Parallelized DeepFlow

Shiyu Huang, Hongxiang Qiu, Zeyu Zhao, Zongren Zou
{shiyuhuang, qiu, zeyu_zhao, z_zou}@g.harvard.edu

CS205: Computing Foundations for Computational Science

Spring 2018

Introduction

Optical flow describes the motion of objects in two consecutive images. Figure 1 below shows one of the gun smoke images and the visualization of their corresponding optical flow. Many algorithms are designed to estimate optical flow because of its importance in the computer vision area.

DeepFlow [1] is one of the state-of-the-art dense optical flow algorithm which has very high accuracy. The algorithm minimizes an energy function composed of the data term, the smoothness term and the matching term (shown in Figure 2). This optimization can be estimated by a **linear system** and the **original implementation uses SOR** (Successive Over Relaxation) method to solve the linear system. DeepFlow only has a serial implementation online and it's very slow especially if one want to process a video. DeepFlow spends most of the time solving the aforementioned linear system and our project is to speed up DeepFlow by parallelizing its linear system solver.

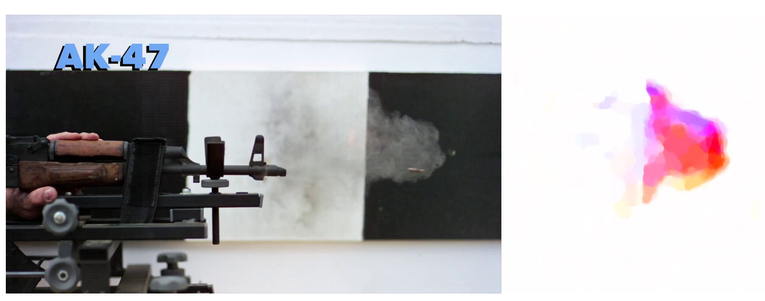

Figure 1. Gun Smoke and Corresponding Optical Flow

$$E(\boldsymbol{w}) = \int_{\Omega} E_D + \alpha E_S + \beta E_M \boldsymbol{dx}$$

$$E_D = \delta\Psi\left(\sum_{i=1}^{c} \boldsymbol{w}^\top \bar{\boldsymbol{J}}_0^i \boldsymbol{w}\right) + \gamma\Psi\left(\sum_{i=1}^{c} \boldsymbol{w}^\top \bar{\boldsymbol{J}}_{xy}^i \boldsymbol{w}\right)$$

$$E_M = c\phi\Psi(\|\boldsymbol{w} - \boldsymbol{w}'\|^2) \qquad E_S = \Psi\left(\|\nabla u\|^2 + \|\nabla v\|^2\right)$$

Figure 2. Energy Function and Terms

Data and Application

DeepFlow can be used to generate flows for any two consecutive images or any videos. In our experiments, we tried DeepFlow both on two images and on videos. For videos, we generate two-way flows (forward and backward) for each pair of the frames since our application examples require them.

DeepFlow has many applications. In our project, we focus on its applications on video processing. Our project website [2] shows a video stylization example and a fake slow motion video generation example. Both of them are based on the flows of the input videos and therefore can benefit from our parallelized implementation of DeepFlow. Our source code is publicly accessible on GitHub [3].

SOR, Jacobi and RBSOR

SOR, Jacobi and Red-Black SOR (RBSOR) are all iterative methods to numerically solve a linear system. Figure 3 shows a visualization of those three methods. In each iteration, we update all cells once with their numbers as the order. We are updating the black cell, the calculation depends on the **last iteration values of the blue cells** and **current iteration values of the red cells**.

