



Taking a Bigger Byte: Wide Hashless Chunking for Data Deduplication

Sreeharsha Udayashankar
University of Waterloo

Abdelrahman Baba
University of Waterloo

Samer Al-Kiswany
University of Waterloo

Abstract

Content-defined Chunking (CDC) algorithms are critical to data deduplication and can be categorized into two families: hash-based and hashless. Hashless chunking algorithms are faster than their hash-based counterparts, especially when accelerated using the Single Instruction Multiple Data (SIMD) capabilities of modern CPUs. However, they achieve lower deduplication space savings on certain datasets.

We present WideCDC, a technique to improve the deduplication efficiency of hashless CDC algorithms by using *wide regions* of multiple bytes to determine chunk boundaries instead of individual bytes. Existing SIMD acceleration techniques for hashless CDC miss chunk boundaries when using such wide regions, as they do not account for overlaps. To address this while preserving performance, WideCDC uses novel *Accumulated Extreme Byte Searches* and *Gated Range Scans*. By evaluating WideCDC on real-world datasets, we show that it improves the deduplication ratios of hashless chunking algorithms up to 3.07× while achieving 2.04× higher chunking throughput than existing vector-accelerated techniques.

1 Introduction

Data generation rates are skyrocketing, with global data production expected to exceed 525 zettabytes in 2029 [1]. Global data storage capacity has not kept pace, prompting cloud storage providers to adopt data-reduction mechanisms [2] to manage the data influx.

Previous studies [3, 4] show that a large amount of redundancy exists within data stored on the cloud. Data deduplication [3,5] is the primary mechanism used by cloud storage providers to identify and eliminate these duplicates, retaining only unique data to conserve storage space. Deduplication consists of three phases [6]; incoming data is first divided into chunks, chunks are then fingerprinted with collision-resistant hashing algorithms [7], and fingerprints are finally compared against those previously seen to identify duplicates.

Data is divided into chunks using Content-Defined Chunking (CDC) algorithms, which come in two families: hash-based and hashless [8, 9]. Hash-based CDC [10–12] uses computationally expensive rolling hash algorithms to determine chunk boundaries. Hashless algorithms [13–15] instead treat every byte independently and determine chunk boundaries using byte minima/maxima, improving throughput.

Previous studies show that hashless algorithms achieve lower deduplication efficiency than their hash-based counterparts on certain datasets [9,16]. We identified that this is mainly due to the limited range of values that can be held within a single byte, leading to minima/maxima gravitating towards `0x00`/`0xFF` on these datasets. We address this by augmenting hashless CDC to use *wide regions*, i.e., 2-byte `words`, 4-byte `dwords`, and 8-byte `qwords` (§4). Our evaluation shows that this practical change significantly improves the efficiency of hashless CDC algorithms.

On the other hand, recent efforts [9, 16, 17] have tried to accelerate CDC algorithms using the Single Instruction Multiple Data (SIMD) vector instruction sets [18] available on modern CPUs. SS-CDC [17] accelerated hash-based algorithms and VectorCDC [9, 16] accelerates hashless algorithms. VectorCDC achieves 21×–46× higher throughput than SS-CDC's accelerated hash-based algorithms by using *Tree-based Extreme Byte Searches* and *Range Scans*. However, VectorCDC's acceleration methods only support byte-based hashless algorithms and not wide regions (§2).

We therefore present WideCDC, a novel method to improve the deduplication efficiency of hashless CDC. WideCDC uses `words`, `dwords`, and `qwords` to determine chunk boundaries, improving deduplication efficiency. WideCDC includes two novel vector-acceleration methods to support wide regions: *Accumulated Extreme Byte Searches* and *Gated Range Scans* (§4). Our evaluation (§5) shows that WideCDC improves the deduplication efficiency of hashless CDC algorithms up to 3.07× on datasets where they traditionally underperform, while achieving 2.04× higher throughput than the state-of-the-art VectorCDC [9, 16] on all datasets.

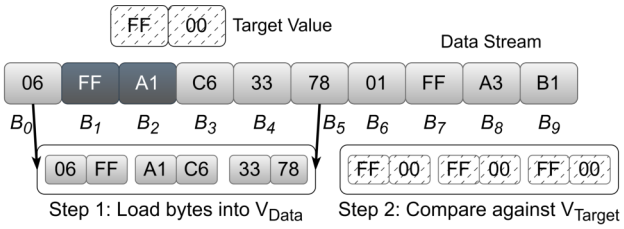


Figure 1: Naively running *Range Scans* with 2-byte `words`

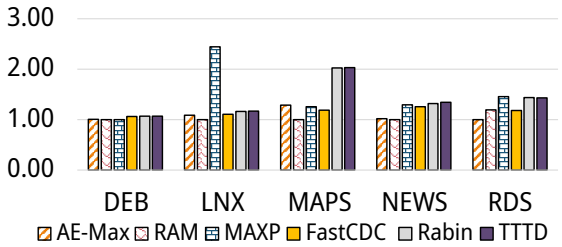


Figure 2: Normalized DERs of CDC algorithms on real-world datasets. Hashless algorithms are shown with patterned bars.

