Root-Down Exposure for Maximal Clique Enumeration on GPUs

Anonymous Author(s)

Abstract

Maximal clique enumeration (MCE) in large-scale graphs is critical across various application domains, including social network analysis, bioinformatics, and computer vision. However, existing GPU-based MCE solutions suffer from inefficient load-balancing mechanisms. These mechanisms force busy workers to pause and hand over workloads to idle workers, which introduces significant synchronization overhead and increases memory usage. To address these limitations, we introduce a root-down exposure mechanism where busy workers dynamically expose their current root, enabling idle workers to pull workloads from the exposed node directly without synchronization. We then propose a bitmap-centric MCE scheme and an aggressive node generation rule to further simplify the memory layout and accelerate enumeration. We combine them into RDMCE, a Root-Down MCE solution on GPUs. Across large real-world graphs with up to 146 billion maximal cliques, RDMCE is 1.25-5.38× faster than any next-best state-of-the-art GPU-based solutions and is the only one that completes enumeration on every test dataset, offering a more efficient and scalable MCE solution.

Keywords: maximal clique enumeration, GPU, load balancing, bitmap.

1 Introduction

In an undirected graph, a clique is a complete subgraph where every pair of vertices is connected by an edge. A maximal clique is a clique that cannot be enlarged by adding more vertices. The maximal clique enumeration (MCE) problem aims to identify all maximal cliques within a given graph. This task is computationally intensive, particularly for large graphs, because the number of maximal cliques can be exponential in the number of vertices in the worst case [22].

MCE is widely used across multiple application domains, including social network analysis [1, 20, 36], bioinformatics [17, 21, 28, 29], and computer vision [38, 39]. For example, in social network analysis, MCE helps identify closely connected groups of users, such as friend circles on social media platforms. Detecting these groups aids in understanding social dynamics and the spread of information, which is vital for targeted advertising and trend analysis.

Mainstream MCE solutions follow the Bron-Kerbosch framework [6], which recursively expands the current clique with possible vertices and checks maximality via an excluded set on a search tree. Subsequent works refine this recursion through pivoting [31], degeneracy ordering [11], hybrid data structures [4, 28, 32], top-down pruning [18], and graph reduction [10]. Some of them parallelize MCE on multicore CPUs by assigning each core an independent subtree [4, 5, 10, 32]. However, CPU-based solutions are ultimately bounded by the limited parallelism of a handful of cores. State-of-the-art solutions therefore target GPUs [2, 35, 40], whose tens of thousands of lightweight cores [24] can process extensive subtrees concurrently. Despite these advances, two fundamental challenges still hinder further performance gains for GPU-based solutions:

(1) Load imbalance. MCE workloads are inherently unbalanced because subtrees vary greatly. Specifically, the power-law distribution of vertex degrees [7] and the wide range of maximal clique sizes result in subtrees with different branching factors and depths. GPUs make the problem worse, because tens of thousands of cores must wait for the slowest thread, wasting compute resources. GBK [35] and G^2-AIMD [40] partition subgraphs on the CPU and then stream one induced subgraph to the GPU at a time, but incur partition overhead and repeated host-device traffic. mce-gpu [2] launches thousands of GPU workers, each handling an independent subtree, and keeps a worker list for load balance. However, the worker list enforces busy workers to pause, split workloads, and hand over workloads to idle workers, incurring synchronization overhead.

(2) Memory pressure. The MCE problem has a large memory footprint, because its search tree can hold an exponential number of maximal cliques in the vertex count [22]. Since GPU dynamic memory allocation is costly [37], GPU solutions must pre-allocate the worst-case memory for every tree traversal. GBK and G^2-AIMD process only a small induced subgraph at a time, but their breadth-first search (BFS) still demands memory exponential in the vertex count, so even a modest subgraph can exhaust device memory. mce-gpu cuts the memory demand by using depth-first search (DFS), but adds many extra objects, such as a global worker list and load-balance buffers per worker. These extra allocations consume large portions of usable memory.

To address these challenges, we propose RDMCE, a high performance GPU-based MCE solution built on three key techniques. First, the *root-down exposure mechanism* drives each worker to expose its DFS path downward to a node with unexplored subtrees, enabling idle workers to pull a subtree from the exposed node without synchronization. This mechanism efficiently reduces load-balancing synchronization overhead, because busy workers never pause to hand over workloads while idle workers pull unexplored subtrees without assistance. Second, the *bitmap-centric MCE scheme* loads

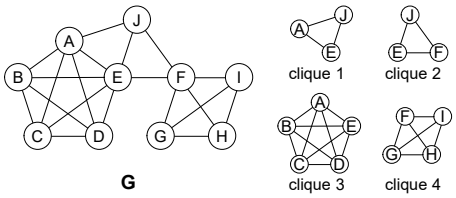


Figure 1. A graph G containing 4 maximal cliques.

the global graph once per worker to initialize subgraphs encoded with bitmaps and then performs the entire enumeration directly on these bitmaps, simplifying the memory layout and accelerating computation via bitwise operations. Third, *aggressive node generation* expands cliques by several vertices at once, replacing sequential vertex-by-vertex clique expansion and further accelerating enumeration.

We evaluate RDMCE on 12 large real-world graphs. Only RDMCE enumerates all maximal cliques on every dataset when executed on an NVIDIA RTX 4090 with 24 GB of global memory. All other state-of-the-art methods exhaust memory and terminate early on Wikipedia-link-en, a graph that contains 335 million edges and 6 billion maximal cliques. For datasets in which at least one baseline completes, RDMCE is 1.25-5.38× faster than any next-best competitors, demonstrating both superior performance and high scalability.

In summary, we make the following contribution:

- A root-down exposure mechanism for dynamic load balancing
- A bitmap-centric MCE scheme to simplify memory layout and accelerate computation
- An aggressive node generation approach for additional speedup.
- An efficient Root-Down MCE solution RDMCE on GPUs, combining three techniques above.
- Extensive experiments show RDMCE outperforms any next-best baselines by 1.25–5.38×, demonstrating both superior performance and high scalability.

2 Background

In this section, we formulate the MCE problem, review the classical Bron-Kerbosch MCE algorithm, and introduce the state-of-the-art GPU-based MCE solutions.

Problem Formulation. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E \subseteq V \times V$. For any vertex $v \in V$ its neighborhood is $N(v) = \{u \in V \mid (u, v) \in E\}$, its degree is denoted as $\deg(v) = |N(v)|$, and the maximum degree of the graph is $\Delta(V) = \max_{v \in V} \deg(v)$. A subset $C \subseteq V$ is a clique if every two distinct vertices in C are connected, and a clique C is maximal if no superset $C' \supsetneq C$ is a clique. The Maximal Clique Enumeration (MCE) problem aims to enumerate all maximal cliques in G. For example, Figure 1 shows a graph G and all maximal cliques in G.

Algorithm 1: Bron–Kerbosch with pivoting and degeneracy ordering

Input: Undirected graph $G = (V, E)$
Output: All maximal cliques in G

1 compute a degeneracy ordering $v_1, v_2, \ldots, v_{|V|}$ of G;
2 **for** $i = 1$ **to** $|V|$ **do**
3 $\quad P \leftarrow N(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_{|V|}\}$;
4 $\quad X \leftarrow N(v_i) \cap \{v_1, \ldots, v_{i-1}\}$;
5 $\quad$ BK$(\{v_i\}, P, X)$;
6 **Procedure** BK (R, P, X)**:**
7 $\quad$ **if** $P = \varnothing$ ***and*** $X = \varnothing$ **then output** R;
8 $\quad$ choose a pivot $u \in P \cup X$ maximizing $|P \cap N(u)|$;
9 $\quad$ **foreach** $v \in P \setminus N(u)$ **do**
10 $\quad\quad$ BK$(R \cup \{v\},\ P \cap N(v),\ X \cap N(v))$;
11 $\quad\quad P \leftarrow P \setminus \{v\}$;
12 $\quad\quad X \leftarrow X \cup \{v\}$;

Bron–Kerbosch with Pivoting and Degeneracy Ordering. Modern MCE algorithm is the Bron–Kerbosch (BK) framework [6] refined by pivoting [31] and degeneracy ordering [11]. The BK algorithm is conducted as a depth-first traversal of a search tree whose nodes are represented by the triple (R, P, X), where R contains the result vertices already fixed in the current clique, P comprises the possible vertices adjacent to every vertex of R and still eligible for extension, and X keeps the exclude vertices that have been processed and are now excluded to avoid duplicate output. *Pivoting* selects a vertex $u \in P \cup X$ and limits branching to $P \setminus N(u)$, thereby pruning redundant subtrees without sacrificing completeness. *Degeneracy ordering* arranges the vertices according to a k-core elimination sequence. As a result, each induced subproblem exhibits a candidate set whose size is bounded by the graph's degeneracy d, which is much more smaller than $|V|$ and $\Delta(V)$. Consequently, the algorithm attains a running time of $O(d\, 3^{d/3}\, |V|)$, whereas the original BK algorithm exhibits the bound $O(3^{|V|/3})$ [11].

