valent since p_i executes its critical operation op_i first. Similarly, the configuration C_2 that immediately follows execution $e_2 = op_j, op'_1, \ldots, op'_k$ (i.e. removing op_i from e_1), is y-valent. Execution e_2 corresponds to the case that p_i stops right before executing op_i. Due to the hypothesis that all op_i's units are overwritten by $op_j, op'_1, \ldots, op'_k$, configurations C_1 and C_2 are indistinguishable to process p_j, a contradiction since C_1 is x-valent and C_2 is y-valent.
- We now prove by contradiction that op_i must also write to two-writer units written by p_i and p_j, where p_j's critical value is different from p_i's, $\forall j \neq i$. Due to Lemma 3.1, op_i and op_j must write to the same object, or there must be units written by both op_i and op_j. Suppose that all units written by both op_i and op_j are overwritten by other processes operations. The configuration C_1 that immediately follows an execution $e_1 = op_i, op_j, op'_1, \ldots, op'_k$ continuing from C^*, where all units written by both op_i and op_j are overwritten by (some of) other processes operations $op'_1, \ldots, op'_k$, is x-valent since p_i executes its critical operation op_i first. Similarly, the configuration C_2 that immediately follows execution $e_2 = op_j, op_i, op'_1, \ldots, op'_k$ (i.e. reversing the order of op_i and op_j in e_1) is y-valent. Due to the hypothesis that all units written by both op_i and op_j are overwritten by $op'_1, \ldots, op'_k$, configurations C_1 and C_2 are indistinguishable to processes p_i and p_j, a contradiction since C_1 is x-valent and C_2 is y-valent.

■

IV. Consensus number of the *svword* model

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

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 = set $\{u_0, u_1, u_2\}$ of $units$: initialized to $\perp$ and used in the first phase. $WR_1[0]$ and $WR_1[2]$ are units written only by p_0 and p_1, respectively. $WR_1[1]$ is a unit written by both processes.
WR_2 = set $\{v_0, \ldots, v_4\}$ of $units$: initialized to $\perp$ and used in the second phase. $WR_2[0]$, $WR_2[2]$ and $WR_2[4]$ are units written only by p_0, p_2 and p_1, respectively. $WR_2[1]$ and $WR_2[3]$ are 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 1**: Achieve an agreement between p_0 and p_1.
2V: **if** $i = 0$ or $i = 1$ **then**
3V: $first \leftarrow$ SVW_FIRSTAGREEMENT(i); // $first$ is a shared variable
4V: **end if**
  // **Phase II**: Achieve an agreement between all three processes.
5V: $winner \leftarrow$ SVW_SECONDAGREEMENT($i, first_{ref}$); // $first_{ref}$ is the reference to $first$
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** $WR_1[(1 - i) * 2] = \perp$ **then**
7SF: **return** i; // The other process hasn't written its value
8SF: **else if** ($WR_1[1] = Higher$ and $i = 0$) or ($WR_1[1] = Lower$ 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** $(1 - i)$;
12SF: **end if**

Algorithm 3 SVW_SECONDAGREEMENT(i: index; $first_{ref}$: reference) 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** (($WR_2[0] \neq \perp$ or $WR_2[4] \neq \perp$) and $WR_2[2] = \perp$) or // The predicates are checked in the writing order.
  ($WR_2[0] \neq \perp$ and $WR_2[1] = Higher$) or
  ($WR_2[4] \neq \perp$ and $WR_2[3] = Higher$) **then**
9SS: **return** $first$; // p_2 is preceded by either p_0 or p_1. $first$ is obtained by dereferencing $first_{ref}$.
10SS: **else**
11SS: **return** 2;
12SS: **end if**

to utilize the size-variation feature of the *svwrite* operation. A *b-svwrite* operation can atomically write up to b values to b consecutive memory units if each of the values can be stored in one memory unit. Therefore, keeping the values to be atomically written as small as possible will maximize the number of processes for which *b-svwrite* can solve the consensus problem. Unlike the wait-free consensus algorithm using the m-word assignment by Herlihy [5], 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 (or $\frac{B}{2}$ if units are single bits) ordering-related values. The new algorithm utilizes process unique identifiers, which are an implicit assumption in Herlihy's consensus model [21].

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, $PROPOSAL[first]$, is the proposal value of the *preceding process*, whose SVWRITE (line 2SF or 4SF) precedes that of the other process (cf. Lemma 4.1).

Due to the memory alignment restriction, in order to be able to allocate memory for the WR_1 variable (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 variable is located in a memory region consisting of 4 consecutive units $\{u_0, u_1, u_2, u_3\}$ of which u_0 is at an address multiple of 2 and u_1 at an address multiple of 3. This memory allocation allows p_0 and p_1 to write atomically to the first two units $\{u_0, u_1\}$ and the last 3 units $\{u_1, u_2, u_3\}$, respectively (cf. Figure 3(a)). The WR_1 variable is the set $\{u_0, u_1, u_2\}$ (cf. the solid squares in Figure 3(a)), namely p_1 ignores u_3 (cf. line 4SF in Algorithm 2).

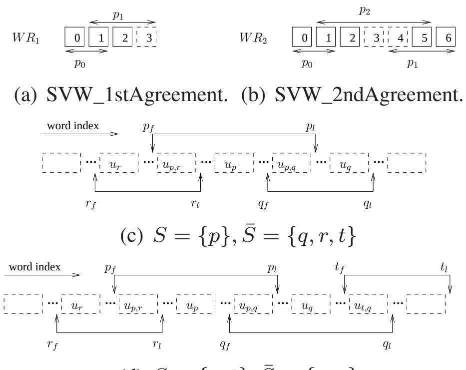
Subsequently, the agreed value will be used as the proposal value of both p_0 and p_1 in the second phase in order to achieve an agreement with the other process p_2 (cf. Algorithm 3). Let p_{first} be the preceding process of p_0 and p_1 in the first phase. The second phase returns p_{first}'s proposal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

Units written by processes' SVWRITEs are illustrated in Figure 3(b). In order to be able to allocate memory for the WR_2 variable, process p_0's, p_1's and p_2's SVWRITEs are chosen as 2-*svwrite*, 3-*svwrite* and 5-*svwrite*, respectively. The WR_2 variable is located in a memory region consisting of 7 consecutive units $\{u_0, \ldots, u_6\}$ of which u_0 is at an address multiple of 2, u_4 at an address multiple of 3 and u_1 at an address multiple of 5. Since 2, 3 and 5 are prime numbers, we can always 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 (cf. Figure 1(a)). This memory allocation allows p_0, p_1 and p_2 to write atomically to the first two units $\{u_0, u_1\}$, the last three units $\{u_4, u_5, u_6\}$ and the five middle units $\{u_1, \ldots, u_5\}$, respectively. The WR_2 variable is the set $\{u_0, u_1, u_2, u_5, u_6\}$ (cf. the solid squares in Figure 3(b)).

