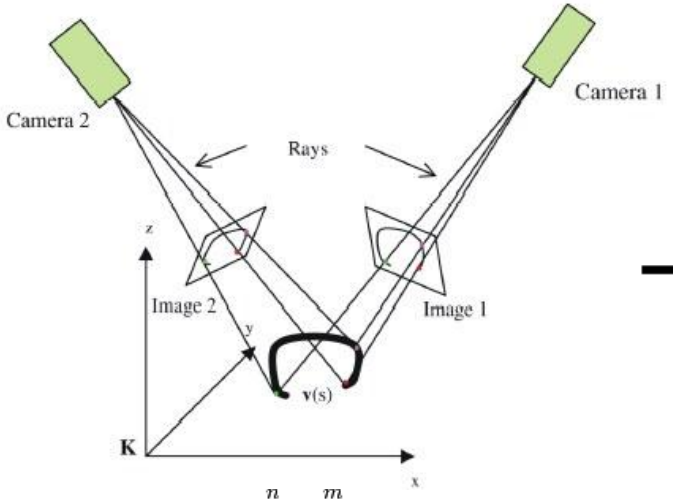
PRACTICAL TIME BUNDLE ADJUSTMENT FOR 3D RECONSTRUCTION ON THE GPU

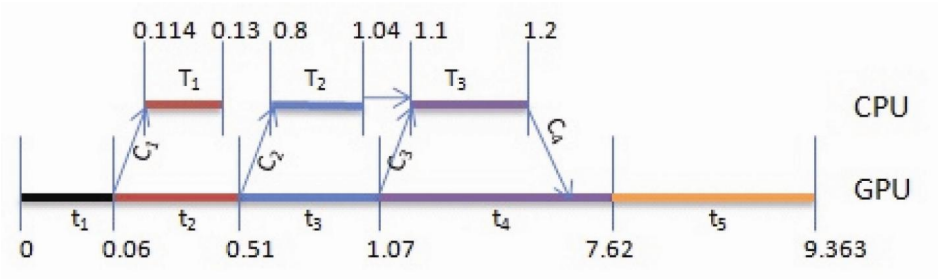
Siddharth Choudhary (IIIT Hyderabad), Shubham Gupta (IIIT Hyderabad),
P J Narayanan (IIIT Hyderabad)

Abstract



$$\min_{\mathbf{a}_j, \mathbf{b}_i} \sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij}\, d(\mathbf{Q}(\mathbf{a}_j, \mathbf{b}_i),\ \mathbf{x}_{ij})^2,$$

+



Takes around 9 seconds to perform one iteration on GPU for 488 cameras, giving it a speedup of around 10 times

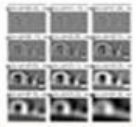
Outline

- Motivation
- Related Work
- Problem Statement
- What is Bundle Adjustment ?
- Sparse Bundle Adjustment on the GPU
- Results and Analysis
- Preliminary Results on Fermi
- Summary
- Future Work

Motivation

Images

Interest Point Detection



Invariant Point Description

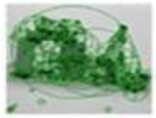
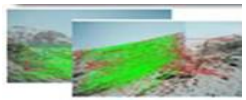
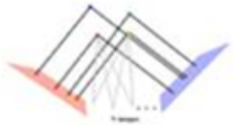


Image Matching

Image Geometry Model Fitting



Connected Component Identification and 3D Point Estimation



Refinement using Bundle Adjustment and Model Output

50% of Computation Time

50% of Computation Time

Related Work

- Building Rome in a Day (ICCV 2009)
  - Uses 500 computer cores to maximize parallelization in the SFM pipeline
- Building Rome in a Cloudless Day (ECCV 2010)

Problem Statement

The goal is to develop a practical time implementation of Bundle Adjustment by exploiting all computing resources of the CPU and the GPU

What is Bundle Adjustment ?

Objective Function:

$$\min_{P,X} \sum_{i=1}^{n} \sum_{j=1}^{m} d\left(Q\left(P_j, X_i\right), x_{ij}\right)^2$$

$Q\left(P_j, X_i\right)$ is the predicted projection of point i on image j
$d(x, y)$ is the Euclidean distance between the image points represented by x and y

Minimizing this function is a sparse non linear least squares problem

What is Bundle Adjustment ?

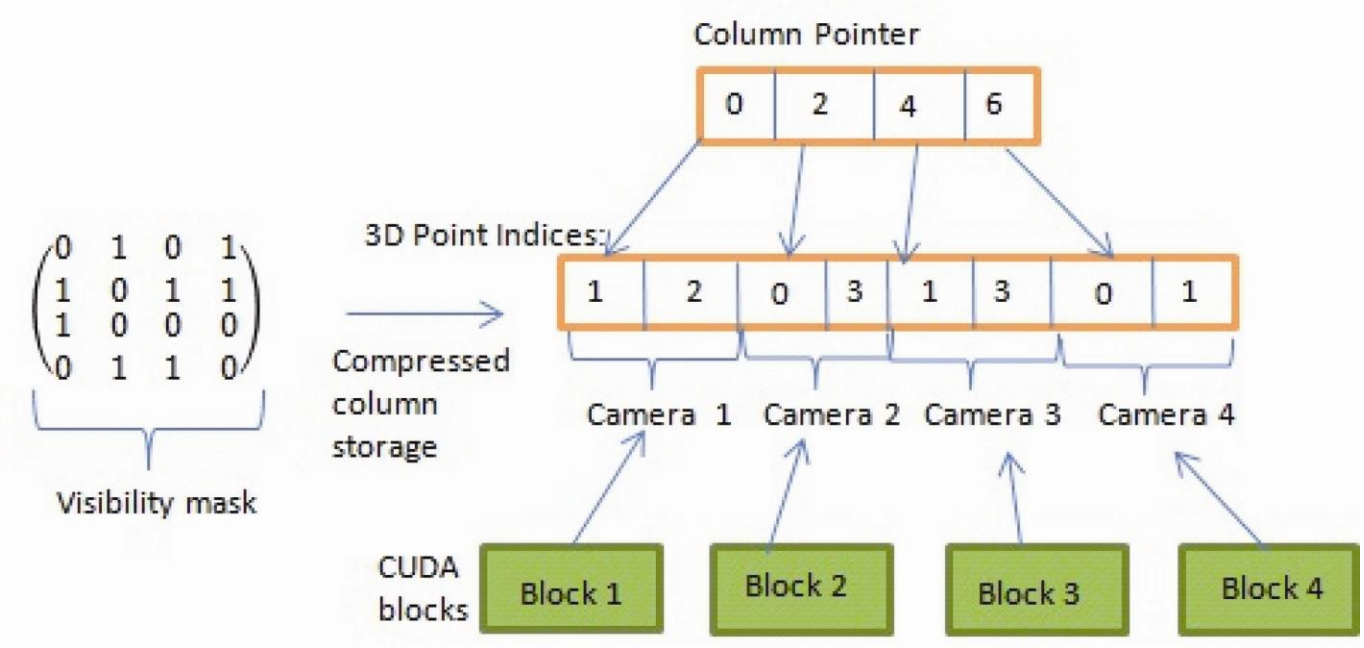
The non-linear least squares objective function is solved using LM Algorithm which is an interpolation of Gauss Newton and Gradient descent iteration.

