Chapter 4
GPU-Accelerated Triangulation in the Stereo-DIC Algorithm Using a Heterogeneous Framework

Mullai Thiagu, Rupesh Nasre, and Sankara J. Subramanian

Abstract Stereo-DIC is a well-established non-contact, optical technique used to measure surface deformation. Recent advancements aim to improve both computation speed and accuracy of the stereo-DIC algorithm, facilitating real-time analysis of stereo image pairs captured during in-situ experiments. Previous studies in literature have demonstrated that leveraging a high-performance parallel computing platform, such as a GPU, can significantly improve computation speed for operations that are data-independent. While there is evidence that GPUs enhance the overall speedup factor, a comprehensive analysis of the implementation of each individual step of the stereo-DIC algorithm is yet to be discussed. The purpose of this work is to efficiently implement the stereo triangulation step using a GPU. It is to be noted that the stereo triangulation step is performed twice to calculate the full-field, surface displacement vectors between two stereo image pairs. In this work, a heterogeneous framework consisting of an Intel Xeon octa-core CPU and a Nvidia Tesla K20C GPU card is deployed. The CPU is used for pre-processing tasks such as transfer of spatial coordinates derived from stereo correlation and camera calibration matrix parameters, whereas the GPU is used to implement the triangulation process. By harnessing parallelism both within a thread block and across multiple thread blocks in a Compute Unified Device Architecture (CUDA) programming model, the triangulation process is executed concurrently. It is observed that when the total number of spatial co-ordinates to be triangulated ranges from 50 to 10,000, the overall speedup factor improves by $\approx 758\times$ to $\approx 232\times$.

Keywords Three-dimensional digital image correlation (3D-DIC) · Stereo triangulation · Camera calibration matrix · Heterogeneous framework · Graphics Processing Unit (GPU)

Introduction

In literature, it is well established that the stereo-DIC algorithm is primarily deployed to measure both in-plane and out-of-plane (i.e. surface) deformations of arbitrarily shaped specimens [2], [3]. A conventional stereo-DIC algorithm encompasses three important steps : a) stereo correlation, b) left/right temporal correlation and c) stereo triangulation. Firstly, the purpose of stereo correlation step is to compute the disparity values of subsets between left and right images captured of a specimen before deformation. On the other hand, the objective of the temporal correlation step is to trace the movement of the subset centroids in the reference (undeformed) stereo image pair along the left/right camera image(s). Here, the in-plane flow of optical vectors for respective subsets is tracked between sequences of left/right camera images captured during the deformation process by using two-dimensional digital image correlation technique. The third step (i.e. stereo triangulation) is carried out individually for reference stereo image pair as well as deformed stereo image pair, to reconstruct the 3D coordinates of point correspondences. Finally, difference between the triangulated reference and triangulated deformed surface(s) is calculated to

Mullai Thiagu
Department of Computer Science, SIAS, Krea University, Sri City - 517 646, Andhra Pradesh, India
e-mail: mullai.thiagu@krea.edu.in

Rupesh Nasre
Department of Computer Science and Engineering, Indian Institute of Technology, Madras, Chennai-600 036, Tamil Nadu, India
e-mail: rupesh@cse.iitm.ac.in

Sankara J. Subramanian
Co-Founder, CTO, PhotoGAUGE
e-mail: shankar.j.subramanian@gmail.com

Chi-Hung Hwang et al. (eds.), *Advancement of Optical Methods & Digital Image Correlation in Experimental Mechanics, Vol. 2*, Conference Proceedings
of the Society for Experimental Mechanics Series, https://doi.org/10.13052/97887-438-0828-2_4

yield full-field, surface displacement/strain data. The purpose of this paper is to implement the stereo triangulation computation of the reference and deformed surface using a GPU. This step has the scope for massive data parallelism and therefore the approach elucidated by the authors in their previous works [4], [5] is adopted here as well to study the impact of a heterogeneous (CPU-GPU) framework towards improving the computation speed of the stereo triangulation step. As explained in our earlier works, we refer to the technique in which parallelism is achieved across thread blocks[1] as *inter-parallelism* and parallelism within a thread block as *intra-parallelism*. We propose to use a heterogeneous (CPU-GPU) framework and evaluate the gain in speedup factor on a typical stereo-DIC dataset. The remainder of the paper is organized as follows: in the next section , we discuss the background details of the heterogeneous framework, followed by the analysis of results in Section 3 and concluding remarks in Section 4.

