Automated Task Scheduling for Cloth and Deformable Body Simulations in Heterogeneous Computing Environments

CHENGZHU HE, Xiamen University, China and Style3D Research, China
ZHENDONG WANG, Style3D Research, China
ZHAORUI MENG, Xiamen University, China
JUNFENG YAO*, Xiamen University, China
SHIHUI GUO, Xiamen University, China
HUAMIN WANG, Style3D Research, China

(a) Multilayer Dress (b) Basketball Jersey

Fig. 1. By performing heterogeneous computation across both the CPU and GPU cores of Apple's M-series processors, an XPBD simulator enhanced with our auto-scheduling method achieves a speedup of 40% to 60% compared to computation using GPU cores alone, as shown in the two above examples.

The concept of the Internet of Things (IoT) has driven the development of system-on-a-chip (SoC) technology for embedded and mobile systems, which may define the future of next-generation computation. In SoC devices, efficient cloth and deformable body simulations require parallelized, heterogeneous computation across multiple processing units. The key challenge in heterogeneous computation lies in task distribution, which must account for varying inter-task dependencies and communication costs. This paper proposes a novel framework for automated task scheduling to optimize simulation performance by minimizing communication overhead and aligning tasks with the specific strengths of each device. To achieve this, we introduce an efficient scheduling method based on the Heterogeneous Earliest Finish Time (HEFT) algorithm, adapted for hybrid systems. We model simulation tasks—such as those in iterative methods like Jacobi and Gauss-Seidel—as a Directed Acyclic Graph (DAG). To maximize the parallelism of nonlinear Gauss-Seidel simulation tasks, we present an innovative asynchronous Gauss-Seidel method with specialized data synchronization across units. Additionally, we employ task merging and tailored task-sorting strategies for Gauss-Seidel tasks to achieve an optimal balance between convergence and efficiency. We validate the effectiveness of our framework across various simulations, including XPBD, vertex block descent, and second-order stencil descent, using Apple M-series processors with both CPU and GPU cores. By maximizing computational efficiency and reducing processing times, our method achieves superior simulation frame rates compared to approaches that rely on individual devices in isolation. The source code with hybrid Metal/C++ implementation is available at *https://github.com/ChengzhuUwU/libAtsSim*.

CCS Concepts: • **Computing methodologies** → **Physical Simulation**; **Modeling Methodologies**.

Additional Key Words and Phrases: task scheduling, heterogeneous computing, asynchronous parallelism

*Corresponding author.

Authors' Contact Information: Chengzhu He, Xiamen University, Xiamen, Fujian, China and Style3D Research, Hangzhou, Zhejiang, China, 38120231150160@xmu.edu.cn; Zhendong Wang, Style3D Research, Hangzhou, Zhejiang, China, wang.zhendong.619@gmail.com; Zhaorui Meng, Xiamen University, Xiamen, Fujian, China, 19020231153638@xmu.edu.cn; Junfeng Yao, Xiamen University, Xiamen, Fujian, China, yao0010@xmu.edu.cn; Shihui Guo, Xiamen University, Xiamen, Fujian, China, guoshihui@xmu.edu.cn; Huamin Wang, Style3D Research, Hangzhou, Zhejiang, China, wanghmin@gmail.com.

SIGGRAPH Conference Papers '25, Vancouver, BC, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Chengzhu He, Zhendong Wang, Zhaorui Meng, Junfeng Yao, Shihui Guo, and Huamin Wang. 2025. Automated Task Scheduling for Cloth and Deformable Body Simulations in Heterogeneous Computing Environments. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Modern computing systems often integrate multiple powerful processing units, including CPUs, GPUs, and specialized accelerators like TPUs and NPUs for AI and data-intensive tasks. An emerging challenge in improving simulation performance lies in determining whether cloth and deformable body simulations can be effectively parallelized across these heterogeneous units. Compared to simulations on single units, such as a CPU or GPU [Wu et al. 2022], this area remains significantly understudied. This is understandable given the nature of cloth and deformable body simulations, which involve numerous iterative tasks requiring frequent data synchronization. The isolation of computing units on the board and the limited bandwidth for data transfer between components often make the high synchronization costs outweigh the potential benefits of parallelization.

The growing adoption of Internet-of-Things (IoT) devices has popularized system-on-a-chip (SoC) designs, particularly in embedded and mobile systems. SoC processors, such as Apple's M series and AMD's Ryzen series, integrate multiple computing units on a single chip. This architecture significantly alleviates memory bandwidth constraints between units, presenting an opportunity to revisit the challenge of heterogeneous computation for cloth and deformable body simulations.

A key issue in achieving efficient heterogeneous computation for these simulations is task scheduling to ensure a balanced workload across different computing units. This problem is particularly challenging due to the varying capabilities and data transfer costs associated with chips of different architectures. To address this issue, we propose a novel framework for automatically scheduling iterative tasks, aiming to accelerate the performance of general iterative simulation methods. Our approach leverages an enhanced Heterogeneous Earliest Finish Time (HEFT) [Topcuoglu et al. 2002] algorithm, tailored to account for the unique communication costs and dependencies in heterogeneous computing systems. By modeling simulation workloads as Directed Acyclic Graphs (DAGs), the framework facilitates efficient task assignment, reducing idle times and synchronization delays. This enables seamless collaboration between different devices, maximizing computational throughput and achieving performance improvements beyond what either device can achieve independently. For tasks executed in a Jacobi manner, our method fully adapts to their inherently parallel nature. For Gauss-Seidel tasks, where sequential processing is required, we employ a graph-coloring method to partition tasks into color blocks, enabling intra-block parallelism. To overcome inter-block sequential execution, we present an novel asychronous Gauss-Seidel method adapted to our scheduling framework to optimize the parallel performance of Gauss-Seidel. Furthermore, our method extends to support general iterative tasks, including Jacobi, Gauss-Seidel, and hybrid Jacobi-Gauss-Seidel approaches. We demonstrate the effectiveness of our framework on representative iterative methods such as Jacobi-preconditioned gradient descent [Wang and Yang 2016], XPBD [Macklin et al. 2016], Vertex Block Descent (VBD) [Chen et al. 2024], and Second-order Stencil Descent (SOSD) [Lan et al. 2023], showcasing significant performance improvements. We summarize main contributions as follows:

- we are the first to apply the HEFT algorithm to schedule iterative simulation tasks on heterogeneous devices. To enhance scheduling performance, we introduce task merging and Gauss-Seidel $rank_u$ ordering.
- we propose a novel inter-interaction parallel asynchronous Gauss-Seidel method.
- we demonstrate the applicability of our method to various nonlinear iterative approaches for physics-based cloth and deformable body simulations.

2 Related Work

Physics-based simulation of cloth and deformable bodies is essential in computer graphics and animation. Early methods like [Baraff and Witkin 1998] introduced implicit integration for stable solution of large sparse linear systems. To improve efficiency, techniques such as multilevel methods [Chen et al. 2021; Wu et al. 2022], and multigrid methods [Tamstorf et al. 2015; Wang et al. 2018; Xian et al. 2019] have been developed. Quasi-Newton methods [Li et al. 2019; Liu et al. 2017] use approximate Hessian to avoid direct Hessian inversion, enabling real-time simulation of hyperelastic materials. Although Newton's method remains the most accurate for exact collision handling [Li et al. 2020a, 2021], its computational expense limits its use in real-time applications like gaming and virtual fashion. Iterative methods, such as PBD [Müller et al. 2007] and XPBD [Macklin et al. 2016], provide more efficient solutions by iteratively solving constraints in a Gauss-Seidel manner. Recent advances, such as hybrid iterative methods [Lan et al. 2023] and GPU-accelerated techniques [Fratarcangeli et al. 2016; Lan et al. 2022], further enhance scalability and performance. Other approaches include using the diagonal of the Hessian as a Jacobi preconditioner with momentum techniques [Wang 2015; Wang and Yang 2016] and vertex-based iterative methods in a pure Gauss-Seidel manner [Chen et al. 2024].