The normal equation to be solved during each linear LM iteration is given as:

$$\left(\mathrm{J}^{\mathrm{T}}\Sigma_{\mathrm{X}}^{-1}\mathrm{J}+\mu\mathrm{I}\right)\delta=\mathrm{J}^{\mathrm{T}}\Sigma_{\mathrm{X}}^{-1}\epsilon\,,$$

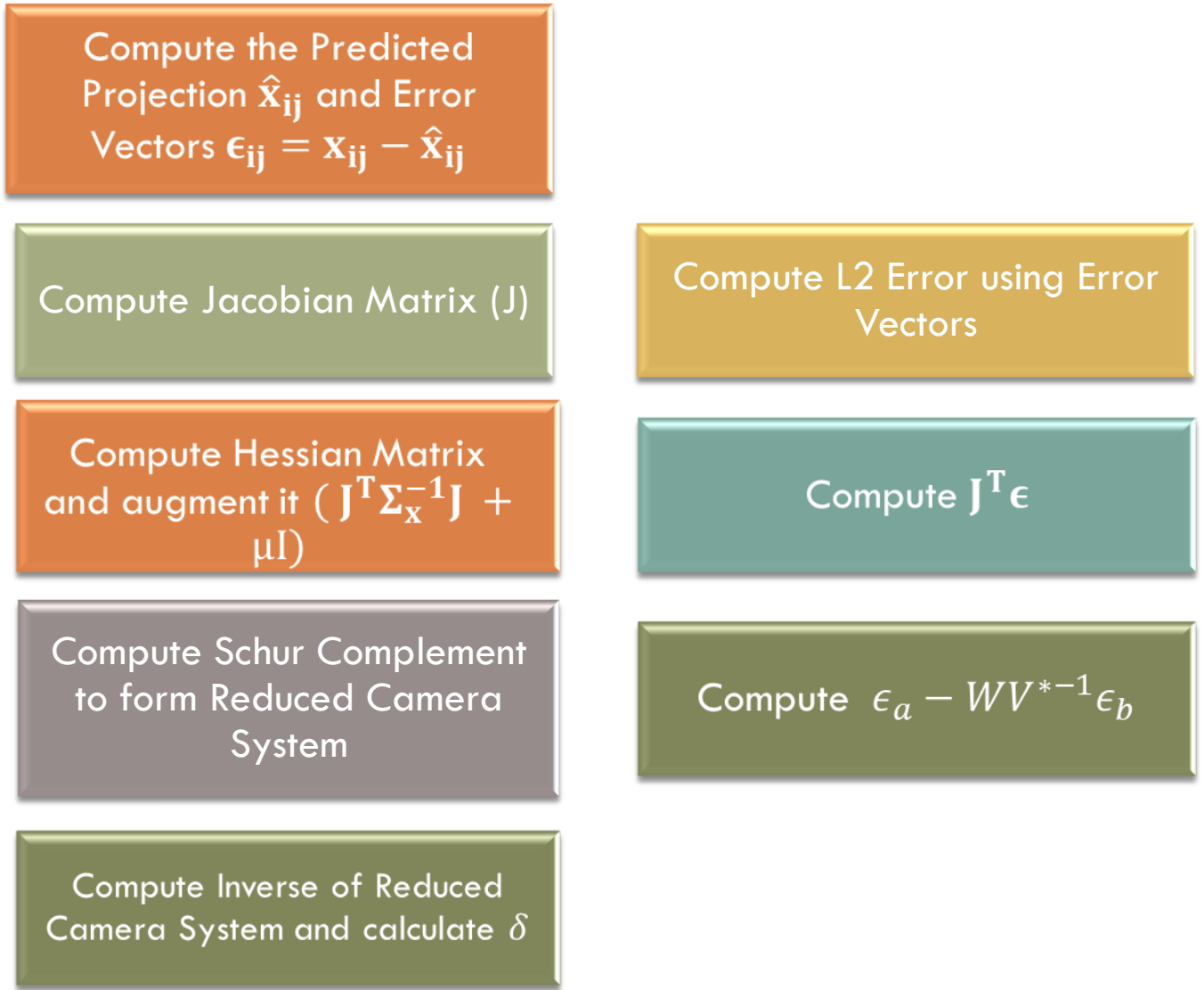
Where $\mathrm{J}=\frac{\partial \mathrm{X}}{\partial \mathrm{P}}$, $\epsilon=\mathrm{X}-\widehat{\mathrm{X}}$ and μ is the damping factor