Lemma 4.1: 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 (i.e. p_0's SVWRITE (line 2SF) precedes p_1's SVWRITE (line 4SF)), their 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 case, $WR_1[2] = \perp$ holds (line 6SF), making the procedure return 0 (line 7SF). In the latter case,



(a) SVW_1stAgreement. (b) SVW_2ndAgreement.

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

(d) $S = \{p, t\}, \bar{S} = \{q, r\}$

Fig. 3. Illustrations for the SVW_FIRSTAGREEMENT, SVW_SECONDAGREEMENT and Lemma 4.5.

$WR_1[2] \neq \perp$ and the procedure checks $WR_1[1]$ at line 8SF. Since predicate ($WR_1[1] = Higher$ and $i = 0$) holds, the procedure returns 0 (line 9SF).

If p_1 precedes p_0, their unit $WR_1[1]$ from line 8SF 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 case cannot happens since p_0 executes its SVWRITE at line 2SF (i.e. before line 6SF) in the SVW_FIRSTAGREEMENT procedure and the procedure is assumed to be invoked by p_0. In the latter case, the procedure returns 1 (line 11SF) since the predicate at line 6SF fails and subsequently the predicate at line 8SF fails. ∎

Lemma 4.2: The SVW_SECONDAGREEMENT 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 (i.e. p_0's SVWRITE (line 2SS) precedes p_2's SVWRITE (line 6SS)), their 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 case, predicate ($WR_2[0] \neq \perp$ and $WR_2[2] = \perp$) holds (line 8SS), making the procedure return $first$ (line 9SS). In the latter case, predicate ($WR_2[0] \neq \perp$ and $WR_2[1] = Higher$) holds (line 8SS), making the procedure return $first$ (line 9SS).

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

Note that since i) p_0 and p_1 invoke the SVW_SECONDAGREEMENT procedure (line 5V, Algorithm 1) only after getting $first$ from the SVW_FIRSTAGREEMENT procedure (line 3V) and ii) the reference to $first$ (instead of a value of $first$) is passed to SVW_SECONDAGREEMENT, the value $first$ returned by SVW_SECONDAGREEMENT in these two cases is defined.

If p_2 precedes both p_0 and p_1 (i.e. p_2's SVWRITE precedes both p_0's and p_1's SVWRITEs), then

- $WR_2[2] \neq \perp$ at line 8SS, and
- their unit $WR_2[1]$ is either $Higher$ (when p_0 has not executed its SVWRITE yet, i.e. $WR_2[0] = \perp$) or $Lower$ (when p_0's SVWRITE has overwritten the value $Higher$ written by p_2's), and
- their unit $WR_2[3]$ is either $Higher$ (when p_1 has not executed its SVWRITE yet, i.e. $WR_2[4] = \perp$) 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).

Note that the five units $WR_2[0], WR_2[1], WR_2[2], WR_2[3]$ and $WR_2[4]$ at line 8SS are read one by one and the order in which the reads are performed for checking each predicate does not matter due to the atomic write-operations SVWRITE at lines 2SS, 4SS and 6SS. ∎

Lemma 4.3: 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 Lemma 4.2, 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 $PROPOSAL[first]$, $first \in \{0, 1\}$ (otherwise). ∎

Lemma 4.4: 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 using only registers and the single-*svword* assignment with $B \geq 5$ (Lemma 4.3), this lemma immediately follows. ∎

Lemma 4.5: The single-*svword* assignment has consensus number at most 3, $\forall B \geq 5$.

Proof: We prove the lemma by contradiction. Assume that there is a wait-free consensus algorithm $\mathcal{ALG}$ for four processes p, q, r and t. From Lemma 3.1, $\mathcal{ALG}$ must have a critical configuration C^* and the critical operations op_i of processes p_i with different critical values must be write-operations. At the critical configuration C^*, we can always divide the set of the four processes into two non-empty subsets S and $\bar{S}$ where S consists of at most two processes with the same critical value called V and $\bar{S}$ consists of processes with critical values different from V (If three of the four processes have the same critical value, the other process is chosen as S). 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. 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 3.2) and none of them are completely covered by ranges $[v_f, v_l]$ of the other processes v (due to the first requirement of Lemma 3.2).

Figures 3(c) and 3(d) illustrate the main idea of the proof when S consists of one and two processes, respectively. In Figure 3(c), the range $[t_f, t_l]$ of process t cannot partly overlap with that of process p without completely covering (or being covered by) the range of process r or q. In Figure 3(d), t and r belong to different subsets S and $\bar{S}$, respectively, but their ranges cannot partly overlap. The detailed proof is as follows.

- If S consists of 1 process, let $S = \{p\}$. Since p's critical value is different from those of the three other processes q, r and t, process p's critical operation must atomically write to 4 units $u_{p,q}$, $u_{p,r}$, $u_{p,t}$ and u_p (cf. Lemma 3.2). The atomic write-operation determines the relative ordering between p and the three other processes with critical values different from p's: if p's operation precedes q's, p is considered preceding q.

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 3(c)).

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 3.2 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 the overlapping ranges $[p_f, p_l]$ and $[r_f, r_l]$, violating the first requirement of Lemma 3.2 for the process q.

Arguing similarly, we have $t_f < p_f \leq t_l < p_l$. If $t_f \leq r_f$, r's range $[r_f, r_l]$ is covered completely by the overlapping 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 the overlapping ranges $[r_f, r_l]$ and $[p_f, p_l]$, a contradiction to the first requirement of Lemma 3.2 for the process t.

- If S consists of 2 processes, let $S = \{p, t\}$. Since p's and t's critical value is 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 3.2). 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 3(c)) or $q_f < p_f \leq q_l < r_f \leq p_l < r_l$ (i.e. exchange r and q in Figure 3(c)). 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 $p_f < q_f \leq p_l < t_f \leq q_l < t_l$ (cf. Figure 3(d)). 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 $r_f < p_f \leq r_l < q_f \leq p_l < q_l$.

■

Lemma 4.6: The single-*svword* assignment (*svwrite*) has consensus number 1 when $B = 2$.

