The Synchronization Power of Coalesced Memory Accesses

Phuong Hoai Ha[1], Philippas Tsigas[2], and Otto J. Anshus[1]

[1] University of Tromsø, Department of Computer Science, Faculty of Science, NO-9037 Tromsø, Norway,
{phuong,otto}@cs.uit.no
[2] Chalmers University of Technology, Department of Computer Science and Engineering, SE-412 96 Göteborg, Sweden,
tsigas@chalmers.se

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 processing cores access the shared memory, directly influence the synchronization capabilities of the multicore processors. 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 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 mechanisms can facilitate strong synchronization between 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 mechanisms have consensus numbers up to sixteen.

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 [13]. Unlike previous GPU architectures, which are single-instruction multiple-data (SIMD), recent GPU architectures (e.g. Compute Unified Device Architecture (CUDA) [2]) 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 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 size and their size (in bytes) is 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 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 [2]. The access coalescence takes place even if some of the threads do not actually access memory. It is well-known that memory access mechanisms, by devising how processing cores access the shared memory, directly influence the synchronization capabilities of multicore processors. 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 [10]. 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. For hard real-time systems, it has been shown that any object with consensus number n is universal[3] for any numbers of processes running on n processors [14]. For systems that allow processes to simultaneously access m objects of type T in one atomic operation (or multi-object operation), upper and lower bounds on the consensus number of the multi-object called type T^m have been provided for the base type T with *consensus number greater than or equal to two* [4, 11, 16]. 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 $m > 1$ [10].

Note that the aforementioned CUDA coalesced memory accesses are neither the atomic m-register assignment [10] nor the multi-object types [4, 11, 16]. They are not the atomic m-register assignment since they do not allow processes 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 2). 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 less than two*.

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

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 mechanisms can facilitate strong synchronization between the threads of multicore processors, without the need of synchronization primitives other than reads and writes.

We first design 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 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 b is no longer restricted to be a power of 2 (e.g. type *float3* in [2]). 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 different size must be aligned from the address base of the memory. This memory alignment constraint 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 subset of bytes 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 B values of size 1 (instead of just one value of size B) to B consecutive memory locations. The 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 design two other models, the *aiword* and *asvword* models, to capture the fundamental features of the mechanism. The mechanism coalesces simultaneous read/write instructions by each thread of a SIMD multiprocessor into a *single* aligned memory access even if some of the threads do not actually access memory [2]. This allows each SIMD multiprocessor (or process) to atomically write to an arbitrary subset of the aligned memory units that can be written by a single coalesced memory access. 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 [10] due to the memory alignment restriction[4]. 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

[4] In this paper, we use term "single" in *single-*word assignment* when we want to emphasize that the assignment is not the *multiple* assignment [10].

instructions of different coalesced global memory accesses can access words of different size [2].

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. 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 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 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 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 4).
- 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 *sword*s of the same size $b, b \in \{1, B\}$ (cf. Section 5). 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 up to 1.1) in which $A = 16$ and $B = 2$, the consensus number of the *asvword* model is sixteen.

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

Due to space limitations, we present here only intuitions behind the consensus number results. Complete proofs of the results can be found in the full version of this paper [9].

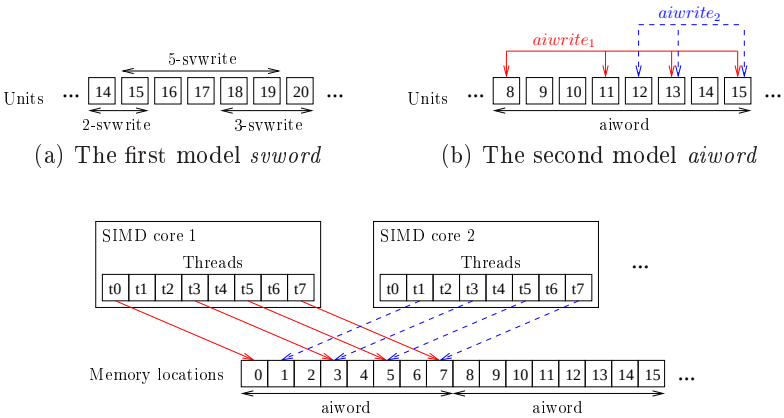
2 Models

Before describing the details of each of the three new memory access models, we present the common properties of all these three models. The shared memory in the three new models is sequentially consistent [3, 12], which is weaker than the linearizable one [5] assumed in most of the previous research on the synchronization power of the conventional memory access models [10]. 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 (without overwriting other unintended bytes/bits). These memory models address individual memory units. Memory is organized so that a group of n consecutive memory units called *word* can be stored or retrieved in a *single* basic write or read operation, respectively, and n is called *word size*. Words of size n must always start at addresses that are multiples of n, which is called *alignment restriction* as defined in the conventional computer architecture.