Background

Stereo triangulation is the process of finding the position of an object point in three-dimensional space given its positions in the stereo image pair acquired using the calibrated stereo rig [1]. As illustrated in Figure 1, this process is applied on a group of corresponding camera coordinate object points derived from stereo correlation step applied between the rectified reference stereo image pair ($\mathbf{I^{RL_1}}, \mathbf{I^{RR_1}}$). The spatial coordinates of corresponding points (i.e., $\mathbf{X_1^L}, \mathbf{Y_1^L}, \mathbf{X_1^R}, \mathbf{Y_1^R}$) within a stereo image pair in the camera coordinate system are transformed onto a set of distinct 3D object points (i.e., $\mathbf{X_1^{3D}}, \mathbf{Y_1^{3D}}, \mathbf{Z_1^{3D}}$).

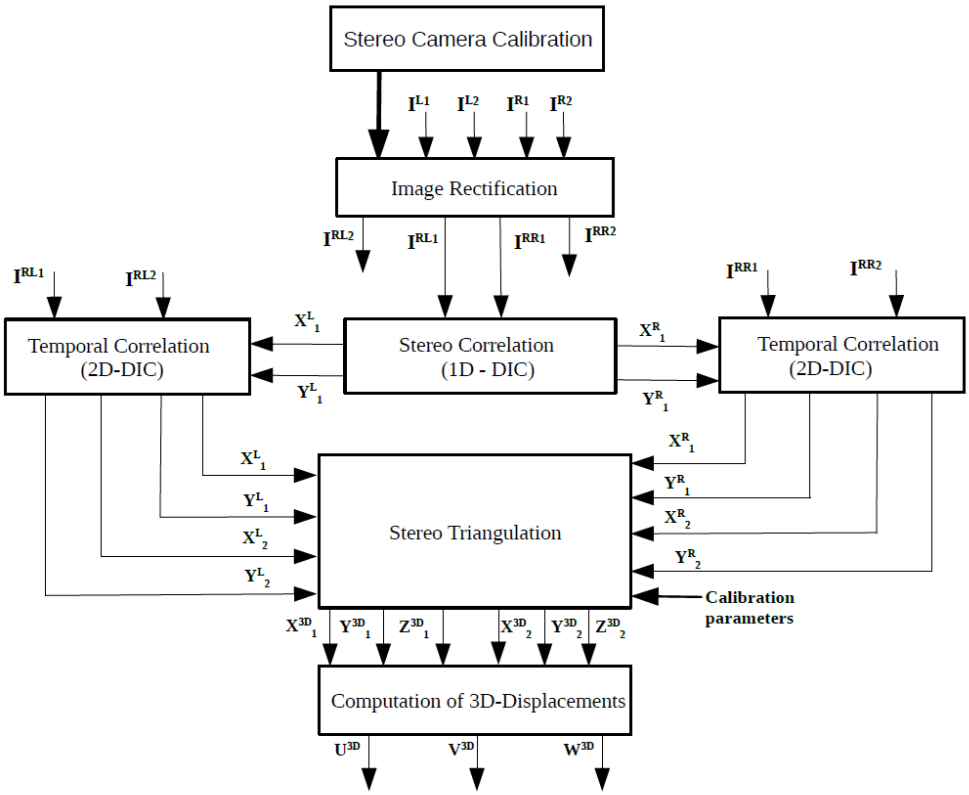

Fig. 1 Computation flow of image processing using Stereo-DIC technique with two stereo image pairs.

Nomenclature:

- $\mathbf{I^{RL_1}}, \mathbf{I^{RR_1}}$: Rectified image pairs of reference stereo image pair ($\mathbf{I^{L1}}, \mathbf{I^{R1}}$)
- $\mathbf{I^{RL_2}}, \mathbf{I^{RR_2}}$: Rectified image pairs of deformed stereo image pair ($\mathbf{I^{L2}}, \mathbf{I^{R2}}$)

[1] terminology used in CUDA programming model, https://developer.nvidia.com/cuda-toolkit

- $\mathbf{X_1^L}, \mathbf{Y_1^L}, \mathbf{X_1^R}, \mathbf{Y_1^R}$: Correspondence points after stereo correlation step in spatial domain
- $\mathbf{X_2^L}, \mathbf{Y_2^L}$: Correspondence points after left temporal correlation step in spatial domain
- $\mathbf{X_2^R}, \mathbf{Y_2^R}$: Correspondence points after right temporal correlation step in spatial domain
- $\mathbf{X_1^{3D}}, \mathbf{Y_1^{3D}}, \mathbf{Z_1^{3D}}$: Reference surface re-projected after stereo triangulation step of ($\mathbf{X_1^L}, \mathbf{Y_1^L}, \mathbf{X_1^R}, \mathbf{Y_1^R}$)
- $\mathbf{X_2^{3D}}, \mathbf{Y_2^{3D}}, \mathbf{Z_2^{3D}}$: Deformed surface re-projected after stereo triangulation step of ($\mathbf{X_2^L}, \mathbf{Y_2^L}, \mathbf{X_2^R}, \mathbf{Y_2^R}$)
- $\mathbf{U^{3D}}, \mathbf{V^{3D}}, \mathbf{W^{3D}}$: Full-field displacement vectors in 3D