Proof: We prove the lemma by contradiction. Assume that there is a wait-free consensus algorithm $\mathcal{ALG}$ for two processes (with different proposal values) p_0 and p_1 using only *svwrite*s and registers. Algorithm $\mathcal{ALG}$ must have a critical configuration C^* (Lemma 3.1) in which p_i's critical operation must atomically write to both p_i's 1W-unit $u_i, i \in \{0, 1\}$, and a 2W-unit $u_{0,1}$ (Lemma 3.2). Since $B = 2$, in order to atomically write to two units, both p_0's and p_1's critical operations must be 2-*svwrite*s, which prevents the two critical operations from *partly* overlapping due to the memory alignment restriction. That means if p_0's critical operation is the first operation writing to the 2-*svword* containing $u_{0,1}$ and u_0, p_1's critical operation, which must write to $u_{0,1}$, will then overwrite the 2-*svword* completely, violating the first requirement of Lemma 3.2 for process p_0. ■

Lemma 4.7: The single-*svword* assignment (*svwrite*) has consensus number 2 when $B \in \{3, 4\}$.

Proof: Since the SVW_FIRSTAGREEMENT procedure (Algorithm 2) solves wait-free consensus for two processes using *svwrite*s and registers (Lemma 4.1) when $B \geq 3$, the single-*svword* assignment (*svwrite*) has consensus number at least 2 when $B \in \{3, 4\}$.

We now prove by contradiction that when $B \in \{3, 4\}$, there is no wait-free consensus algorithm for three processes using only *svwrite*s and registers. Assume that there is a wait-free consensus algorithm $\mathcal{ALG}$ for three processes p, q and r. Algorithm $\mathcal{ALG}$ must have a critical configuration (Lemma 3.1) in which we can always divide the set of these three processes into two non-empty subsets S and $\bar{S}$, where S consists of a process p with critical value V and $\bar{S}$ consists of processes q and r with critical 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 $k \in \{p, q, r\}$ atomically writes using its critical operation op_k. 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 3.2) and none of them are completely covered by ranges $[v_f, v_l]$ of the other processes v (due to the first requirement of Lemma 3.2).

Arguing similarly to the proof of Lemma 4.5 results in that ranges $[p_f, p_l]$, $[q_f, q_l]$ and $[r_f, r_l]$ must partly overlap each other as shown in Figure 3(c).

- If $B = 3$, range $[p_f, p_l]$, which contains $u_{p,r}$, u_p and $u_{p,q}$, must be a 3-*svword* starting at an address multiple of 3, namely $p_f = 3a, a \in \mathbb{N}$ (integers). In order to *partly* overlap with range $[p_f, p_l]$, ranges $[r_f, r_l]$ and $[q_f, q_l]$ must be 2-*svword*s due to the memory alignment restriction. That means r_f and q_f are addresses multiple of 2: $r_f = 2b$ and $q_f = 2c$ where $b, c \in \mathbb{N}$. On the other hand, since range $[p_f, p_l]$ is a 3-*svword*, there is exactly one unit between r_l and q_f (cf. Figure 3(c)), or $q_f = r_l + 2 = (r_f + 1) + 2 = 2(b + 1) + 1$, a contradiction to $q_f = 2c$ (an even address).
- If $B = 4$, range $[p_f, p_l]$ must be a 4-*svword* starting at an address multiple of 4, namely $p_f = 4a, a \in \mathbb{N}$. Indeed, if range $[p_f, p_l]$ is a 3-*svword*, arguing similarly to the case $B = 3$ will result in a contradiction since r_l and q_f must be odd and even, respectively, while $q_f = r_l + 2$.

  Since range $[p_f, p_l]$ is a 4-*svword*, in order to *partly* overlap with range $[p_f, p_l]$, ranges $[r_f, r_l]$ and $[q_f, q_l]$ must be 3-*svword*s due to the memory alignment restriction. That means r_f and q_f are addresses multiple of 3: $r_f = 3b$ and $q_f = 3c$ where $b, c \in \mathbb{N}$. Since range $[p_f, p_l]$ must not be covered completely by ranges $[r_f, r_l]$ and $[q_f, q_l]$, $c \geq (b+2)$ must hold (cf. Figure 3(c)). On the other hand, since range $[p_f, p_l]$ is a 4-*svword*, there are at most 2 units between r_l and q_f (cf. Figure 3(c)). That means $q_f - r_l \leq 3$, or $3(c - b) \leq 5$, a contradiction to $c \geq (b + 2)$.

■

From Lemmas 4.4, 4.5, 4.6 and 4.7, we have the following

Theorem:

Theorem 4.1: The single-*svword* assignment has consensus number exactly N, where

$$N = \begin{cases} 3, & \text{if } B \geq 5 \\ 2, & \text{if } B = 3, 4 \\ 1, & \text{if } B = 2 \end{cases} \tag{2}$$

V. 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 a 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 the *aiwrite* operation and registers. Subsequently, we prove that there is no wait-free consensus algorithm for $N+1$ processes using only the *aiwrite* operation 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 units that must be written atomically by the *aiword* operation (cf. Lemma 5.4). The algorithm consists of N rounds and a process $p_i, i \in [1, N]$, participates in rounds $r_i \dots r_N$. 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 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 $p_i, 1 \leq i \leq j$.

A process p_i participates in the consensus protocol from round i by initializing its agreed value $A^i[i]$ in round i to its proposal value buf_i (line 1I). In order to determine whether it precedes all $(i-1)$ other processes p_k participating in round i, $k = 1, \dots, (i-1)$, p_i atomically writes $Higher$ to its unit U_i^i and $(i-1)$ units $U_{i,k}^i$ using *aiwrite* (line 3I). Each process p_k atomically writes $Lower$ to its units U_k^i and $U_{i,k}^i$ using *aiwrite* (line 13I). Process p_k precedes p_i if p_k's *aiwrite* precedes p_i's *aiwrite*. If one of processes p_k precedes p_i, p_i agrees on p_k's proposal value $A^i[k]$ by writing this value to $A^i[i]$ (lines 4I - 6I). Note that all processes p_k participating in round i, $k < i$, have the same proposal value, which is their agreed value in the previous round $(i-1)$ (line 12I and Lemma 5.1). Otherwise, p_i keeps its proposal value buf_i as its agreed value in round i and subsequently enters the next round $(i+1)$ (line 11I).