Algorithm 1 formalizes this combined strategy. After degeneracy ordering (line #1), each vertex v_i seeds a procedure BK with a node $(\{v_i\}, P, X)$, where P holds its neighbours appearing later in the order (lines #2-5), ensuring the vertex count in P never exceeds the degeneracy d. BK (R, P, X) outputs R when P and X are empty (line #7), chooses a pivot $u \in P \cup X$ that maximizes $|P \cap N(u)|$ (line #8), and branches only on $v \in P \setminus N(u)$ (lines #9-11).

Example 1. *Figure 2 illustrates the search tree produced by Algorithm 1 on graph G. We use subscripts to indicate the vertex set associated with an enumeration node. For example, R_x denotes the set R of node x. Initially, the vertices are reordered according to the degeneracy ordering (line #1), as shown at the root node. We start from the root node and generate the search tree through backtracking. First, we enter node x by traversing J.* $R_x = \{J\}$, $P_x = N(J) \cap \{J, G, H, I, F, A, B, C, D, E\} =$

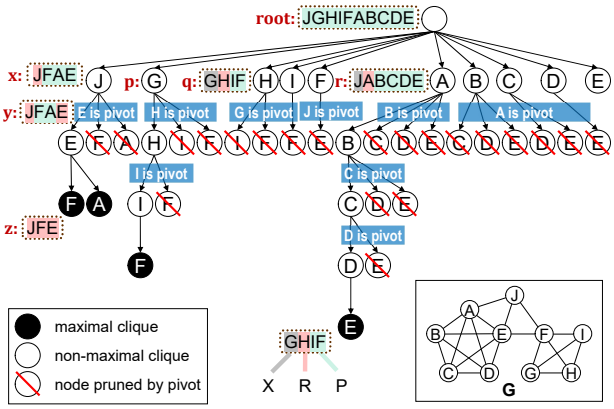


Figure 2. Search tree of Bron-Kerbosch with pivoting and degeneracy ordering on graph G. Each node labels the vertex added to R. Background shading distinguishes its role within the corresponding (R, P, X) triple.

$\{F, A, E\}$, and $X_x = \varnothing$ (lines #3–5). Then, node x selects vertex E as pivot because $|P_x \cap N(E)| = |\{F, A\}| = 2$ is greater than $|P_x \cap N(F)| = |P_x \cap N(A)| = 1$. Node x explores the set $P_x \setminus N(E) = \{E\}$ and prunes branches induced by F and A (line #9). Next, node x traverses vertex E to node z. $R_z = R_y \cup \{F\} = \{J, F, E\}$, $P_z = P_y \cap N(F) = \varnothing$, and $X_z = \varnothing$ (lines #10–12), thus outputting a maximal clique $\{J, F, E\}$ (line #7). The remaining nodes are generated similarly.

Parallelizing MCE on GPUs. Unlike the modest parallelism offered by multi-core CPUs, modern GPUs expose extensive lightweight cores grouped into streaming multiprocessors (SMs). Under the SIMT (Single-Instruction, Multiple-Thread) model, threads are scheduled in 32-thread warps, offering massive parallelism. However, porting the BK algorithm to such hardware is non-trivial because the search tree is both large and irregular.

GBK [35] streams small subgraphs from the CPU and lets each warp independently perform pivot selection or maximality checks on a single vertex. mce-gpu [2] decomposes the search tree into independent subtrees, assigns each to a thread block, and allows blocks to donate unfinished branches to a lightweight worker list for dynamic load balancing. G^2-AIMD [40] slices the graph into chunks, processes several chunks in a BFS manner at a time, and adjusts chunk size on demand to stay within device memory. Despite these efforts, state-of-the-art solutions are still limited by either the synchronization overhead of dynamic load-balancing primitives or the repeated CPU-GPU data transfers.

3 Challenges

This section shows the two main challenges of MCE on GPUs—load imbalance and memory pressure, reviews current solutions, and pinpoints potential for improvement.

3.1 Load Imbalance

Primary reason. The imbalance arises from variations in the sizes of the search trees generated by the BK algorithm (i.e., Algorithm 1). For any node (R, P, X), the branching factor is bounded by $|P|$, while the subtree depth is bounded by the graph's degeneracy d under degeneracy ordering. As $|P|$ ranges from one to thousands, subtree sizes vary greatly, leading to a wide spread in GPU workload.

Current practice. State-of-the-art GPU load-balancing strategies are distinguished by *where* the split decision is made. We take a worker to be the smallest thread block that handles a subtree independently. *Host-side scheme* let the CPU decide the granularity of each GPU launch. GBK [35] streams one induced subgraph at a time and frees memory when a branch completes, whereas G^2-AIMD [40] partitions the graph into fine-grained chunks and dispatches them in batches. Both of them incur partitioning overhead and repeated host-device transfers. *Device-side scheme* places the split inside the GPU and redistributes workloads at runtime. Typically, overloaded workers are notified to split their workloads and hand them to idle counterparts. mce-gpu [35] forces busy workers to check a work list and push unexplored subtrees when signaled. Likewise, GMBE [26, 27] for maximal biclique enumeration requires busy workers to split workloads into a global task queue. STMatch [34] for graph pattern matching lets busy workers to interrupt traversal and copy context to idle peers.

Potential for improvement. Host-side scheme is bound by repeated CPU-GPU traffic, so the greater speed-up potential lies with device-side scheme. However, we observe that **current device-side designs still force busy workers shoulder the balancing cost**: each overloaded worker pauses its traversal, splits its workload, and then hands pieces to idle peers. This pause extends the busy worker's critical path while idle workers spin and waste cycles waiting for the hand-off. Consequently, a mechanism is needed that allows idle workers to pull workloads without ever interrupting the busy one.

3.2 Memory Pressure

Primary reason. The memory pressure comes from the large footprint of MCE, because the number of maximal cliques in its search tree can reach $3^{|V|/3}$ [22]. Given the high cost of dynamic memory allocation on GPUs, we must pre-allocate $O(p\Delta(V)d)$ memory to parallelize p workers, because each node (R, P, X) stores $O(\Delta(V))$ vertices and DFS holds up to d such nodes, equal to the search tree's maximum depth. For example, running MCE on Wikipedia-link-en (Table 1) with an NVIDIA RTX 4090 (24 GB, 128 SMs) [24] requires $128 \times 1{,}052{,}326 \times 1{,}114 \times$ `sizeof(int)` $\approx$ 559 GB, far beyond the GPU's capacity.

Current practice. Recent GPU-based MCE solutions alleviate memory pressure from two sides. GBK [35] and G^2-AIMD [40] leave the full graph on the CPU and stream only an induced subgraph to the GPU. GBK always sends a chunk whose worst-case memory footprint never exceeds the remaining GPU space, while G^2-AIMD adopts the TCP AIMD congestion-control algorithm to enlarge or shrink the chunk on demand. In contrast, mce-gpu [35] keeps the graph resident on the GPU, encoding R and P as bitmaps and shares a compact X within each worker, cutting memory per worker from $O(\Delta(V)d)$to $O(\Delta(V) + d^2)$.

Potential for improvement. To avoid CPU-GPU traffic, we focus on mce-gpu, a fully on-device solution. We observe that **mce-gpu uses numerous auxiliary GPU memory blocks**, including a global CSR/COO copy of the graph, per-worker bitmaps for R and P, a compressed X structure, a global worker list, a private buffer for load balance per worker, and so on. These scattered allocations leave holes and waste space, so we need to eliminate as many of these blocks as possible.