Data Structure for the SBA

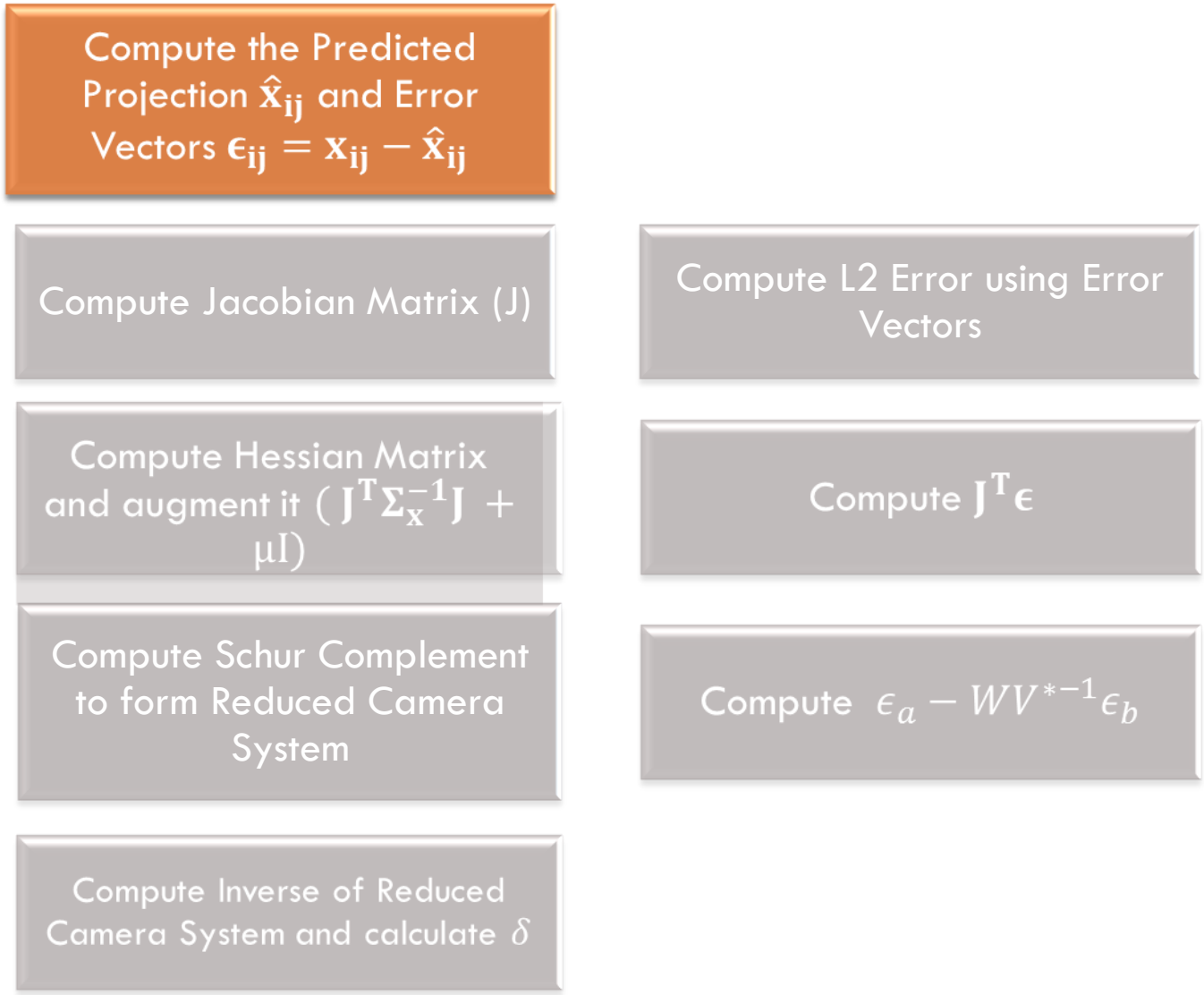


Compressed Column Storage of Visibility Mask having 4 Cameras and 4 3D Points. Each CUDA Block processes one set of 3D points.

Sparse Bundle Adjustment on the GPU

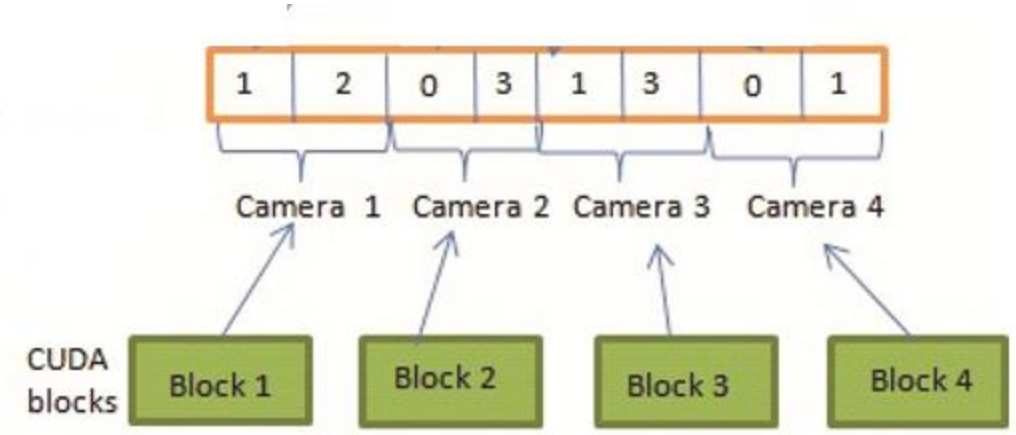


Sparse Bundle Adjustment on the GPU



Sparse Bundle Adjustment on the GPU

- Computation of the Predicted Projection and Error Vector
  - For *m* cameras, m blocks are launched, with block *j* computing projections corresponding to camera *j*.
  - The computation is limited by the number of registers available per block and a maximum limit of number of threads per block.



Sparse Bundle Adjustment on the GPU

Compute the Predicted Projection $\hat{\mathbf{x}}_{ij}$ and Error Vectors $\boldsymbol{\epsilon}_{ij} = \mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}$

Compute Jacobian Matrix (J)

Compute L2 Error using Error Vectors

Compute Hessian Matrix and augment it ($\mathbf{J}^{\mathbf{T}}\boldsymbol{\Sigma}_{\mathbf{x}}^{-1}\mathbf{J} + \mu\mathrm{I}$)

Compute $\mathbf{J}^{\mathbf{T}}\boldsymbol{\epsilon}$

Compute Schur Complement to form Reduced Camera System

Compute $\epsilon_a - WV^{*-1}\epsilon_b$

Compute Inverse of Reduced Camera System and calculate δ

Sparse Bundle Adjustment on the GPU

- Computation of the Jacobian Matrix and L2 Error
  - The Jacobian Matrix is calculated as $\frac{\partial X}{\partial P}$, with a grid structure similar to the computation of initial projections.

$$\begin{pmatrix} A_{10} & 0 & 0 & 0 & 0 & B_{10} & 0 & 0 \\ A_{20} & 0 & 0 & 0 & 0 & 0 & B_{20} & 0 \\ 0 & A_{01} & 0 & 0 & B_{01} & 0 & 0 & 0 \\ 0 & A_{31} & 0 & 0 & 0 & 0 & 0 & B_{31} \\ 0 & 0 & A_{12} & 0 & 0 & B_{12} & 0 & 0 \\ 0 & 0 & A_{32} & 0 & 0 & 0 & 0 & B_{32} \\ 0 & 0 & 0 & A_{03} & B_{03} & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{13} & 0 & B_{13} & 0 & 0 \end{pmatrix}$$

Compressed Column Storage

$$\mathbf{A}ij \leftarrow \frac{\partial \hat{x}_{ij}}{\partial a_j} = \frac{\partial \mathbf{Q}(a_j, b_i)}{\partial a_j} \left(\frac{\partial \hat{x}_{ij}}{\partial a_k} = 0 \; \forall i \neq k\right)$$

$$\mathbf{B}ij \leftarrow \frac{\partial \hat{x}_{ij}}{\partial b_i} = \frac{\partial \mathbf{Q}(a_j, b_i)}{\partial b_i} \left(\frac{\partial \hat{x}_{ij}}{\partial b_k} = 0 \; \forall j \neq k\right)$$

$$\mathbf{J} = (A_{10}, B_{10}, A_{20}, B_{20}, A_{01}, B_{01}, A_{31}, B_{31}, A_{12}, B_{12}, A_{32}, B_{32}, A_{03}, B_{03}, A_{13}, B_{13})$$

