

Reduction of CPU-GPU Synchronization Overhead for Accelerating Implicit Clothing Simulator

Sangbin Lee¹, Donghan Ryu² and Hyeong-Seok Ko¹

¹ Seoul National University, Korea

² Nvidia Corporation

Abstract

When trying to make the conjugate gradient (CG) method exploit GPU technology, this paper notes that the communication between CPU and GPU to transfer the residual value and waiting for the CPU's decision whether to continue further iterations is a new source of delay that has been overlooked and turns out not negligible. By examining the residual decrease pattern in log scale, this paper proposes so-called the Secant Lazy Residual Evaluation (Secant LRE) method to skip needless synchronization. We experimented the method for a clothing simulator and found that the proposed method reduces the sync overhead significantly, leading to 10~60% performance gain.

CCS Concepts

•Computing methodologies → Massively parallel and high-performance simulations; Parallel algorithms; Physical simulation; Graphics processors;

1. Introduction

This paper is related to making a marginal speed up when executing the conjugate gradient (CG) method on GPU. To show how the proposed method applies to a real application, this paper will showcase a clothing simulator, the framework of which is established in Baraff and Witkin [BW98]. Implicit time integration of the governing equations describing the cloth movement is reduced to solving a system of linear equations. Since the system matrix is sparse, an iterative solver such as preconditioned conjugate gradient (PCG) method shown in Figure 1 is used to solve the system [BW98]. In implementing CG method to run on CPU + GPU, this paper proposes a method to reduce the amount of CPU-GPU communication.

Since the major bottleneck of CG method is Sparse-matrix Vector Multiplication (SpMV), studies on speeding up CG has mostly focused on optimizing SpMV by devising various data structures (e.g., exploiting memory coalescing) for representing the sparse system matrix [TTN*13, WBS*13]. This paper notes a new source of delay in solving CG, which is not curable by accelerating the algebraic operations.

In the CG method (or PCG without loss of generality in regard to the problem this paper tries to solve), as diagrammed in Figure 1, no communication between host and device is needed while executing the inside of the while loop. After each round of iteration, however, communication has to occur because CPU has to make a decision whether to continue the next iteration or to stop based on the residual value which is calculated on the GPU. (In recent GPUs, it is possible to make the GPU make the decision for itself

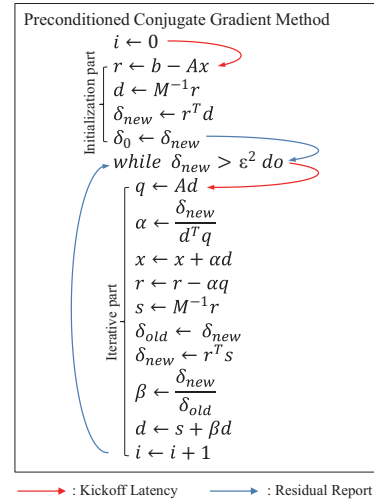


Figure 1: Flow of PCG when utilizing GPU.

via so-called the *Dynamic Parallelism*. In some GPUs, however, self-decision making is not possible. The proposed method does not depend on the style of CPU-GPU collaboration mechanism.)

The above CPU-GPU communication causes two types of latency. The first one is called the *residual report*, which is the latency

for transferring the residual value from device to host. The second one is called *the kickoff latency*, which is the latency introduced in launching a sequence of kernels when CPU finds further iterations need to be carried out. We will call the union of the above, i.e., the residual report plus the kickoff latency collectively as the *sync overhead*.

Sync overhead is not negligible. To measure the latency caused by the sync overhead, we performed a simple swinging handkerchief simulation (see Table 3 (d)) in two different resolutions and summarized the results in Table 1. (The testing environment will be detailed in Section 4.) Columns (a) and (b) show the total time consumed for PCG method including and excluding the sync overhead, respectively, after 128 frames of simulation. As Column (c) shows, the overhead is more than 20%, although it depends on the mesh size.

Table 1: Measurement of sync overhead.

# of vert	(a) With sync overhead	(b) Without sync overhead	(c) Ratio of sync overhead
2.5k	27.71 sec	18.48 sec	33.30%
50k	274.19 sec	212.91 sec	22.35%

This paper introduces a new method to reduce sync overhead between host and device in the context of accelerating PCG method utilizing GPU. The proposed method has no dependency on the data structure used for storing the sparse matrix or the SpMV algorithm. Pre-existing optimized data structures and algorithms can make the speed-up by employing this method.