4 Design of RDMCE

We present a GPU-based MCE solution, RDMCE, featuring three novel techniques: (1) a root-down exposure mechanism to balance workloads, (2) a bitmap-centric MCE scheme, and (3) aggressive node generation approach.

4.1 Root-Down Exposure

To eliminate load-balancing overhead, we introduce the root-down exposure mechanism. Its guiding principle is simple: *busy workers do not pause to split or copy workloads but only expose the current DFS root, so that any idle worker can then fetch a ready-to-execute subtree without synchronization.*

Specifically, the mechanism only needs each worker to augment its DFS stack with two extra words: a pointer `ptr` that always points to the exposed root node (R, P, X) and a counter `cnt` holding the remaining candidate count $|VP|$, where $VP = P \setminus N(\text{pivot})$. At start-up, `ptr` is pinned to the root of the subtree and `cnt` is set to $|VP|$. While the worker continues its own DFS, idle peers check `cnt`. Whenever `cnt` > 0, the idle worker performs an atomic fetch-and-decrement, obtaining index $i = \texttt{cnt} - 1$, copies the node at `ptr` into local memory, and pulls a new subtree rooted by node (R', P', X') via

$$R' = R \cup \{VP[i]\},$$
$$P' = P \cap N(VP[i]) \setminus \{VP[0, 1, \ldots, i-1]\},$$
$$X' = (X \cup \{VP[0, 1, \ldots, i-1]\}) \cap N(VP[i]).$$

Once `cnt` reaches zero, the worker moves `ptr` one level deeper along the DFS path and recomputes `cnt`. This process repeats until the enumeration is complete.

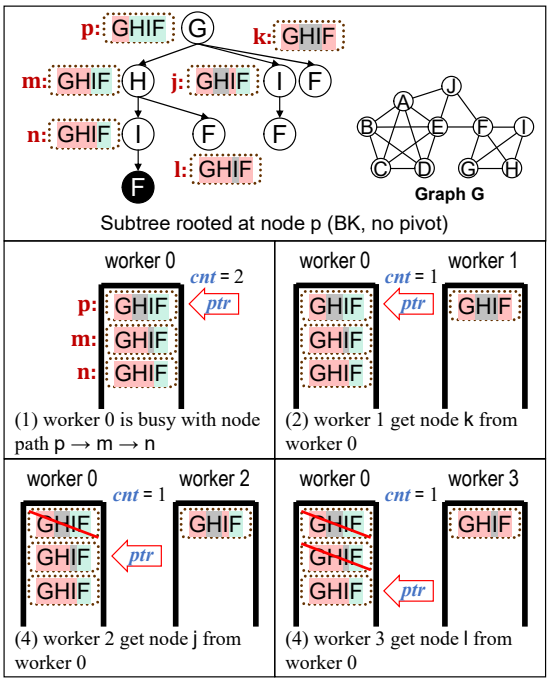


Figure 3. An example of root-down exposure mechanism on the subtree rooted at p in Figure 2, BK pivot rule is disabled.

Example 2. *Figure 3 illustrates the root-down exposure mechanism on the subtree rooted at p in Figure 2. For clarity, we disable pivot rule to retain more nodes, so VP simply equals P.*

Initially, Worker 0 holds the nodes p, m, and n. Step (1): Worker 0 sets `ptr` *to the root node p, computes $VP_p = P_p = \{I, F\}$, and sets* `cnt` *to $|VP_p| = 2$. Step (2): Idle Worker 1 notices that Worker 0's* `cnt` *is positive and then atomically decrements it to 1, receiving index i=* `cnt - 1` *= 1. It then computes $R_k = R_p \cup VP_p[1] = \{G, F\}$, $P_k = P_p \cap N(VP_p[1]) \setminus \{VP_p[0]\} = \varnothing$, and $X_k = (X_p \cup VP_p[0]) \cap N(VP_p[1]) = \{H, I\}$, yielding the node k. Step (3):* `cnt` *is decremented from 1 to 0, giving index $i = 0$. Worker 2 applies the same formulas and also pulls node j. Worker 0 then observes* `cnt` $= 0$*, moves* `ptr` *one level deeper to node m, and recomputes* `cnt` *to $|VP_m| = |P_m| = 1$. Step (4):* `cnt` *drops from 1 to 0 with index $i = 0$, and Worker 3 pulls node l. Worker 0 finds* `cnt` $= 0$ *again, moves* `ptr` *further down the path, and updates* `cnt` *accordingly.*

We analyze the advantages of the root-down exposure mechanism by comparing the load-balancing sequence diagrams of mce-gpu and RDMCE in Figure 4.

Advantage 1: lower overhead and shorter critical path on busy workers. In mce-gpu, a busy worker that receives a balancing request must split its current workload and copy the selected load into a shared buffer—both steps entail non-trivial computation and memory traffic. In RDMCE, the busy worker only checks whether `cnt` $= 0$ and, if so, moves `ptr` one level deeper. These lightweight operations leave the critical path almost untouched.

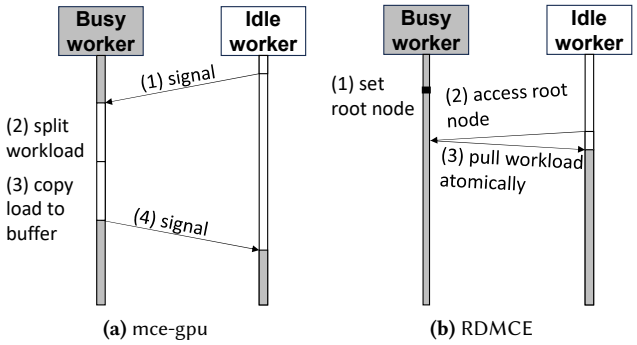


Figure 4. Comparison of load-balancing mechanisms.

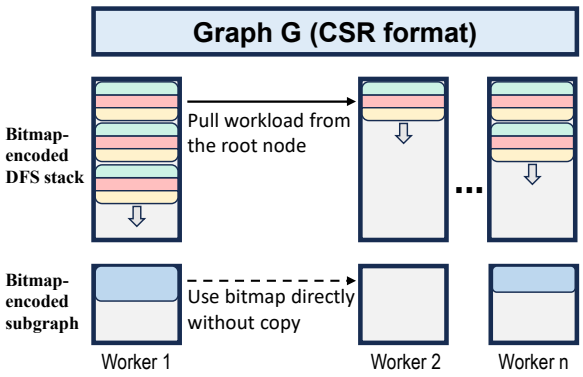


Figure 5. GPU memory layout of RDMCE.

Advantage 2: reduction of global balancing cost. The overall cost of mce-gpu comprises the time spent by the busy worker in steps (2)–(3) and the idle worker's waiting time from step (1) to step (4) in Figure 4a. RDMCE replaces these steps with a single atomic fetch-and-decrement of `cnt` and a local copy of the exposed root; the idle worker never interrupts the busy one, eliminating the synchronization overhead.

Advantage 3: simple and effective workload splitting. Unlike other schemes that require the busy worker to search for splittable workloads, RDMCE always exposes workloads located at the DFS root. This root is the node of minimum depth among all unexplored nodes in the worker's stack, hence its candidate set P has undergone the fewest eliminations and remains largest in cardinality. Consequently, this load splitting is more likely to yield near-perfect load distribution with minimal programmer effort.

The key insight is that the root-down exposure mechanism enables efficient load balancing by **exposing a reference to the workload rather than transferring the workload itself**. Idle workers atomically pull workloads from this reference, which removes synchronization and memory overhead. This method applies broadly to search-tree-based algorithms that can distribute load by exposing root nodes.

4.2 Bitmap-Centric MCE Scheme

We present a bitmap-centric MCE scheme that first dynamically constructs bitmap-encoded subgraphs G_{sub} from the graph's compressed sparse row (CSR) representation. The scheme then enumerates all maximal cliques within G_{sub}, thereby eliminating further access to the global graph G. Our goal is to *simplify the GPU memory layout and focus on high-performance bitmap-based MCE algorithms*. The aforementioned root-down exposure mechanism fits this goal because it needs no extra buffers.