Block *j* computes the A_{ij} & B_{ij} submatrices corresponding the *j*th camera

Sparse Bundle Adjustment on the GPU

- Computation of the Jacobian Matrix and L2 Error
  - The Jacobian Matrix is calculated as $\frac{\partial X}{\partial P}$, with a grid structure similar to the computation of initial projections.
  - While the GPU is computing Jacobian Matrix, CPU computes the L2 Error using the Error vector

Sparse Bundle Adjustment on the GPU

Compute the Predicted Projection $\hat{\mathbf{x}}_{\mathbf{ij}}$ and Error Vectors $\boldsymbol{\epsilon}_{\mathbf{ij}} = \mathbf{x}_{\mathbf{ij}} - \hat{\mathbf{x}}_{\mathbf{ij}}$

Compute Jacobian Matrix (J)

Compute L2 Error using Error Vectors

Compute Hessian Matrix and augment it ($\mathbf{J}^{\mathbf{T}}\boldsymbol{\Sigma}_{\mathbf{x}}^{-\mathbf{1}}\mathbf{J} + \mu\mathrm{I}$)

Compute $\mathbf{J}^{\mathbf{T}}\boldsymbol{\epsilon}$

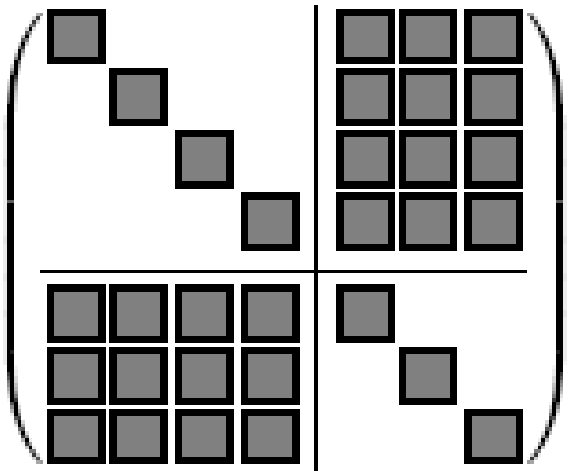
Compute Schur Complement to form Reduced Camera System

Compute $\epsilon_a - WV^{*-1}\epsilon_b$

Compute Inverse of Reduced Camera System and calculate δ

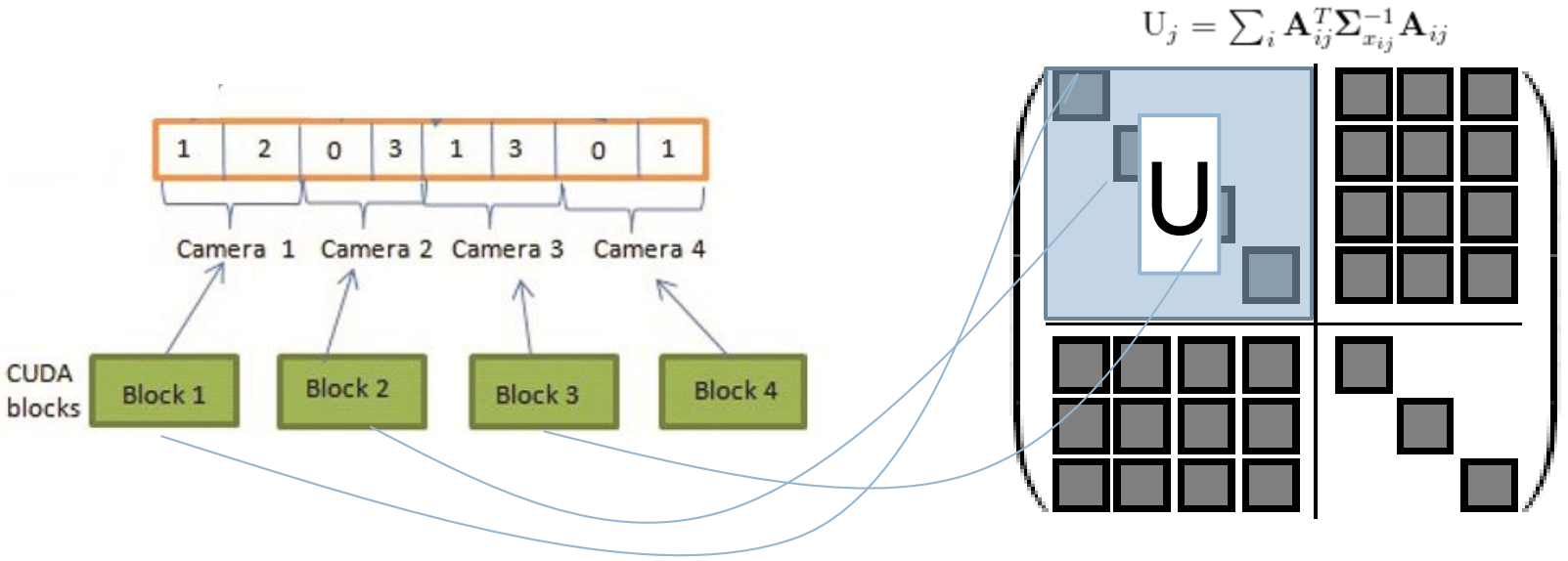
Sparse Bundle Adjustment on the GPU

- Computation of the Hessian Matrix and $J^T \epsilon$
  - The Hessian Matrix is computed using three different and independent kernels, two of which computes the diagonal sub-matrices and one of them computes the non diagonal sub-matrix



Sparse Bundle Adjustment on the GPU

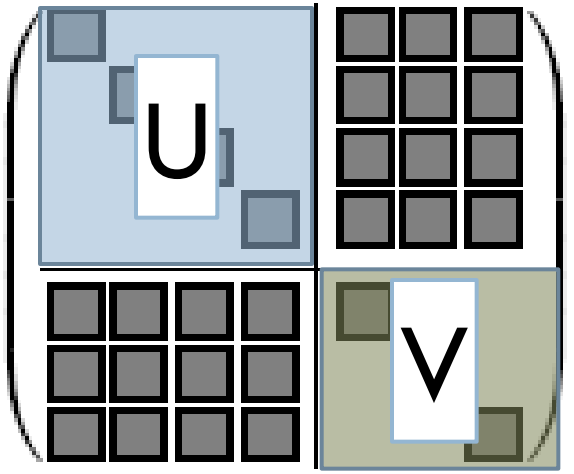
- Computation of U
  - The grid structure consists of m blocks with each block processing one Uj where j is the block id
  - Summation is done using segmented scan