Process p_i participates in rounds $j, j = (i+1), \dots, N$, by initializing its agreed value $A^j[i]$ in round j to its agreed value $A^{j-1}[i]$ in the previous round $(j-1)$ (line 12I). Since all processes p_k participating in round j, $k < j$, have agreed on the same value in the previous round $(j-1)$ (cf. Lemma 5.1)), they have the same proposal value in round j. Therefore, p_i needs to change its agreed value $A^j[i]$ only if p_j precedes all processes $p_k, k < j$. After atomically writing $Lower$ to its units U_i^j and $U_{j,i}^j$, p_i checks if p_j precedes all other processes $p_k, k < j$ (lines 14I - 21I). If so, p_i agrees on p_j's proposal value $A^j[j]$ by writing this value to $A^j[i]$ (line 23I). After obtaining its agreed value $A^N[i]$ in round N, p_i agrees with all other processes on the same value (Lemma 5.1) and finishes the consensus protocol (line 27I).

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

$A^r[i]$: the value upon which p_i agrees with other processes in round r;
$U_{i,j}^r$: the unit written only by processes p_i and p_j in round r. U_i^r: the unit written only by process p_i in round r;

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: **if** $i \geq 2$ **then**
3I: AIWRITE($\{U_i^i, U_{i,1}^i, \dots, U_{i,i-1}^i\}$, $\{Higher, Higher, \dots, Higher\}$) // Atomic assignment
4I: **for** $k = 1$ to $(i-1)$ **do** // Check if p_i precedes all other processes p_k in round i
5I: **if** $(U_k^i \neq \perp)$ and $(U_{i,k}^i = Higher)$ **then** // The predicates are checked from left to right
6I: $A^i[i] \leftarrow A^i[k]$; // p_k precedes $p_i \Rightarrow$ Update p_i's agreed value with the set S's agreed value
7I: **break**;
8I: **end if**
9I: **end for**
10I: **end if**
// Participate in rounds $(i+1) \dots N$
11I: **for** $j = i+1$ to N **do**
12I: $A^j[i] \leftarrow A^{j-1}[i]$; // Initialized p_i's agreed value for round j
13I: AIWRITE($\{U_i^j, U_{j,i}^j\}$, $\{Lower, Lower\}$; // Atomic assignment
14I: **if** $(U_j^j \neq \perp)$ and $(U_{j,i}^j = Lower)$ **then** // The predicates are checked from left to right
15I: $WinnerIsJ \leftarrow$ **true**; // p_j precedes p_i
16I: **for** $k = 1$ to $j-1$ **do** // Check if p_j precedes all $p_k, \forall k < j$
17I: **if** $(U_k^j \neq \perp)$ and $(U_{j,k}^j = Higher)$ **then** // The predicates are checked from left to right
18I: $WinnerIsJ \leftarrow$ **false**; // p_k precedes p_j;
19I: **break**;
20I: **end if**
21I: **end for**
22I: **if** $WinnerIsJ =$ **true then**
23I: $A^j[i] \leftarrow A^j[j]$; // p_j precedes $p_k, \forall k < j, \Rightarrow p_j$'s value is the agreed value in round j.
24I: **end if**
25I: **end if**
26I: **end for**
27I: **return** $A^N[i]$;

Definition 5.1: A *correct* process is a process that does not crash.

Definition 5.2: The *agreed value* v of a correct process p_i in round $r_j, j \geq i$, is the value of $A^j[i]$ when p_i reaches either line 10I (if $i = j$) or line 25I in iteration j of the for-loop 11I - 26I (if $i < j$). We say that p_i *agrees* on v in r_j.

Lemma 5.1: All correct processes 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_j are the same (line 12I) 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_j will be either A_S^j or A_j^j. At this moment, we assume that AIWRITE (at line 3I or 13I) is atomic (cf. Figure 4 for the layout of units U_j^j, U_i^j and $U_{j,i}^j$ on an *aiword* when $j = N$). We will prove that in round r_j the agreed value of participating processes will be A_j^j if p_j precedes *all* the other processes $p_i, 1 \leq i < j$ (i.e. p_j's AIWRITE precedes all the other processes AIWRITE), or 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 3I for p_j or line 13I for $p_i, i < j$). Let $p_l, l = 1, \ldots, j$, 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 their unit $U_{j,i}^j$, which is only written by p_j's and p_i's AIWRITEs (lines 4I-9I). Since p_j precedes all the other processes p_i, predicate $U_{j,i}^j = Higher$ holds only if p_i did not yet execute its AIWRITE, which would overwrite value $Higher$ written by p_j with value $Lower$. This leads to $U_i^j = \perp$, making predicate ($U_i^j \neq \perp$ and $U_{j,i}^j = Higher$) at line 5I false, $\forall i < j$. Consequently, p_j keeps its proposal value A_j^j as its agreed value.

  If $l = i, i < j$, process p_i determines its relative ordering with process p_j using their unit $U_{j,i}^j$ (line 14I). Since p_j precedes all p_i, predicate ($U_j^j \neq \perp$ and $U_{j,i}^j = Lower$)) holds (line 14I) but predicate ($U_k^j \neq \perp$ and $U_{j,k}^j = Higher$) does not hold for all $k \in [1, j-1]$ (line 17I). This makes p_i agree on A_j^j (line 23I).

- If there is a process $p_I, I < j$, that precedes p_j, predicate ($U_I^j \neq \perp$ and $U_{j,I}^j = Higher$) at line 5I holds, making p_j agree on A_S^j (line 6I). For a process $p_i, i < j$, that is executing the AIW_CONSENSUS procedure, if it is preceded by p_j, it will find that the predicate ($U_I^j \neq \perp$ and $U_{j,I}^j = Higher$) at line 17I holds and consequently keep its proposal value A_S^j as its agreed value.

■

With the assumption that AIWRITE can atomically write to p_j's units at line 3I or p_i's units at line 13I, it follows directly from Lemma 5.1 that all the N processes will achieve an agreement in round r_N.

Lemma 5.2: 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 11I and 16I. Therefore, the AIW_CONSENSUS algorithm is wait-free.

From Lemma 5.1, the AIW_CONSENSUS algorithm can solve the consensus problem for $N = \lfloor \frac{A+1}{2} \rfloor$ processes if AIWRITE can *atomically* write to p_j's j units at line 3I or p_i's 2 units at line 13I. Since AIWRITE can write to an *arbitrary* subset 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 the requirement is satisfied for the case $j = N$.