Traditional iterative methods like Jacobi and Gauss-Seidel rely on synchronous updates, where all processing units wait for the slowest, limiting performance in multi-processor systems. Asynchronous methods address this by allowing tasks to execute independently, using the latest available data without global synchronization, enhancing efficiency in heterogeneous systems. Chaotic relaxation, introduced by [Chazan and Miranker 1969], enabled fully asynchronous computations but relied on potentially stale data, with convergence conditions later generalized by [Baudet 1978] for nonlinear contracting operators. Subsequent work [Bahi et al. 2007; Bertsekas and Tsitsiklis 2015; Nishida and Kuang 2005; Zhongzhi and Deren 1997] has improved asynchronous methods, focusing on convergence and scalability for modern parallel architectures. In physics simulation, asynchronous techniques assign varying time-steps to regions or vertices based on velocity or material stiffness [Harmon

et al. 2009; Reinhardt et al. 2017; Thomaszewski et al. 2008; Zhao et al. 2016], reducing computational costs and improving parallelism.

With limited computational resources, researchers have developed simulation techniques leveraging multi-processor systems for tasks such as collision detection [Pabst et al. 2010], cloth simulation [Hutter et al. 2014; Li et al. 2020b; Liang and Lin 2018; Selle et al. 2008], fluid simulation [Chu et al. 2017; Liu et al. 2016], and material point methods [Wang et al. 2020]. These approaches rely on high resolutions to offset communication costs, though optimal task scheduling in such systems is NP-complete [Coffman and Bruno 1976; Ullman 1975]. This complexity intensifies in heterogeneous environments with varying computational efficiency and dynamic contact patterns. Modern HPC frameworks like OpenMP [Dagum and Menon 1998] and OpenCL [Stone et al. 2010] address these challenges through cross-platform abstractions, while domain(graphics)-specific-languages (DSLs) [Herholz et al. 2024; Hu et al. 2019; Li et al. 2024; Yu et al. 2022; Zheng et al. 2024, 2022] enable compiler-driven optimizations for enhanced performance and multi-backend deployment such as CUDA, Metal and CPU.

Task scheduling is a key factor for performance optimization in heterogeneous computing systems, divided into static scheduling and dynamic scheduling. Among static approaches, heuristic-based methods, such as Heterogeneous Earliest Finish Time (HEFT) [Topcuoglu et al. 2002], are widely studied. Variants like Predict Earliest Finish Time (PEFT) [Arabnejad and Barbosa 2013] and other methods, including Critical Path on a Processor (CPOP) [Topcuoglu et al. 2002] and High-Performance Task Scheduling (HPS) [Maurya and Tripathi 2018], address challenges such as communication overhead and workload balancing. Studies [Maurya and Tripathi 2018] show that HEFT and PEFT consistently outperform other algorithms in robustness and efficiency.

3 Background

3.1 Physics-based Simulation

When a soft body moves forward a time step h from the current frame t to the next frame $t+1$, the corresponding implicit Euler time integration of dynamic simulation of deformable objects can be formulated as an optimization problem with an incremental potential objective

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}} \Phi(\mathbf{x}) + \Psi(\mathbf{x}), \tag{1}$$

w.r.t. the position vector $\mathbf{x} \in \mathbb{R}^{3n}$ of n vertices, in which $\Phi(\mathbf{x}) = \frac{1}{2}h^{-2}\|\mathbf{M}^{\frac{1}{2}}(\mathbf{x}-\tilde{\mathbf{x}})\|^2$ is a term relative to kinetic energy, $\tilde{\mathbf{x}} = \mathbf{x}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{ext}$ is a predicted position under the effect of velocity $\mathbf{v} \in \mathbb{R}^{3n}$ and external force $\mathbf{f}_{ext} \in \mathbb{R}^{3n}$, $\mathbf{M} \in \mathbb{R}^{3n\times 3n}$ is a diagonal lumped mass matrix, and $\Psi(\mathbf{x})$ is the total potential energy of all kinds of constraints, including finite-element hyperelastic energy, discrete bending energy, contact energy, etc.

Optimizing Eq. 1 is challenging due to the high nonlinearity of $\Psi(\mathbf{x})$ w.r.t the variable $\mathbf{x}$. Newton's method offers rapid quadratic convergence but is computationally expensive for real-time interactive applications as it requires solving a large sparse linear system at each iteration. A more efficient alternative is performing a single Newton iteration without line-searching [Baraff and Witkin 1998; Wu et al. 2022], but this can cause instability with nonlinear constraints or complex collisions.

3.2 Iterative Solvers

Alternatively, nonlinear iterative methods, such as Jacobi and Gauss-Seidel iterations, can be employed. These methods avoid the overhead of solving linear systems and significantly reduce the computational cost associated with evaluating the Hessian. Additionally, in nonlinear iterative methods, tasks are lightweight, making them well-suited for efficient scheduling across different devices.

3.2.1 Nonlinear Jacobi. Instead of computing the full Hessian and solving a large sparse linear system, the Hessian can be approximated by its (block) diagonal, leading to the Jacobi-preconditioned gradient descent method [Wang and Yang 2016]. This approach is also applied in Projective Dynamics (PD) [Bouaziz et al. 2014], a nonlinear method that minimizes the incremental potential objective in Eq. 1 using a Local/Global strategy. The local step handles constraints via nonlinear projection, while the global step solves a linear system with a constant weighting matrix. On GPUs, the global step can be efficiently performed using a single Jacobi iteration [Wang 2015], with Chebyshev weights accelerating convergence.

3.2.2 Nonlinear Gauss-Seidel. Constraint-based methods like Position-based Dynamics (PBD) [Müller et al. 2007] and its extension XPBD [Macklin et al. 2016] iteratively process constraints using a sequential Gauss-Seidel approach. XPBD improves upon PBD by accurately handling material stiffness with Lagrangian multipliers, making it widely used for simulating simple garment behavior. Derived from compliant constraints [Tournier et al. 2015], XPBD represents total potential energy as the sum of quadratic constraint energies. In Vertex Block Descent (VBD), Chen et al. [2024] introduced a Gauss-Seidel method with block-diagonal preconditioning, assigning each vertex to a non-overlapping block for efficient computation.

3.2.3 Hybrid Jacobi-GS. Jacobi methods are fully parallelizable but have slow convergence, while Gauss-Seidel methods converge faster but are less parallel-friendly, even with graph coloring. To balance these, Second-Order Stencil Descent (SOSD) [Lan et al. 2023] combines both approaches. SOSD represents constraints as stencils, each involving a few vertices, and solves compact stencil-wise linear systems using PCG. It employs a hybrid Jacobi-Gauss-Seidel strategy to enhance convergence and parallelism.