Sparse Bundle Adjustment on the GPU

- Computation of V
  - Computation of V is done in a way similar to U, with each block computing Vi for the ith point

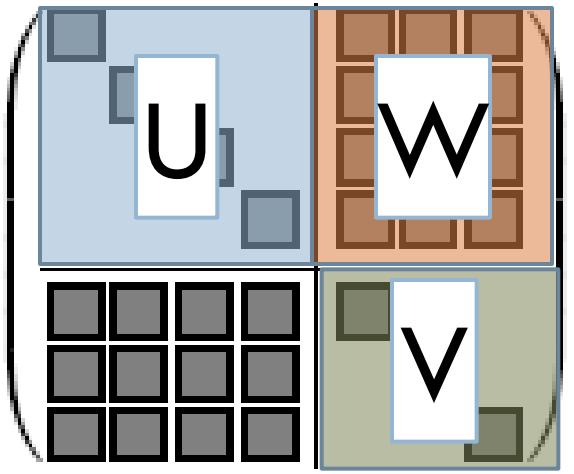
$$\mathrm{V}_i = \sum_j \mathbf{B}_{ij}^T \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$$



Sparse Bundle Adjustment on the GPU

- Computation of W
  - Computation of Ws are independent of each other
  - nnz/10 blocks are launched with each block computing 10 Ws

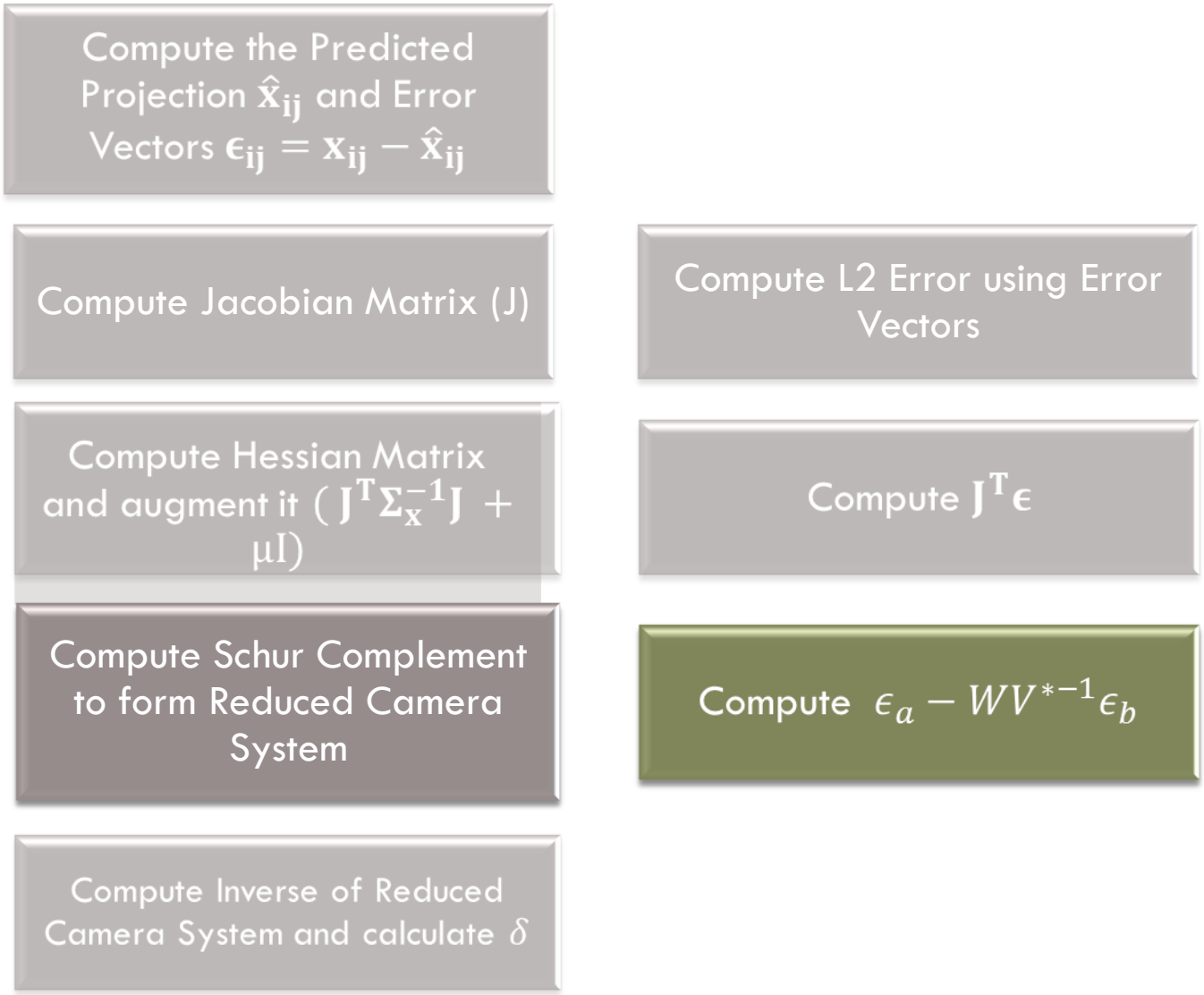
$$W_{ij} = \mathbf{A}_{ij}^{T} \mathbf{\Sigma}_{x_{ij}}^{-1} \mathbf{B}_{ij}$$



Sparse Bundle Adjustment on the GPU

- Computation of the Hessian Matrix and $J^T \epsilon$
  - The Hessian Matrix is computed using three different and independent kernels, two of which computes the diagonal sub-matrices and one of them computes the non diagonal sub-matrix
  - After U,V,W is calculated, the augmentation is done using another kernel, which calculates μ and add to the diagonal terms.
  - While the H matrix is computation is done on GPU, $J^T \epsilon$ computation is done on CPU.

Sparse Bundle Adjustment on the GPU



Sparse Bundle Adjustment on the GPU

- Computation of the Schur Complement to from the Reduced Camera System
  - It is the most computationally expensive step of all the modules.
  - S is a symmetric matrix. So, we calculate only the upper diagonal of S.
  - The CUDA grid structure consists of $m \times m$ blocks with each block computing a 9×9 sub-matrix in the upper diagonal
  - The computation is limited by the amount of shared memory available and number of registers available per block.