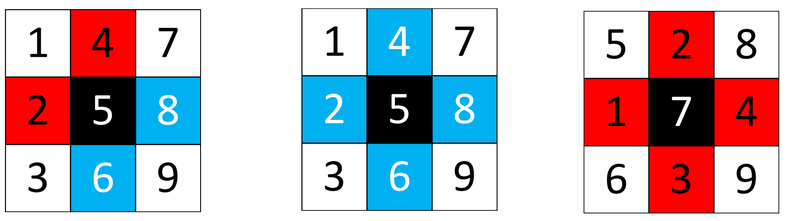

Figure 3. Visualization. Left to right: SOR, Jacobi, RBSOR

We now want to compute multiple cells together (parallelization) in one step. Clearly, **SOR is not parallelizable** due to its serial dependency while Jacobi is pretty simple to parallelize. RBSOR can be parallelized by dividing a step into 2. We can first calculate all red cells together and then all blue cells (and the black cell) together. **RBSOR converges better than Jacobi and we use RBSOR in our final models.**

MapReduce + OpenMP

Intuitively, in the video processing, the **flows for all pairs of frames are independent**. We use MapReduce to assign **frame pairs to different nodes** and each node will use DeepFlow (OpenMP implementation) to compute both **the forward and backward flows simultaneously** (higher throughput) of the assigned pairs.

OpenMP and OpenACC

As shown in Figure 4, the idea behind OpenMP and OpenACC is that we can calculate red cells (odd cells) together and then blue cells (even cells) together since cells with the same color are all independent. For OpenACC, data is only copied to GPU once and copied out after the last iteration.

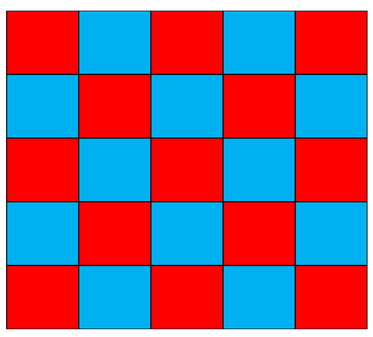

Figure 4. Shared-memory Parallelization

Results

We report execution time (**seconds**) for DeepFlow on two 1080p images and 1 second long 1080p video.

	Image	Video
Original Implementation	42.582	2129
OpenMP (4 threads)	19.730	1004
OpenACC	28.385	1414
MPI (4 processes)	102.224	Not Tested
MapReduce (8 workers)	N/A	147

OpenACC and MPI results are bad due to their big overheads. But OpenMP and MapReduce perform well.

MPI

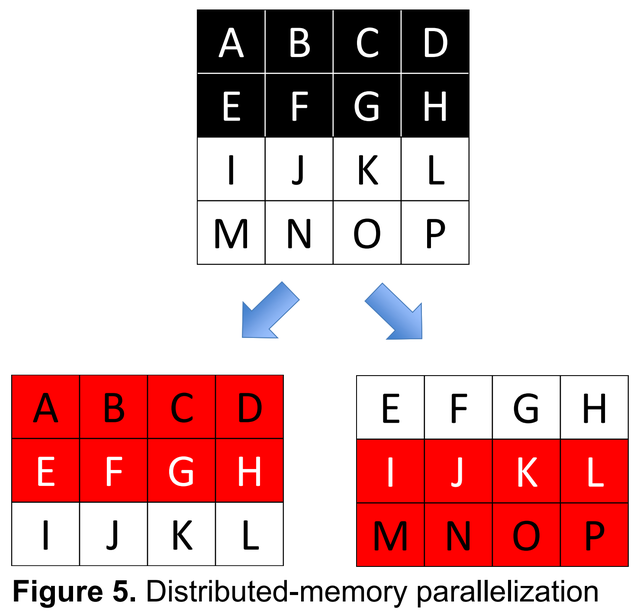

Figure 5. Distributed-memory parallelization

Figure 5 above shows the idea behind our MPI implementation. The original matrix is distributed to 2 worker processes and each process **only updates the red cells**. After each iteration, each process will **send results of the white letters** to other processes and **receive new white cell results** from other processes.

Citations and Links

[1] Weinzaepfel, P., Revaud, J., Harchaoui, Z., & Schmid, C. (2013). DeepFlow: Large displacement optical flow with deep matching. *IEEE International Conference on Computer Vision (ICCV)*, pp. 1385-1392.

[2] Website: **https://zeruniverse.github.io/CS205-project**

[3] Code: **https://github.com/zeruniverse/CS205-project**