3.3 HEFT-based Task Scheduling

3.3.1 DAG Representation and Task Sorting. The HEFT method represents tasks as nodes in a Directed Acyclic Graph (DAG), with edges capturing dependencies and weighted by communication costs. Each node includes computation times on different devices and maintains lists of predecessor and successor nodes for efficient dependency management. A topological sorting algorithm determines execution order by iteratively selecting nodes with zero in-degree, ensuring acyclic dependencies. Tasks are then executed sequentially, with computation times recorded in computation matrices and communication costs stored in communication matrices. In non-unified memory architectures, communication time depends on data size,

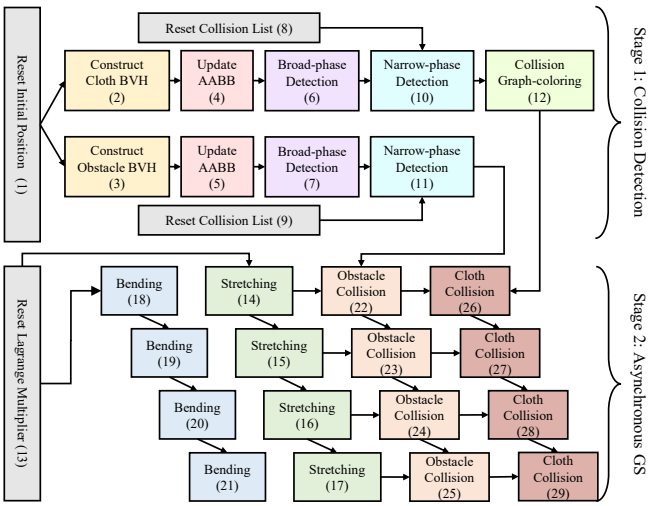


Fig. 2. The DAG of coarse-grained simulation tasks in our method includes two main stages, alongside some data preprocessing tasks: the collision detection stage, which involves BVH updates and collision detection, and the asynchronous Gauss-Seidel stage for iterative tasks. Using this DAG, we perform topological task sorting and $rank_u$ sorting. The sorted tasks are then assigned to different devices using a greedy strategy aimed at minimizing the total execution time.

while unified memory architectures assume a fixed delay (e.g., 0.2 ms) for CPU-GPU synchronization.

3.3.2 Rank Calculation. HEFT employs a breadth-first approach to calculate the rank of each node, starting from the terminal nodes and propagating backward through the DAG. The rule for computing the $rank_u$ of a task i is

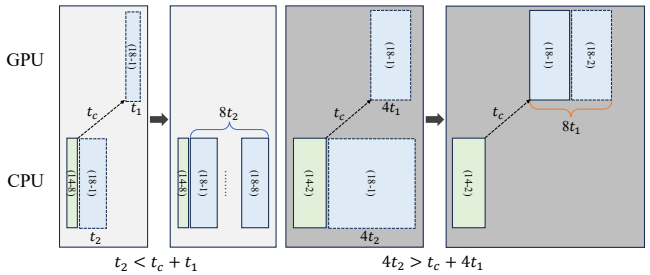
$$rank_u[i] = \text{avg(costs)} + \max_{j \in \text{successors}} (rank_u[i][j] + comm[i][j]),$$

in which avg(costs) is the average computation cost of the task across devices and $comm[i][j]$ is the communication cost between the task i and its successor task j. This process assigns lower rank values to nodes closer to the terminal nodes. Once task ranks are calculated, HEFT sorts tasks in descending order of rank and employs a greedy algorithm for scheduling.

3.3.3 Greedy Task Assignment. For each task in the sorted order, the algorithm calculates the earliest start time based on the latest finish times of its predecessors. It then identifies the optimal insertion point in each processor's queue and selects the slot with the shortest expected finish time. To reduce synchronization overhead, consecutive tasks on the same device are grouped with a shared start and end time, minimizing inter-device synchronization events and overall execution time.

4 Our Method

In this paper, we propose a unified scheduling method for simulation tasks within the framework of iterative methods for nonlinear cloth and deformable body simulations. To optimize efficiency, we introduce an asynchronous Gauss-Seidel approach that maximizes parallelism in iterative simulation tasks.



(a) Without merging (b) With merging

Fig. 3. Task merging in topology sorting. Green and blue boxes represent different blocks of stretching and bending iteration tasks respectively. (a) Without merging small bending tasks, 8 consecutive tasks are assigned to the CPU, resulting in a total execution time of $8t_2$, leaving the more efficient GPU idle. (b) With task merging, two large bending tasks are assigned to the GPU because $4t_2 > t_c + 4t_1$. The total execution time becomes $t_c + 8t_1 < 8t_2$, significantly improving the scheduling efficiency. The total execution time becomes $t_c + 8t_1 < 8t_2$, significantly improving the scheduling efficiency.

4.1 Simulation Tasks Scheduling

To apply HEFT for scheduling simulation tasks on CPUs and GPUs, a DAG is constructed to represent task dependencies. In nonlinear iterative physics-based simulations, the smallest units, such as calculating forces for a single triangle or handling collisions, are impractical to track due to their sheer number of nodes and performance measurement challenges. Instead, tasks are abstracted at a higher level, like "update AABB" or "calculate stretching deformation for all triangles." For example, an XPBD simulation involves over 30 tasks, including stretching constraints, bending constraints, collision detection, and collision handling. As shown in Fig. 2, a DAG of coarse-grained tasks is used in our method.

4.1.1 Task Merging in Topology Sorting. To ensure an execution order that respects task dependencies, topological sorting is performed, arranging tasks so that later tasks may depend on earlier ones but not vice versa. In nonlinear iterative simulations, tasks are often fine-grained; for example, computing stretching deformation is typically divided into 5-8 sequential blocks. Let t_1 and t_2 represent GPU and CPU computation times ($t_1 < t_2$), and t_c the communication cost of switching tasks between devices. As shown in Fig. 3(a), if the first bending block is assigned to the CPU and $t_2 < t_c + t_1$, all bending blocks are assigned to the CPU, leaving the GPU idle. Conversely, if the first block is assigned to the GPU and $t_1 < t_c + t_2$, all blocks are assigned to the GPU, leaving the CPU idle. Both cases waste resources. To address this, we propose merging consecutive small tasks into larger coarse-grained tasks. As illustrated in Fig. 3(b), merging every four bending tasks into one satisfies $4t_2 > t_c + 4t_1$, allowing the merged tasks to be assigned to the GPU. This reduces the total execution time to $t_c + 8t_1$, significantly improving scheduling efficiency compared to the $8t_2$ required in Fig. 3(a).

4.1.2 Gauss-Seidel $rank_u$. In our asynchronous Gauss-Seidel method, tasks within each iteration follow the Gauss-Seidel sequential order, but tasks across iterations may not, as shown in Fig.9, potentially

leading to suboptimal results as demonstrated in Fig. 4(a). To resolve this, we propose synchronous Gauss-Seidel $rank_u$, assigning smaller $rank_u$ values to tasks from later iterations, ensuring proper sequential order. As demonstrated in Fig.9, this rectified $rank_u$ sorting improves convergence.

4.2 Asynchronous Gauss-Seidel

In multi-device setups, such as a CPU and GPU in Apple's M series with unified memory, the traditional Gauss-Seidel (GS) method is not directly applicable. GS typically uses graph coloring to enable parallelism within color blocks while ensuring sequential execution across blocks. To leverage multiple devices, tasks are distributed across them to balance completion times. Each device executes tasks in a Gauss-Seidel manner, while tasks across devices run independently in a Jacobi manner without inter-device communication. After all tasks are completed, results are synchronized through weighted averaging.

