An Energy Efficient Time-sharing Pyramid Pipeline for Multi-resolution Computer Vision

Qiuling Zhu*, Navjot Garg†, Yun-Ta Tsai†, Kari Pulli†
*Dept. of Electrical and Comp. Eng., Carnegie Mellon University, Pittsburgh, PA, USA
†NVIDIA Research, Santa Clara, CA, USA
Email: qiulingz@andrew.cmu.edu, ytsai@nvidia.com

***Abstract*—We introduce an energy efficient time-sharing pyramid pipeline architecture designed for multi-resolution image analysis in mobile computer vision. The time-sharing pipeline efficiently reduces the off-chip memory traffic by re-organizing the data storage and processing order of an image pyramid. We build a parameterized image pyramid hardware generator and successfully evaluate the overall pyramid design space. Our results demonstrate that the time-sharing pyramid pipeline achieves about 50% of hardware savings in terms of area and power consumption compared to the traditional linear pipeline implementation. We also implement the multi-resolution Lucas-Kanade optical flow algorithm on the time-sharing pipeline, and demonstrate an order of magnitude savings in off-chip memory traffic and system energy consumption.**

***Index Terms*—time-sharing; pipeline; image pyramid; multi-resolution computer vision; optical flow**

I. Introduction

Mobile camera applications, from computational photography to computer vision, require efficient processing of large amounts of data, and can greatly benefit from hardware acceleration. The overall performance and energy use is often limited by the memory bandwidth [1], [2], [3], [4]. This problem has been addressed by multiresolution processing [5].

Image pyramid is the basic structure for multi-resolution image processing, and it provides a complete image representation that supports fast search and direct access to pixels at different image resolutions [6], [7], [3]. More importantly, the image pyramid also provides a hierarchical framework for efficient implementation of multi-resolution algorithms (e.g., motion and stereo analysis). In image pyramid processing, the pixels of an image in the pyramid are recursively processed and up-sampled or down-sampled to create an increasingly finer or coarser image for analysis.

This paper proposes a time-sharing pipeline that exploits the special pyramid access pattern, and minimizes the off-chip memory traffic by streaming the intermediate data storage only through an on-chip buffer. This pipeline architecture presents superior power and energy efficiency, and it is also configurable and general enough, both for constructing all types of basic pyramid structures (e.g., Gaussian pyramid) and performing various multi-resolution vision tasks (e.g., hierarchical optical flow).

To achieve this, our time-sharing pyramid pipeline architecture combines the advantages of conventionally used linear and segment pyramid pipelines and performs the multi-resolution tasks at different pyramid levels in a particular spatial and temporal order. The time-sharing pyramid pipeline is designed for a specialized front-end computer vision processor with the primary goal of improving the energy efficiency. It implements only one copy of the processing element (PE) for an image pyramid and runs it at full clock rate for the computations of data at all pyramid levels, in a time-sharing manner. This allows exploiting the special pyramid access pattern and preserving good spatial and temporal data locality throughout the pyramid processing so that the active working dataset is small and can be temporarily buffered on-chip while streaming through the linebuffer of the pyramid, eliminating all unnecessary off-chip memory traffic. We demonstrate the application of the pipeline architecture in the implementation of multi-resolution Lucas-Kanade optical flow and how our design yields superior hardware efficiency and substantial energy savings compared to the traditional designs.

Related work Pyramid architectures for computer vision engines [4], [8], [9], [10] can be divided into two categories: segment pipeline (i.e., *SP*) and linear pipeline (i.e., *LP*) [7], [5]. In the *SP* architecture, one processing element (PE) works for all the pyramid levels, one level after another. As the on-chip memory is not able to hold a whole pyramid level, the PE writes the results of computing one pyramid level to the main memory. To start the computation for the next level, the results are read again from the main memory again. Therefore, a segment pipeline results in heavy memory traffic as it requires reading and writing each level of the image pyramid from and to the main memory. On the other hand, in the *LP* architecture, the PE is duplicated for every pyramid level and all the PEs work in parallel for all the pyramid levels simultaneously. A certain amount of SRAM linebuffers are assigned to each PE for storing the latest working data set at each level. The intermediate data just streams through the local memory linebuffer, eliminating the unnecessary data access to/from the main memory. However, the inefficient usage of the computational resources due to the unbalanced workloads among the PEs at different pyramid levels is a severe problem with this architecture.