Lemma 1. *Every descendant of a recursion node (R, P, X) depends only on the bipartite adjacency between P and $P \cup X$.*

Proof. By induction on recursion depth: the node itself satisfies the claim; any child is created by picking $v \in P$ and setting $R' = R \cup \{v\}$, $P' = P \cap N(v)$, $X' = X \cap N(v)$, so $P' \subseteq P$ and $(R' \setminus R) \cup P' \cup X' \subseteq P \cup X$, hence the child—and by induction all its descendants—only needs edges between vertices already contained in P and $P \cup X$. □

Hybrid graph representation is an emerging accelerator design for GPU graph mining [8, 27, 35]. The key insight is to exploit locality: hot connections are encoded as dense bitmaps and processed via massively-parallel bitwise operations. For MCE, Lemma 1 guarantees that every descendant of a node (R, P, X) accesses only the bipartite adjacency between P and $P \cup X$; combined with a degeneracy ordering that bounds $|P| \leq d$, the resulting slice is compact. Indeed, as shown in Table 1, $d < 1200$ for the majority of real-world graphs, making this slice ideally suited for bitmap encoding.

Memory layout. The GPU memory layout is straightforward: a global CSR graph $G(V, E)$, a per-worker bitmap subgraph G_{sub}, and a per-worker bitmap DFS stack S. All buffers, worker lists, and other auxiliary structures are removed, as shown in Figure 5.

Graph G is stored in CSR format, enabling efficient vertex neighbourhood accesses with minimum memory footprint bounded by $O(|E|)$. When a worker receives a node (R^*, P^*, X^*) from G, it builds G_{sub}—an array of length $|P^* \cup X^*|$ whose i-th entry is a $|P^*|$-bit mask recording the neighbours of vertex v_i inside P^*. Constructing this bitmap costs $O(\Delta(V)\, d)$ because $|P^* \cup X^*|$ is bounded by $O(\Delta(V))$ and $|P^*| \leq d$. The DFS stack S stores each node (R, P, X) together with the chosen pivot v_{pivot} as a triple (R_b, P_b, VP_b) of $|P^*|$-bit vectors: R_b encodes $R \setminus R^*$, P_b encodes P, and VP_b encodes $P \setminus N(v_{\text{pivot}})$. The omitted set X is implicit in the subgraph bitmap, while VP_b pinpoints the branches to explore. Each triple adds $3|P|$ bits, yielding $O(d^2)$ per worker.

Overall, the *space complexity* is $O(|E| + p\,(\Delta(V)\, d))$, where p is the number of workers. This layout (i) removes all auxiliary structures, (ii) turns irregular CSR accesses into regular bitmap operations, and (iii) enables *zero-copy sharing of the bitmap subgraph during load balancing*, because load balancing is invoked only after the entire graph G has been

Algorithm 2: Bron-Kerbosch on bitmaps

Input: Graph G; initial node (R, P, X)
Output: All maximal cliques in the search tree rooted at (R, P, X)

1 **Procedure** BK_BITMAP(G, R, P, X)**:**
2 $G_{\text{sub}} \leftarrow \{\}$;
3 **for** $v \in P \cup X$ **do**
4 mask $\leftarrow N(v) \cap P$ encoded as $|P|$-bit bitmap;
5 $G_{\text{sub}}[v] \leftarrow$ mask;
6 $R_b \leftarrow |P|$-bit bitmap of 0's; $P_b \leftarrow |P|$-bit bitmap of 1's;
7 $u \leftarrow v \in P \cup X$ maximizing $|G_{\text{sub}}[v]|$;
8 $VP_b \leftarrow P_b \setminus G_{\text{sub}}[u]$;
9 BIT_KERNEL$(G_{sub}, R_b, P_b, VP_b)$
10 **Procedure** BIT_KERNEL$(G_{sub}, R^*_b, P^*_b, VP^*_b)$**:**
11 $l \leftarrow 0$; ptr $\leftarrow -1$; cnt $\leftarrow 0$;
12 $S[0] \leftarrow (R^*_b, P^*_b, VP^*_b)$;
13 **while** $l \geq 0$ **do**
14 **while** ptr $< l$ *and* cnt $= 0$ **do**
15 ptr $\leftarrow$ ptr $+ 1$;
16 $(R_b, P_b, VP_b) \leftarrow S[\text{ptr}]$;
17 cnt $\leftarrow |VP_b|$;
18 **if** ptr $= l$ **then**
19 $idx \leftarrow$ **atomicSub**(cnt, 1);
20 **if** $idx < 0$ **then**
21 **break**;
22 **else**
23 $(R_b, P_b, VP_b) \leftarrow S[l]$;
24 $R'_b \leftarrow R_b \cup \{VP_b[idx]\}$;
25 $P'_b \leftarrow P_b \cap G_{\text{sub}}[VP_b[idx]] \setminus \{VP_b[0], VP_b[1], \ldots VP_b[idx-1]\}$;
26 $u \leftarrow v \in P'_b$ maximizing $|G_{\text{sub}}[v] \cap P'_b|$;
27 $VP'_b \leftarrow P'_b \setminus G_{\text{sub}}[u']$;
28 $l \leftarrow l + 1$; $S[l] \leftarrow (R'_b, P'_b, VP'_b)$;
29 $(R_b, P_b, VP_b) \leftarrow S[l]$;
30 **if** $VP_b = \emptyset$ **then**
31 **if** $R_b \not\subseteq G_{sub}[v]$ *for all* $v \in P \cup X$ **then**
32 **Output** $R \cup R_b$
33 $l \leftarrow l - 1$;
34 **else**
35 $u \leftarrow$ a vertex in VP_b;
36 $VP_b \leftarrow VP_b \setminus \{u\}$;
37 $R'_b \leftarrow R_b \cup \{u\}$; $P'_b \leftarrow P_b \cap G_{\text{sub}}[u]$;
38 $C \leftarrow \{ v \in P \cup X \mid R'_b \subseteq G_{\text{sub}}[v] \}$;
39 $u' \leftarrow v \in C$ maximizing $|G_{\text{sub}}[v] \cap P'_b|$;
40 $VP'_b \leftarrow P'_b \setminus G_{\text{sub}}[u']$;
41 $l \leftarrow l + 1$; $S[l] \leftarrow (R'_b, P'_b, VP'_b)$;

processed, busy workers never overwrite their subgraph. mce-gpu cannot do this, since its compressed-X structure is dynamic and must be copied at cost $O(\Delta(V))$.

Computation on Bitmaps. Exploiting bitmap-encoded subgraphs and a DFS stack, we present a GPU-friendly MCE algorithm that reorganizes the classic Bron-Kerbosch routine into three concise steps: (1) replacing recursion with an iterative kernel that processes compact, bitmap-encoded node descriptors, avoiding the high cost of GPU recursion; (2) collapsing all set operations—including node generation, neighbor pruning, and maximality checks—into efficient bitwise instructions on these descriptors; (3) integrating our root-down exposure mechanism, which seamlessly hands load-balancing duties to idle workers without extra synchronization or data movement. Next, we detail the kernel design, bitmap representation, and their seamless integration that together enable high-throughput execution on GPUs.

Algorithm 2 details the workflow BK_BITMAP. Each worker begins by scanning every vertex in $P \cup X$ of the received root (R, P, X) and encoding their neighbours inside P as $|P|$-bit masks, producing the array G_{sub} (lines #2-5). From these masks it forms a bitmap triple (R_b, P_b, VP_b), producing an all-zero mask for R_b, an all-one mask for P_b, and the pivot-pruned set VP_b (lines #6-8). Then, kernel BIT_KERNEL is launched with G_{sub} and that node (line #9). The iterative kernel BIT_KERNEL replaces recursion: starting from level $l = 0$ (line #11), it repeatedly pops the current node (R_b, P_b, VP_b) (line #29); when VP_b is empty, it tests maximality by scanning G_{sub}, checking whether no vertex in G_{sub} contains R_b in its neighbourhood and outputs the clique before decrementing l (lines #31-33). Whenever VP_b is non-empty, the next vertex is removed without replacement (lines #35-36), the successor triple is computed via bitwise intersections (lines #37-40), and the new entry is pushed onto the stack (line #41).