The lack of inter-device communication reduces the convergence rate of the Gauss-Seidel method. To address this, we propose an asynchronous Gauss-Seidel approach with data synchronization between devices, enhancing convergence while utilizing multiple devices effectively. As illustrated in Fig. 5, when a GPU task T_c completes and task T_e is about to execute, T_e retrieves the latest data from T_c and the most recently completed CPU task T_b. The strategy for combining data from T_c and T_b is crucial for ensuring optimal convergence. Designing an asynchronous Gauss-Seidel method involves two key considerations: the asynchronization strategy and the data communication mechanism. In this paper, we explore both aspects in detail within our proposed method.

4.2.1 Inter-Iteration Asynchronization. To improve Gauss-Seidel parallelism, two asynchronization strategies are considered: intra-iteration, where tasks within the same iteration overlap, and inter-iteration, where tasks across iterations overlap. These strategies can be combined in various ways. Our method uses inter-iteration asynchronization combined with intra-iteration synchronization. As shown in the DAG in Fig. 2, there are four Gauss-Seidel iterations. Within each iteration, stretching and bending tasks can execute concurrently, while obstacle and cloth collision tasks must follow sequentially. Tasks from different iterations can overlap, including the overlap of stretching and bending tasks on collision tasks, but the execution order of the same task type across iterations must be maintained to prevent mixing. Generally, n-iteration concurrency ($n \geq 2$) can be used, with $n = 1$ corresponding to traditional Gauss-Seidel. We select $n = 2$ as it provides sufficient tasks for HEFT to maximize parallelism while maintaining convergence in our experiments. We evaluate the convergence performance of various strategies, as shown in Fig. 4(b). Our adopted strategy, inter-iteration and intra-iteration asynchronization with collision constraints at the end (Asyn-Serial Collision), outperforms all other approaches.

4.2.2 Data Synchronization. The core concept of Gauss-Seidel iteration is to use the most recently updated information for the next computation, which is ensured in sequential execution but violated in asynchronous Gauss-Seidel (GS). For instance, in Fig. 5, GPU task T_e receives inputs from GPU task T_c and CPU task T_b.

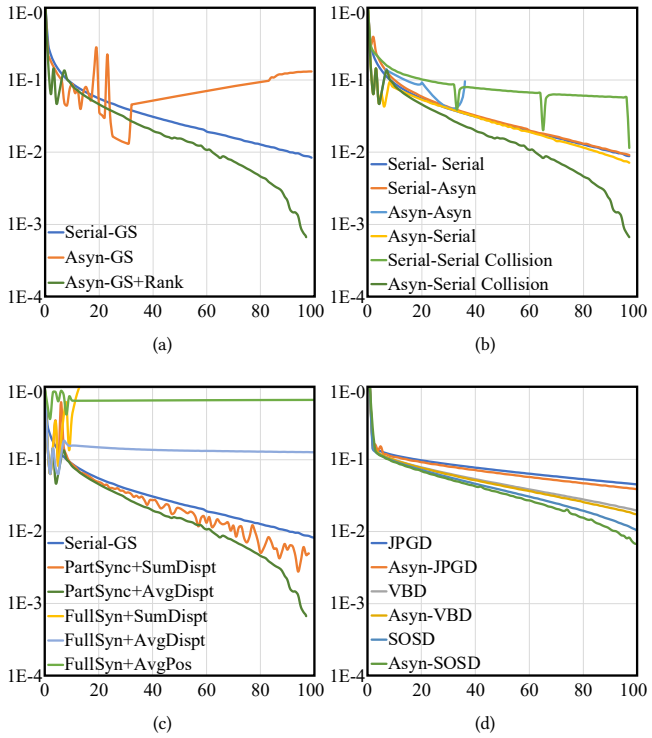


Fig. 4. The convergence of the relative energy error of key strategies in our method is demonstrated in the Dress example over 100 iterations. The relative energy error is calculated as $y^k = (E^k - E^*)/(E^0 - E^*)$, where E is the objective energy function being optimized. (a) Asyn-GS+Rank: Asynchronous Gauss-Seidel with specific $rank_u$ ordering outperforms Serial-GS and Asyn-GS without $rank_u$, the latter failing to converge. (b) Asyn-Serial Collision: Inter-iteration and intra-iteration asynchronization with collision constraints at the end surpasses other strategies. (c) PartSync+AvgDisp: Partial CPU strategy combined with average displacement for data synchronization achieves the best overall performance. (d) Our method applied to various iterative simulation methods consistently outperforms their traditional counterparts.

Combining these inputs follows a Jacobi-like approach, assuming a common starting point. In physics-based simulations like XPBD, this data often represents positions or displacements, which impact convergence. When using positions, the starting point is implicitly ensured. For example, combining positions x_a and x_c results in $x = (x_a + x_c)/2$, equivalent to averaging displacements with a shared starting point x_s: $x = x_s + (\Delta x_a + \Delta x_c)/2$. However, this simple averaging can hinder convergence. Using displacements instead requires an explicit starting point since Δx_a and Δx_c cannot be meaningfully combined otherwise. To address this, our method precomputes a data synchronization map using the task scheduling map, ensuring explicit starting points for synchronization. The data synchronization map is illustrated by the data communication arrows in the scheduling map in Fig. 10. This enables more advanced strategies beyond simple averaging, improving the convergence of asynchronous GS, as demonstrated in Fig. 4(c).

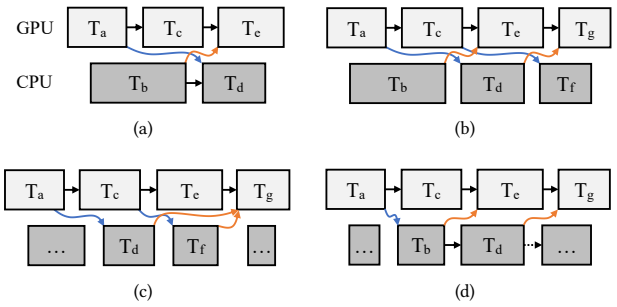


Fig. 5. Asynchronous communication strategies. (a) A CPU task T_d fetches data from its previous CPU task T_b and the most recently completed GPU task T_a, while a GPU task T_e does the reverse. (b) To improve convergence, a CPU task T_d drops the data from its previous CPU task T_b if it receives the data from a GPU task T_a. This dropping operation is not applied on GPU tasks. (c) To maintain data consistence after dropping operations, a GPU task T_g may receive data from multiple CPU tasks, such as T_a and T_f. (d) The data of a CPU task T_b can affects GPU tasks multiple times due to dropping operations on CPU tasks. This localized influence benefits for convergence.