II. Time-sharing Pipeline Pyramid Architecture

We propose an efficient time-sharing pyramid pipeline (i.e., *TP*), shown in Fig. 1 (a). It combines the good properties of the linear and segment pipeline pyramid architectures while

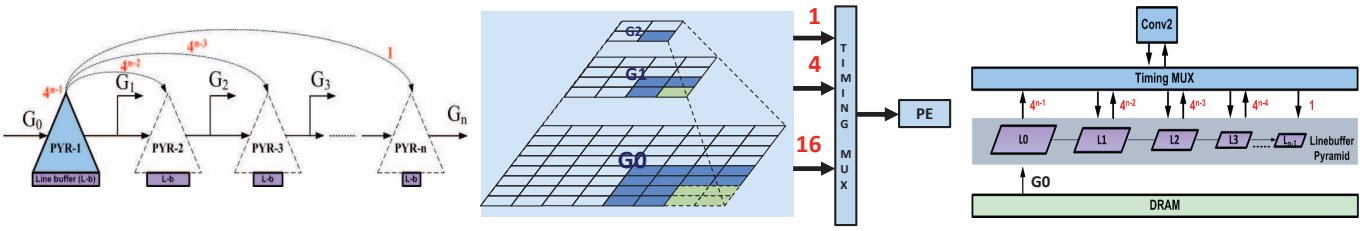

Fig. 1. (a) Time-sharing Pyramid Pipeline Illustration (b) Time-sharing Pyramid Pipeline Workload Distribution (c) Time-sharing Pyramid Pipeline Architecture

avoiding their shortcomings. That is, it implements physically only a single copy of each PE, but logically it processes all the pyramid levels in parallel, in a time-sharing manner.

Fig. 1 (b) shows the *TP* working mechanism on a partial Gaussian pyramid. Assuming that the resolution scaling ratio is four (two for both x and y), we first divide the overall work set of a pyramid into multiple work units. We highlight one of the work units in Fig. 1 (b) which comprises one pixel at level *G2*, four pixels at *G1* and 16 pixels at *G0*. The PE will process one work unit after another. In each work unit, the PE can process from coarse-to-fine, or from fine-to-coarse. In either case, it spends one time slot to process one pixel at the coarsest level, and four times more time processing four times more pixels at each successive pyramid level. After finishing processing all the pixels in one work unit, the PE starts to process the next work unit. The red numbers shown in the figures represent the relative number of time slots spent at each level. In this way, the time-sharing pipeline has a single PE working for all pyramid levels in a time-sharing pipeline manner. A control block (i.e., *timing MUX*) sits between the PE and the pyramid memory and is responsible for scheduling work of the single PE.

Fig. 1 (c) shows the overall architecture of a time-sharing Gaussian pyramid pipeline where the processing element is a simple convolution engine. For a more complicated multi-resolution pyramid system, the convolution engine can be replaced with other processing elements. We implement only one convolution engine that is connected to linebuffer SRAM arrays through the *Timing MUX* control block. The linebuffers at the coarser pyramid levels have smaller sizes than those of the finer pyramid levels, resulting in a *linebuffer pyramid*. The linebuffer pyramid holds the most recent data, which is a small subset of the whole image pyramid. The *Timing MUX* schedules how the processing elements access data from one level of the linebuffer pyramid and write the results into the next level. The design only needs to access the source image (*G0*) from the main memory. The intermediate data (e.g., the pixels in the middle levels of the Gaussian pyramids) are just temporarily buffered in the linebuffer pyramid, from which it can be requested by other processing elements in the system.

III. Multi-resolution Optical Flow

We next map the time-sharing pipeline to a real multi-resolution computer vision application. The measurement of optical flow is a crucial and challenging task in processing of video sequences, and its hardware acceleration has been regarded as critically important for low-power real-time computer vision systems [11]. We will demonstrate that the Lucas-Kanade (L-K) algorithm [12], [13], [14], an optical flow method with a relatively low computational hardware overhead and reasonably good accuracy, can be efficiently implemented in the proposed time-sharing pipeline.

The hierarchical L-K optical flow estimation proceeds from coarse to fine pyramid levels and has a very regular pyramid structure. As shown in Fig. 2, two image pyramids are first constructed for the two consecutive source images A and B. The motion estimations are first performed at the coarsest level of the image pyramids and the estimated motion vectors (velocities) are upsampled and used as the seed to be refined at the next finer pyramid level. To achieve this, each pixel of image A maintains a memory address from the upsampled velocity results as the starting point, and then the motion estimation is performed at each pixel between the current level of the updated images A and B. The updating process is called "warp" as the memory address is offset by the previous velocity seed. This process is iterated until the finest level, and it yields the optical flow between the two images.