Indeed, since $N = \lfloor \frac{A+1}{2} \rfloor$, an A-unit *aiword* (or A-*aiword* for short) can accommodate both $(N-1)$ units $U_{N,i}^N, 1 \leq i < N$, and N units $U_k^N, 1 \leq k \leq N$, used in round r_N. Figure 4 illustrates the 2-dimensional layout of the $(2N-1)$ units to be mapped on the 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 $p_k, 1 \leq k \leq (N-1)$, can atomically write to its units U_k^N and $U_{N,k}^N$, and p_N can can atomically write to its unit U_N^N and $(N-1)$ units $U_{N,1}^N, \ldots, U_{N,N-1}^N$. This guarantees that a unit U_i^N is written only by p_i and a unit $U_{N,i}^N$ is written only by p_N and p_i. ■

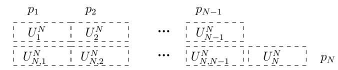


Fig. 4. The layout of units U_i^N and $U_{N,i}^N$ on an A-*aiword*, where $N = \lfloor \frac{A+1}{2} \rfloor$.

Lemma 5.3: 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 5.2) using only the single-*aiword* assignment and registers, this lemma immediately follows. ■

Lemma 5.4: 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 $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$. Due to Lemma 3.1, $\mathcal{ALG}$ must have a critical configuration C^* and the critical operations op_i of processes p_i with different critical values must be write-operations. At the critical configuration C^*, we divide N processes into $k \geq 2$ subsets $s_1, \ldots, s_k$ each of which consists of processes with the same critical value. Let $n_1, \ldots, n_k$ to be the sizes 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 critical value is different from that of $\left(\sum_{l \neq i} n_l\right)$ processes in the $(k-1)$ other subsets, p_j^i's critical operation must atomically write to its 1W-unit and $\left(\sum_{l \neq i} n_l\right)$ 2W-units (cf. Lemma 3.2). The write-operation determines the relative ordering between p_j^i and the $\left(\sum_{l \neq i} n_l\right)$ other processes with critical values different from p_j^i's: if p_j^i's operation precedes p_*^l's, $l \neq i$, p_j^i is considered preceding p_*^l.

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 critical 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 *aiword*s due to the memory alignment restriction, units u_j^i and $u_{j,j'}^{i,i'}$ must be located in the same *aiword* corresponding to p_j^i. Similarly, units $u_{j'}^{i'}$ and $u_{j,j'}^{i,i'}$ must be located in the same *aiword* corresponding 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, 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 must be located in the same A-*aiword* called AI. Let M be the number of 1W-/2W-units that 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:

$$\begin{aligned}M &= N + n_1(n_2 + \cdots + n_k) + n_2(n_3 + \cdots + n_k) + \cdots + n_{k-1}n_k \\ &\quad \text{/*N: the number of 1W-units*/} \\ &\geq N + n_1(n_i + \cdots + n_k) + n_2(n_i + \cdots + n_k) + \\ &\quad \cdots + n_{i-1}(n_i + \cdots + n_k), \text{where } 2 \leq i \leq k \\ &= N + (n_1 + n_2 + \cdots + n_{i-1})(n_i + \cdots + n_k) \end{aligned} \tag{3}$$

That means M will be less if there are only two subsets s_I, s_{II} of processes with the same critical 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

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

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 $M \leq A$. ■

From Lemmas 5.3 and 5.4, we have the following theorem.

Theorem 5.1: The single-*aiword* assignment has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$.

VI. CONSENSUS NUMBER OF THE *asvword* MODEL

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

$$N = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \text{ (positive integers)} \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B, t \in \mathbb{N}^* \\ \lfloor \frac{A+1}{2} \rfloor, & \text{if } B = tA, t \in \mathbb{N}^* \end{cases} \tag{4}$$

The intuition behind the higher consensus number N of the *asvword* model (cf. Equation (4)) compared with the *aiword* model is that process p_N in Algorithm 4 can atomically write to $A \cdot B$ units using AxB-*asvwrite* instead of only A units using A-*aiwrite* (line 3I). As illustrated in Figure 2, an 8x2-*asvwrite* can atomically write to 16 memory units (i.e. write to 8 consecutive 2-*svword*s each of which is comprised of 2 memory units) (cf. row $b = 2$) whereas an 8x1-*asvwrite* (or 8-*aiwrite*) can atomically write to only 8 memory units (cf. row $b = 1$). However, since an 8x2-*asvwrite* uses 2-*svword*s as its minimum units, it cannot write to only one *memory* unit. For instance, using 8x2-*asvwrite*, SIMD core 4 cannot write to only memory unit 14 (cf. row $b = 1$), but it must write to both memory units 14 and 15 that comprise 2-*svword* 7 (cf. row $b = 2$). Therefore, to prevent p_N from overwriting unintended memory units when using AxB-*asvwrite*, each B-*svword* located in $A_l, 1 \leq l \leq B$, contains either units U_i^N or units $U_{N,i}^N, i < N$, but not both as illustrated in Figure 6(a), where B-*svword*s labeled "1W" contain only units U_i^N and B-*svword*s labeled "2W" contain only units $U_{N,i}^N$. This allows p_N to atomically write to only B-*svword*s with units $U_{N,i}^N$ (and keep unit $U_i^N, i < N$, untouched) using AxB-*asvwrite*. For each process $p_i, i \neq N$, its units U_i^N and $U_{N,i}^N$ are located in two B-*svword*s labeled "1W" and "2W", respectively, that belong to the same A_l. This allows p_i to atomically write to only its two units using Ax1-*asvwrite*.

We first prove that the *asvwrite* operation has consensus number at least N (cf. Equation (4)). We prove this by presenting a wait-free consensus algorithm ASVW_CONSENSUS for N processes using only the *asvwrite* operation and registers. Subsequently, we prove that there is no wait-free consensus algorithm for $N+1$ processes using only the *asvwrite* operation and registers. The rest of this section presents a complete proof of the exact consensus number.

Lemma 6.1: The single-*asvword* assignment has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$ for $B = tA, t \in \mathbb{N}^*$.

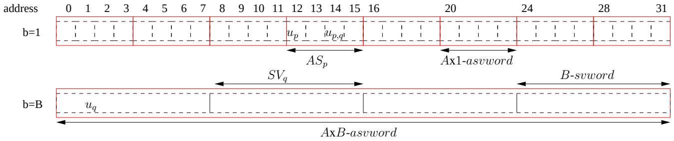
Proof: Since the *asvword* model is an extension of the *aiword* model, the *asvwrite* operation has consensus number at least $N = \lfloor \frac{A+1}{2} \rfloor$ (cf. Theorem 5.1). When the size b of *svword*s is the *same* for *all asvwrite*s, the *asvword* model degenerates to the *aiword* model. The *asvwrite* operation can achieve a higher consensus number when the size b of *svword*s is allowed to be different between *asvwrite*s, namely both Ax1-*asvwrite*s and AxB-*asvwrite*s are utilized.