2 Background and Related Work

Data deduplication consists of three major phases [6]; a) *Data Chunking* where incoming data is divided into chunks, b) *Fingerprinting* where chunks are hashed using collision resistant hashing algorithms like SHA-256 [7] and compared, and c) *Data Storage* where unique chunks are stored and their fingerprints added to a database.

Metrics. Deduplication Elimination Ratio (DER) [10, 12] is a measure of the overall space conserved by deduplication:

$$DER = \frac{Original\ Size}{Deduplicated\ Size} \quad (1)$$

Data Chunking. Data chunking and fingerprinting are traditionally the most compute-intensive phases in the deduplication pipeline [5]. As fingerprinting has been accelerated using GPUs [19, 20] and faster algorithms [21], the bottleneck has moved towards data chunking [16]. Data chunking relies on Content-Defined Chunking (CDC) algorithms [10–15, 22], which are categorized as hash-based or hashless [9].

Hash-based algorithms [10–12, 22] slide a fixed-size window over the data, recomputing hashes efficiently with rolling hash functions. When the hash value of the window's contents satisfies a predefined constraint, they insert a chunk boundary, creating a new data chunk. Note that, unlike fingerprinting, these hashing algorithms are not collision-resistant. Hashless CDC algorithms [13–15] instead use local minima/maxima to insert chunk boundaries; for instance, a boundary may be inserted whenever a local maximum is detected for a pre-specified window size [15]. As these do not involve rolling hashes, they are typically faster than hash-based algorithms. However, they achieve lower deduplication space savings on certain datasets [9].

Vector Acceleration. Recent efforts [9, 16, 17] focus on using vector instructions [18] to accelerate CDC algorithms. SS-CDC [17] targeted hash-based CDC, but achieves low speedups due to adjacent byte dependencies [9]. VectorCDC [9, 16] instead accelerates hashless algorithms, identifying two common phases that they use: *Extreme Byte Searches* and *Range Scans*.

VectorCDC's [9, 16] *Tree-based Extreme Byte Searches* find minima/maxima using a tree-based approach, where bytes are packed into vector registers and compared pairwise with *VMAX/VMIN* instructions. Resulting vector registers are again compared against each other, repeating until the extreme value is found. Their *Range Scans* pack multiple adjacent bytes into a vector register and use vector comparison operations (such as *GEQ*) to compare them against a target value. Although VectorCDC may appear compatible with a wide range of regions (`words` or larger), it misses boundaries because it ignores overlaps.

Let us consider a VectorCDC [9] *GEQ Range Scan* that looks for `bytes` greater than or equal to a target `byte`. This uses AVX-512 `mm512_cmpge_epu8` vector instructions to compare single `byte` values. To extend it to 2-byte `words`, we could simply use `mm512_cmpge_epu16` instructions instead. Figure 1 illustrates this attempt. Step 1 loads the bytes B_0 – B_5 into a vector register V_{Data} and Step 2 compares the `words` they form (`0x06FF`, `0xA1C6`, and `0x3378`) against the target `word` `0xFF00`. However, this comparison fails to account for the word `0xFFA1` formed by B_1 and B_2, leading to a missed match. Similar issues occur with `dwords` and `qwords`, as well as with *Tree-based Extreme Byte Searches*. WideCDC uses alternative vector-compatible methods instead to correctly accelerate wide hashless algorithms (§4).

Related Work. Several other efforts exist to optimize the other phases in data deduplication. Some use locality [23] and multiple threads [24, 25] to speed up data chunking, while others target fingerprinting [19, 20, 26, 27] and data storage [28]. WideCDC is compatible with all these efforts.

3 Motivation: Hash-based vs Hashless CDC

Figure 2 shows the Deduplication Elimination Ratios (DERs) exhibited by hash-based [10–12] and hashless [13–15] CDC algorithms across five real-world datasets described in §5, when targeting an average chunk size of 8KB. We use an Intel Emerald Rapids machine described in §5 for this experiment. As DERs vary per dataset, we have normalized them per dataset to improve plot clarity.

Figure 2 shows that while hashless algorithms match or outperform hash-based ones on datasets such as `DEB` and `LNX`, they achieve significantly lower DERs on datasets such as `MAPS`. Furthermore, on `NEWS` and `RDS`, some hashless algorithms (such as *AE-Max* [13]) lag behind hash-based ones. *This indicates that specific data patterns exist where hashless*

algorithms underperform in deduplication efficiency.

However, VectorCDC [9] showed that with vector-acceleration, hashless algorithms achieve 21×–46× higher throughput than hash-based ones. This creates a conundrum for production systems with these data patterns, forcing them to choose between higher DER with accelerated hash-based CDC or higher throughput with accelerated hashless CDC. WideCDC targets such systems, aiming to improve the DER of vector-accelerated hashless algorithms while retaining their throughput advantage.