While the Gaussian pyramids of two image frames are first constructed from the finest scale to the coarsest scale, the motions between the two images frames are estimated starting from the coarsest scale back to the finest scale. The sequential processing order significantly delays the processing. However, in the time-sharing pipeline implementation (Fig. 2), we can perform the Gaussian pyramid construction and motion estimation simultaneously. To achieve this, the processing elements (e.g., two downsampling filters, one motion estimation (ME) engine and another image warping block) are connected to three linebuffer pyramids through different *Timing MUX* blocks. The down-sampling filters first start to work in the time-sharing manner to construct the Gaussian pyramids. As long as there are enough pixels computed at the coarsest scale, the motion estimation engine will start to work and the motion refinement operation propagates gradually from the coarsest scale to the finest scale in the time-sharing manner. Meanwhile, the two downsampling filters continuously generate new pixels at all scales of pyramids to feed the motion estimation engines. The resulting architecture significantly reduces the latency of the whole process.

Besides the two fine-to-coarse linebuffer pyramids, there is another coarse-to-fine linebuffer pyramid for the storage of the motion vectors at each pyramid resolution due to the feedback loop of the algorithm. Our time-sharing pipeline only needs

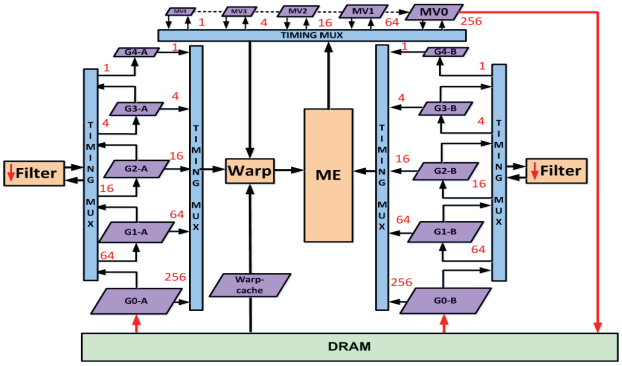

Fig. 2. Time-sharing Pyramid Pipeline Implementing L-K Optical Flow

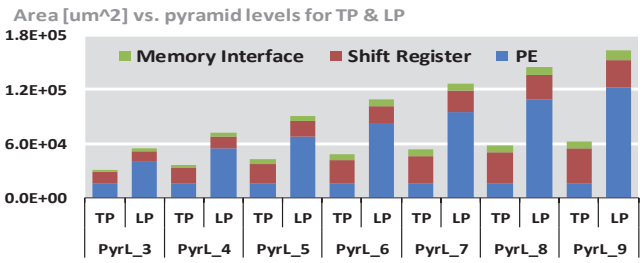

Fig. 3. Area Comparison with LP

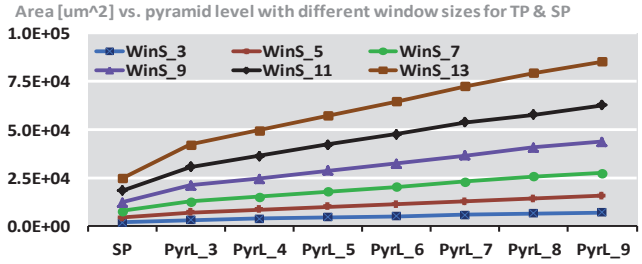

Fig. 4. Area Comparison with SP

to read the two source images from the main memory, and write the resulting motion vectors back to the memory. The intermediate results, e.g., the two image Gaussian pyramids and the intermediate motion vectors at intermediate levels, just stream through the linebuffer and are consumed in the processing feedback loop. A segment pipeline, on the other hand, would have to store the intermediate results back to DRAM, and then access them again when the computation proceeds to the next level.

IV. Implementation, Evaluation and Results

In this section, we implement the prototype hardware and model all the three discussed pyramid pipelines. We will show that the proposed time-sharing pyramid pipeline demonstrates superior hardware efficiency compared to linear pyramid pipeline, and significant memory bandwidth and energy savings compared to segment pipeline.

A. Hardware Cost Evaluation

As the time-sharing pyramid pipeline is a fundamental structure in a low-power computer vision SoC engine, its area and power requirements need to be as low as possible [15]. The pyramid engine is composed of four major parts: the processing engine logic PE, the window registers to hold the current working windows, the linebuffer SRAM arrays, and the control logic including *Timing MUX*. In order to efficiently explore the design space, we use the *Genesis2* design tool to build the image pyramid engine chip generator that encapsulates all the possible combinations of the design parameters[16], from which the optimized synthesizable hardware for a specific design point can be automatically generated for fast evaluation. We also implement the corresponding segment and linear pipelines for comparison. Area and power are measured from the automated physical synthesis of the designs on a commercial 32 nm CMOS process at 500 *MHz* using Synopsys synthesis tools. Below, we use *PyrL*, *WinS*, *ImSize* to denote the number of pyramid levels, the window size, and image width.