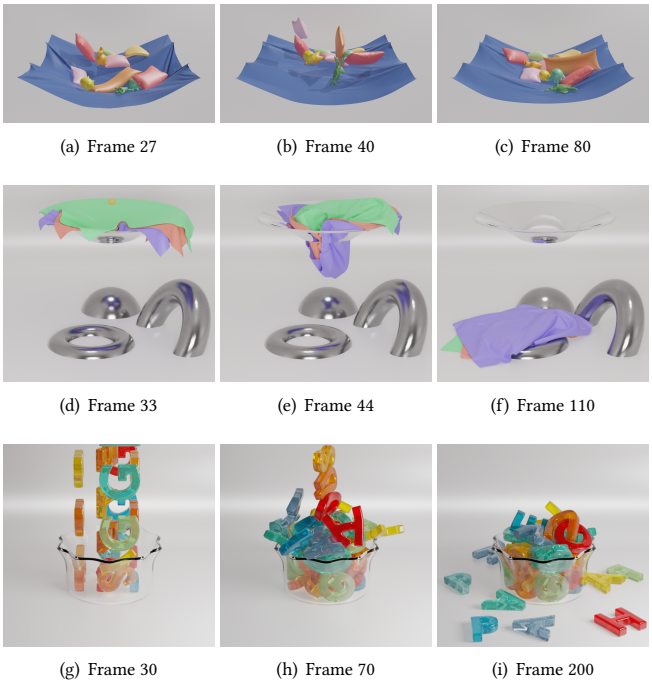


Fig. 6. Deformable pillows (a-c) are dropped onto a hanging cloth, stretching it before bouncing back and settling by the 80th frame. (d-f) A deformable sphere pushes three cloth layers through a funnel, with all layers resting on obstacles by the 110th frame. (g-i) Forty-five deformable letters drop into a glass bowl, stacking or settling by the 200th frame. Collisions and deformations are robustly handled using our asynchronous Gauss-Seidel method within the XPBD framework.

When a CPU task completes, there is a fixed waiting time t_c for the GPU to read its output, and similarly, t_g for the CPU to access GPU output, in unified memory architectures. For a GPU task T_e to access CPU task T_b's data, explicit synchronization is needed. Dynamically identifying which CPU task to wait for is inefficient, so we use a scheduling task map to predefine the data synchronization map. This ensures tasks know in advance which task to wait for, incorporating t_c and t_g without additional overhead, improving the efficiency of our asynchronous Gauss-Seidel method. As shown in Fig. 7, using positions with simple averaging slows convergence, while our method, employing displacements and explicit starting points, achieves faster convergence.

As shown in Fig. 10, frequent data communication is observed between iterative asynchronous Gauss-Seidel tasks in our method. When a CPU task T_e executes, it uses data from the last CPU task T_d if no GPU data is available; otherwise, it uses data from GPU task T_a, discarding T_d's data. This ensures each CPU task receives data from only one GPU task, avoiding interference from outdated information. Since T_a's data already includes information from earlier GPU tasks like T_z, retaining T_z's data is redundant and may disrupt convergence. This dropping strategy maintains a consistent data stream on the GPU, the primary device in our method, with the CPU serving as support. Actually, frequent data communication with the dropping strategy is the reason for our method can achieve better convergence performance than conventional XPBD using synchronous Gauss-Seidel iterations.

5 Applications

We apply our method to various iterative methods for nonlinear cloth and deformable body simulation. As illustrated in Fig. 4(d), our method demonstrates superior convergence compared to conventional approaches.

XPBD: constraint-wise Gauss-Seidel. Conventional XPBD [Macklin et al. 2016] updates each constraint iteratively using a sequential Gauss-Seidel approach. The task scheduling map for asynchronous XPBD, optimized with our method, is shown in Fig. 10.

VBD: vertex-wise Gauss-Seidel. The conventional Vertex Block Descent (VBD) method [Chen et al. 2024] updates each vertex iteratively by solving small 3×3 block linear systems in a sequential Gauss-Seidel manner. The task scheduling map for asynchronous VBD, optimized using our method, is shown in Fig. 11.

SOSD: constraint-wise Hybrid Jacobi-GS. Conventional Second-order Stencil Descent (SOSD) [Lan et al. 2023] is a hybrid Jacobi-Gauss-Seidel (GS) method that iteratively updates positions using stencils. The stencils are grouped into n blocks, with stencils within each block being independent and executable in parallel. However, blocks must be processed sequentially in a Gauss-Seidel manner. To optimize GPU performance, the main GS-block is executed while the other blocks run in a Jacobi manner. Essentially, each computational unit in SOSD performs a Jacobi iteration, but the key difference lies in the data synchronization: the displacement from the main GS-block takes precedence over that from the Jacobi blocks.

Gradient Descent: vertex-wise Jacobi. The conventional block-diagonal preconditioned gradient descent method (JBD) [Wang

Table 1. The statistics of our examples and their performances on an Apple's M3 Max Chip. We run XPBD simulations with a large frame time step $\Delta t = 1/60s$ for all examples, with each frame containing 32 or 64 substeps, and each substep using 8 or 12 nonlinear Gauss-Seidel iterations. Compared to a GPU-only implementation, our hybrid CPU-GPU implementation achieves over 30% speedup in execution time and 40% in FPS. When compared to CPU-only implementation, the speedups are even more significant, reaching 80% in execution time and 400% in FPS. All timing are provided in milliseconds.

Example	Mesh Setting			Frame Setting		#Coll.	Avg. Frame Timing			Time Speedup		FPS Speedup	
	#Vert.	#Tri.	#Tet.	#Sub.	#Iter.		CPU	GPU	Hybrid	CPU	GPU	CPU	GPU
Dress	104 K	207 K	0	32	8	96.9 K	815.9	268.6	174.5	79%	35%	368%	54%
Jersey	69 K	138 K	0	32	8	78.6 K	506.1	173.1	122.2	76%	29%	314%	42%
Pillows	176 K	90 K	493 K	64	8	17.7 K	2.18K	813.5	463.8	79%	43%	369%	75%
Funnel	132 K	260 K	2.7 K	64	12	225.7 K	2.21K	935.6	528.5	76%	44%	319%	77%
Letters	190 K	0	635 K	32	8	7.4 K	1.49K	368.9	266.0	82%	27%	459%	38%

Table 2. The performance of various simulation methods, evaluated on an Apple M2 Max chip using the Dress example, is presented. All timings are reported in milliseconds.

Methods	Avg. Frame Timing			Time Speedup		FPS Speedup	
	CPU	GPU	Hybrid	CPU	GPU	CPU	GPU
VBD	3.07K	678.3	555.1	82%	18%	454%	22%
SOSD	4.95K	1.33K	1.03K	79%	22%	380%	28%
JPGD	3.00K	941.3	614.1	80%	35%	389%	53%

Table 3. The performance of our examples, evaluated on an Apple M2 Max chip using XPBD simulation, is presented. All timings are reported in milliseconds.

Example	Avg. Frame Timing			Time Speedup		FPS Speedup	
	CPU	GPU	Hybrid	CPU	GPU	CPU	GPU
Dress	1.22 K	309.4	240.4	80%	22%	407%	29%
Jersey	706.1	205.2	131.8	87%	36%	648%	56%
Pillows	4.65 K	1.14 K	816.3	82%	28%	470%	39%
Funnel	2.89 K	1.07 K	573.1	80%	47%	406%	87%
Letters	3.31 K	424.9	322.2	90%	24%	928%	32%

and Yang 2016] iteratively updates the nonlinear configuration of physics-based simulations in a Jacobi manner, allowing for full parallelism. However, explicit data synchronization between iterations must be ensured. When using our method to schedule JBD, we optimize task assignment and synchronization, ensuring efficient parallel execution while managing dependencies across iterations.