4 Design

We analyzed real-world datasets with data patterns that cause hashless CDC to underperform, identifying the root cause as the limited range of values that can be held by an individual `byte` (§4.1). WideCDC instead uses *wide regions* (`words`, `dwords`, and `qwords`), while determining chunk boundaries.

However, the state-of-the-art SIMD acceleration technique for hashless chunking, VectorCDC [9], is incompatible with this approach (§2). To address this, we present two alternative vector-compatible approaches: simple (§4.2.1) with performance penalties and optimized (§4.2.2).

4.1 Using Wide Regions for Chunk Boundary Detection

Hashless chunking algorithms [13–15] use minima/maxima bytes to determine chunk boundaries. However, an individual byte can hold only a limited range of values, `0x00`–`0xFF`, resulting in the minima/maxima always leaning toward `0x00` and `0xFF` on certain datasets [29]. This reduces chunk boundary quality, leaving them vulnerable to *byte-shifting* [10].

To address this, WideCDC modifies hashless algorithms to use *wide regions*; `words`, `dwords`, and `qwords`. This significantly increases the range of values that these minima or maxima can hold and improves their distribution; for instance, 2-byte `words` can hold values between `0x0000`–`0xFFFF`. Improved distributions in turn result in better chunk boundary quality and deduplication efficiency.

We use this concept to rebuild four hashless CDC algorithms *AE-Max* [13], *AE-Min* [13], *RAM* [14], and *MAXP* [15], creating *WAE-Max*, *WAE-Min*, *WRAM*, and *WMAXP*. Each of these algorithms can operate on `words`, `dwords`, or `qwords`; we only discuss `words` here for simplicity.

- *WAE-Max*: Like *AE-Max* [13] for `bytes`, *WAE-Max* scans for a `word` that is larger than all its predecessors. When such a `word` is found and is also larger than a fixed number of `words` after it, a chunk boundary is inserted.
- *WAE-Min*: Like *AE-Min* [13] for `bytes`, *WAE-Min* looks for a `word` that is lower in value than all its predecessors. If it's also lower than a fixed number of `words` after it, a chunk boundary is inserted.

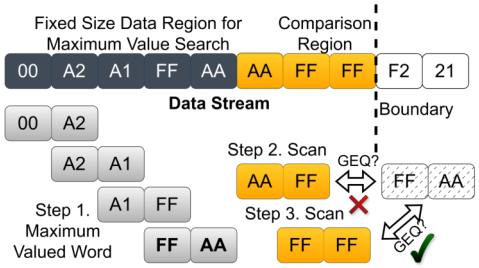


Figure 3: Chunking with *WRAM* operating on 2-byte `words`. *GEQ* means greater than or equal to comparison.

- *WMAXP*: Like *MAXP* [15] for `bytes`, *WMAXP* looks for regions with `word` local maxima at their center, i.e., greater than a fixed number of `words` before and after.
- *WRAM*: Like *RAM* [14] does for `bytes`, *WRAM* first scans for a maximum-valued `word` in a fixed-size region and then inserts a boundary when it finds a `word` outside this region that is at least as large as the found maximum.

Figure 3 shows an illustration of a chunk generated by *WRAM* operating on 2-byte `words`. First, all the overlapping `words` from the fixed-size region are scanned to determine the maximum `0xFFAA`. Subsequently, overlapping `words` outside this region are serially compared against `0xFFAA` to see if they exceed or equal it. When `0xFFFF` that satisfies this condition is found, a boundary is inserted.

4.2 Acceleration with Vector Instructions

To avoid missing boundaries due to overlaps (§2), we present two methods: Simple (§4.2.1) and Optimized (§4.2.2).

4.2.1 Simple Acceleration Methods

We present simple modifications to VectorCDC's [9, 16] *Range Scans* and *Tree-based Extreme Byte Searches* (§2) to account for overlaps. We only discuss `words` here for simplicity, but they also apply to `dwords` and `qwords`.

Range Scans. To fix the problem shown in Figure 1, a simple approach is to perform two parallel range scans offset by one byte. The first loads data starting at B_0 into the vector register V_{Data1}, and the second loads data starting at B_1 into the vector register V_{Data2}, after which both are compared against the target values. If there is no match, we move ahead by the vector register size, i.e., start the next scans at B_6 and B_7.

This accounts for overlaps, but comes with high performance costs at larger granularities (§5.2). For instance, `qwords` need eight range scans offset by one byte each.