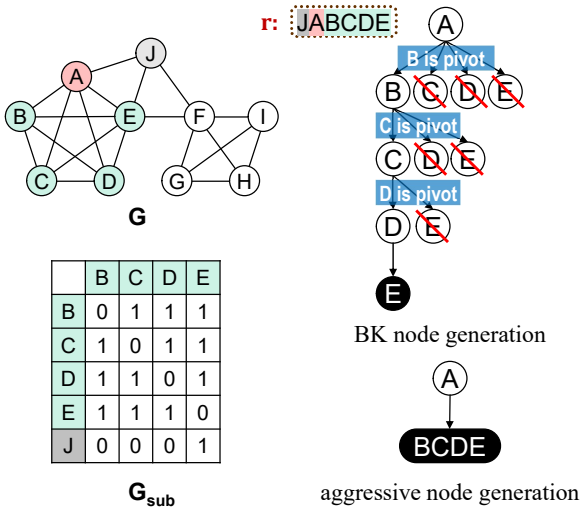
Blue lines add root-down exposure via an extra pointer ptr and counter cnt. Workers initialize both variables (line #11) and, whenever the exposed node ptr equals the current level l, the local worker first checks whether cnt is zero; if so, BIT_KERNEL exits (lines #20-21). Otherwise, it atomically fetches the next index and pushes the new triple onto the stack (lines #23-28). Any idle worker performs the same atomic fetch-and-push sequence (lines #23-28) without synchronization with the busy worker.

4.3 Aggressive Node Generation

To reduce intermediate nodes in the search tree, we propose aggressive node generation that *moves multiple vertices from P to R in one step instead of adding them one by one.*

Lemma 2. *For a node (R, P, X), any vertex $v \in P$ with $|N(v) \cap P| = |P| - 1$ belongs to every maximal clique that is eventually output by some node in the subtree rooted at (R, P, X).*

Proof. Such v satisfies $N(v) \supseteq R \cup (P \setminus \{v\})$. For any node (R', P', X') in the subtree outputting maximal clique R', we have $R' \subseteq R \cup P$ and $R' \cap X' = \varnothing$. If $v \notin R'$, then $R' \subseteq R \cup (P \setminus \{v\}) \subseteq N(v)$, thus $R' \cup \{v\}$ is a larger clique—contradicting the maximality of R'. Therefore, the lemma holds. □



	B	C	D	E
B	0	1	1	1
C	1	0	1	1
D	1	1	0	1
E	1	1	1	0
J	0	0	0	1

Figure 6. An example of aggressive node generation. Extracted from Figure 2: left is graph G and node r induced subgraph G_{sub}; right is the subtrees rooted at r produced by standard BK (top) and the aggressive method (bottom).

We exploit Lemma 2 to accelerate enumeration. Whenever we process a node (R, P, X), we immediately move every vertex $v \in P$ with $|N(v) \cap P| = |P| - 1$ into R in a single step. This overhead is negligible, since the pivot-selection step already computes $|P \cap N(v)|$ to choose the vertex with the largest remaining candidate set (line #8 in Algorithm 1; lines #7, 26, 39 in Algorithm 2). We simply flag the qualifying vertices and merge them into R together.

Example 3. *Figure 6 illustrates aggressive node generation approach for the subtree rooted at node r in Figure 2. The baseline BK inserts vertices one at a time, creating three intermediate nodes while expanding R with B, C, D, E. Instead, our method observes $|N(v) \cap P_r| = 3 = |P_r| - 1$ for each vertex in {B,C,D,E} during pivot computation, so it adds all four vertices to R_r in one step and produces no intermediates.*

5 Implementation

Workflow of RDMCE. Initially, we sort vertices in degeneracy order and convert the graph to CSR format. Next, we seed procedure `BK_BITMAP` in Algorithm 2 with nodes (R, P, X), each derived from an edge (v_x, v_y) by setting $R = \{v_x, v_y\}$, $P = N(v_x) \cap N(v_y) \cap \{v_{y+1}, \ldots, v_{|V|}\}$, and $X = N(v_x) \cap N(v_y) \cap \{v_1, \ldots, v_{y-1}\}$. To give every worker a unique edge, we use an atomic edge counter that distributes edges via the AtomicInc primitive. We initialize with edges rather than vertices because $|E| > |V|$ provides more parallelism. After all global edge seeds are exhausted,the root-down mechanism activates for load balancing. A worker that finishes its enumeration repeatedly checks exposed nodes until every worker completes.

Parallel strategy. We adopt the warp-centric scheme of [9, 26], regarding each warp as a worker that processes a subtree independently. Initially, we launch sufficient warps to saturate GPUs, creating `warpPerSM` warps per SM on average under the persistent thread programming model [12]. The selection of `warpPerSM` trades overall parallelism against the resources available to each warp. We further examine this trade-off in Section 6.4.

Exposure threshold. The root-down exposure mechanism exposes a node only when its candidate set size $|P|$ exceeds a threshold τ. This threshold embodies a crucial trade-off: a value too low results in excessive exposure, flooding the GPU with fine-grained atomic operations; conversely, a value too high severely restricts workload sharing, leading to underutilization as warps sit idle. After empirical testing, we set $\tau = 24$, a value that effectively balances these concerns—maintaining high warp occupancy while minimizing the cost of atomic operations, as detailed in Section 6.4.

6 Evaluation

In this section, we select the most time-consuming real-world datasets from recent studies to validate the effectiveness of RDMCE and our proposed techniques.

6.1 Experimental Setup

Platform. All experiments run on Ubuntu 24.04.2 LTS with Linux 6.8.0-71-generic. The host has an Intel Xeon Platinum 8358P CPU (32 cores, 2.60 GHz) and 512 GB of RAM. The system is equipped with eight NVIDIA RTX 4090 GPUs [24], each providing 24 GB of global memory and 128 SMs, with CUDA 12.4 installed. Unless stated otherwise, all runs use this configuration.

Datasets. Table 1 lists the 12 real-world graphs used in our experiments. These graphs are taken from three public repositories—KONECT [15], SNAP [16], and Network Repository [30]. Each is pre-processed into a simple undirected graph by removing self-loops and duplicate edges. They represent the most time-consuming instances used in recent studies [2, 10, 40]. We list them in ascending order of maximal-clique count, because running time is strongly correlated with this metric.

Baselines. State-of-the-art GPU-based MCE efforts comprise GBK [35], mce-gpu [2], and G^2-AIMD [40] . Because GBK is early and its code is unavailable, while both mce-gpu and G^2-AIMD report significantly better performance than GBK, we take the latter two as our baselines. We use the open-source releases of mce-gpu (variants mce-gpu-P and mce-gpu-PX) and G^2-AIMD and run each with its default settings on identical hardware and datasets for a fair comparison. We omit CPU baselines, because even the latest 32-thread parRMCE [10] needs about 1.5 hours on com-orkut, whereas every GPU baseline finishes within one minute.

Table 1. Evaluation datasets.

Dataset	Abbr	Source	$\|V\|$	$\|E\|$	$\Delta(V)$	d	Maximal Clique
Flickr	fi	KONECT	105,938	2,316,948	5,425	573	199,203,908
Twitter-higgs	hig	SNAP	456,626	12,508,413	51,386	125	433,411,770
Wiki-link	wlk	NETWORK	25,921,548	543,183,611	4,271,341	1,120	568,730,123
Facebook	fb	SNAP	4,039	88,234	1,045	115	869,325,383
StackOverflow	st	NETWORK	2,584,164	28,183,518	44,065	198	918,563,917
web-ClueWeb09-50m	cw9	NETWORK	147,925,593	446,766,953	308,477	192	1,001,323,679
soc-Sinaweibo	ssn	NETWORK	58,655,850	261,321,033	278,489	193	1,117,416,174
soc-Orkut-dir	okd	NETWORK	3,072,441	117,185,083	33,313	253	2,269,631,973
com-Orkut	or	KONECT	3,072,441	117,184,899	33,313	253	2,270,456,447
com-Friendster	frd	SNAP	65,608,366	1,806,067,135	5,214	304	3,364,773,700
Wikipedia-link-en	wen	KONECT	13,593,032	334,591,525	1,052,326	1,114	6,079,659,726
Dogster	dg	KONECT	426,816	8,543,549	46,503	248	145,906,757,880

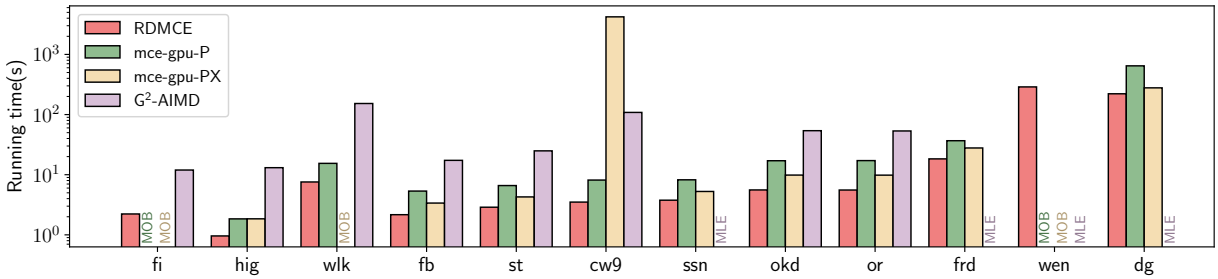


Figure 7. Overall evaluation (log scaled). MOB means variants of mce-gpu aborted by Memory Out-of-Bounds and MLE means G²-AIMD aborted by Memory Limit Exceeded.