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 can be any integer between 1 and an upper bound B and is called *svword size*. 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. types *float1, float2, float3* and *float4* in [2], 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.

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.

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*



(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.

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 [2]. The CUDA architecture can be generalized to an abstract model of a MIMD[5] chip with multiple SIMD cores sharing memory. Each core can process A threads simultaneously in a SIMD manner, but different cores can simultaneously execute different instructions. The instance of a program that is being sequentially executed by one SIMD core is called *process*. Namely, each process consists of A parallel threads that are running in SIMD manner. The process accesses the shared memory using the CUDA memory access models. In CUDA, the simultaneous global memory accesses by each thread of a SIMD core during the execution of a single read/write instruction can be coalesced into a *single* aligned memory access. The coalescence happens even if some of the threads do not actually access memory (cf. [2], Figure 5-1). This allows a SIMD core (or a process consisting of A parallel threads running in a SIMD manner) to atomically access multiple memory locations that are not at consecutive addresses.

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

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

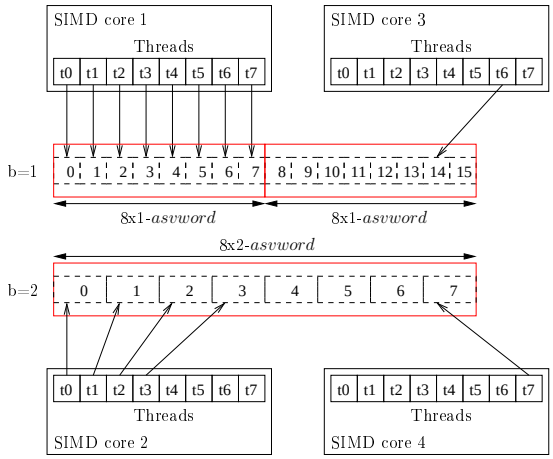


Fig. 2. An illustration for the *asvword* model.

the CUDA architecture allows threads from different SIMD cores to communicate through the global shared memory [7].

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* is 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. Let Axb-*asvword* be the *asvword* that is composed of A *svword*s of which each 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 $B < A$). (At the moment, CUDA supports the *atomic* coalesced memory access to only words of size 4 and 8 bytes (i.e. only *svword*s consisting of 1 and 2 *units* in our definition), cf. Section 5.1.2.1 in [2]). For the sake of simplicity, we assume that $b \in \{1, B\}$ holds. A more general model with $b = 2^c, c = 0, 1, \cdots, \log_2 B$, can be established from this model. 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 *asv-*

word 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* assignment in the second model, which can atomically write to at most 8 units (or A units), the *asvwrite* assignment 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).

Terminology This paper uses the conventional terminology from bivalency arguments [8, 10, 15]. 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*. The *critical value* of a process is the value that would get decided if that process takes the next step after the critical configuration.

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 *asvword*. It has been proven that such an 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 $\mathcal{O}$ [10]. The object $\mathcal{O}$ consists of *memory units*.

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-units for short) $u_{i,j}$ written only 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 [10]. □

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 $Init$ 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 = set $\{v_0, \cdots, v_4\}$ 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 1**: 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($i, first_{ref}$); // $first_{ref}$ is the reference to $first$
6V: **return** $PROPOSAL[winner]$

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 to utilize the size-variation feature of the *svwrite* operation. Since *b-svwrite* can atomically write b values of size 1 unit (instead of just one value of size b units) to b consecutive memory units, keeping the size of values to be atomically written as small as 1 unit will maximize the number of processes for which *b-svwrite*, together with registers, can solve the consensus problem. Unlike the seminal wait-free consensus algorithm using the m-word assignment by Herlihy [10], 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 [6].

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 (lines 2SF and 4SF) precedes that of the other process (lines 6SF-11SF).

