P-Cloth: Interactive Complex Cloth Simulation on Multi-GPU Systems using Dynamic Matrix Assembly and Pipelined Implicit Integrators

Supplementary Material

1 ALGORITHMS FOR TIME INTEGRATION

Some algorithms used for parallel time integration is described in the following pseudo-code.

Algorithm 1 Work Queue Generation Algorithm used for Pipelined SpMV

1: \textbf{sw}: index of the switch
2: \textbf{offset}: offset for sub-vector index
3: \textbf{lvl}: level in the binary tree, 0 for leaf nodes
4: \textbf{GetGPURange}: 
5: Returns the indices of the GPU that the given switch interconnects. For a leaf switch, returns exact 2 GPU indices.
6: \textbf{GetChildSwitches}: 
7: Return the indices of 2 child switches of a given switch.
8: 
9: // Call GenerateWorkQueues(0, 0, log2n) for the overall
10: // work queues.
11: \textbf{procedure} GenerateWorkQueues(sw, offset, lvl)
12: // step (1):
13: // Recursively generate optimal work queues for
14: // its children.

15: \textbf{if} lvl != 0 \textbf{then}
16: sw0, sw1 \leftarrow GetChildSwitches(sw)
17: GenerateWorkQueues(sw0, offset, lvl - 1)
18: GenerateWorkQueues(sw1, offset, lvl - 1)
19: \textbf{else}
20: // Each leaf switch interconnects exact 2 GPUs.
21: G0, G1 \leftarrow GetGPURange(sw)
22: node0 \leftarrow (G1, G0 + offset)
23: node1 \leftarrow (G0, G1 + offset)
24: Push node0 to Q(G0)
25: Push node1 to Q(G1)
26: \textbf{end if}
27: // step (2):
28: // Transfer minimized amount of data between
29: // its children.
30: range0 \leftarrow GetGPURange(sw0)
31: range1 \leftarrow GetGPURange(sw1)
32: for G0, G1 \in range0, range1 \textbf{do}
33: node0 \leftarrow (G1, G0 + offset)
34: node1 \leftarrow (G0, G1 + offset)
35: Push node0 to Q(G0)
36: Push node1 to Q(G1)
37: \textbf{end for}
38: // step (3):
39: // Delegate rest of the work queue generation
40: // tasks to its children.
41: offset0 \leftarrow offset + 2^{lvl}
42: offset1 \leftarrow offset + 2^{lvl}
43: // An offset is applied so that the work queue is
44: // generated for the delegated sub-vectors.
45: GenerateWorkQueues(sw0, offset0, lvl - 1)
46: GenerateWorkQueues(sw1, offset1, lvl - 1)
47: \textbf{end procedure}

Algorithm 2 Sparse Matrix Filling Algorithm

1: // Index Table Allocating:
2: \textbf{for each } GPU \textbf{do in parallel}
3: IndexTable \leftarrow AllocateIndexTable(i)

Permissions: Authors’ addresses: Cheng Li, Zhejiang University, licharmy@yahoo.com; Min Tang, Zhejiang University, tang_mz@zju.edu.cn; Ruofeng Tong, Zhejiang University, trf@zju.edu.cn; Ming Cai, Zhejiang University, corresponding author, cm@zju.edu.cn; Jieyi Zhao, University of Texas Health Science Center at Houston, jieyi.zhao@uth.tmc.edu; Dinesh Manocha, University of Maryland at College Park https://min-tang.github.io/home/PCloth/
2 STITCHING ALGORITHM

The stitching algorithm is described in the following pseudo-code.

Algorithm 3 Stitching Algorithm

1: **Input**: Stitching node pairs $NP$. 
2: **Output**: Stitched and refined cloth mesh .
3: 
4: // Stitching cloth pieces with linking constraints
5: // for all the $NP$. 
6: StitchingCoarsePieces($NP$)
7: 
8: // Merge the pieces together by merging
9: // all the node pairs $NP$
10: MergePieces($NP$)
11: 
12: // Refine the merged cloth mesh to higher resolution
13: // by subdividing.
14: RefinePieces()

3 ADDITIONAL BENCHMARKS

We use some complex cloth simulation benchmarks for regular/irregular-shaped cloth simulation (Fig. 1 and Fig. 2).
Fig. 2. Multi-layer Garment Benchmarks: We used these benchmarks: (a) Miku with 1.33M triangles, (b) Zoey with 569K triangles, (c) Andy with 538K triangles, and (d) Kimono with 1M triangles, for multi-layer garment simulation.