2. Related Work

There have been a number of studies to exploit GPU technology which can be used for speeding up clothing simulation. At the early stage, [Zel05] proposed a method to implement clothing simulation utilizing the GPU based on Verlet integration. [BG08] introduced various types of data structures for storing the sparse matrix and proposed SpMV kernels that work with those data structures. [TTN*13] noted that, for a structured quad mesh, if the vertices are numbered carefully, non-zero blocks form 17 diagonal lines. Based on that observation, they proposed a modified diagonal format called the CDF (Compressed Diagonal Format), which was optimized for GPU memory access pattern without creating padded zero terms. [WBS*13] proposed a new compressed sparse matrix format called the BIN-CSR (BIN - Compressed Sparse Row) which could exploit memory coalescing even for unstructured meshes. Additionally, they noted that kernel call overhead can hamper the performance and proposed that such an overhead can be reduced by merging several kernels into one.

3. Lazy Residual Evaluation

Seeing that the sync overhead is not negligible, a question arises: do we have to check the residual after every round of iteration? This paper proposes that we can skip some part of the checks. More specifically, it introduces the lazy residual evaluation (LRE) technique. For the subsequent theoretical discussions, we define $n_{converge}$ as the number of iterations to escape from the PCG loop (which is unknown when starting the PCG).

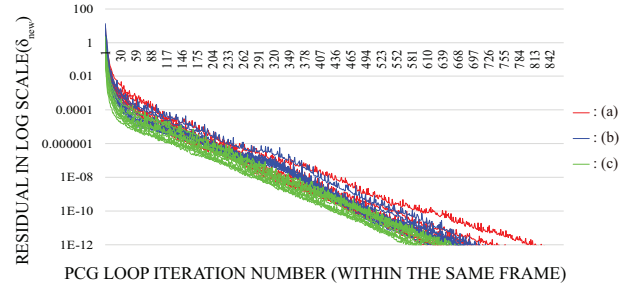


Figure 2: Residual decrease pattern (in log-scale) in simulating three sample garments (a)~(c) of Table 3.

3.1. Static LRE

If we check the residual once in every k iterations, communication between host and device will be reduced from $n_{converge}$ to $\lceil \frac{n_{converge}}{k} \rceil$. When a constant k is used in the above, we will call it the k -static LRE. In choosing the constant integer k , if k is too small, the sync overhead reduction can be insignificant. If k is too large, some amount of unnecessary iterations can occur beyond $n_{converge}$, which will cancel the benefit of LRE. Therefore, a proper k need to be chosen. In this paper, instead of the static LRE, we propose so-called the dynamic LRE in which the number of unchecked iterations is dynamically determined on the fly.

3.2. Dynamic LRE

3.2.1. Phenomenological Analysis of Residual Decrease

If we can predict the residual decrease pattern (even approximately), more effective k could be chosen. When the PCG method is executed, the iteration-residual plot typically takes the form shown in Figure 2. This plot is extracted while simulating three samples of clothing – blouse, one-piece dress, and pants (see Table 3 (a)~(c)) – which are shown in red, green, and blue, respectively. For all three samples, the simulation was run for 128 frames and Figure 2 shows how the PCG method converges, in which the x -axis represents the PCG loop iteration number and the y -axis represents the residual value in log scale. To avoid excessive overlapping in the plot, we randomly chose only 10 frames for each sample.

After making observations on the log-scale PCG convergence pattern in a number of clothing samples (e.g. the ones in Figure 2), we note that there are some common tendencies in the residual reduction:

- The residual decrease can be divided into two stages. In the first stage, the residual decreases rapidly. After the first stage, it enters into the second stage, in which the residual decrease is slower and stable.
- The exact pattern or the length of the first stage is hard to predict. However, for the second stage, if plotted in log scale, we find that the curve shape generally follows approximately a straight line with some noise.
- With a sensible ϵ , the PCG method does not end during the first stage; The PCG loop escape happens in the second stage. According to all the experiments performed so far, the number n_2

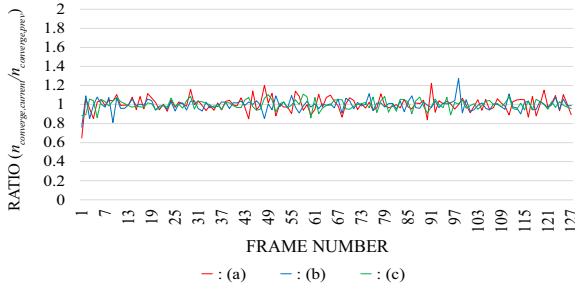


Figure 3: Ratio of $n_{converge}$ between two consecutive frames in the simulation of three sample garments (a)~(c) of Table 3.

of iterations in the second stage is larger than the number n_1 of iterations in the first stage.

Based on the above observations, this paper proposes two strategies that can apply to the first and second stages, respectively.

3.2.2. Skipping the 1st Half

The phenomenological fact that PCG method does not finish in the first stage implies that sync can be omitted altogether for the whole duration of the first stage. More specifically, we propose that the sync should be skipped for the first $0.5 * n_{converge}$ iterations. The question now is how to predict $n_{converge}$. Once again, we use a phenomenological fact.