Measurements. We record running time from the moment the host sends the graph to the GPU until the GPU returns the total count of maximal cliques, excluding the common degeneracy ordering, graph loading from disks, and G²-AIMD's graph partitioning. Each value is the average of five runs on a single NVIDIA RTX 4090. Unless noted otherwise, RDMCE runs with `warpPerSM` = 20, exposure threshold $\tau = 24$, and aggressive node generation enabled.

6.2 Overall Evaluation

Figure 7 presents a comparison of running times between RDMCE and other GPU-based competitors. RDMCE completes on all datasets, while the other solutions only succeed on a subset. Among completed runs, RDMCE is 1.25-5.38× faster than any next-best solutions. All three competitors fail on Wikipedia-link-en due to its high maximum degree ($\Delta V > 10^6$), large degeneracy ($d > 10^3$), and extensive number of maximal cliques ($> 6{\times}10^9$). Variants of mce-gpu fail on Flickr and Wiki-link due to out-of-bounds accesses in load-balancing buffers. G²-AIMD fails on StackOverflow, com-Friendster, Wikipedia-link-en, and Dogster due to memory-limit errors. This occurs because even with small chunks, its BFS enumeration requires $O(3^{|V'|/3})$ memory per chunk [31], which often exceeds GPU capacity. Furthermore, its frequent CPU-GPU data transfers incur significant overhead, making it the slowest baseline in most cases. RDMCE achieves superior performance and robustness thanks to its efficient load-balancing mechanism and compact memory layout.

6.3 Breakdown Evaluation

Effect of the root-down exposure mechanism. To validate the root-down exposure mechanism in Section 4.1, we disable it in variant RDMCE-NB. Figure 8 shows that RDMCE is 1.03–69.61× faster than RDMCE-NB, because this mechanism successfully balances workloads across all datasets. Specifically, on Dogster with 146 billion maximal cliques, this mechanism reduces the running time from 4.3 hours to 221.6 seconds, because larger maximal clique counts tend to create more unbalanced workloads, and root-down exposure addresses this problem. On com-Friendster with 1.8 billion edges, the speed-up is only 1.03×, because numerous edges initialize extensive parallel workloads (Section 5) that tend to hide the longest critical path.

Effect of the bitmap-centric MCE scheme. To quantify the memory savings of our bitmap-centric MCE scheme, we compare the peak GPU memory usage of RDMCE and mce-gpu-PX under the same configuration (12 warps per SM) across all datasets supported by mce-gpu-PX. As both

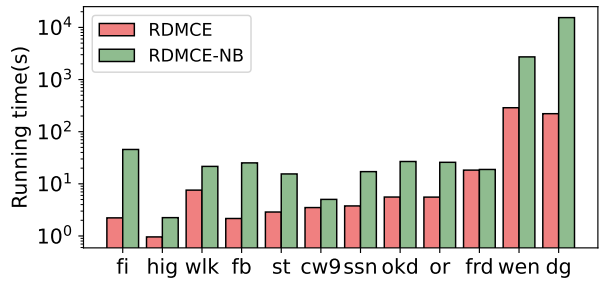


Figure 8. Effect of the root-down exposure mechanism (log scale). RDMCE-NB is RDMCE with this mechanism disabled.

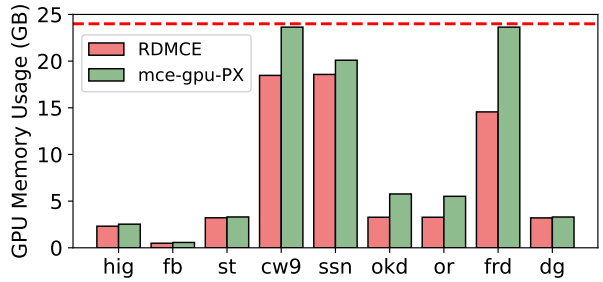


Figure 9. Effect of the bitmap-centric MCE scheme. The red line marks the 24 GB capacity of an NVIDIA RTX 4090.

algorithms run fully on-device and each warp encodes the connections of P and $P \cup X$ using bitmaps, the comparison directly reflects memory layout efficiency. As shown in Figure 9, RDMCE uses only 56.8%–97.3% of the memory required by mce-gpu-PX, with an average of 80.0%, due to its simplified memory layout.

Effect of aggressive node generation. To evaluate the aggressive node generation proposed in Section 4.3, we disabled this optimization in a variant named RDMCE-NP. Table 2 shows RDMCE is 1.018× faster than RDMCE-NP on average due to its efficient removal of intermediate nodes, exemplified in Figure 6. The actual speedup depends on the number of such nodes during execution. RDMCE achieves significant improvements of 4.8% on soc-Sinaweibo and 6.4% on StackOverflow. In contrast, on Twitter-higgs and web-ClueWeb09-50m, where intermediate nodes are scarce, the additional checking overhead outweighs the benefits, resulting in minor slowdowns of 0.006 and 0.002 seconds, respectively.

6.4 Sensitivity Analysis

Edge vs. vertex node initialization. To evaluate the node seeding strategy in Section 5, we create RDMCE-vertex, a variant that seeds `BK_BITMAP` with nodes initialized by vertices instead of edges. Figure 10 shows that RDMCE outperforms RDMCE-vertex on all graphs except Flickr, web-ClueWeb09-50m and Wikipedia-link-en, and achieves a 2.52× average speed-up because $|E| > |V|$ supplies more initial works and thus higher parallelism.

Table 2. Effect of aggressive node generation. RDMCE-NP is RDMCE with the optimization disabled.

Dataset	fi	hig	wlk	fb	st	cw9
RDMCE	2.225	0.960	7.574	2.164	2.881	3.505
RDMCE-NP	2.270	0.954	7.616	2.243	3.067	3.503

Dataset	ssn	okd	or	frd	wen	dg
RDMCE	3.769	5.581	5.563	18.330	288.255	221.579
RDMCE-NP	3.952	5.639	5.624	18.810	288.345	222.773

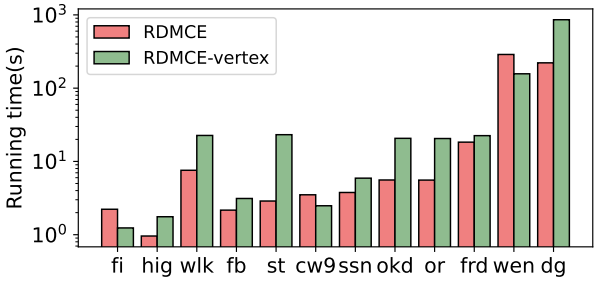


Figure 10. Edge vs. vertex node initialization (log scale).

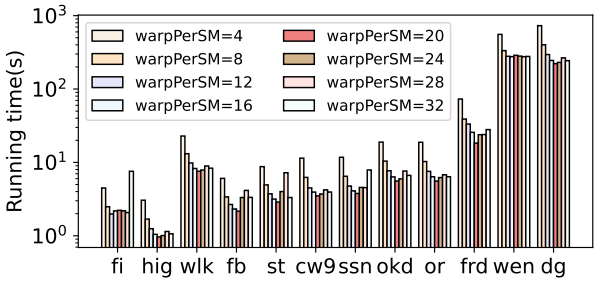


Figure 11. Impact of `warpPerSM` (log scale).

Impact of the number of warps in each SM. To decide `warpPerSM` in Section 5, we run RDMCE on every dataset with `warpPerSM` set to 4, 8, 12, 16, 20, 24, 28, and 32, as shown in Figure 11. The selection balances two factors: more warps increase parallelism, while fewer warps grant each warp larger register and shared-memory shares. As the warp count increases, running time first falls and then rises, forming a U-shape. We set `warpPerSM=20`, as it yields the best time on every dataset except Flickr and Wikipedia-link-en.