Extreme Byte Searches (EBS). VectorCDC uses tree-based EBS; let us consider a maximum value search. To find the maximum valued `word` without missing overlaps, we can perform two tree-based searches. The first considers `words`

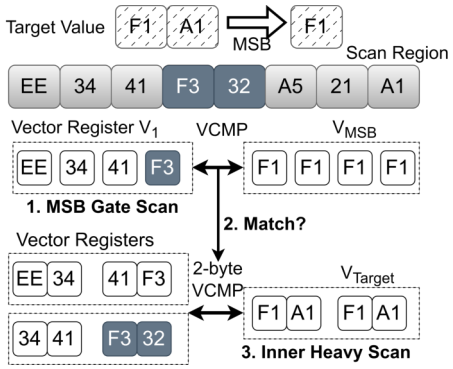


Figure 4: *Gated GEQ Range Scan* for 2-byte `words`

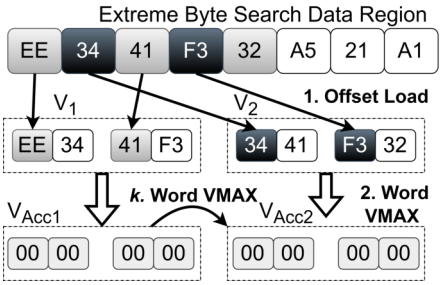


Figure 5: *Accumulated Extreme Byte Search* for 2-byte `words`

starting at odd offsets, while the second considers `words` starting at even offsets. Each tree operates independently, using `mm512_max_epu16` (VMAX) instructions to obtain a single register with maximum values from its region. We can combine the results from both trees with a single VMAX operation between the final registers, and sequentially scan the combined register to get the maximum 2-byte words. Similarly, we can perform minima searches, or extend this to `dwords` and `qwords` with four / eight parallel trees.

However, tree-based EBS is vector register-intensive. A limited number of these registers are present in modern CPUs, and data is moved to the L1 cache when evicted from one of them. Running multiple parallel tree-based searches puts even more pressure on register consumption, leading to reduced performance gains with increasing granularity (§5.2).

4.2.2 Optimized Acceleration Methods

This section presents WideCDC's acceleration methods.

Gated Range Scans. To avoid the redundant computational cost of running multiple parallel range scans, WideCDC uses *Gated Range Scans*. We decompose *wide region* comparisons into smaller constituents, using the insight that, for a *wide region* to match the comparator requirement, its first k significant bytes must as well. For instance, when running GEQ Range Scans on `words`, its most significant byte (MSB) has to be GEQ that of the target `word`.

We use this insight to hide the multiple offset comparisons behind a *Gate* to only be run if the MSB comparator matches. Figure 4 shows an example of a *Gated GEQ Range Scan* looking for `word` matches. We pack individual bytes from the data region into a vector register V_1, and compare them against the target `word's` MSB held in V_{MSB} (Step 1). Comparisons generate a bitmask (Step 2), containing a set bit if a byte in V_1 is GEQ the target MSB in V_{MSB}. If any bit is set, we run the expensive `word` comparisons accounting for overlaps (Step 3).

Accumulated EBS. We forgo the tree-based search approach, instead adopting an accumulated approach (Figure 5). The figure shows the search for `word` maxima, but the same concepts hold for minima and `dwords`/`qwords`.

We maintain two accumulator registers, V_{Acc1} and V_{Acc2}, which hold the current global maximums; they are initially set to zero. `Words` from the region are loaded into two vector registers V_1 and V_2 in Step 1, in offset fashion. In Step 2, the pairwise maximums between the accumulator and data registers are calculated with VMAX instructions; for instance, a VMAX between V_1 and V_{Acc1} would result in V_{Acc1} retaining the maximum between its and V_1's corresponding byte for each of its 64 bytes. We repeat this with a VMAX between V_2 and V_{Acc2}. We then load the next set of region words (starting at `0x32` and `0xA5`) and repeat Steps 1 and 2, continuing until the region is exhausted. At the end, pairwise maximums between V_{Acc1} and V_{Acc2} are calculated, and the resulting register is sequentially scanned to find the maximum `word`.

Unlike the simple tree-based approach (§4.2.1), we only need a fixed number of accumulators equal to the *granularity size*, resulting in better performance at larger granularities. Incidentally, our accumulated approach also outperforms the tree-based approach for `bytes` (§5.2).

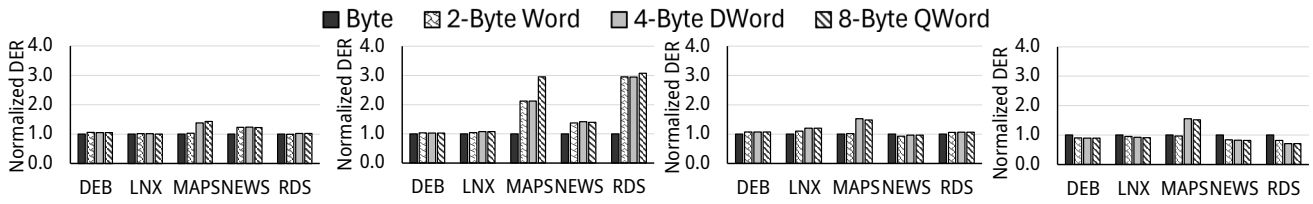
5 Evaluation

This section outlines our evaluation efforts for WideCDC. Note that the deduplication efficiency improvements (§5.1) stem from using *wide regions* for chunk boundaries (§4.1) while the throughput improvements (§5.2) stem from WideCDC's optimized acceleration techniques (§4.2.2).