In Fig. 3, we break down the area consumption of *TP* and compare it to the corresponding *LP* designs for constructing the $1080p$ image pyramids. We see that *TP* consumes much less PE cost due to the time-sharing of the same PE among different pyramid levels. Although it consumes slightly more shift-register and controlling logic for accommodating the time-sharing configurations, that are negligible compared with the reduction of the PE cost. Therefore, *TP* significantly saves area cost over *LP*, but maintains the minimum off-chip memory bandwidth requirements as we will explain below.

It is also important to understand the extra hardware overhead of *TP* compared to *SP*. We plot the areas of *SP* for different window sizes on the left in Fig. 4. For each window size, we also plot the corresponding areas of *TP* with different pyramid levels. As expected, *TP* consumes increasingly more area compared to *SP* as the pyramid levels grow. On the other hand, we see that the slope of the curves is proportional with the window size, and the bottom curve which corresponds to the window size of three is almost flat. That implies that the overhead of *TP* over *SP* is fairly small for designs with small windows. As *TP* consumes the same amount of area as the PE with *SP*, the real extra cost actually comes from the linebuffers, shift registers, and their control logic.

B. Performance and Energy Evaluation

We next evaluate the performance and energy efficiency of the proposed pipeline. In Fig. 5 (a) we demonstrate the latency of performing optical flow estimation for one pair of image frames. We see that *TP* is almost two times faster than *SP* as it allows performing the pyramid construction and motion estimation in parallel (see Section III). On the other hand, *TP* is only slightly slower than *LP*, while it eliminates the duplication of the processing elements to save area and energy.

To understand the memory traffic savings, we implement the L-K optical flow based on the time-sharing pipeline and collect the memory traffic statistics. In Fig. 5 (b), we break down the L-K DRAM traffic statistics for all the *SP*, *TP* and *LP* designs. Basically there are two different types of data traffic in the optical flow implementation: the traffic for the movement of the image pixels and the calculated motion vectors at all pyramid levels. The movement of the image

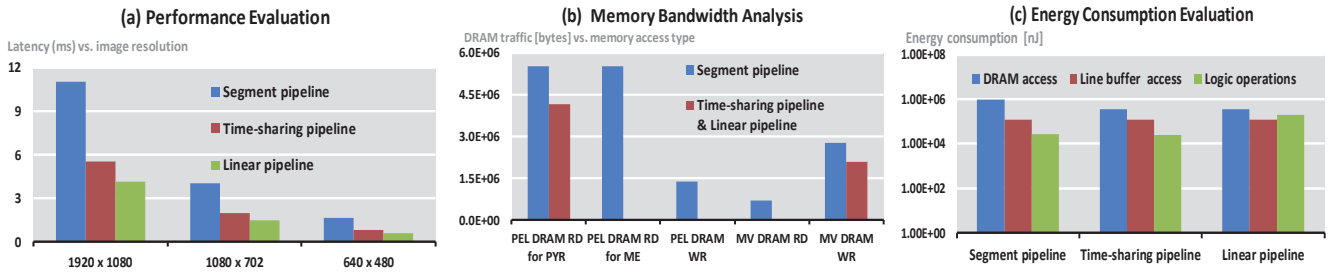

Fig. 5. Performance and Energy Evaluation

pixels has two different purposes: the construction the image pyramids and the motion estimation. The plots show that the *SP* implementation reads the two image pyramids from the DRAM twice, first for the Gaussian pyramid construction (*PEL_RD_FOR_PYR*) and then for motion estimation (*PEL_RD_FOR_ME*). It also requires memory bandwidth to write the constructed Gaussian pyramids back to the DRAM (*PEL_DRAM_WR*). *SP* also needs to write the motion vectors at all pyramid levels to the DRAM (*MV_DRAM_WR*) and read them again (*MV_DRAM_RD*) for refining the motion vectors at successive pyramid levels. In the time-sharing and linear pipeline, however, the memory traffic is reduced to only accessing the two source images from the DRAM, and to returning the resulting motion vectors back to the DRAM. All other intermediate memory traffic is completely eliminated.