The matrix representation to compute the 3D position of any point, $\tilde{\mathbf{X}} = (\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ in the left camera is written as below. This typically involves solving a system of linear equations.

$$\mathbf{[V]_{4\times 3}}\{\tilde{\mathbf{X}}\}_{3\times 1} = \{\mathbf{r}\}_{4\times 1} \tag{1}$$

where,

$$\mathbf{[V]_{4\times 3}} = \begin{bmatrix} f_x^L & 0 & (x^L - c_x^L) \\ 0 & f_y^L & (y^L - c_y^L) \\ -f_x^R & 0 & (x^R - c_x^R) \\ 0 & f_y^R & (y^R - c_y^R) \end{bmatrix} \tag{2}$$

$$\mathbf{\{r\}_{4\times 1}} = \begin{bmatrix} 0 \\ 0 \\ -f_x^R T \\ 0 \end{bmatrix} \tag{3}$$

The elements of the matrix $\mathbf{V}$ represent intrinsics parameters that are derived from the camera calibration process. In this context, f_x^L and f_y^L represent the focal lengths of the left camera along the $\mathbf{x}$ and $\mathbf{y}$ axes, respectively. Similarly, c_x^L and c_y^L denote the principal point of the left camera along the $\mathbf{x}$ and $\mathbf{y}$ axes. For the right camera, f_x^R and f_y^R represent the focal lengths along the $\mathbf{x}$ and $\mathbf{y}$ axes, respectively, while c_x^R and c_y^R represent the principal point of the right camera along the $\mathbf{x}$ and $\mathbf{y}$ axes. Next, the vector $\{\mathbf{r}\}$ is a 4×1 column vector, which is related to the translation between the stereo camera pair. The solution of the above system of equations is obtained by the linear least square method represented in Equation 4 below:

$$\{\tilde{\mathbf{X}}_\mathbf{1}^\mathbf{L}\} = [\mathbf{V^T V}]^{-\mathbf{1}}[\mathbf{V}]^\mathbf{T}\mathbf{r} \tag{4}$$

Next, equation 5 is used to transform a point $\{\tilde{\mathbf{X}}_\mathbf{1}^\mathbf{L}\}$ in the left camera coordinate system to the right camera coordinate system.

$$\{\tilde{\mathbf{X}}_\mathbf{2}^\mathbf{R}\} = \mathbf{R}\{\tilde{\mathbf{X}}_\mathbf{1}^\mathbf{L}\} + \mathbf{T} \tag{5}$$

Here, the rotation ($\mathbf{R}$) matrix and translation ($\mathbf{T}$) vector describes the rotation as well as translation of the left camera coordinate system relative to the right camera coordinate system. These are the extrinsic parameters calculated during the camera calibration process. By repeating the above process for multiple points in the stereo correlated images, 3D surface(s) of the object in its undeformed and deformed states is reconstructed. It is well known that the stereo rectification process simplifies the number of intrinsic parameters by removing distortion. Thus, the orientation of right camera with respect to left camera is represented by an identity matrix as in Equation 6. In a similar way, the translation between the cameras in the rectified stereo rig is represented by a vector $\mathbf{T}$ as given in equation 7.

$$\mathbf{[R]} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$\mathbf{[T]} = \begin{bmatrix} \mathbf{T_x} \\ \mathbf{T_y} \\ \mathbf{T_z} \end{bmatrix} = \begin{bmatrix} \mathbf{T[1,1]} \\ \mathbf{T[2,1]} \\ \mathbf{T[3,1]} \end{bmatrix} \tag{7}$$

Typically, the stereo triangulation process can be viewed as a structure/shape reconstruction problem. Here, the point correspondences estimated by using stereo correlation step is projected to determine the point(s) of intersection (i.e., 3D points). It can be understood that the number of times the stereo triangulation process is repeated is same as the number of stereo rectified image pairs available during post processing. If there are `t` corresponding points between rectified stereo images $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RL_2}}$, re-projection of the 3D points is given by the Equation 8.

Input: $\mathbf{X_1^L}, \mathbf{Y_1^L}, \mathbf{X_1^R}, \mathbf{Y_1^R}, \mathbf{c_x^L}, \mathbf{c_y^L}, \mathbf{c_x^R}, \mathbf{c_y^R}, \mathbf{f_x^L}, \mathbf{f_y^L}, \mathbf{f_x^R}, \mathbf{f_y^R}, \mathbf{R}, \mathbf{T}$

Output: $\mathbf{X_1^{3D}}, \mathbf{Y_1^{3D}}, \mathbf{Z_1^{3D}}$

for $i \leftarrow 1$ **to** t **do**

$\mathrm{x_t}[1,\mathrm{i}] = \frac{\mathrm{X_1^L}[1,\mathrm{i}]-c_x^L}{f_x^L}$ ▷ Left camera - homogeneous coordinates