Implementation and AI Usage. We implemented `word`,

Dataset	Size	Information
DEB	40 GB	65 Debian VM Images released by VectorCDC [30]
LNX	65 GB	160 Linux kernel distributions in TAR format [31]
MAPS	981 GB	OpenStreetMaps [32] backups of a specific country
NEWS	478 GB	Complete snapshots of a news website at a depth of three links.
RDS	122 GB	100 Redis [33] snapshots with `redis-benchmark` runs

Table 1: Dataset Information



(a) AE-Max [13] (b) AE-Min [13] (c) MAXP [15] (d) RAM [14]

Figure 6: WideCDC DER improvements. Note that DERs are scaled against byte-granularity (original) versions.

Dataset	Unique		Top-1		Top-2		Top-10	
	Byte	*Qword*	*Byte*	*Qword*	*Byte*	*Qword*	*Byte*	*Qword*
RDS	2	41947	95.7%	42.7%	100%	48.5%	100%	48.9%
MAPS	1	615	100%	67.9%	100%	91.5%	100%	99.9%

Table 2: Extrema Distribution Analysis with *Byte* and *Qword* versions of *AE-Min*.

`dword`, and `qword`-based *Extreme Byte Searches* and *Range Scans*; both simple and optimized (§4.2). We accelerated four hashless CDC algorithms [13–15] with WideCDC using AVX-256 and AVX-512 [18] instructions. Our new implementations span ~3500 lines of C++ code and are released within DedupBench [8]. We used GitHub Copilot [34] with Claude Opus 4.5 [35] to aid with implementation, and Google Gemini 3/3.5 Pro [36] for grammatical checks.

Testbed. We run all our experiments using an Intel Emerald Rapids (*c6620*) machine from Cloudlab [37], that supports the AVX-256 and AVX-512 vector instruction sets [18]. The machine has a 28-core Intel Xeon Gold 5512U with hyperthreading, 128 GB of RAM, and two 800 GB NVMe SSDs. Our throughput results are ten run averages with a standard deviation under 5%.

Alternatives. We evaluate the hash-based *FastCDC* [11], *Rabin-Karp* [10] and *TTTD* [12] algorithms and the hashless *AE* [13], *RAM* [14] and *MAXP* [15]. In addition, we evaluate the following AVX-512 accelerated algorithms:

- *VAE / VRAM / VMAXP*: VectorCDC [9, 16] versions of *AE*, *RAM* and *MAXP*.
- *WAE / WRAM / WMAXP*: WideCDC versions of *AE*, *RAM* and *MAXP*. We evaluate `word`, `dword`, and `qword` versions of each algorithm. We gate *Range Scans* using the first two MSBs for `dword` and `qword` versions.

Datasets. Table 1 shows the datasets used within our evaluation. We note that they represent diverse domains. We use VectorCDC's [9] released `DEB` dataset [30].

5.1 Deduplication Efficiency

This section compares the Deduplication Elimination Ratios (DERs) of all alternatives with an average chunk size of 8 KB.

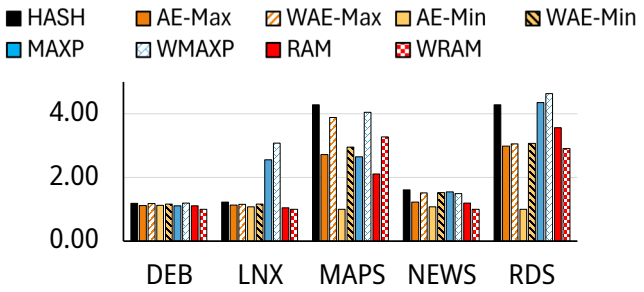
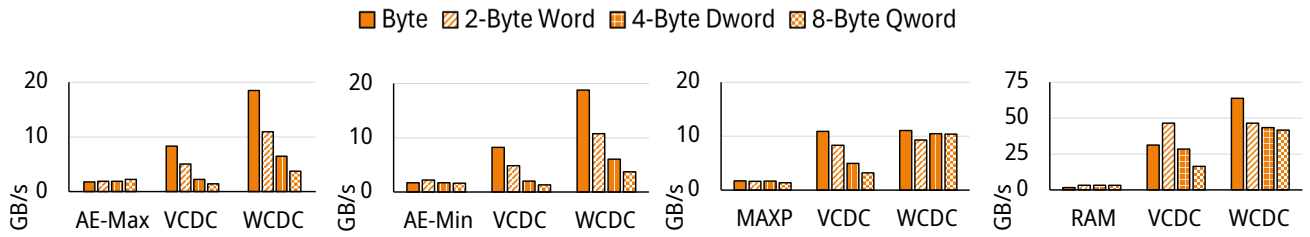


Figure 7: Normalized DERs of WideCDC algorithms against hash-based. `HASH` is the best among hash-based CDC [10–12]. Each WideCDC algorithm is its best granularity.