However, we will prove by contradiction that when $B = tA$, the combination of Ax1-*asvwrite*s and AxB-*asvwrite*s does not provide any additional strength. Assume that there is a wait-free consensus algorithm $\mathcal{ALG}$ for N processes, where $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$, using *asvwrite*s and registers. Due to Lemma 3.1, $\mathcal{ALG}$ must have a critical configuration C^* and the critical operations op_i of processes p_i with different critical values must be write-operations. At the critical configuration C^*, assume that there are two processes p and q that have different critical values and use the Ax1-*asvwrite* and AxB-*asvwrite* as their critical operations to write to their 2W-unit $u_{p,q}$ (cf. Lemma 3.2), respectively. Since p must atomically write to its 1W-unit u_p and 2W-unit $u_{p,q}$ using an Ax1-*asvwrite*, the two units must be located in the same Ax1-*asvword* AS_p that starts and ends at addresses $kA, k \in \mathbb{N}$, and $(k+1)A-1$, respectively, due to the memory alignment restriction as illustrated in Figure 5(a). Two rows $b = 1$ and $b = B$ in Figure 5(a) illustrate the memory alignment corresponding to the size b of *svword*s on the same 32 consecutive memory units with addresses from 0 to 31. Let $k = at + b$, where $a, b \in \mathbb{N}, t = \frac{B}{A}, b \leq t - 1$. Since AxB-*asvwrite* uses B-*svword*s as its working units, q whose write-operation is AxB-*asvwrite*, must write to the B-*svword* SV_q that overlaps $u_{p,q}$. The starting address and ending address of SV_q are aB and $(a+1)B - 1$, respectively, due to the memory alignment restriction (cf. Figure 5(a)). We have $aB = atA \leq kA$ and $(a+1)B = (ta+t)A \geq (k+1)A$, namely 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 3.2 for process p. ■

Lemma 6.2: The single-*asvword* assignment has consensus number at least

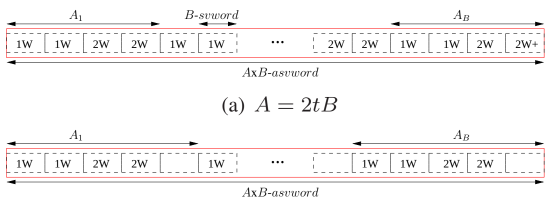
$$m = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, t \in \mathbb{N}^* \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B, t \in \mathbb{N}^* \end{cases} \tag{5}$$

Proof: We prove this lemma by presenting a wait-free consensus algorithm ASVW_CONSENSUS for m processes using only *asvwrite*s and registers. The ASVW_CONSENSUS algorithm is similar to the AIW_CONSENSUS algorithm (Al-



(a) $B = 2A : A = 4, B = 8$.

Fig. 5. An illustration for the proof of Lemma 6.1.



(a) $A = 2tB$

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

Fig. 6. An illustration for the proof of Lemma 6.2.

gorithm 4) except that the AIWRITE operations used at lines 3I and 13I are replaced by the ASVWRITE operations.

Similarly to the proof of Lemma 5.2, we will prove that in round m: i) p_m's ASVWRITE can atomically write to *only* m units $\{U_m^m, U_{m,1}^m, \ldots, U_{m,m-1}^m\}$ (line 3I) and ii) p_i's ASVWRITE can atomically write to *only* 2 units $\{U_i^m, U_{m,i}^m\}$, where $1 \leq i < m$ (line 13I). We will present a distribution of the units on an AxB-*asvword* that satisfies all the requirements. Figures 6(a) and 6(b) illustrate this proof.

- $A = 2tB$: Due to the memory alignment, an AxB-*asvword* AS is always aligned with B Ax1-*asvword*s called $A_1, \ldots, A_B$, of which each can be atomically written by an Ax1-*asvwrite* (cf. Figure 6(a)).

  The units U_i^m and $U_{m,i}^m$ of each process $p_i, i < m$, are located in the same $A_l, 1 \leq l \leq B$, so that p_i can atomically write to *only* its two units using an Ax1-*asvwrite* operation. This makes the operation satisfy the requirement ii) for p_i. Each B-*svword* located in A_l contains either units U_i^m or units $U_{m,i}^m$ but not both, which allows p_m to modify only B-*svword*s with units $U_{m,i}^m$ and keep units U_i^m untouched by using one AxB-*asvwrite*. The distribution of units U_i^m and $U_{m,i}^m$ on A_l is shown in Figure 6(a), where t B-*svword*s labeled "1W" contain only units U_i^m and t B-*svword*s labeled "2W" contain only units $U_{m,i}^m$. Since each process p_i requires one unit U_i^m and one unit $U_{m,i}^m$, the number of processes that A_l can support is $n_l = tB$. Consequently, the number of units $U_{m,i}^m$ in A_l that can be written atomically by p_m's ASVWRITE is $n_l = tB$.

  The last *svword* labeled "2W+" in Figure 6(a) contains also p_m's unit U_m^m, which, together with units $U_{m,i}^m$, will be written atomically by p_m's AxB-*asvwrite*. Therefore, the number of units $U_{m,i}^m$ located in A_B is $n_B = tB - 1$. The total number of units $U_{m,i}^m$ to which p_m can write atomically together with its 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 *only* its unit U_m^m and $(m-1)$ units $U_{m,i}^m, 1 \leq i \leq (m-1)$, using an ASVWRITE. This makes the operation satisfy the requirement i) for p_m.

- $A = (2t+1)B$: Similarly, the units U_i^m and $U_{m,i}^m$ of each process $p_i, i < m$, are located in the same $A_l, 1 \leq l \leq B$, so that p_i can atomically write to *only* its two units using an Ax1-*asvwrite*. This makes the operation satisfy the requirement ii) for p_i. For each A_l with $(2t+1)$ B-*svword*s, t B-*svword*s labeled "1W" contain only units U_i^m, t B-*svword*s labeled "2W" contain only units $U_{m,i}^m$ and the other B-*svword* without label is not used (cf. Figure 6(b)). Therefore, the number of processes that A_l can support is $n_l = tB$. It follows that the number of units $U_{m,i}^m$ in A_l that can be written atomically by p_m's ASVWRITE is $n_l = tB$.