$\mathrm{x_t}[2,\mathrm{i}] = \frac{\mathrm{Y_1^L}[1,\mathrm{i}]-c_y^L}{f_y^L}$

$\mathrm{x_t}[3,\mathrm{i}] = 1.0$

$\mathrm{x_{tt}}[1,\mathrm{i}] = \frac{\mathrm{X_1^R}[1,\mathrm{i}]-c_x^R}{f_x^R}$ ▷ Right camera - homogeneous coordinates

$\mathrm{x_{tt}}[2,\mathrm{i}] = \frac{\mathrm{Y_1^R}[1,\mathrm{i}]-c_y^R}{f_y^R}$

$\mathrm{x_{tt}}[3,\mathrm{i}] = 1.0$

$\mathrm{u}[1:3,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{R}[1:3,\mathrm{m}] * \mathrm{x_t}[\mathrm{m},\mathrm{i}])$

$\mathrm{nx_{t2}}[1,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{x_t}[\mathrm{m},\mathrm{i}])^2$

$\mathrm{nx_{tt2}}[1,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{x_{tt}}[\mathrm{m},\mathrm{i}])^2 + \mathrm{u}[3,\mathrm{i}] * \mathrm{x_{tt}}[3,\mathrm{i}]$

$\mathrm{DD}[1,\mathrm{i}] = (\mathrm{nx_{t2}}[1,\mathrm{i}] * \mathrm{nx_{tt2}}[1,\mathrm{i}]) - \{\sum_{\mathrm{m}=1}^{3} (\mathrm{u}[\mathrm{m},\mathrm{i}] * \mathrm{x_{tt}}[\mathrm{m},\mathrm{i}])\}$

$\mathrm{u_T}[1,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{u}[\mathrm{m},\mathrm{i}] * \mathrm{T}[\mathrm{m},1])$

$\mathrm{x_{ttT}}[1,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{x_{tt}}[\mathrm{m},\mathrm{i}] * \mathrm{T}[\mathrm{m},1])$

$\mathrm{x_{ttu}}[1,\mathrm{i}] = \sum_{\mathrm{m}=1}^{3} (\mathrm{u}[\mathrm{m},\mathrm{i}] * \mathrm{x_{tt}}[\mathrm{m},1])$

$\mathrm{NN_1}[1,\mathrm{i}] = (\mathrm{x_{ttu}}[1,\mathrm{i}] * \mathrm{x_{ttT}}[1,\mathrm{i}]) - (\mathrm{nx_{tt2}}[1,\mathrm{i}] * \mathrm{u_T}[1,\mathrm{i}])$

$\mathrm{NN_2}[1,\mathrm{i}] = (\mathrm{nx_{t2}}[1,\mathrm{i}] * \mathrm{x_{ttT}}[1,\mathrm{i}]) - (\mathrm{u_T}[1,\mathrm{i}] * \mathrm{x_{ttu}}[1,\mathrm{i}])$

$\mathrm{Z_t}[1,\mathrm{i}] = \frac{\mathrm{NN1}[1,\mathrm{i}]}{\mathrm{DD}[1,\mathrm{i}]}$

$\mathrm{Z_{tt2}}[1,\mathrm{i}] = \frac{\mathrm{NN2}[1,\mathrm{i}]}{\mathrm{DD}[1,\mathrm{i}]}$

$\mathrm{X_1}[1:3,\mathrm{i}] = \mathrm{x_t}[1:3,\mathrm{i}] * \mathrm{Z_t}[1:3,\mathrm{i}]$

$\mathrm{X_2}[1:3,\mathrm{i}] = \sum_{\mathrm{j}=1}^{3} \mathrm{R}[\mathrm{j},\mathrm{i}] * \mathrm{X_1}[\mathrm{j},1:3] + \mathrm{T}[\mathrm{j},1]$

$\mathrm{X_L}[1:3,\mathrm{i}] = \frac{\mathrm{X_1}[1:3,\mathrm{i}]+\mathrm{X_2}[1:3,\mathrm{i}]}{2}$

$\mathbf{X_1^{3D}}[\mathbf{1},\mathbf{i}] = \mathrm{X_L}[1,\mathrm{i}]$

$\mathbf{Y_1^{3D}}[\mathbf{1},\mathbf{i}] = \mathrm{X_L}[2,\mathrm{i}]$

$\mathbf{Z_1^{3D}}[\mathbf{1},\mathbf{i}] = \mathrm{X_L}[3,\mathrm{i}]$

end

Algorithm 1: Pseudo-code for stereo triangulation between $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ using `Sequential version`

$$\{\mathbf{X^L}\} = \begin{bmatrix} \mathbf{X_1^{3D}} \\ \mathbf{Y_1^{3D}} \\ \mathbf{Z_1^{3D}} \end{bmatrix} \tag{8}$$