Extrema Distribution Table 2 shows the number of unique extrema values exhibited by *AE-Min* on `MAPS` and `RDS`, and the percentage share of total occurrences for the top-1, top-2, and top-10 occurring values combined. On both datasets, *Byte* versions of *AE-Min* exhibit skewed distributions, with only 1-2 unique extrema values as described in §1. Using *Qwords* significantly improves this distribution. For instance, on `RDS`, the number of unique extrema goes from 2 values to 41947 values, improving chunk boundary quality. Thus, *WideCDC fixes the skewed minimum/maximum value distributions in hashless CDC.*

Algorithmic Impact. Figures 6a - 6d show the impact of WideCDC on hashless CDC algorithms. We note that, except for *RAM* [14], all algorithms see improvements to DER across datasets. For instance, the `qword` version of *AE-Max* (Figure 6a) achieves 1.43× higher DER than the original byte-granularity version on `MAPS`. *AE-Min* (Figure 6b) benefits the most among all algorithms, achieving up to 2.95× and 3.07× improvements in DER on `MAPS` and `RDS` respectively. While *RAM* (Figure 6d) suffers from slightly decreasing DER on many datasets, it still sees up to 1.55× DER improvement on `MAPS`, indicating that WideCDC still has its benefits on specific data patterns. Thus, *WideCDC is largely beneficial for improving the deduplication efficiency of hashless algorithms.* Note that the average chunk size between the byte and WideCDC versions differs only by a maximum of 1.1%.

Datasets and Granularity. `MAPS`, `NEWS`, and `RDS` see the greatest improvements across algorithms, while `DEB` and `LNX`



(a) AE-Max [13] (b) AE-Min [13] (c) MAXP [15] (d) RAM [14]

Figure 8: Throughput on `DEB`. `VCDC` is VectorCDC extended with simple techniques (§4.2.1), while `WCDC` is WideCDC.

see smaller ones, as these datasets suffer from the skewed extrema distributions shown above. This shows *that the DER improvements are due to WideCDC addressing the pathological data patterns that plague hashless CDC algorithms*.

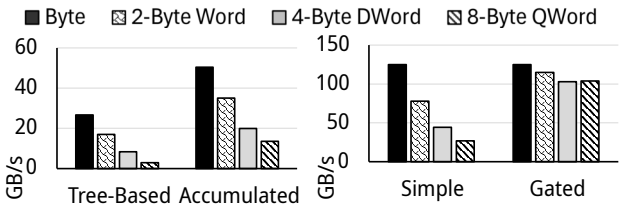
Increasing granularities are beneficial only up to `dwords` for most algorithms; going further to `qwords` has limited benefit. The only exception is *AE-Min* on `MAPS`, where `qwords` see a sizeable DER improvement over `dwords`. Thus, *in most cases*, `words`/`dwords` *are the optimal choice*.

Comparison vs hash-based CDC. Figure 7 takes the best performing WideCDC variant of each algorithm and compares it against the best hash-based [10–12] one from Figure 2. We note that, except for *RAM*, *WideCDC brings the DER of all hashless algorithms much closer to their hash-based counterparts.* The improvement is larger on datasets like `MAPS` that had larger differences between hashless and hash-based CDC.

Vector-Acceleration Impact. Similar to the results observed by previous studies [9, 16, 17], accelerating wide hashless algorithms with vector instructions has no DER impact.

5.2 Chunking Throughput

Figure 8 shows the throughput of WideCDC algorithms compared to unaccelerated and VectorCDC [9] versions on `DEB` with an 8 KB average chunk size, while Figure 9 compares the throughputs of VectorCDC's and WideCDC's individual phases on randomized data with a microbenchmark. Note that VectorCDC [9] does not support higher granularities (§2); we extend it with the simple techniques described in §4.2.1, while WideCDC uses the optimized ones from §4.2.2. Note that this experiment measures single-threaded throughput, and that the trends were similar on other datasets and chunk sizes.



(a) Extreme Byte Search (b) Range Scan

Figure 9: Throughput breakdown on randomized data

Byte Granularity. Figure 8 shows that WideCDC algorithms match or outperform VectorCDC [9] for `bytes`. In Figure 8d, `byte` *WRAM* achieves 63.8 GB/s, 2.04× and 37.57× higher throughput than *VRAM* and *RAM*, respectively. This is due to *Accumulated EBS* being faster than its counterpart, the *Tree-based EBS* (Figure 9). Thus, *WideCDC outperforms VectorCDC [9] for* `bytes`, *showing benefits even for datasets without DER gains from higher granularity chunking*.