  Unlike in the case of $A = 2tB$, in this case p_m's unit U_m^m can be located in the unused B-*svword* of A_B and thus the number of units $U_{m,i}^m$ located in A_B is $n_B = tB$ as in other $A_l, 1 \leq l < B$. The total number of units $U_{m,i}^m$ to which p_m can write atomically together with its 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 only its unit U_m^m and $(m-1)$ units $U_{m,i}^m, 1 \leq i \leq (m-1)$, using an ASVWRITE. This makes the operation satisfy the requirement i) for p_m.

■

Lemma 6.3: The single-*asvword* assignment has consensus number at most

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

Proof: We prove this lemma by contradiction. Assume that there is a wait-free consensus algorithm $\mathcal{ALG}$ for N processes where $N > M$. Due to Lemma 3.1, $\mathcal{ALG}$ must have a critical configuration C^* and the critical operations op_i of processes p_i with different critical values must be write-operations. At the critical configuration C^*, we divide N processes into $k \geq 2$ subsets $s_1, \ldots, s_k$ each of which consists of processes with the same critical value. Let $n_1, \ldots, n_k$ 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 critical value is different from that of $\left(\sum_{l \neq i} n_l\right)$ processes in the $(k-1)$ other subsets, p_j^i's critical operation must atomically write to its 1W-unit and $\left(\sum_{l \neq i} n_l\right)$ 2W-units (cf. Lemma 3.2). The operation determines the relative ordering between p_j^i and the $\sum_{l \neq i} n_l$ other processes with critical values different from p_j^i's: if p_j^i's operation precedes p_*^l's, $l \neq i$, p_j^i is considered preceding p_*^l.

Since N is larger than $\lfloor \frac{A+1}{2} \rfloor$, the consensus number of single-*aiword* assignments (or A-*aiwrite*s), processes in the $\mathcal{ALG}$ algorithm must use both AxB-*asvwrite*s and Ax1-*asvwrite*s. Note that if all processes use only either AxB-*asvwrite* or Ax1-*asvwrite*, the *asvword* model degenerates into the *aiword* model. Let p_b^a and $p_{b'}^{a'}, a \neq a'$, be the processes that use an AxB-*asvwrite* and an Ax1-*asvwrite* as their critical operations to modify their 2W-unit $u_{b,b'}^{a,a'}$, respectively. Let AS be the AxB-*asvword* written by p_b^a's AxB-*asvwrite*. 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-*asvword*s that contains this unit. Let A_c be this Ax1-*asvword*. Since p_j^i and $p_{j'}^{i'}$ belong to different subsets, they have different critical 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-*asvwrite*, its 2W-unit $u_{j,j'}^{i,i'}$ must be located in AS since AxB-*asvwrite* cannot atomically write to two units belonging to two different AxB-*asvword*s. If p_j^i uses Ax1-*asvwrite*, its 2W-unit $u_{j,j'}^{i,i'}$ must be located in $A_c \in AS$ since Ax1-*asvwrite* cannot atomically write to two units belonging to two different Ax1-*asvword*s. 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 deduce 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. Indeed, since p_b^a's 1W-unit u_b^a is located in AS, all $\left(\sum_{l \neq a} n_l\right)$ processes p_j^l in the $(k-1)$ other 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. It follows that all 2W-units written by two 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 $\mathcal{M}$ of the 1W- and 2W-units used by the N processes. Using similar argument to the proof of Lemma 5.4, it follows that $\mathcal{M}$ achieves the minimum $(2N-1)$ when there are only two subsets: one containing $(N-1)$ processes p with the same critical value and the other containing only 1 process q.

Lastly, we prove that N cannot be larger than M defined in Equation 6.

If q uses an Ax1-*asvwrite*, let A_q be the Ax1-*asvword* written by q. A_q contains q's 1W-unit u_q and all $(N-1)$ 2W-units $u_{p,q}$. We prove that the number of A_q's 1-*svword*s required by a process p is at least 2. Indeed, if p uses Ax1-*asvwrite*, both its 1W-unit u_p and 2W-unit $u_{p,q}$ must be located in A_q since Ax1-*asvwrite*s cannot atomically write to two 1-*svword*s located in different Ax1-*asvword*s. Therefore, p requires two 1-*svword*s of A_p. If p uses an AxB-*asvwrite*, its 2W-unit $u_{p,q}$ must be B-*svword* so that p's AxB-*asvwrite* 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-*svword*s of A_q. That means the number of A_q's 1-*svword*s required by N processes including q is at least $2(N-1)+1 = 2N-1$. Since the Ax1-*asvword* A_q has A 1-*svword*s, it follows that $N = \lfloor \frac{A+1}{2} \rfloor$, a contradiction to the hypothesis that $N > M$.

If q uses an AxB-*asvwrite*, let AS be the AxB-*asvword* written by q. Due to the memory alignment, the AxB-*asvword* AS is aligned with B Ax1-*asvword*s called $A_l, 1 \leq l \leq B$ (cf. Figure 6(a)). 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-*svword*s (instead of B-*svword*s). It follows that all $(N-1)$ processes $p \neq q$ must use Ax1-*asvwrite*. Each process p must have its 1W-unit u_p and 2W-unit $u_{p,q}$ located in the same Ax1-*asvword* 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-*asvword* A_l can accommodate their 1W- and 2W-units is $n_l = tB$. For the Ax1-*asvword* $A_{l'}$ that contains q's 1W-unit u_q, $n_{l'} = \lfloor \frac{2tB-1}{2} \rfloor = tB - 1$. Therefore, the maximum number of processes (including q) that the AxB-*asvword* AS can support is $N = (B-1)tB + (tB-1) + 1 = tBB = \frac{AB}{2} = M$, a contradiction to the hypothesis that $N > M$.
- If $A = (2t+1)B$, we prove that each Ax1-*asvword* $A_l, 1 \leq l \leq B$, 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. It follows that there is at least one of A_l's B-*svword*s that contains *both* 1W-units $u_r, r \neq q$, and 2W-units $u_{r',q}$, where r' can be r (cf. Figure 6(b)). Since q writes to the 2W-units $u_{r',q}$ using an AxB-*asvwrite*, which uses B-*svword*s as its basic units, q will overwrite $u_r, r \neq q$, a contradiction to the first requirement of Lemma 3.2 for process r.