The pseudo-code for stereo triangulation between $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ is presented as Algorithm 1. This open source implementation is based on the camera calibration toolbox for Matlab by Jean-Yves Bouguet.[2] The inputs to this module are

[2] http://robots.stanford.edu/cs223b04/JeanYvesCalib/.

spatial point correspondences, focal length, principal points, rotation matrix and translation vector values. A for loop is initialized at line 4 for `t` subsets between images $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ to compute triangulated points corresponding to the reference surface as the output. At the end of the first iteration, this results in a 3D point for the first subset ($\mathbf{X_1^{3D}}[1,1]$, $\mathbf{Y_1^{3D}}[\mathbf{1},\mathbf{1}]$, $\mathbf{Z_1^{3D}}[\mathbf{1},\mathbf{1}]$). Line 5 to line 25 represent the sequence of calculations during an iteration cycle. After initialization of `for` loop, the first step is to normalize the image projections in homogeneous coordinates as represented by line 5 upto line 10. Next, equations 4 and 5 are implemented by line 10 upto line 23. Finally, this process is repeated for `t` iterations.

Analysis

The pseudo-code to reconstruct a 3D surface (single reference and multiple deformed surface(s)) using a GPU is presented as Algorithm 2. In this pseudo-code, three kernels are launched to transform the calculations run using CPU onto a GPU. The first kernel is responsible to normalize the spatial coordinates of left camera ranging from line 4 to line 9. Likewise, code from line 10 to line 14 is used to normalize the coordinates of the right camera. Here, the focus of the third kernel is to calculate the 3D object point corresponding to each subset in a given stereo image pair. It is observed that this step has massive scope for data parallelism (i.e. `intra-parallelism`, `inter-parallelism` and `inter-image parallelism`). It is made feasible by storing the spatial coordinates of the point correspondences between $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RL_2}}$ together with point correspondences between $\mathbf{I^{RR_1}}$ and $\mathbf{I^{RR_2}}$ in the global memory of the GPU thereby enabling simultaneous data access by individual processor cores. As mentioned earlier, a stereo triangulation step is executed two times during processing of two stereo image pairs as per the stereo-DIC algorithm. The first stereo triangulation step is useful to re-project a reference surface of the specimen. The second invocation of triangulation step is responsible to reconstruct the deformed surface of the specimen. Because of the data independent nature of calculations within the stereo triangulation module, it is evident that its computation speed shall improve by mapping the steps to hardware resources of the GPU.

Before presenting the results for stereo-DIC computation, we present the configuration details of the framework used to test the stereo image pairs. In this study, the heterogeneous framework comprises of Intel Xeon octa-core CPU and NVIDIA Tesla K20C GPU. The software package utilized to read digital images and implement `sequential version` of the stereo-DIC algorithm is developed using MATLAB R2017a.

To study the performance of the stereo-triangulation algorithm, `Stereo Sample 1 (35mm – Experimental images)` dataset from Stereo DIC challenge repository [3] is considered. This dataset consists of experimental images with resolution of 2048×2448 pixels. In this research work, the stereo images are used after rectification of stereo pairs. For the purpose of this experiment, a ROI in both $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ is selected as shown in Figure 2. Firstly, stereo correlation module is executed between reference stereo image pairs ($\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$).

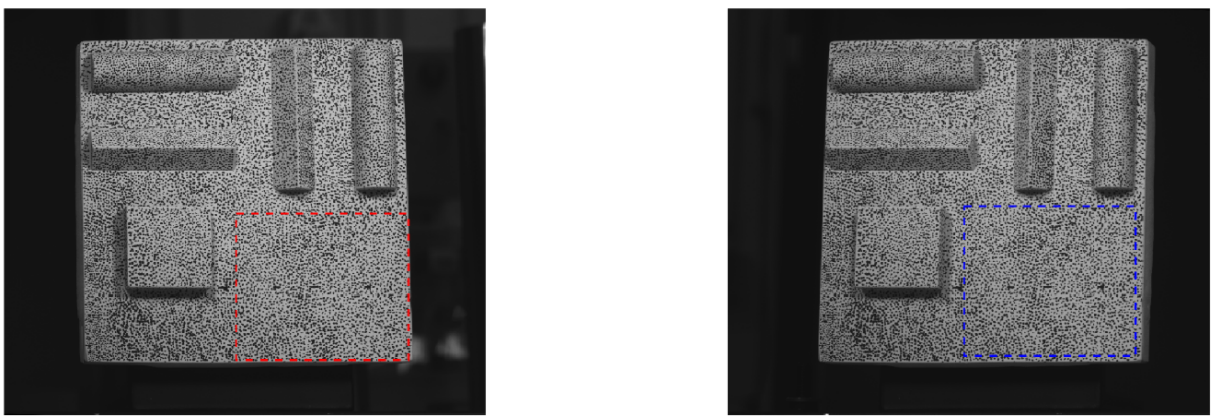

Fig. 2 Stereo Sample 1 dataset from DIC Challenge Repository: (a) Rectified left reference image (I^{RL_1}) with a chosen region-of-interest, (b) Rectified right reference image (I^{RR_1}) of undeformed specimen.