Figure 3 shows the ratio $\frac{n_{converge.current}}{n_{converge.prev}}$ of $n_{converge}$ between two consecutive frames in the three clothing samples, where $n_{converge.current}$ and $n_{converge.prev}$ are $n_{converge}$ of the current and previous frames, respectively. Disregarding the noise for the moment, the curve is horizontally flat at the height one. Excluding the first frame, note that the ratio is bounded within $[0.8, 1.3]$. It implies that $n_{converge.current}$ can be approximated by $n_{converge.prev}$ since $n_{converge.current}$ is larger than $0.5 * n_{converge.prev}$. Therefore we can safely skip $0.5 * n_{converge.prev}$ iterations in current frame. We call it *Skipping the 1st half*. If the ratio never goes below 0.5, skipping the first half will never bypass $n_{converge}$ of the current frame. (Although we show only three samples here, the ratio is larger than 0.5 even when all the experiments we have done so far are included.) For the case of the first frame, which has no record for the previous frame yet, we just use the Secant LRE method (presented in the next section) without skipping the 1st half.

3.2.3. Secant LRE

We propose a dynamic LRE method called *the Secant LRE* which is based on the Secant method (a well-known method for root-finding). In the Secant LRE, the log of the residual value δ_i of the i -th step is used as the function value and iteration number x_i of that step is used as the function argument as shown in Figure 4. Then, the predicted number k_i of iterations for which the sync can be exempted is given by

$$k_i = \left\lceil \left(\log \epsilon^2 - \log \delta_i \right) \frac{x_i - x_{i-1}}{\log \delta_i - \log \delta_{i-1}} \right\rceil. \quad (1)$$

Because the residual decrease pattern is not exactly a straight

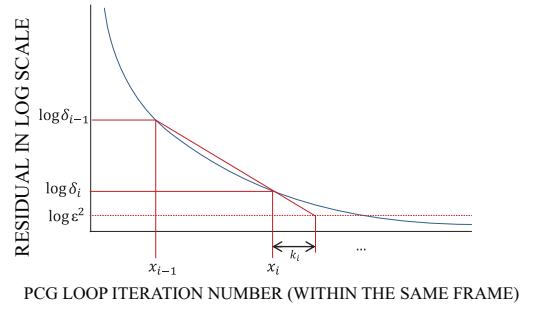


Figure 4: Use of the Secant method to predict the number of iterations for which CPU-GPU sync can be omitted.

line and has some noise, there exists a possibility that k_i given in Equation 1 overshoots or undershoots the situation. For this reason, we propose the clamped version of the Secant LRE, in which the minimum and maximum step values K_{min} and K_{max} , respectively, are used to clamp k_i . More specifically, we use

$$k_i = clamp \left(K_{min}, K_{max}, \left\lceil \left(\log \epsilon^2 - \log \delta_i \right) \frac{x_i - x_{i-1}}{\log \delta_i - \log \delta_{i-1}} \right\rceil \right), \quad (2)$$

where K_{min} and K_{max} are user-controlled parameters.

4. Results

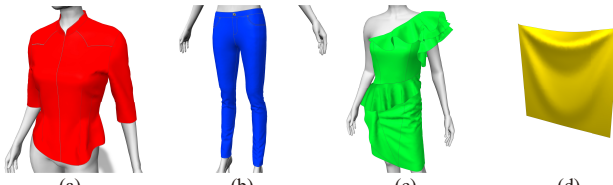
This section reports experimental results and analyzes performance enhancement due to LRE. The proposed method was implemented with C++ and CUDA 9.0 and run on Intel core i7 3770K CPU with 16GB RAM and Nvidia GTX titan X. The linear system was formatted as block compressed sparse row (BCSR), and to get the better performance we employed BSRMV algorithm (by-block) from [EH16]. Dot product kernel was taken from [Nvi17] with CUBLAS_POINTER_MODE_DEVICE option. We adopted the second-order backward difference formula for building the linear system [CK02] and used block Jacobi preconditioner. The force model is from [BW98]. Every kernel was programmed in single precision. We set the K_{min} and K_{max} for clamping the Secant LRE to 5 and 50, respectively, and ϵ to 10^{-6} .

4.1. Analysis of Sync Reduction

We simulated four clothing samples in motion to measure the sync count. Table 3 lists the number of vertices and the number of non-zero 3×3 blocks in the system matrix for each of the clothing samples. Each sample was simulated with i) Original PCG solver, ii) Secant LRE (SecLRE) and iii) Secant LRE after skipping 1st half (SecLRE+SFH). Table 2 summarizes the result. Every sample was simulated for 128 frames, during which we accumulated the number of iterations and the number of syncs. SecLRE method significantly reduced the sync overhead, the 1st half skipping policy reducing only about additional 1 % point. As the total iteration counts show, SecLRE and SecLRE+SFH could overshoot the $n_{converge}$. However, the benefit from reducing synchronization overhead clearly surpassed the loss due to the overshoot PCG iterations.