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

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** ($WR_1[1] = Higher$ and $i = 0$) or ($WR[1] = Lower$ 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** $(\neg i)$;
12SF: **end if**

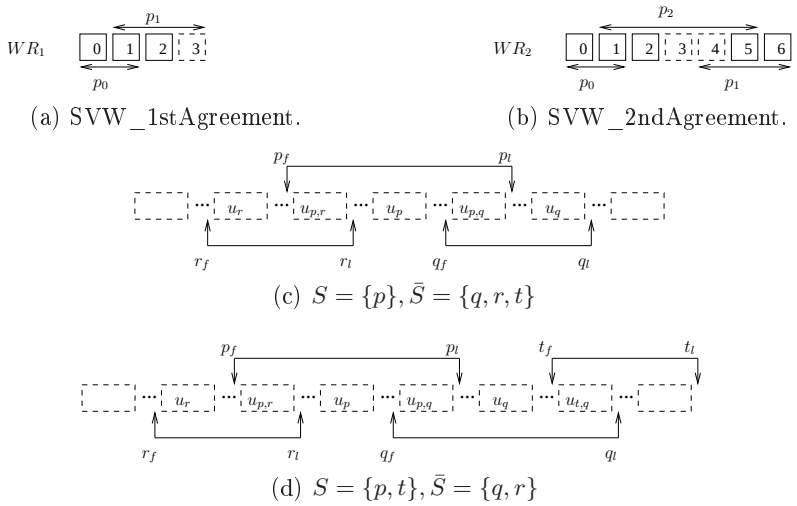
Algorithm 3 SVW_SECONDAGREEMENT(i: index; $first_{ref}$: reference) invoked by process $p_i, i \in \{0,1,2\}$

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**

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 critical 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) and returns p_2's proposal value otherwise.

Units written by processes' SVWRITE 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 $\{u_0, \cdots, u_6\}$ 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 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 (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



(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 5.

units $\{u_4, u_5, u_6\}$ and the five middle units $\{u_1, \cdots, u_5\}$, 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 2. *The* SVW_SecondAgreement *procedure returns index 2 if p_2 precedes both p_0 and p_1. Otherwise, it returns index $first$.*

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

Lemma 5. *The single-*svword *assignment has consensus number at most 3, $\forall B \geq 2$.*

Proof. (*Intuition; the full proof is in [9]*) 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. At the critical configuration of the algorithm, 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 (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).

Figures 3(c) and 3(d) illustrate 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. □

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

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 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 1W- and 2W-units that must be written atomically by the *aiword* operation (cf. Lemma 9). 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 6. *All correct processes*[6] p_i *agree on the same value in round* r_j*, where* $1 \leq i \leq j \leq N$.

With the assumption that AIWrite can atomically write to p_j's units at line 2I and p_i's units at line 11I, it follows directly from Lemma 6 that all the N processes will achieve an agreement in round r_N.