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

■

As a remark, we can apply a similar analysis and get the same upper bound $M = \frac{AB}{2}$ (when $A > B$) for a variant *asvword** of the Axb-*asvword* model where $b = 2^c, c = 0, 1, \ldots, \log_2 B$, and A, B are powers of 2. Since Lemma 6.2 is applicable to the *asvword** model, the *asvword** model has consensus number exactly $N = \frac{AB}{2}$ when $A > B$.

For the *asvword** model, Lemma 6.1 implies that in case $A \leq B$, any two operations Axb_1-*asvwrite* and Axb_2-*asvwrite*, where $\frac{b_1}{b_2} = tA, t \in \mathbb{N}^*$, cannot be used as critical operations in any wait-free consensus algorithm $\mathcal{ALG}$ for N processes where $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$. That means any two operations Axb_1-*asvwrite* and Axb_2-*asvwrite* ($b_1 \geq b_2$) used as critical operations in $\mathcal{ALG}$ must satisfy $\frac{b_1}{b_2} < A, \forall b_1, b_2 = 2^{min_c}, \ldots, 2^{max_c}$. Let $B' = 2^{max_c - min_c}$, we have $b = 2^0, 2^1, \ldots, B'$, and $B' < A$, the case in which the *asvword** model has consensus number exactly $N = \frac{AB'}{2}$ as argued above. When $B' = \frac{A}{2}$, N reaches its maximum of $\frac{A^2}{4}$. That means the *asvword** model has consensus number exactly $\frac{A^2}{4}$ when $A < B$.

VII. Conclusions

This paper has investigated the consensus number of the new memory access mechanisms implemented in current graphics processor architectures. We have first defined three new memory access models to capture the fundamental features of the new memory access mechanisms, and subsequently have proven the synchronization power of these models. The first model is the size-varying word model called *svword*, which has consensus number exactly 3. The second model is the aligned-inconsecutive word model called *aiword*, which has consensus number exactly $\lfloor \frac{A+1}{2} \rfloor$. The second model is stronger than the first model when $A \geq 7$. The third model is the combination of the first and second models called *asvword*, which is the strongest model. The third model has consensus number N, where

$$N = \begin{cases} \frac{AB}{2}, & \text{if } A = 2tB, \text{ where } t, B \in \mathbb{N}^*, B \geq 2 \\ \frac{(A-B)B}{2} + 1, & \text{if } A = (2t+1)B \\ \lfloor \frac{A+1}{2} \rfloor, & \text{if } B = tA, \text{ where } t, A \in \mathbb{N}^*, A \geq 2 \end{cases} \tag{7}$$

The results of this paper show that the new memory access mechanisms can facilitate strong synchronization between the threads of multicore architectures, without the need of synchronization primitives other than reads and writes.

Acknowledgements The authors wish to thank the anonymous reviewers for their helpful and thorough comments on the earlier version of this paper. Phuong Ha's and Otto Anshus's work was supported by the Norwegian Research Council (grant numbers 159936/V30 and 155550/420). Philippas Tsigas's work was supported by the Swedish Research Council (VR) (grant number 37252706).

References

[1] P. H. Ha, P. Tsigas, and O. J. Anshus, "The synchronization power of coalesced memory accesses," in *Proc. of the Intl. Symp. on Distributed Computing (DISC)*, 2008, pp. 320–334.

[2] 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*, vol. 26, no. 1, pp. 80–113, 2007.

[3] *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, version 2.1.* NVIDIA Corporation, December 2008.

[4] *Cell Broadband Engine Architecture, version 1.01.* IBM, Sony and Toshiba Corporations, 2006.

[5] M. Herlihy, "Wait-free synchronization," *ACM Transaction on Programming and Systems*, vol. 11, no. 1, pp. 124–149, Jan. 1991.

[6] P. Jayanti, "Robust wait-free hierarchies," *J. ACM*, vol. 44, no. 4, pp. 592–614, 1997.

[7] 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)*, 1996, pp. 233–242.

[8] 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*, 1996, pp. 213–222.

[9] P. Jayanti and S. Khanna, "On the power of multi-objects," in *WDAG '97: Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997, pp. 320–332.

[10] E. Ruppert, "Consensus numbers of multi-objects," in *Proc. of Symp. on Principles of Distributed Computing (PODC)*, 1998, pp. 211–217.

[11] P. H. Ha, P. Tsigas, and O. J. Anshus, "Wait-free programming for general purpose computations on graphics processors," in *Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, 2008, pp. 1–12.

[12] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *J. ACM*, vol. 34, no. 1, pp. 77–97, 1987.

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[14] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[15] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *Computer*, vol. 29, no. 12, pp. 66–76, 1996.

[16] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess progranm," *IEEE Trans. Comput.*, vol. 28, no. 9, pp. 690–691, 1979.

[17] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. John Wiley and Sons, Inc., 2004.

[18] I. Castano and P. Micikevicius, "Personal communication," *NVIDIA*, 2008.

[19] E. Ruppert, "Determining consensus numbers," in *Proc. of Symp. on Principles of Distributed Computing (PODC)*, 1997, pp. 93–99.

[20] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[21] H. Buhrman, A. Panconesi, R. Silvestri, and P. Vitanyi, "On the importance of having an identity or, is consensus really universal?" *Distrib. Comput.*, vol. 18, no. 3, pp. 167–176, 2006.

Phuong H. Ha received the BEng degree from the Department of Information Technology, Ho-Chi-Minh City University of Technology, Vietnam and the Ph.D. degree from the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden. Currently, he is a postdoc at the Department of Computer Science, University of Tromsø, Norway. His research interests are parallel/distributed computing and systems, including efficient fault-tolerant inter-process coordination mechanisms, concurrent data structures and parallel programming (www.cs.uit.no/∼phuong).

Philippas Tsigas 's research interests include concurrent data structures for multiprocessor systems, communication and coordination in parallel systems, fault-tolerant computing, mobile computing and information visualization. He received a BSc in Mathematics from the University of Patras, Greece and a Ph.D. in Computer Engineering and Informatics from the same University. Philippas was at the National Research Institute for Mathematics and Computer Science, Amsterdam, the Netherlands (CWI), and at the Max-Planck Institute for Computer Science, Saarbrücken, Germany, before. At present he is a professor at the Department of Computing Science at Chalmers University of Technology, Sweden (www.cs.chalmers.se/∼tsigas).

Otto J. Anshus is a professor of computer science at the University of Tromsø. His research interests include operating systems, parallel and distributed architectures and systems, scalable display systems, data-intensive computing, high-resolution visualizations, and human-computer interfaces. He is a member of the IEEE Computer Society, the ACM, and the Norwegian Computer Society. Contact him at otto.anshus@uit.no.