Table 2: Comparison of sync overhead among different methods.

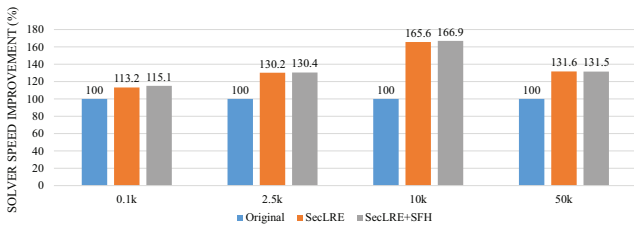
	(a)			(b)			(c)			(d)		
	Total Iteration	# of Syncs	Ratio	Total Iteration	# of Syncs	Ratio	Total Iteration	# of Syncs	Ratio	Total Iteration	# of Syncs	Ratio
Original	98,402	98,402	100%	67,940	67,940	100%	75,536	75,536	100%	45,634	45,634	100%
SecLRE	100,111	2,201	2.2%	69,106	1,591	2.3%	76,603	1,728	2.3%	46,891	1,048	2.2%
SecLRE+SFH	99,898	1,259	1.3%	69,321	932	1.3%	76,776	1,011	1.3%	47,138	580	1.2%

Table 3: Simulation samples and some statistics.


	(a)	(b)	(c)	(d)
# of vertices	5,603	13,992	10,783	2,604
# of non-zero	71,423	181,082	135,903	32,472

4.2. Performance Gain in Various Mesh Resolutions

For the sample (d) in Table 3, we ran the simulation in four different mesh resolutions, i.e., 0.1k, 2.5k, 10k, 50k in the number of vertices. Figure 5 shows how the performance gain of original, SecLRE and SecLRE+SFH varies with those resolutions. SecLRE method reduced the solver time conspicuously, producing 10~60% performance gain. The extra performance gain of SecLRE+SFH from that of SecLRE was marginal.

**Figure 5:** Performance gain (for the case of handkerchief) in various resolutions. From left to right, 0.1k, 2.5k, 10k, 50k in the number of vertices.

5. Conclusion

This paper proposed a new way to reduce the sync overhead when solving the linear system with CPU and GPU. We noted that the communication between CPU and GPU to transfer the residual value and waiting for the CPU's decision whether to continue further iterations can be a source of delay that has been overlooked. By analyzing the log-scale residual decrease pattern, we proposed Lazy Residual Evaluation (LRE) method to skip unnecessary syncs.

We find that the Secant LRE is very effective. It produces

10~60% performance gain. We also tested another version of LRE based on linear regression on the 2nd half, but it was not clearly superior to the Secant LRE.

Acknowledgment

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (MEST) (No. 2015R1A2A1A10055178), in part by the Brain Korea 21 Project 2018 funded by the Ministry of Education (MOE), and in part by ASRI (Automation and Systems Research Institute at Seoul National University).

References

- [BG08] BELL N., GARLAND M.: *Efficient sparse matrix-vector multiplication on CUDA*. Tech. rep., Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008. 2
- [BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 43–54. URL: <http://doi.acm.org/10.1145/280814.280821>, doi:10.1145/280814.280821. 1, 3
- [CK02] CHOI K.-J., KO H.-S.: Stable but responsive cloth. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 604–611. URL: <http://doi.acm.org/10.1145/566570.566624>, doi:10.1145/566570.566624. 3
- [EH16] EBERHARDT R., HOEMMEN M.: Optimization of block sparse matrix-vector multiplication on shared-memory parallel architectures. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (May 2016), pp. 663–672. doi:10.1109/IPDPSW.2016.42. 3
- [Nvi17] NVIDIA: Cublas library user guide. <http://developer.nvidia.com/cublas>. 3
- [TTN*13] TANG M., TONG R., NARAIN R., MENG C., MANOCHA D.: A gpu-based streaming algorithm for high-resolution cloth simulation. *Computer Graphics Forum* 32, 7 (2013), 21–30. URL: <http://dx.doi.org/10.1111/cgf.12208>, doi:10.1111/cgf.12208. 1, 2
- [WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D.: Efficient gpu data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum* 32, 1 (2013), 16–26. URL: <http://dx.doi.org/10.1111/j.1467-8659.2012.03227.x>, doi:10.1111/j.1467-8659.2012.03227.x. 1, 2
- [Zel05] ZELLER C.: Cloth simulation on the gpu. In *ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), SIGGRAPH '05, ACM. URL: <http://doi.acm.org/10.1145/1187112.1187158>, doi:10.1145/1187112.1187158. 2