Fig. 5 (c) shows the energy consumption evaluation results. We see that the energy consumption of the pipeline systems are dominated by DRAM accesses. Compared to *SP*, our *TP* achieves almost an order of magnitude of energy savings on DRAM access (note the logarithmic scale), while consuming similar amount of energy as *SP* for linebuffer (SRAM) access and logic operations. The *LP* design is as efficient as *TP* for memory performance, however, it consumes more energy on the logic processing elements.

Fig. 6 presents the calculated optical flow (velocity) on a benchmark image with a left-to-right movement. The proposed *TP*-based implementation produces the same motion vectors as the *SP*-based implementation, validating the approach.

V. Conclusion

The real-time and low-power demands of implementing computer vision applications on a mobile device are often met with application specific hardware accelerators. The image pyramid data structure that provides a hierarchical framework to implement multi-resolution algorithms amplifies such demands. A specialized time-sharing pipeline architecture that overcomes the inefficiencies of the conventional segment and linear pipeline architectures has been developed to provide the required efficiencies. Our hardware evaluation for 32nm CMOS process shows significant improvements in both area and power. Moreover, we present opportunities of exploiting the time-sharing pipeline to eliminate off-chip memory traffic, resulting in energy savings for performing a real-time multi-resolution computer vision task. This paper demonstrates that a specialized pipeline architecture that exploits the inherent pyramid data access pattern can achieve superior area, power, and energy efficiency of the SOC design for low-power multi-resolution image processing.

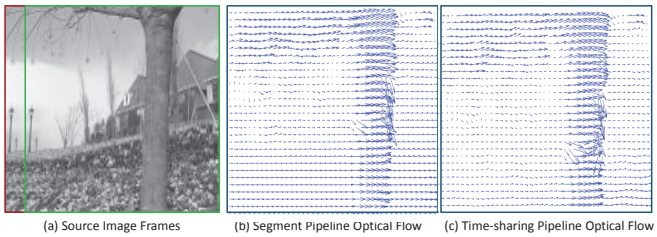

Fig. 6. Optical Flow Results

References

[1] G. P. Stein, E. Rushinek, G. Hayun, and A. Shashua, "A computer vision system on a chip: a case study from the automotive domain," *IEEE Computer Vision and Pattern Recognition*, 2005.

[2] S. Muramatsu, Y. Otsuka, H. Takenaga, Y. Kobayashi, I. Furusawa, and T. Monji, "Image processing device for automotive vision systems," *IEEE Intelligent Vehicle Symposium*, 2002.

[3] P. Burt, "A pyramid-based front-end processor for dynamic vision applications," *Proceedings of the IEEE*, vol. 90, no. 7, 2002.

[4] G. van der Wal, M. W. Hansen, and M. R. Piacentino, "The Acadia vision processor," in *CAMP*, 2000.

[5] P. Burt and G. van der Wal, "An architecture for multiresolution, focal, image analysis," in *Pattern Recognition*, vol. 2, 1990.

[6] D. C. Zhang, "Method of image fusion and enhancement using mask pyramid," *FUSION*, 2011.

[7] G. S. van der Wal and P. J. Burt, "A VLSI pyramid chip for multiresolution image analysis," *International Journal of Computer Vision*, vol. 8, no. 3, 1992.

[8] E. Rushinek and P. Del Vecchio, "Multi-threaded design tackles SOC performance bottlenecks," *EmbeddedSystems Europe*, 2006.

[9] S. Kyo and S. Okazak, "In-vehicle vision processors for driver assistance systems," in *ASPDAC*, 2008.

[10] A. Techmer, "Application development of camera-based driver assistance systems on a programmable multi-processor architecture," in *IEEE Intelligent Vehicles Symposium*, 2007.

[11] F. Barranco, M. Tomasi, J. Diaz, M. Vanegas, and E. Ros, "Parallel architecture for hierarchical optical flow estimation based on FPGA," *IEEE Transactions on VLSI systems*, vol. 20, no. 6, 2012.

[12] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th international joint conference on Artificial intelligence*, 1981.

[13] V. Mahalingam, K. Bhattacharya, N. Ranganathan, H. Chakravarthula, R. R. Murphy, and K. S. Pratt, "A VLSI architecture and algorithm for Lucas-Kanade-based optical flow computation," *IEEE Transactions on VLSI systems*, vol. 18, no. 1, 2010.

[14] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, 2001.

[15] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation," *International Conference on Embedded Computer Systems*, 2010.

[16] O. Shacham., "Chip multiprocessor generator: automatic generation of custom and heterogeneous compute platforms," *PhD Thesis, Stanford*, 2011.