Sparse Bundle Adjustment on the GPU

Compute the Predicted Projection $\hat{\mathbf{x}}_{\mathbf{ij}}$ and Error Vectors $\boldsymbol{\epsilon}_{\mathbf{ij}} = \mathbf{x}_{\mathbf{ij}} - \hat{\mathbf{x}}_{\mathbf{ij}}$

Compute Jacobian Matrix (J)

Compute L2 Error using Error Vectors

Compute Hessian Matrix and augment it ($\mathbf{J}^{\mathbf{T}}\boldsymbol{\Sigma}_{\mathbf{x}}^{-\mathbf{1}}\mathbf{J} + \mu\mathrm{I}$)

Compute $\mathbf{J}^{\mathbf{T}}\boldsymbol{\epsilon}$

Compute Schur Complement to form Reduced Camera System

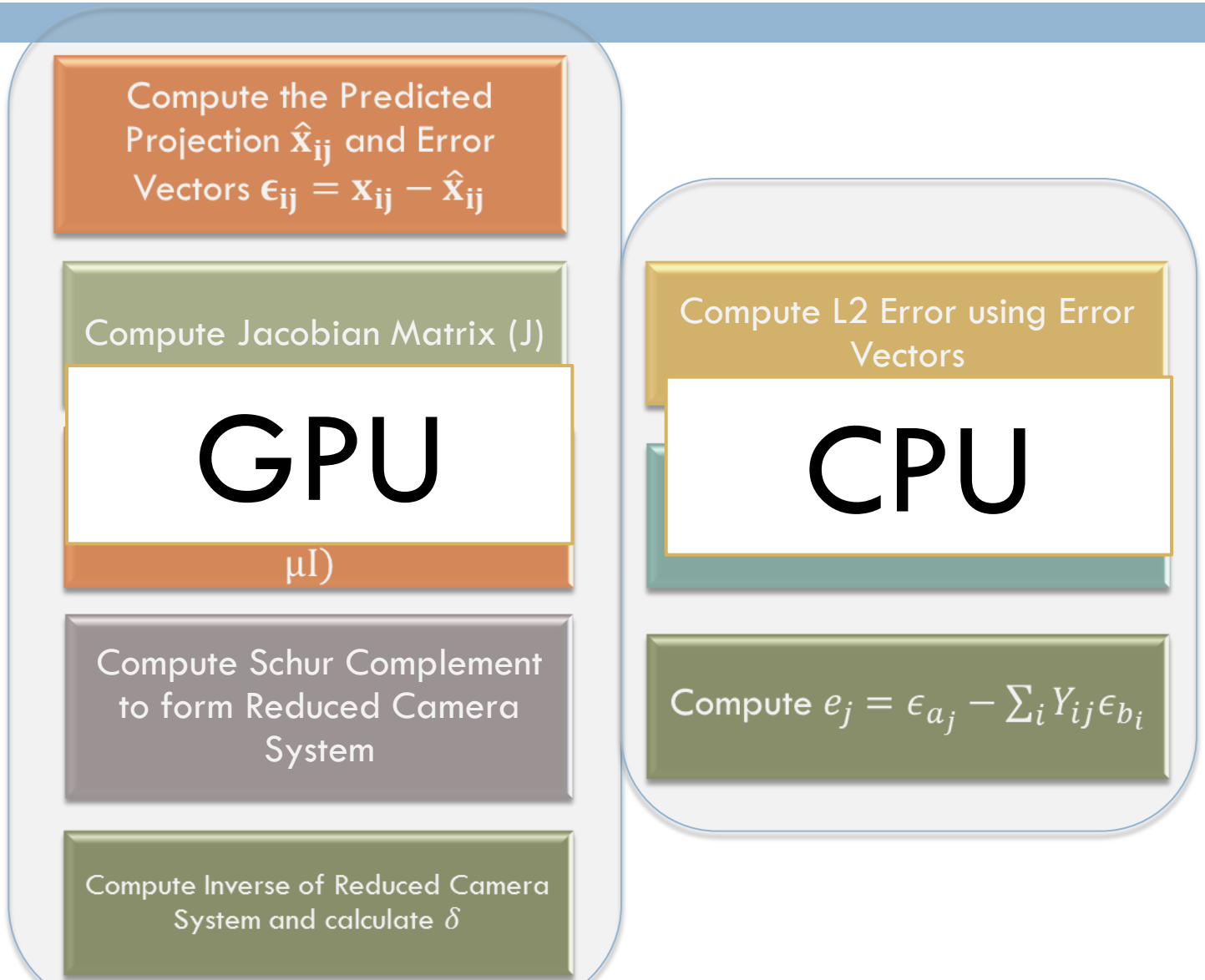
Compute $\epsilon_a - WV^{*-1}\epsilon_b$