[3] https://idics.org/challenge/

The average timing results for stereo triangulation of reference surface with 437 subsets using `heterogeneous version` is 17.43 milliseconds (inclusive of data transfer overhead) and 0.30 milliseconds (exclusive of data transfer overhead). This implies that the speedup factor shall improve with increase in the number of subsets.

Input: $\mathbf{X_1^L}$, $\mathbf{Y_1^L}$, $\mathbf{X_1^R}$, $\mathbf{Y_1^R}$, $\mathbf{c_x^L}$, $\mathbf{c_y^L}$, $\mathbf{c_x^R}$, $\mathbf{c_y^R}$, $\mathbf{f_x^L}$, $\mathbf{f_y^L}$, $\mathbf{f_x^R}$, $\mathbf{f_y^R}$, $\mathbf{R}$, $\mathbf{T}$

Output: $\mathbf{X_1^{3D}}$, $\mathbf{Y_1^{3D}}$, $\mathbf{Z_1^{3D}}$

for *($t \times N$) parallel threads* **do**

tid = getThreadId() ▷ `Intra-parallelism and Inter-parallelism`

$x_t[2 * tid] = (X_1^L[2 * tid] - c_x^L[0])/f_x^L[0]$

$x_t[(2 * tid) + 1] = (Y_1^L[(2 * tid) + 1] - c_y^L[0])/f_y^L[0]$

$xt[0] = x_t[2 * tid]$

$xt[1] = x_t[(2 * tid) + 1]$

$xt[2] = 1.0f$

$x_{tt}[2 * tid] = (X_1^R[2 * tid] - c_x^R[0])/f_x^R[0]$

$x_{tt}[(2 * tid) + 1] = (Y_1^R[(2 * tid) + 1] - c_y^R[0])/f_y^R[0]$

$xtt[0] = x_{tt}[2 * tid]$

$xtt[1] = x_{tt}[(2 * tid) + 1]$

$xtt[2] = 1.0f$

$u[0] = (R[0] * xt[0]) + (R[3] * xt[1]) + (R[6] * xt[2])$

$u[1] = (R[1] * xt[0]) + (R[4] * xt[1]) + (R[7] * xt[2])$

$u[2] = (R[2] * xt[0]) + (R[5] * xt[1]) + (R[8] * xt[2])$

$nx_{t2} = (xt[0])^2 + (xt[1])^2 + (xt[2])^2$

$nx_{tt2} = (xtt[0])^2 + (xtt[1])^2 + (xtt[2])^2$

$DD = (nx_{t2} * nx_{tt2}) - \{(u[0] * xtt[0]) + (u[1] * xtt[1]) + (u[2] * xtt[2])\}^2$

$u_T = \sum_{m=0}^{2} (u[m] * T[m])$

$x_{ttT} = \sum_{m=0}^{2} (xtt[m] * T[m])$

$x_{ttu} = \sum_{m=0}^{2} (u[m] * xtt[m])$

$NN1 = (xttu * xttT) - (nx_{tt2} * u_T)$

$NN2 = (nx_{t2} * x_{ttT}) - (nx_{tt2} * u_T)$

Algorithm 2: Pseudo-code for stereo triangulation between $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ in stereoDIC computation realized using GPU (`Heterogeneous version`) continued ...

for *($t \times N$) parallel threads* **do**

$\mathrm{Z_t} = \frac{\mathrm{NN1}}{\mathrm{DD}}$

$\mathrm{Z_{tt}} = \frac{\mathrm{NN2}}{\mathrm{DD}}$

$\mathrm{X_1}[0] = \mathrm{xt}[0] * \mathrm{Z_t}$

$\mathrm{X_1}[1] = \mathrm{xt}[1] * \mathrm{Z_t}$

$\mathrm{X_1}[2] = \mathrm{xt}[2] * \mathrm{Z_t}$

$\mathrm{diff_{new}}[0] = (\mathrm{xtt}[0] * \mathrm{Z_{tt}}) - \mathrm{T}[0]$

$\mathrm{diff_{new}}[1] = (\mathrm{xtt}[1] * \mathrm{Z_{tt}}) - \mathrm{T}[1]$

$\mathrm{diff_{new}}[2] = (\mathrm{xtt}[2] * \mathrm{Z_{tt}}) - \mathrm{T}[2]$

$\mathrm{R}^{'} = \mathrm{transpose}(\mathrm{R})$

$\mathrm{X_2}[0] = (\mathrm{R}^{'}[0] * \mathrm{diff_{new}}[0]) + (\mathrm{R}^{'}[3] * \mathrm{diff_{new}}[1]) + (\mathrm{R}^{'}[6] * \mathrm{diff_{new}}[2])$