Impact of the threshold for root-down exposure. To determine the exposure threshold τ for root-down exposure in Section 5, we run RDMCE on every dataset with τ set to 16, 24, 32, 48, 64, and 96, as shown in Figure 12. A small τ splits work into finer pieces, so load is balanced but more synchronizations are needed, while a large τ cuts the synchronization cost but can leave some workers idle. We set τ to 24, because it reports the shortest running time on over 50% testing datasets.

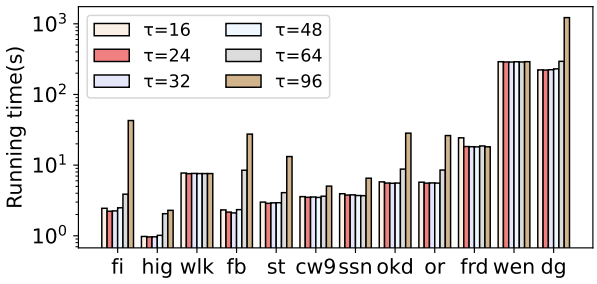


Figure 12. Impact of τ for root-down exposure (log scale).

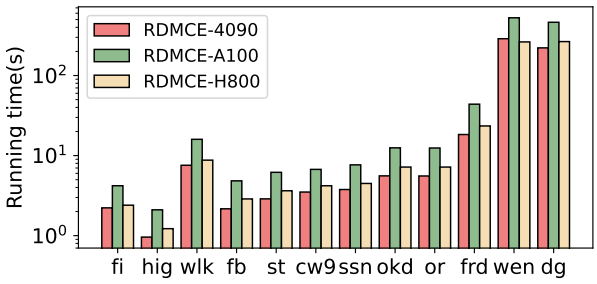


Figure 13. Adaptability of RDMCE on different GPUs (log scale).

Adaptability on different GPUs. To evaluate the adaptability of RDMCE across GPU architectures, we test it on an NVIDIA RTX 4090 (128 SMs, 24 GB), an NVIDIA A100 (108 SMs, 40 GB) [25], and an NVIDIA H800 (132 SMs, 80 GB) [23]. On all datasets except Wikipedia-link-en, RDMCE performs fastest on the RTX 4090, followed by the H800, and slowest on the A100. The H800 outperforms the A100 by 1.76× on average, owing to its higher SM count, double the memory, and greater bandwidth. Despite having fewer SMs, less memory, and lower bandwidth, the RTX 4090 surpasses the H800 by 1.21× on average, benefiting from its abundant CUDA cores which accelerate integer-intensive MCE computation. On Wikipedia-link-en, which exhibits high memory pressure due to its large scale and enormous cliques, the H800 achieves a 1.10× speedup over the RTX 4090, leveraging its higher memory bandwidth and additional SMs.

Scalability on multiple GPUs. To evaluate the scalability of RDMCE, we implement a multi-GPU version of RDMCE by sharing the edge counter across all GPUs so that each worker can obtain a unique edge via the AtomicInc_system primitive. The root-down exposure mechanism is applied within each GPU to avoid costly cross-GPU memory traffic. Figure 14 shows the running time of RDMCE on 1, 2, 4, and 8 NVIDIA RTX 4090 GPUs on the Dogster graph, which contains 146 billion maximal cliques. The running time drops steadily as more GPUs are added, and every GPU finishes almost simultaneously. With eight GPUs, RDMCE reduces the running time from 221.6 seconds to 45.9 seconds, achieving a 4.83× speed-up.

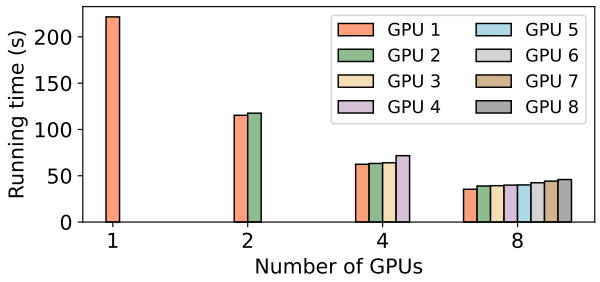


Figure 14. Scalability of RDMCE on 8× NVIDIA RTX 4090 GPUs using Dogster dataset with 146B maximal cliques.

7 Related Work

Maximal Clique Enumeration on CPUs. Most mainstream MCE solutions build on the classic Bron–Kerbosch framework [6], augmenting it with pivoting [31] to prune branches early and degeneracy ordering [11] to bound recursion depth. Recent work refines this framework through top-down pruning [18], hash-joined set intersection [4], hybrid data structures [14], differentiate the set intersection [32], and graph reduction optimizations [10]. Subsequent efforts have ported these optimizations to multicore CPUs [4, 5, 10, 32] to pursue the best performance. However, limited by the finite CPU core count, they achieve only limited speed-ups compared to the state-of-the-art GPU solution [2] on large datasets with billions of maximal cliques.

Graph Mining on GPUs. Graph mining on GPUs has become a hot topic in recent years, covering maximal clique enumeration [2, 35, 40], maximal biclique enumeration [13, 26, 27], general graph pattern mining [8, 9, 19, 34], and other specialized patterns [3, 33, 41]. For large graph patterns, search trees diverge sharply in size, causing severe load imbalance. Existing load-balancing mechanisms keep busy workers active in rebalancing through worker lists [2], task queues [26], or hierarchical work stealing [34]. We instead adopt a root-down exposure mechanism that lets busy workers only expose the root node, leaving balancing to idle peers.

8 Conclusion

This paper presents RDMCE, a high-performance GPU-based solution for maximal clique enumeration (MCE). Existing GPU MCE solutions require busy workers to pause and redistribute workloads, incurring synchronization overhead and extra memory usage. In contrast, we propose a root-down exposure mechanism, in which workers continuously expose their current root node, enabling idle workers to pull workloads directly without synchronization. This method can be generalized to any search-tree-based algorithm. Complemented by bitmap-centric MCE scheme and aggressive node generation, RDMCE expedites enumeration and reduces memory usage. Extensive experiments show that RDMCE outperforms all existing GPU solutions by 1.25-5.38×.

References

[1] Meriem Adraoui, Asmaâ Retbi, Mohammed Khalidi Idrissi, and Samir Bennani. 2022. Maximal cliques based method for detecting and evaluating learning communities in social networks. *Future Generation Computer Systems* 126 (2022), 1–14.

[2] Mohammad Almasri, Yen-Hsiang Chang, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. 2023. Parallelizing Maximal Clique Enumeration on GPUs. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 162–175.

[3] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. 2022. Parallel K-clique counting on GPUs. In *Proceedings of the 36th ACM International Conference on Supercomputing* (Virtual Event) *(ICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 21, 14 pages.

[4] Jovan Blanuša, Radu Stoica, Paolo Ienne, and Kubilay Atasu. 2020. Manycore clique enumeration with fast set intersections. *Proc. VLDB Endow.* 13, 12 (July 2020), 2676–2690.

[5] Jovan Blanuša, Radu Stoica, Paolo Ienne, and Kubilay Atasu. 2020. Parallelizing Maximal Clique Enumeration on Modern Manycore Processors. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 211–214.

[6] Coen Bron and Joep Kerbosch. 1973. Finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (Sept. 1973), 575–577.

[7] Martin Burtscher, Rupesh Nasre, and Keshav Pingali. 2012. A Quantitative Study of Irregular Programs on GPUs. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*. 141–151.

[8] Weichen Cao, Ke Meng, Zhiheng Lin, and Guangming Tan. 2025. GLumin: Fast Connectivity Check Based on LUTs For Efficient Graph Pattern Mining. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Las Vegas, NV, USA) *(PPoPP '25)*. Association for Computing Machinery, New York, NY, USA, 455–468.

[9] Xuhao Chen and Arvind. 2022. Efficient and Scalable Graph Pattern Mining on GPUs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 857–877. https://www.usenix.org/conference/osdi22/presentation/chen

[10] Wen Deng, Weiguo Zheng, and Hong Cheng. 2024. Accelerating Maximal Clique Enumeration via Graph Reduction. *Proceedings of the VLDB Endowment* 17, 10 (June 2024), 2419–3431.