6 Results

We evaluated the effectiveness of our method across various complex simulation scenarios. As illustrated in Fig. 1, when applied to XPBD, our method efficiently and stably simulates scenarios such as a multilayer dress with complex collisions and a basketball jersey undergoing dramatic motion. In Fig. 6 (a-c), deformable pillows are dropped onto a hanging cloth. The pillows initially stretch the cloth as they push against it, then bounce back into the air, and eventually come to rest statically on the cloth. In Fig. 6 (d-f), a deformable sphere pushes three layers of cloth through a funnel, with all layers ultimately coming to rest on ground obstacles. In Fig. 6 (g-i), forty-five deformable letters are dropped into a glass bowl. During the drop, the letters are heavily compressed, and by the end, they either stack together inside the bowl or rest statically on the ground. Across all these simulations, complex collisions between multilayered cloth and deformable bodies are robustly handled using our asynchronous Gauss-Seidel method within the XPBD framework. Our experiments are conducted on an Apple Mac Studio equipped with an M2 Max chip, featuring a 3.5 GHz 12-core CPU, a 1.4 GHz 30-core GPU, and 32 GB of unified memory and an Apple MacBook Pro equipped with an M3 Max chip, featuring a 4.05 GHz 16-core CPU, a 2.75 GHz 40-core GPU, and 128 GB of unified memory. Our method is implemented in standard C++ for the CPU and Metal for the GPU.

6.1 Performance Analysis

The parameter settings and detailed performance data for each example are summarized in Table 1. Our method employs an asynchronous Gauss-Seidel approach to iterate XPBD. To ensure fair comparisons, baseline CPU/GPU implementations use synchronous execution, as asynchronous algorithms introduce additional overhead from data communications. All examples use a timestep of 1/60 seconds. For the Dress, Jersey and Letters examples, we use 32 substeps per frame, while the Pillows and Funnel examples use 64 substeps per frame. Each substep involves 8 Gauss-Seidel iterations for all examples, except the Funnel, which uses 12 iterations per substep. Compared to a GPU-only implementation, our hybrid CPU-GPU implementation achieves over 30% speedup in execution time and 40% in FPS. When compared to CPU-only implementation, the speedups are even more significant, reaching 90% in execution time and 300% in FPS.

As shown in Fig. 8, the Dress example from the XPBD simulation demonstrates that our method's runtime closely matches theoretical scheduling predictions and significantly outperforms GPU-only performance. During initialization, each task is executed multiple times on both the CPU and GPU to record its average execution cost. To adapt to dynamic contact, we update task execution times every frame and re-schedule tasks using a computation matrix refined through runtime smoothing. Our scheduler operates efficiently, taking under 0.1 ms per frame. Differences in performance, as seen in Fig. 8, are partly due to data copying and weighting during asynchronous iterations.

Our method achieves higher performance than the theoretical speedup limit, as different tasks benefit from varying degrees of speedup through optimized scheduling. For instance, stretching deformation tasks see a 70% speedup, while bending tasks achieve

around 30%. Additionally, the separation and scheduling of collision and deformation tasks provide a significant boost to overall speedup.

6.2 Conclusion

In this paper, we present a novel framework for efficient heterogeneous computation in cloth and deformable body simulations, targeting systems with multiple processing units (CPUs, GPUs, and accelerators). Our approach uses an enhanced Heterogeneous Earliest Finish Time (HEFT) algorithm to optimize task scheduling, minimizing idle times and synchronization delays. We address the challenges of parallelizing iterative tasks, including Jacobi, Gauss-Seidel, and hybrid methods, with improvements such as asynchronous Gauss-Seidel for better parallelism. Through experiments on iterative methods like XPBD, VBD, and SOSD, we demonstrate significant performance gains, surpassing traditional single-unit simulations. Our framework enables high-performance simulation in modern hybrid CPU-GPU systems, offering a robust solution for real-time and resource-constrained environments.

7 Limitations and Future Work

While our method significantly improves task scheduling for CPU-GPU cloth and deformable body simulations, it has several limitations. These include inherent communication overhead between devices, scalability challenges with multiple accelerators, and reduced performance for highly nonlinear simulations or complex constraints. While our architecture-agnostic scheduler works across platforms, implementation validation remains limited to specific GPU configurations. Providing separate implementations for different platforms also leads to more manual work. Future directions include multi-device optimization and adaptive scheduling, nonlinear simulation enhancements, real-time parallel preconditioning, cross-platform support and experiment through DSL-implementation like LuisaCompute [Zheng et al. 2022] and extended benchmarking across simulation and general computing scenarios.

Acknowledgement

This work is supported by National Natural Science Foundation of China (62472364, 62072383), the Public Technology Service Platform Project of Xiamen City (No.3502Z20231043), Xiaomi Young Talents Program/Xiaomi Foundation and the Fundamental Research Funds for the Central Universities (20720240058). The work is partially supported by Fujian Provincial Science and Technology Major Project (No. 2024HZ022003), Jiangxi Provincial Natural Science Foundation Key Project (No. 20244BAB28039).

References

Hamid Arabnejad and Jorge G Barbosa. 2013. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (2013), 682–694.

Jacques Mohcine Bahi, Sylvain Contassot-Vivier, and Raphaël Couturier. 2007. *Parallel Iterative Algorithms: from Sequential to Grid Computing*. Chapman and Hall/CRC.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 43–54.

Gerard M Baudet. 1978. Asynchronous Iterative Methods for Multiprocessors. *Journal of the ACM (JACM)* 25, 2 (1978), 226–244.

Dimitri Bertsekas and John Tsitsiklis. 2015. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, Article 154 (July 2014), 11 pages.

Daniel Chazan and Willard Miranker. 1969. Chaotic Relaxation. *Linear algebra and its applications* 2, 2 (1969), 199–222.

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. 2024. Vertex Block Descent. *ACM Trans. Graph.* 43, 4, Article 116 (jul 2024), 16 pages.

Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. 2021. Multiscale Cholesky Preconditioning for Ill-conditioned Problems. *ACM Trans. Graph.* 40, 4, Article 81 (July 2021), 13 pages.

Jieyu Chu, Nafees Bin Zafar, and Xubo Yang. 2017. A Schur Complement Preconditioner for Scalable Parallel Fluid Simulation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.

Edward G. Coffman and John Bruno. 1976. Computer and Job-shop Scheduling Theory.

Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* 5, 1 (1998), 46–55.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: A Practical Gauss-Seidel Method for Stable Soft Body Dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)* 35, 6, Article 214 (Nov. 2016), 9 pages.

David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. In *ACM SIGGRAPH 2009 papers*. 1–12.

Philipp Herholz, Tuur Stuyck, and Ladislav Kavan. 2024. A Mesh-based Simulation Framework using Automatic Code Generation. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–17.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

Marco Hutter, Martin Knuth, and Arjan Kuijper. 2014. Mesh Partitioning for Parallel Garment Simulation. (2014).

Lei Lan, Minchen Li, Chenfanfu Jiang, Huamin Wang, and Yin Yang. 2023. Second-Order Stencil Descent for Interior-Point Hyperelasticity. *ACM Trans. Graph.* 42, 4, Article 108 (jul 2023), 16 pages.

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-Free Projective Dynamics on the GPU. *ACM Trans. Graph.* 41, 4, Article 69 (jul 2022), 16 pages.

Cheng Li, Min Tang, Ruofeng Tong, Ming Cai, Jieyi Zhao, and Dinesh Manocha. 2020b. P-cloth: Interactive Complex Cloth Simulation on Multi-GPU Systems using Dynamic Matrix Assembly and Pipelined Implicit Integrators. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020a. Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (jul 2020), 20 pages.

Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed Optimization Time Integrator for Large-Step Elastodynamics. *ACM Trans. Graph.* 38, 4, Article 70 (July 2019), 10 pages.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph.* 40, 4, Article 170 (jul 2021), 24 pages.

Yong Li, Shoaib Kamil, Keenan Crane, Alec Jacobson, and Yotam Gingold. 2024. I MESH: A DSL for Mesh Processing. *ACM Transactions on Graphics* 43, 5 (2024), 1–17.

Junbang Liang and Ming C Lin. 2018. Time-Domain Parallelization for Accelerating Cloth Simulation. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 21–34.

Haixiang Liu, Nathan Mitchell, Mridul Aanjaneya, and Eftychios Sifakis. 2016. A Scalable Schur-complement Fluids Solver for Heterogeneous Compute Platforms. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 4, Article 116a (May 2017), 16 pages.

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: Position-Based Simulation of Compliant Constrained Dynamics. In *Proceedings of the 9th International Conference on Motion in Games* (Burlingame, California) *(MIG '16)*. Association for Computing Machinery, New York, NY, USA, 49–54.

Ashish Kumar Maurya and Anil Kumar Tripathi. 2018. On Benchmarking Task Scheduling Algorithms for Heterogeneous Computing Systems. *The Journal of Supercomputing* 74, 7 (2018), 3039–3070.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position Based Dynamics. *J. Vis. Comun. Image Represent.* 18, 2 (April 2007), 109–118.

Hiroshi Nishida and Hairong Kuang. 2005. Experiments on Asynchronous Partial Gauss-Seidel Method. In *Advanced Parallel Processing Technologies: 6th International Workshop, APPT 2005, Hong Kong, China, October 27-28, 2005. Proceedings 6*. Springer, 111–120.

Simon Pabst, Artur Koch, and Wolfgang Straßer. 2010. Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1605–1612.

Stefan Reinhardt, Markus Huber, Bernhard Eberhardt, and Daniel Weiskopf. 2017. Fully asynchronous SPH simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–10.

Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. 2008. Robust High-resolution Cloth using Parallelism, History-based Collisions, and Accurate Friction. *IEEE transactions on visualization and computer graphics* 15, 2 (2008), 339–350.

John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010), 66.

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 34, 6, Article 245 (Oct. 2015), 13 pages.

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous cloth simulation. In *Computer Graphics International*, Vol. 2. Citeseer, 2.

Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274.

Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. 2015. Stable Constrained Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 34, 4, Article 132 (July 2015), 10 pages.

J.D. Ullman. 1975. NP-complete Scheduling Problems. *J. Comput. System Sci.* 10, 3 (1975), 384–393.

Huamin Wang. 2015. A Chebyshev Semi-iterative Approach for Accelerating Projective and Position-Based Dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)* 34, 6, Article 246 (Oct. 2015), 9 pages.

Huamin Wang and Yin Yang. 2016. Descent Methods for Elastic Body Simulation on the GPU. *ACM Trans. Graph. (SIGGRAPH Asia)* 35, 6, Article 212 (Nov. 2016), 10 pages.

Xinlei Wang, Minchen Li, Yu Fang, Xinxin Zhang, Ming Gao, Min Tang, Danny M Kaufman, and Chenfanfu Jiang. 2020. Hierarchical Optimization Time Integration for CFL-rate MPM Stepping. *ACM Transactions on Graphics (TOG)* 39, 3 (2020), 1–16.

Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel Multigrid for Nonlinear Cloth Simulation. *Computer Graphics Forum (Pacific Graphics)* 37, 7 (2018), 131–141.

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-Based Multilevel Additive Schwarz Preconditioner for Cloth and Deformable Body Simulation. *ACM Trans. Graph.* 41, 4, Article 63 (jul 2022), 14 pages.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Trans. Graph.* 38, 6, Article 162 (nov 2019), 13 pages.

Chang Yu, Yi Xu, Ye Kuang, Yuanming Hu, and Tiantian Liu. 2022. MeshTaichi: A compiler for efficient mesh-based operations. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–17.

Danyong Zhao, Yijing Li, and Jernej Barbic. 2016. Asynchronous implicit backward Euler integration.. In *Symposium on Computer Animation*. 1–9.

Shaokun Zheng, Xin Chen, Zhong Shi, Ling-Qi Yan, and Kun Xu. 2024. GPU Coroutines for Flexible Splitting and Scheduling of Rendering Tasks. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–24.

Shaokun Zheng, Zhiqian Zhou, Xin Chen, Difei Yan, Chuyan Zhang, Yuefeng Geng, Yan Gu, and Kun Xu. 2022. Luisarender: A high-performance rendering framework with layered and unified interfaces on stream architectures. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–19.

Bai Zhongzhi and Wang Deren. 1997. Asynchronous parallel multisplitting nonlinear Gauss-Seidel iteration. *Applied Mathematics-A Journal of Chinese Universities* 2, 12 (1997), 179–194.

(a) (b)

Fig. 7. The data synchronization strategy greatly impacts simulation accuracy. (a) Averaging positions can result in penetrations, while (b) averaging displacements ensures correct collision handling.

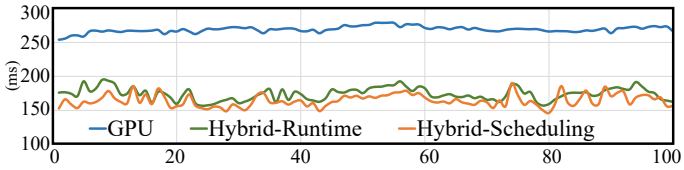
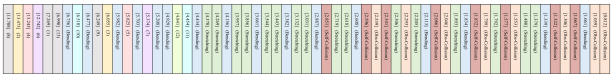
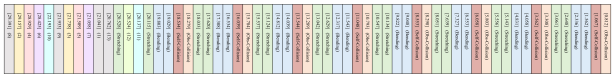


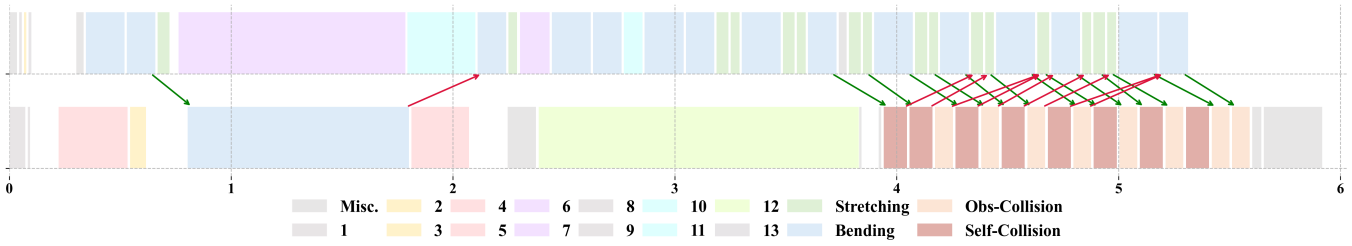
Fig. 8. Illustration of the execution time for 100 frames in the Dress example from the XPBD simulation. The runtime performance of our method closely matches the theoretical scheduling performance predicted before simulation and significantly outperforms the GPU-only performance.