$\mathrm{X_2}[1] = (\mathrm{R}^{'}[1] * \mathrm{diff_{new}}[0]) + (\mathrm{R}^{'}[4] * \mathrm{diff_{new}}[1]) + (\mathrm{R}^{'}[7] * \mathrm{diff_{new}}[2])$

$\mathrm{X_2}[2] = (\mathrm{R}^{'}[2] * \mathrm{diff_{new}}[0]) + (\mathrm{R}^{'}[5] * \mathrm{diff_{new}}[1]) + (\mathrm{R}^{'}[8] * \mathrm{diff_{new}}[2])$

$\mathrm{X_L}[0] = (\mathrm{X_1}[0] + \mathrm{X_2}[0])/2$

$\mathrm{X_L}[1] = (\mathrm{X_1}[1] + \mathrm{X_2}[1])/2$

$\mathrm{X_L}[2] = (\mathrm{X_1}[2] + \mathrm{X_2}[2])/2$

$\mathbf{X_1^{3D}}[\mathbf{tid}] = \mathrm{X_L}[0]$

$\mathbf{Y_1^{3D}}[\mathbf{tid}] = \mathrm{X_L}[1]$

$\mathbf{Z_1^{3D}}[\mathbf{tid}] = \mathrm{X_L}[2]$

Algorithm 2: Pseudo-code for stereo triangulation between $\mathbf{I^{RL_1}}$ and $\mathbf{I^{RR_1}}$ in stereoDIC computation realized using GPU (in `Heterogeneous version`)

- `reference surface` 36.82 milli seconds
- `deformed surface` 11.94 milli seconds

The proposed heterogeneous framework also implements the pseudo-code presented in Algorithm 2. As discussed earlier, the stereo triangulation is implemented to explore both `intra-parallelism` and `inter-parallelism`. Table 1 presents average computation times in milliseconds for different pixel counts (from 1 to 10,000,000) are compared across CPU, $\mathrm{GPU_1}$, and $\mathrm{GPU_2}$ implementations. As pixel count increases, the computation time on the CPU remains signifi-

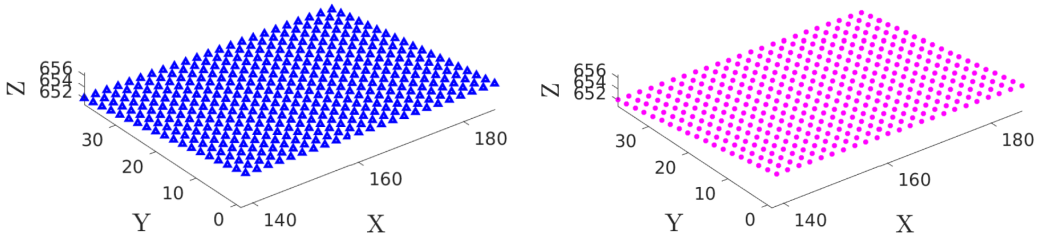

Fig. 3 Stereo triangulation of reference surface computed using (a) `Sequential` and (b) `Heterogeneous version`.

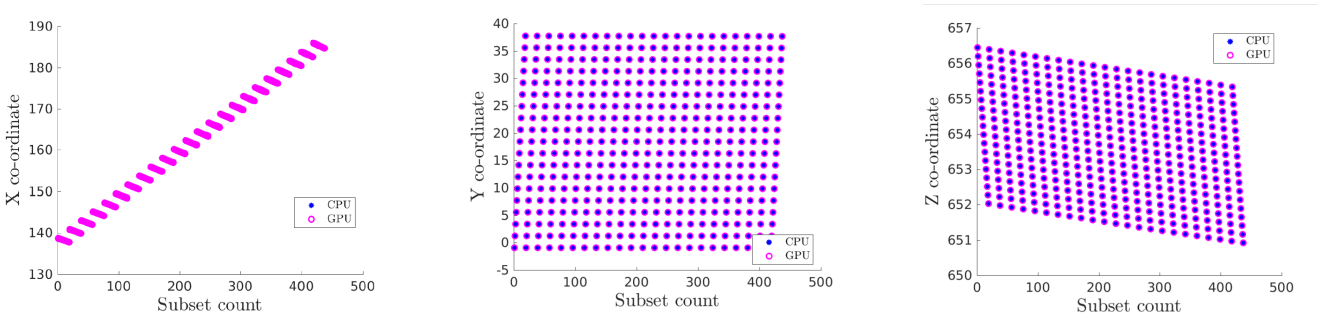

Fig. 4 Stereo triangulation : Compare 3-D co-ordinates of reference surface along (a) X-direction, (b) Y-direction , (c) Z-direction computed using `Sequential version` and `Heterogeneous version`