[11] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *Algorithms and Computation*, Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 403–414.

[12] Kshitij Gupta, Jeff A. Stuart, and John D. Owens. 2012. A study of Persistent Threads style GPU programming for GPGPU workloads. In *2012 Innovative Parallel Computing (InPar)*. 1–14.

[13] Chou-Ying Hsieh, Chia-Ming Chang, Po-Hsiu Cheng, and Sy-Yen Kuo. 2024. Accelerating Maximal Biclique Enumeration on GPUs. *arXiv preprint:2401.05039* (2024). arXiv:2401.05039 [cs.DC]

[14] Yan Jin, Bowen Xiong, Kun He, Yangming Zhou, and Yi Zhou. 2022. On fast enumeration of maximal cliques in large graphs. *Expert Systems with Applications* 187 (2022), 115915.

[15] Jérôme Kunegis. 2013. KONECT – The Koblenz Network Collection. In *Proc. of the 22nd International Conference on World Wide Web Companion*. 1343–1350.

[16] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data

[17] Jinxi Li, James D. Glover, Haiguo Zhang, Meifang Peng, Jingze Tan, Chandana Basu Mallick, Dan Hou, Yajun Yang, Sijie Wu, Yu Liu, Qianqian Peng, Shijie C. Zheng, Edie I. Crosse, Alexander Medvinsky, Richard A. Anderson, Helen Brown, Ziyu Yuan, Shen Zhou, Yanqing Xu, John P. Kemp, Yvonne Y.W. Ho, Danuta Z. Loesch, Lizhong Wang, Yingxiang Li, Senwei Tang, Xiaoli Wu, Robin G. Walters, Kuang Lin, Ruogu Meng, Jun Lv, Jonathan M. Chernus, Katherine Neiswanger, Eleanor Feingold, David M. Evans, Sarah E. Medland, Nicholas G. Martin, Seth M. Weinberg, Mary L. Marazita, Gang Chen, Zhengming Chen, Yong Zhou, Michael Cheeseman, Lan Wang, Li Jin, Denis J. Headon, and Sijia Wang. 2022. Limb development genes underlie variation in human fingerprint patterns. *Cell* 185, 1 (06 01 2022), 95–112.e18.

[18] Yinuo Li, Zhiyuan Shao, Dongxiao Yu, Xiaofei Liao, and Hai Jin. 2019. Fast Maximal Clique Enumeration for Real-World Graphs. In *Database Systems for Advanced Applications*, Guoliang Li, Jun Yang, Joao Gama, Juggapong Natwichai, and Yongxin Tong (Eds.). Springer International Publishing, Cham, 641–658.

[19] Zhiheng Lin, Ke Meng, Chaoyang Shui, Kewei Zhang, Junmin Xiao, and Guangming Tan. 2024. Exploiting Fine-Grained Redundancy in Set-Centric Graph Pattern Mining. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Edinburgh, United Kingdom) *(PPoPP '24)*. Association for Computing Machinery, New York, NY, USA, 175–187.

[20] Zhenqi Lu, Johan Wahlström, and Arye Nehorai. 2018. Community Detection in Complex Networks via Clique Conductance. *Scientific Reports* 8, 1 (13 04 2018), 5982.

[21] Fatemeh Mohebbi, Alex Zelikovsky, Serghei Mangul, Gerardo Chowell, and Pavel Skums. 2024. Early detection of emerging viral variants through analysis of community structure of coordinated substitution networks. *Nature Communications* 15, 1 (02 04 2024), 2838.

[22] J. W. Moon and L. Moser. 1965. On cliques in graphs. *Israel Journal of Mathematics* 3, 1 (1965), 23–28.

[23] NVIDIA Corporation. 2023. *NVIDIA H800 Tensor Core GPU Datasheet*. NVIDIA. https://chaoqing-i.com/upload/20231128/NVIDIA%20H800%20GPU%20Datasheet.pdf Accessed: 2025-08-27.

[24] NVIDIA Corporation. 2025. GeForce RTX 4090. https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/ Accessed: 2025-08-27.

[25] NVIDIA Corporation. 2025. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/ Accessed: 2025-08-27.

[26] Zhe Pan, Shuibing He, Xu Li, Xuechen Zhang, Rui Wang, and Gang Chen. 2023. Efficient Maximal Biclique Enumeration on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 16, 13 pages.

[27] Zhe Pan, Shuibing He, Xu Li, Xuechen Zhang, Rui Wang, Yanlong Yin, and Gang Chen. 2025. Advanced Maximal Biclique Enumeration on GPUs Using Bitmaps. *IEEE Trans. Comput.* 74, 8 (2025), 2552–2566.

[28] Jonas Paulsen, Tharvesh M. Liyakat Ali, Maxim Nekrasov, Erwan Delbarre, Marie-Odile Baudement, Sebastian Kurscheid, David Tremethick, and Philippe Collas. 2019. Long-range interactions between topologically associating domains shape the four-dimensional genome during differentiation. *Nature Genetics* 51, 5 (01 05 2019), 835–843.

[29] Chao Pei, Shu-Lin Wang, Jianwen Fang, and Wei Zhang. 2017. GSMC: Combining Parallel Gibbs Sampling with Maximal Cliques for Hunting DNA Motif. *Journal of Computational Biology* 24, 12 (2017), 1243–1253. PMID: 29116820.

[30] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proc. of the 29th AAAI Conference on Artificial Intelligence*. 4292–4293.

[31] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science* 363, 1 (2006), 28–42. Computing and Combinatorics.

[32] Hans Vandierendonck. 2024. Differentiating Set Intersections in Maximal Clique Enumeration by Function and Subproblem Size. In *Proceedings of the 38th ACM International Conference on Supercomputing*

(Kyoto, Japan) *(ICS '24)*. Association for Computing Machinery, New York, NY, USA, 150–163.

[33] Zhibin Wang, Longbin Lai, Yixue Liu, Bing Shui, Chen Tian, and Sheng Zhong. 2023. I/O-Efficient Butterfly Counting at Scale. *Proc. ACM Manag. Data* 1, 1, Article 34 (May 2023), 27 pages.

[34] Yihua Wei and Peng Jiang. 2022. STMatch: accelerating graph pattern matching on GPU with stack-based loop optimizations. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Dallas, Texas) *(SC '22)*. IEEE Press, Article 53, 13 pages.

[35] Yi-Wen Wei, Wei-Mei Chen, and Hsin-Hung Tsai. 2021. Accelerating the Bron-Kerbosch Algorithm for Maximal Clique Enumeration Using GPUs. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021), 2352–2366.

[36] Xuyun Wen, Wei-Neng Chen, Ying Lin, Tianlong Gu, Huaxiang Zhang, Yun Li, Yilong Yin, and Jun Zhang. 2017. A Maximal Clique Based Multiobjective Evolutionary Algorithm for Overlapping Community Detection. *IEEE Transactions on Evolutionary Computation* 21, 3 (2017), 363–377.

[37] Martin Winter, Mathias Parger, Daniel Mlakar, and Markus Steinberger. 2021. Are dynamic memory managers on GPUs slow? a survey and benchmarks. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 219–233.

[38] Yue Wu, Xidao hu, Yongzhe Yuan, Xiaolong Fan, Maoguo Gong, Hao Li, Mingyang Zhang, Qiguang Miao, and Wenping Ma. 2024. PointMC: Multi-instance Point Cloud Registration based on Maximal Cliques. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

[39] Jiaqi Yang, Xiyu Zhang, Peng Wang, Yulan Guo, Kun Sun, Qiao Wu, Shikun Zhang, and Yanning Zhang. 2024. MAC: Maximal Cliques for 3D Registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (2024), 10645–10662.

[40] Lyuheng Yuan, Akhlaque Ahmad, Da Yan, Jiao Han, Saugat Adhikari, Xiaodong Yu, and Yang Zhou. 2024. G2-AIMD: A Memory-Efficient Subgraph-Centric Framework for Efficient Subgraph Finding on GPUs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 3164–3177.

[41] Zhigao Zheng, Guojia Wan, Jiawei Jiang, Chuang Hu, Hao Liu, Shahid Mumtaz, and Bo Du. 2025. Lock-Free Triangle Counting on GPU. *IEEE Trans. Comput.* 74, 3 (2025), 1040–1052.