[6] A *correct* process is a 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;
$U^r_{i,j}$: the 2W-unit written only by processes p_i and p_j in round r. U^r_i: the 1W-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] ← buf_i; // Initialized p_i's agreed value for round i
2I: AIWRITE({U^i_i, U^i_{i,1}, ··· , U^i_{i,i−1}}, {Higher, Higher, ··· , Higher}) // Atomic assignment
3I: for k = 1 to (i − 1) do
4I:     if U^i_k ≠⊥ and U^i_{i,k} = Higher then
5I:         A^i[i] ← 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] ← A^{j−1}[i]; // Initialized p_i's agreed value for round j
11I:    AIWRITE({U^j_i, U^j_{j,i}}, {Lower, Lower}; // Atomic assignment
12I:    if U^j_j ≠⊥ and U^j_{j,i} = Lower then
13I:        WinnerIsJ ← true; // Check if p_j precedes p_k, ∀k < j.
14I:        for k = 1 to j − 1 do
15I:            if U^j_k ≠⊥ and U^j_{j,k} = Higher then
16I:                WinnerIsJ ← false; // p_k precedes p_j;
17I:                break;
18I:            end if
19I:        end for
20I:        if WinnerIsJ = true then
21I:            A^j[i] ← A^j[j]; // p_j precedes p_k, ∀k < j, ⇒ 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];
```

Lemma 7. *The* AIW_CONSENSUS *algorithm is wait-free and can solve the consensus problem for* $N = \lfloor \frac{A+1}{2} \rfloor$ *processes.*

Proof. (*Intuition; the full proof is in [9]*) 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 6, 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 units at line 2I and p_i's units at line 11I. Indeed, since $N = \lfloor \frac{A+1}{2} \rfloor$, an A-unit *aiword* (or A-*aiword* for short) can accommodate both $(N-1)$ 2W-units $U^N_{N,i}, 1 \leq i < N$, and N 1W-units $U^N_k, 1 \leq k \leq N$, used in round r_N. 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*[7] its 1W and 2W units. □

Lemma 8. *The single-*aiword *assignment has consensus number at least* $\lfloor \frac{A+1}{2} \rfloor$.

[7] "Only" here means to leave other units untouched.

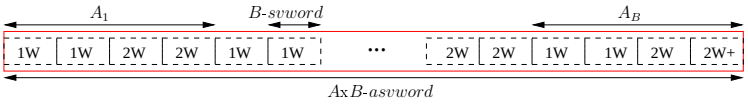


Fig. 4. An illustration for grouping units in the *asvword* model.

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

Proof. (*Intuition; the full proof is in [9]*) 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 two subsets S and $\bar{S}$ where S consists of processes with the same critical value called V and $\bar{S}$ consists of processes with critical values different from V. Let $|S|$ and $|\bar{S}|$ to be the sizes of the subsets, we have $|S| + |\bar{S}| = N$. Due to the memory alignment restriction, all the 1W-units and 2W-units used by critical assignments in the $\mathcal{ALG}$ algorithm must be located in the same *A-aiword* called AI. Let M be the number of the 1W-/2W-units, we have $M \leq A$.

Since $\mathcal{ALG}$ is a wait-free consensus algorithm for N processes, it follows from Lemma 1 that there are N 1W-units and $|S| \cdot |\bar{S}|$ 2W-units, i.e. $M = N + |S| \cdot |\bar{S}|$. Since $1 \leq |S| \leq (N-1)$, $M \geq (2N-1)$. Since $N \geq \lfloor \frac{A+1}{2} \rfloor + 1$ due to the hypothesis, $M \geq (A+1)$ must hold. This contradicts the requirement $M \leq A$. □

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

5 Consensus number of the *asvword* model

The intuition behind the higher consensus number of the *asvword* model compared with the *aiword* model (cf. Equation (1)) 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*. To prevent p_N from overwriting unintended units (as illustrated by SIMD core 4 in Figure 2), each *B-svword* located in $A_l, 1 \leq l \leq B$, contains either 1W-units or 2W-units but not both as illustrated in Figure 4, where *B-svword*s labeled "1W" contain only 1W-units and *B-svword*s labeled "2W" contain only 2W-units. This allows p_N to atomically write to only *B-svword*s with 2W-units $U^N_{N,i}$ (and keep 1W-unit $U^N_i, i \neq N$, untouched) using *AxB-asvwrite*. For each process $p_i, i \neq N$, its 1W-unit U^N_i and 2W-unit $U^N_{N,i}$ 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*. A complete proof of the exact consensus number can be found in the full version of this paper [9].

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. *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. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* John Wiley and Sons, Inc., 2004.
6. 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.
7. I. Castano and P. Micikevicius. Personal communication. *NVIDIA*, 2008.
8. 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.
9. P. H. Ha, P. Tsigas, and O. J. Anshus. The synchronization power of coalesced memory accesses. *Technical report CS:2008-68, University of Tromsø, Norway*, 2008.
10. M. Herlihy. Wait-free synchronization. *ACM Transaction on Programming and Systems*, 11(1):124–149, Jan. 1991.
11. P. Jayanti and S. Khanna. On the power of multi-objects. In *WDAG '97: Proceedings of the 11th International Workshop on Distributed Algorithms*, pages 320–332, 1997.
12. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess progranm. *IEEE Trans. Comput.*, 28(9):690–691, 1979.
13. 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.
14. 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.
15. E. Ruppert. Determining consensus numbers. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, pages 93–99, 1997.
16. E. Ruppert. Consensus numbers of multi-objects. In *Proc. of Symp. on Principles of Distributed Computing (PODC)*, pages 211–217, 1998.