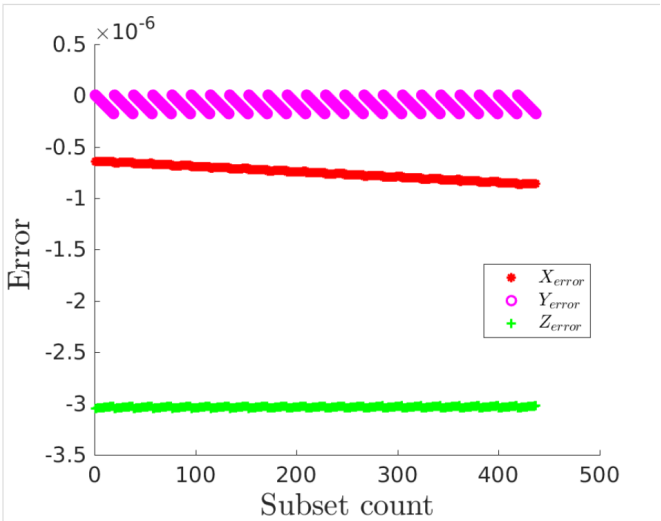

Fig. 5 Stereo triangulation of reference surface : Error between `Sequential` and `Heterogeneous version`

cantly higher than that on the GPUs. The speedup factors illustrate substantial performance improvements, particularly with GPU_{25}, where speedup factor reaches up to $758.4\times$ for smaller pixel counts. However, as the pixel count increases to $10,000,000$, the speedup factors drop to approximately $1.8\times$, indicating diminishing returns due to data transfer overhead. Overall, the results underscore the efficiency of heterogeneous processing, particularly for smaller pixel counts, while also highlighting the impact of data transfer overhead on performance.

Table 1 Stereo triangulation - Impact of data transfer overhead : Pixel count versus computation time and speedup factor (inclusive and exclusive of data transfer overhead) specifically realized using `sequential` and `heterogeneous` version.

Number of pixels	Average computation time (in milli seconds)			Speedup factor	
	CPU	**GPU_1**[4]	**GPU_2**[5]	**S_1**[6]	**S_2**[7]
1	221.7	16.7	0.3	13.3×	749.3×
50	225.3	16.8	0.3	13.4×	758.4×
100	228.5	16.8	0.3	13.6×	720.5×
500	227.3	16.4	0.3	13.9×	730.2×
1000	237.5	16.7	0.3	14.2×	729.9×
5000	234.8	19.4	0.3	12.1×	724.3×
10000	223.7	20.1	0.3	11.1×	732.9×
50000	224.6	23.6	0.4	9.5×	548.5×
100000	224.0	28.8	1.0	7.8×	232.9×
5000000	995.9	511.4	8.1	1.9×	123.4×
10000000	1757.7	991.7	15.1	1.8×	116.5×

[GPU_{14}, S_{16}] Including data transfer overhead
[GPU_{25}, S_{27}] Excluding data transfer overhead

Conclusion

To summarize, this paper addresses the parallelization approach followed to improve the computation speed of a stereo triangulation algorithm. By increasing the total number of spatial co-ordinates to be triangulated from 50 to 10,000, it is observed that by using `heterogeneous` framework, the overall speedup factor of the stereo triangulation step improves by $\approx 758\times$ to $\approx 232\times$. It also highlights the impact of data transfer overhead on the overall gain in speedup factor. Another advantage of the proposed framework is that it is modular, scalable, and can be easily ported on to other NVIDIA-based GPU cards.

Acknowledgments Indian Institute of Technology Madras and Krea University, India.

References

1. R. I. Hartley and P. Sturm. Triangulation. *In International Conference on Computer Analysis of Images and Patterns, Springer, Berlin, Heidelberg*, pages 190–197, 1995.
2. M. A. Sutton, J. J. Orteu, and H. Schreier. Image correlation for shape, motion and deformation measurements: basic concepts, theory and applications. 2009.
3. M.A. Sutton and F. Hild. Recent advances and perspectives in digital image correlation. *Experimental Mechanics*, 55:1–8, 2015.
4. Mullai Thiagu, Sankara J Subramanian, and Rupesh Nasre. Fast, sub-pixel accurate digital image correlation algorithm powered by heterogeneous (cpu-gpu) framework. In *Advancement of Optical Methods & Digital Image Correlation in Experimental Mechanics, Volume 3: Proceedings of the 2018 Annual Conference on Experimental and Applied Mechanics*, pages 95–102. Springer, 2019.
5. Mullai Thiagu, Sankara J Subramanian, and Rupesh Nasre. High-speed, two-dimensional digital image correlation algorithm using heterogeneous (CPU-GPU) framework. *Strain*, 56(3):e12342, 2020.