Compute Inverse of Reduced Camera System and calculate δ

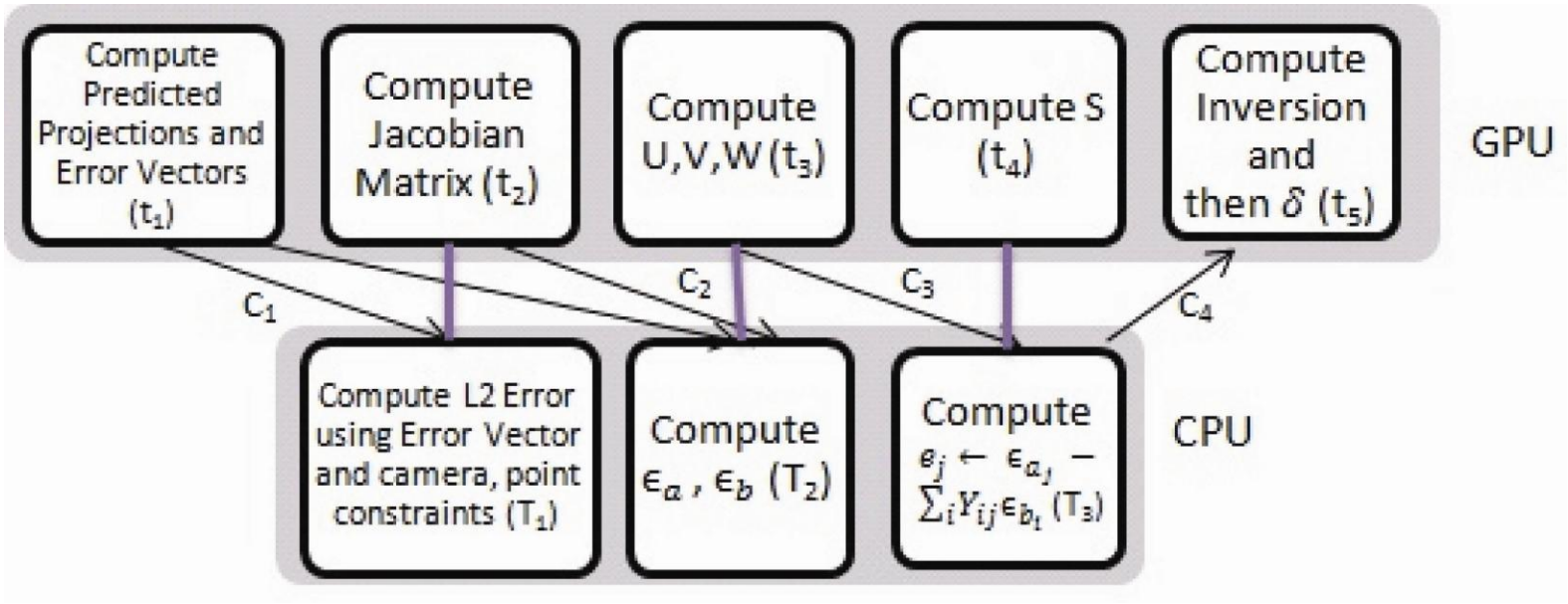
Sparse Bundle Adjustment on the GPU

- Computation of the Inverse of the Reduced Camera System and computation of δ_a
  - We use Cholesky Decomposition implemented in MAGMA library in order to compute the inverse.
  - It benefits from the hybrid computation by using both CPUs and GPUs and achieves a speedup of over 10 times than CPU.

Sparse Bundle Adjustment on the GPU

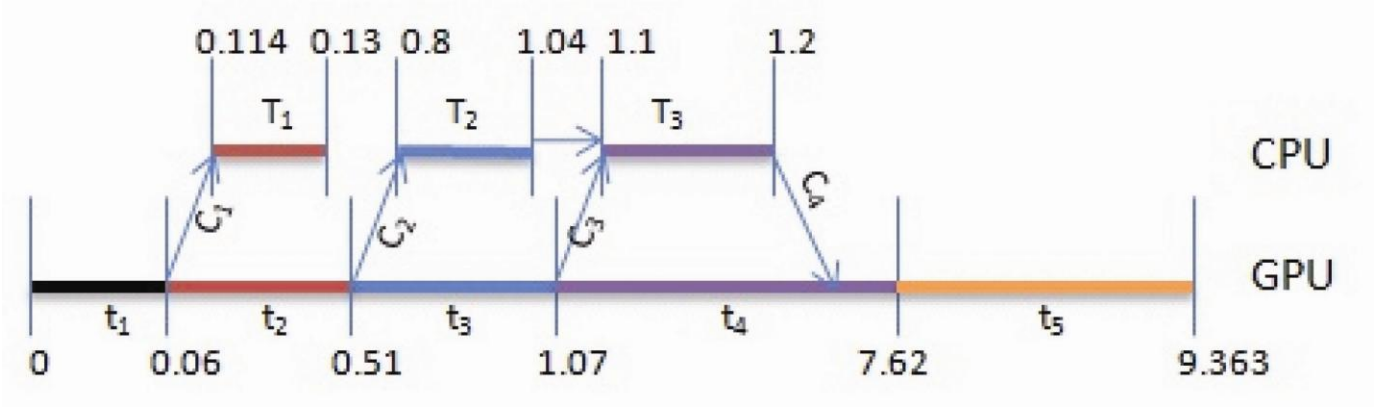


Hybrid Computation



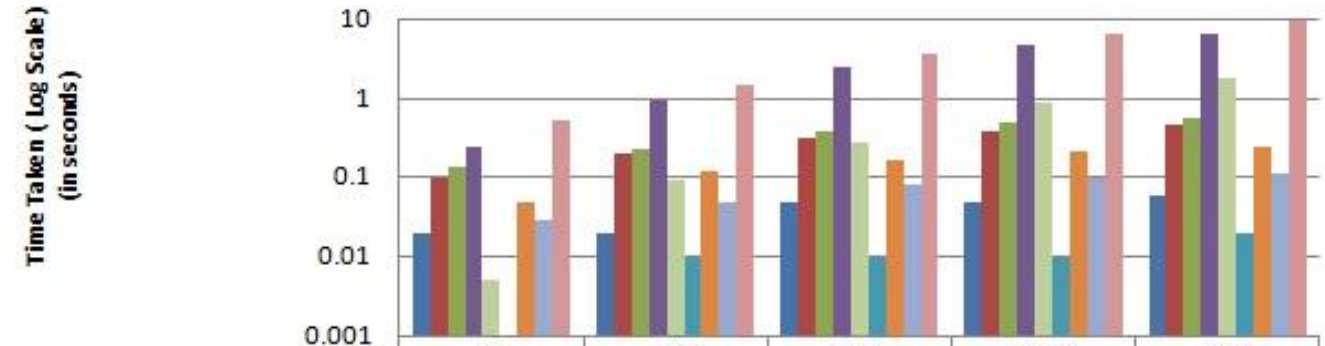
Computation blocks are efficiently scheduled either on CPU or GPU. Arrows show the data dependency between various modules on CPU and GPU. Modules connected through a vertical line are computed in parallel on CPU and GPU

Results and Analysis



Timings shown are the time taken by each component either on CPU or GPU including the memory transfer time in one iteration for 488 cameras.