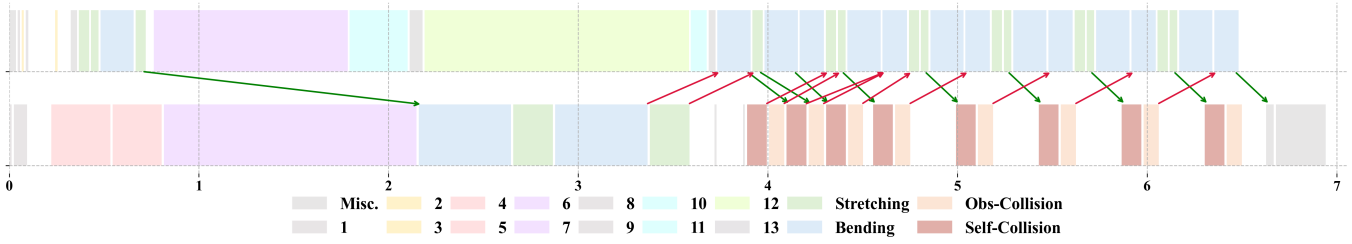
(a) Sorting without Gauss-Seidel $rank_u$



(b) Sorting with Gauss-Seidel $rank_u$



(c) Scheduling without Gauss-Seidel $rank_u$



(d) Scheduling with Gauss-Seidel $rank_u$

Fig. 9. (a) Without Gauss-Seidel $rank_u$ in task sorting, collision tasks are delayed, as reflected in the corresponding scheduling map (c). (b) With Gauss-Seidel $rank_u$, deformation and collision tasks are correctly interleaved. As a result, in the scheduling map (d), deformation and collision tasks communicate earlier, improving convergence, as shown in Fig. 4(a).

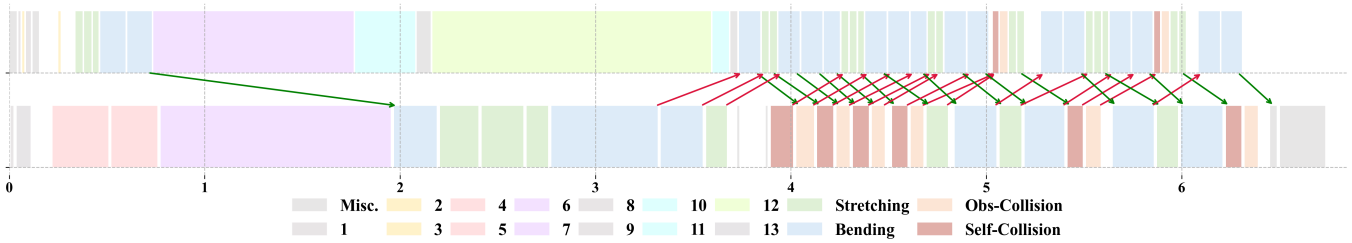


Fig. 10. The scheduling map of XPBD in the Dress example shows data flow between tasks. Green arrows represent a CPU task fetching data from a GPU task, while red arrows indicate a GPU task fetching data from a CPU task. Frequent data communication is observed between iterative asynchronous Gauss-Seidel tasks in our method, which is the reason for our method can achieve better convergence performance than conventional XPBD using synchronous Gauss-Seidel iterations.

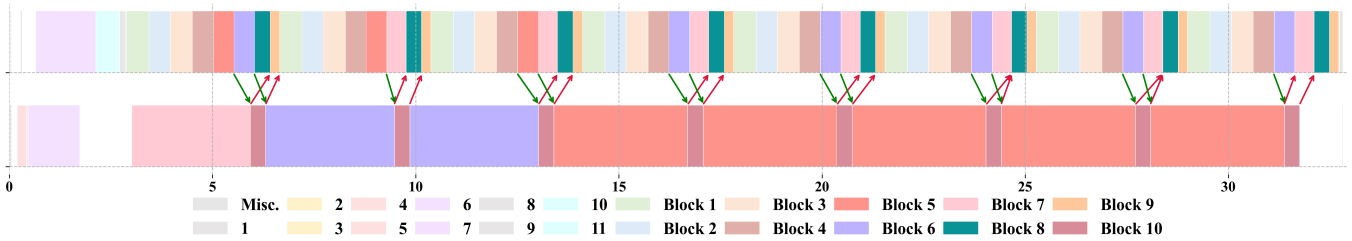


Fig. 11. The scheduling map of Vertex Block Descent (VBD) [Chen et al. 2024] in the Dress example shows data flow between tasks. Green arrows represent a CPU task fetching data from a GPU task, while red arrows indicate a GPU task fetching data from a CPU task. Data communication is not as frequent as XPBD because many CPU tasks have long execution time.

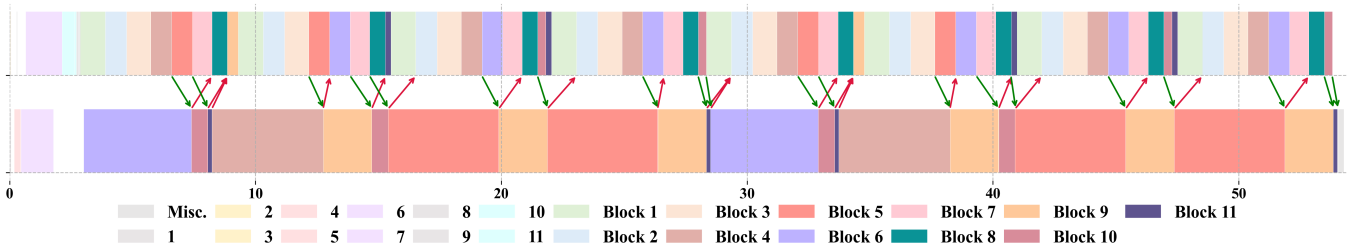


Fig. 12. The scheduling map of Second-order Stencil Descent (SOSD) [Lan et al. 2023] in the Dress example shows data flow between tasks. Green arrows represent a CPU task fetching data from a GPU task, while red arrows indicate a GPU task fetching data from a CPU task. The data communication frequency is similar to that of VBD.

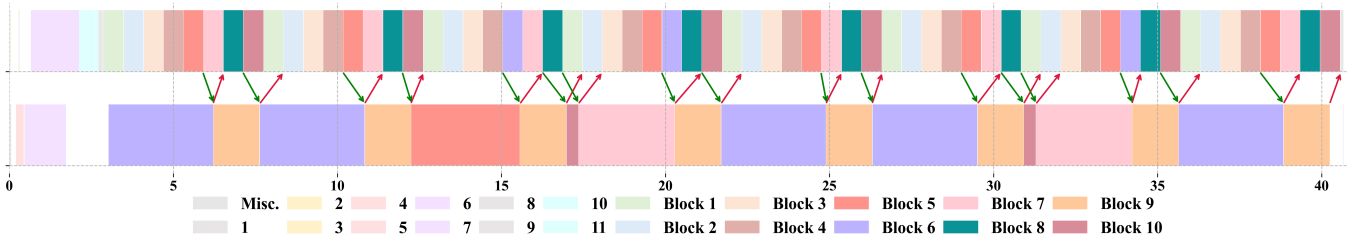


Fig. 13. The scheduling map of Jacobi-preconditioned Gradient Descent (JPGD) [Wang and Yang 2016] in the Dress example shows data flow between tasks. Green arrows represent a CPU task fetching data from a GPU task, while red arrows indicate a GPU task fetching data from a CPU task. The data communication frequency is similar to that of VBD.