Wide Regions. Figure 8 also shows the throughputs at increasing granularities; WideCDC outperforms VectorCDC across granularities and scales better with increasing granularity. This is due to both *Accumulated EBS* and *Gated Range Scans* outperforming their counterparts (Figure 9). Thus, *WideCDC's Accumulated EBS and Gated Range Scan allow it to achieve higher throughput at higher granularities.*

Algorithmic Impact. The throughput benefits vary by algorithm. For instance, `word`, `dword`, and `qword` versions of *WRAM* (Figure 8d) achieve nearly identical throughputs, while *WAE-Max*'s throughput drops with increasing granularity (Figure 8a). This is because of algorithmic characteristics; *AE-Max* relies heavily on Extreme Byte Searches (EBS) running multiple EBS phases per chunk, while *RAM* only needs one EBS per chunk [9].

Hash-based CDC. VectorCDC's [9, 16] hashless algorithms are 21× – 46× faster than SS-CDC's [17] hash-based ones; we observed similar results on our testbed. We omit these from Figure 8 for clarity.

Portability. While we mainly discuss AVX-512 instructions [18], WideCDC is compatible with other instruction sets while retaining its throughput advantages. For instance, `byte` and `dword` *WRAM* with AVX-256 achieve 59.8 GB/s and 29.1 GB/s respectively. We omit these due to space constraints.

6 Conclusion

We present WideCDC, a method to improve hashless chunking algorithms by using `words`, `dwords`, and `qwords`. WideCDC uses novel vector-compatible methods to accelerate wide hashless CDC. Our evaluation shows that WideCDC improves the DERs of hashless algorithms by up to 3.07× while being up to 2.04× faster than the state-of-the-art.

A Artifact Appendix

Abstract

We have made the code for WideCDC publicly available with DedupBench [8] on GitHub [1].

Scope

DedupBench allows for the comparison of numerous content-defined chunking algorithms on custom datasets. It has been used in the development and release of our previous projects, such as SeqCDC [38] and VectorCDC [9, 16].

Using DedupBench, the following claims from our paper can be validated:

- **Space Savings**: The space savings of WideCDC's hashless algorithms (such as `word`, `dword`, and `qword` versions) are higher than their respective `byte` versions. For instance, *AE-Max* with `words` achieves higher space savings than with `bytes`. Note that the gains vary depending on dataset characteristics.

- **Chunking Throughput**: The chunking throughput achieved by WideCDC's algorithms are higher than those achieved without vectorization. The chunking throughput achieved by WideCDC's `byte` versions are higher than VectorCDC's `byte` versions. Note that we **do not** provide the code for the simple VectorCDC extensions described in §4.2.1.

Contents

We have bundled the following WideCDC algorithms into the DedupBench repository:

- *AE-Min:* The hashless Asymmetric Extremum [13] algorithm using local minima.

- *AE-Max:* The hashless Asymmetric Extremum [13] algorithm using local maxima.

- *MAXP:* The hashless MAXP [15] algorithm.

- *RAM:* Rapid Asymmetric Maximum [14] algorithm.

We provide only the AVX-512 versions of each algorithm. For ease of use, we have also provided scripts to run all algorithms on any user-defined dataset and plot graphs. The repository README file contains further details and usage instructions.

[1] https://github.com/UWASL/dedup-bench

Hosting

The code is currently available on the `artifact_fast27` branch of the DedupBench [8] repository. Commit `51208ae` or later should hold all the relevant code. We plan to merge this branch into `main` in the future.

The `DEB` dataset used in our evaluation is publicly hosted on Kaggle [2] [30]. It can be downloaded and used with DedupBench to obtain some of the results from our paper.

Requirements

The machines used to develop DedupBench were obtained from CloudLab [37]. The details can be found in §5.

All of the native (unaccelerated) CDC algorithms in DedupBench are universally compatible with all processors. WideCDC's vectorized algorithms require CPUs with AVX-512 instruction set support.

References

[1] Statista. Worldwide data created from 2010 to 2025, 2024.

[2] Maohua Lu, David Chambliss, Joseph Glider, and Cornel Constantinescu. Insights for data reduction in primary storage: a practical analysis. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. Primary Data Deduplication — Large scale study and system design. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 285–296, 2012.

[4] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *USENIX Conference on File and Storage Technologies (FAST)*, volume 12, pages 4–4, 2012.

[5] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):1–20, 2012.

[6] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016.

[2] https://www.kaggle.com/datasets/sreeharshau/vm-deb-fast25

[7] Dian Rachmawati, JT Tarigan, and ABC Ginting. A comparative study of Message Digest 5 (MD5) and SHA256 algorithm. In *Journal of Physics: Conference Series*, volume 978, page 012116. IOP Publishing, 2018.

[8] Alan Liu, Abdelrahman Baba, Sreeharsha Udayashankar, and Samer Al-Kiswany. DedupBench: A Benchmarking Tool for Data Chunking Techniques. In *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 469–474. IEEE, 2023.

[9] Sreeharsha Udayashankar, Abdelrahman Baba, and Samer Al-Kiswany. VectorCDC: Accelerating data deduplication with vector instructions. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 513–522, 2025.

[10] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 174–187, 2001.