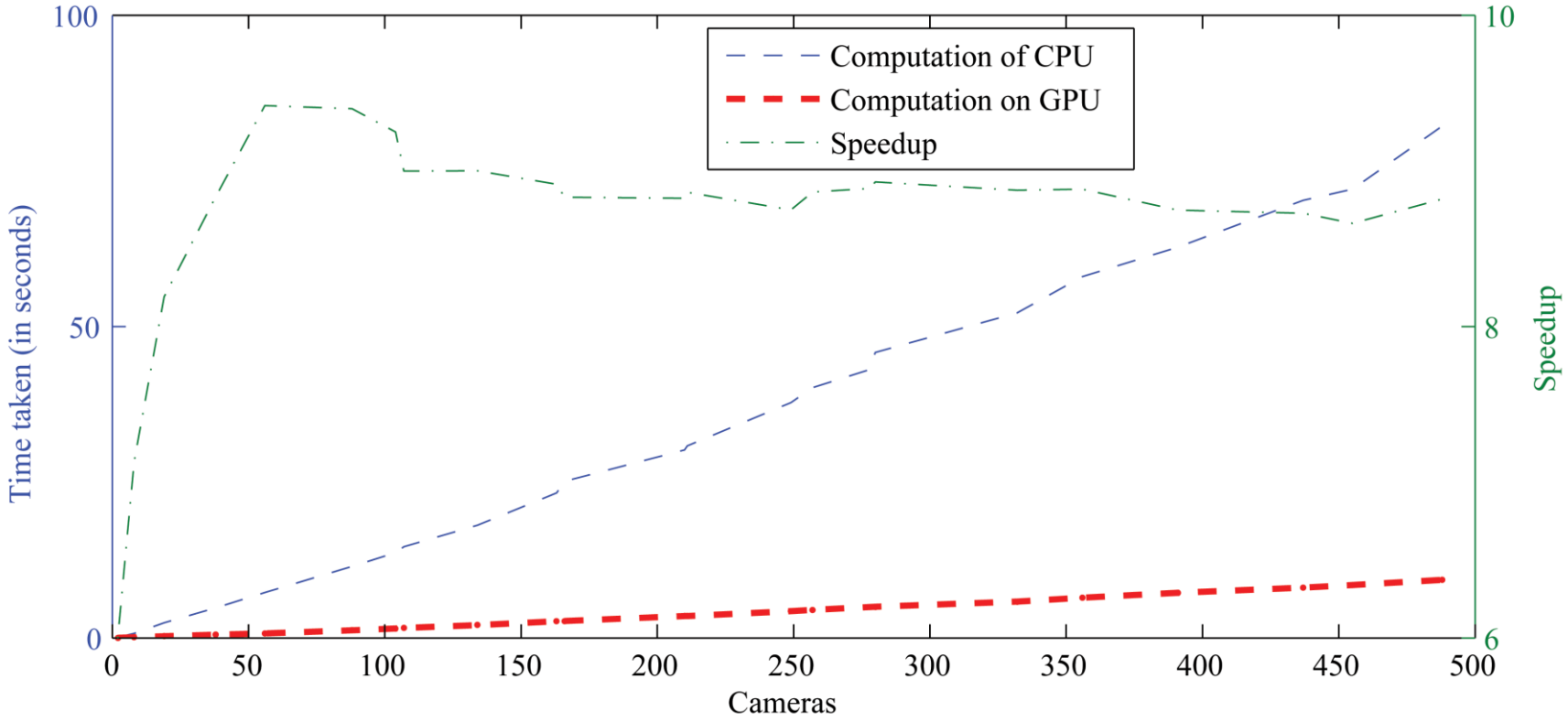
Results and Analysis



	38	104	210	356	488
GPU - Init Proj Computation	0.02	0.02	0.05	0.05	0.06
GPU - Jacobian Computation	0.1	0.2	0.32	0.39	0.45
GPU - U,V,W Computation	0.14	0.23	0.39	0.5	0.56
GPU - S Computation	0.25	0.97	2.5	4.63	6.55
GPU - Inverse Calculation	0.005	0.09	0.282	0.8675	1.743
CPU - L2 Error Computation	0	0.01	0.01	0.01	0.02
CPU - ϵa, ϵb Computation	0.05	0.12	0.17	0.21	0.24
CPU - e Computation	0.03	0.05	0.08	0.1	0.11
Total Time	0.515	1.51	3.542	6.4375	9.363

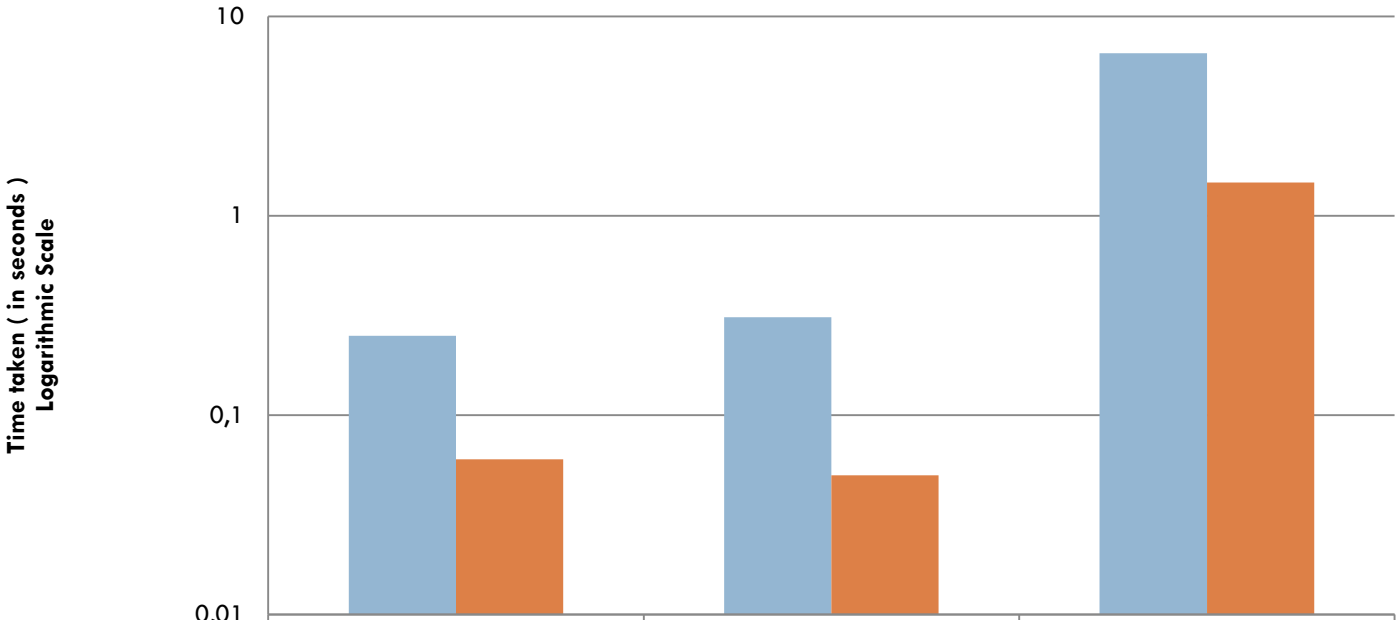
Time(sec) taken for each step in one iteration of Bundle Adjustment on GPU and CPU for various number of cameras. Total time is the time taken by hybrid implementation of BA using CPU and GPU in parallel

Results and Analysis



Time Comparison (one iteration) of Bundle Adjustment Computation on CPU and GPU

Preliminary Results on Fermi



	Jacobian Computation	UVW Computation	S Computation
S1070 Tesla GPU	0,25	0,31	6,53
GTX 480 Fermi GPU	0,06	0,05	1,47

Summary

- We introduced a hybrid algorithm using the GPU and the CPU to perform practical time bundle adjustment.

- We achieve a speedup of around 8 – 10 times over the CPU implementation on one quarter of Nvidia Tesla S1070 GPU

Future Work

- We are adapting our approach to the Fermi and expecting significant speedups on it.

- A multi GPU implementation is also being explored for faster overall processing

Thank You