[11] Wen Xia, Yukun Zhou, Hong Jiang, Dan Feng, Yu Hua, Yuchong Hu, Qing Liu, and Yucheng Zhang. FastCDC: A fast and efficient content-defined chunking approach for data deduplication. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 101–114, 2016.

[12] Kave Eshghi and Hsiu Khuern Tang. A framework for analyzing and improving content-based chunking algorithms. *Hewlett-Packard Labs Technical Report TR*, 30(2005), 2005.

[13] Yucheng Zhang, Hong Jiang, Dan Feng, Wen Xia, Min Fu, Fangting Huang, and Yukun Zhou. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1337–1345. IEEE, 2015.

[14] Ryan NS Widodo, Hyotaek Lim, and Mohammed Atiquzzaman. A new content-defined chunking algorithm for data deduplication in cloud storage. *Future Generation Computer Systems*, 71:145–156, 2017.

[15] Nikolaj Bjørner, Andreas Blass, and Yuri Gurevich. Content-dependent chunking for differential compression, the local maximum approach. *Journal of Computer and System Sciences*, 76(3-4):154–203, 2010.

[16] Sreeharsha Udayashankar, Abdelrahman Baba, and Samer Al-Kiswany. Accelerating data chunking in deduplication systems using vector instructions. *ACM Transactions on Storage*, February 2026.

[17] Fan Ni, Xing Lin, and Song Jiang. SS-CDC: A two-stage parallel content-defined chunking for deduplicating backup storage. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 86–96, 2019.

[18] James E Smith, Greg Faanes, and Rabin Sugumar. Vector instruction set support for conditional operations. *ACM SIGARCH Computer Architecture News*, 28(2):260–269, 2000.

[19] Kiatchumpol Suttisirikul and Putchong Uthayopas. Accelerating the cloud backup using GPU based data deduplication. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 766–769. IEEE, 2012.

[20] Samer Al-Kiswany, Abdullah Gharaibeh, Elizeu Santos-Neto, George Yuan, and Matei Ripeanu. StoreGPU: Exploiting Graphics Processing Units to Accelerate Distributed Storage Systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, HPDC '08, page 165–174, New York, NY, USA, 2008. Association for Computing Machinery.

[21] Nigel Tan, Jakob Luettgau, Jack Marquez, Keita Teranishi, Nicolas Morales, Sanjukta Bhowmick, Franck Cappello, Michela Taufer, and Bogdan Nicolae. Scalable incremental checkpointing using gpu-accelerated de-duplication. In *Proceedings of the 52nd International Conference on Parallel Processing*, ICPP '23, page 665–674, New York, NY, USA, 2023. Association for Computing Machinery.

[22] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Yukun Zhou. Ddelta: A deduplication-inspired fast delta compression approach. *Performance Evaluation*, 79:258–272, 2014. Special Issue: Performance 2014.

[23] Fan Ni and Song Jiang. RapidCDC: Leveraging Duplicate Locality to Accelerate Chunking in CDC-Based Deduplication Systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 220–232, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. P-Dedupe: Exploiting Parallelism in Data Deduplication System. In *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*, pages 338–347, 2012.

[25] Youjip Won, Kyeongyeol Lim, and Jaehong Min. MUCH: Multithreaded Content-Based File Chunking. *IEEE Transactions on Computers*, 64(5):1375–1388, 2015.

[26] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. Similarity and Locality Based Indexing for High Performance Data Deduplication. *IEEE Transactions on Computers*, 64(4):1162–1176, 2015.

[27] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezis, and Peter Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *USENIX Conference on File and Storage Technologies (FAST)*, volume 9, pages 111–123, 2009.

[28] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. HYDRAstor: A scalable secondary storage. In *USENIX Conference on File and Storage Technologies (FAST)*, volume 9, pages 197–210, 2009.

[29] Marcel Gregoriadis, Leonhard Balduf, Björn Scheuermann, and Johan Pouwelse. A thorough investigation of content-defined chunking algorithms for data deduplication. *arXiv preprint arXiv:2409.06066*, 2024.

[30] Sreeharsha Udayashankar, Abdelrahman Baba, and Samer Al-Kiswany. VM Images for Deduplication. https://www.kaggle.com/dsv/10561721, 2025.

[31] Linux. The Linux Kernel Archives. https://www.kernel.org/, 2023.

[32] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 2008.

[33] Redis. Redis. https://redis.io/, 2023.

[34] Github. GitHub Copilot · Your AI pair programmer. https://github.com/features/copilot, 2026.

[35] Anthropic. Introducing Claude Opus 4.5. https://www.anthropic.com/news/claude-opus-4-5.

[36] Google. Gemini 3 Model. https://aistudio.google.com/models/gemini-3.

[37] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1–14, Renton, WA, July 2019. USENIX Association.

[38] Sreeharsha Udayashankar, Abdelrahman Baba, and Samer Al-Kiswany. SeqCDC: Hashless Content-Defined Chunking for Data Deduplication. In *Proceedings of the 25th International Middleware Conference*, pages 292–298, 2024.