EUCLIDEAN DISTANCE TRANSFORM ON XAVIER

Vincent Bao, Stanley Tzeng, Ching Hung

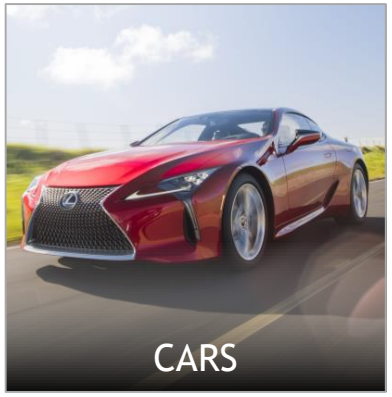
AGENDA

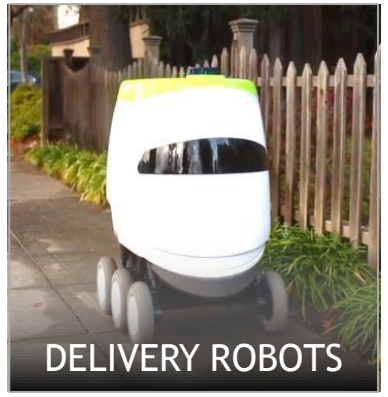
This talk is going to cover

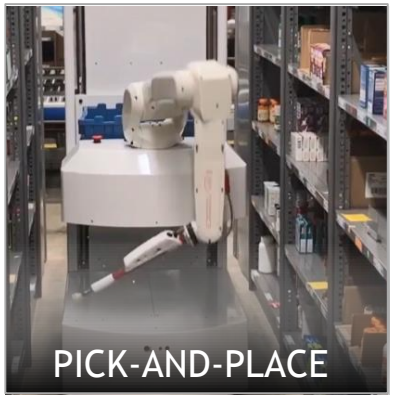
- Autonomous Machines Processor: Xavier
- A New Engine: Programmable Vision Accelerator (PVA)
- Introduction of Euclidean Distance Transform (EDT) with Different Algorithms
- Accelerating EDT by embedded Volta GPU
- PVA is another choice
- Conclusion and future work

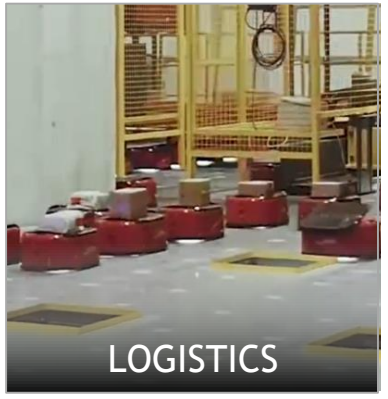
AUTONOMOUS MACHINES

Xavier is Designed for the Next Waves of Autonomous Machines









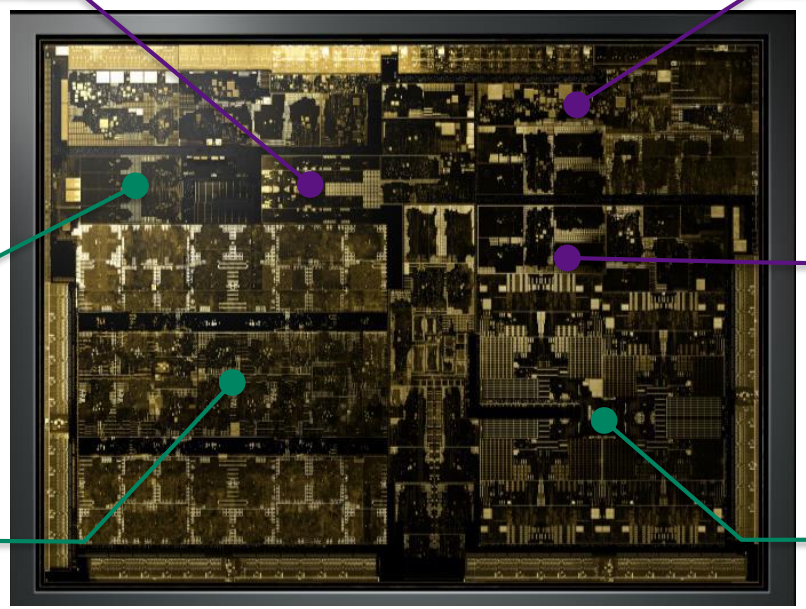
XAVIER

World First Autonomous Machines Processor

DLA
5.7 TFLOPS FP16
11.4 TOPS INT8

PVA
7-slot VLIW DSP
1.7 TOPS

Volta GPU
512 CUDA tensor cores



9 Billion Transistors, 350mm^2, 12 FFN

Multimedia Engines

Stereo & Optical Flow Engines

Carmel CPU
8 custom cores
ARM V8

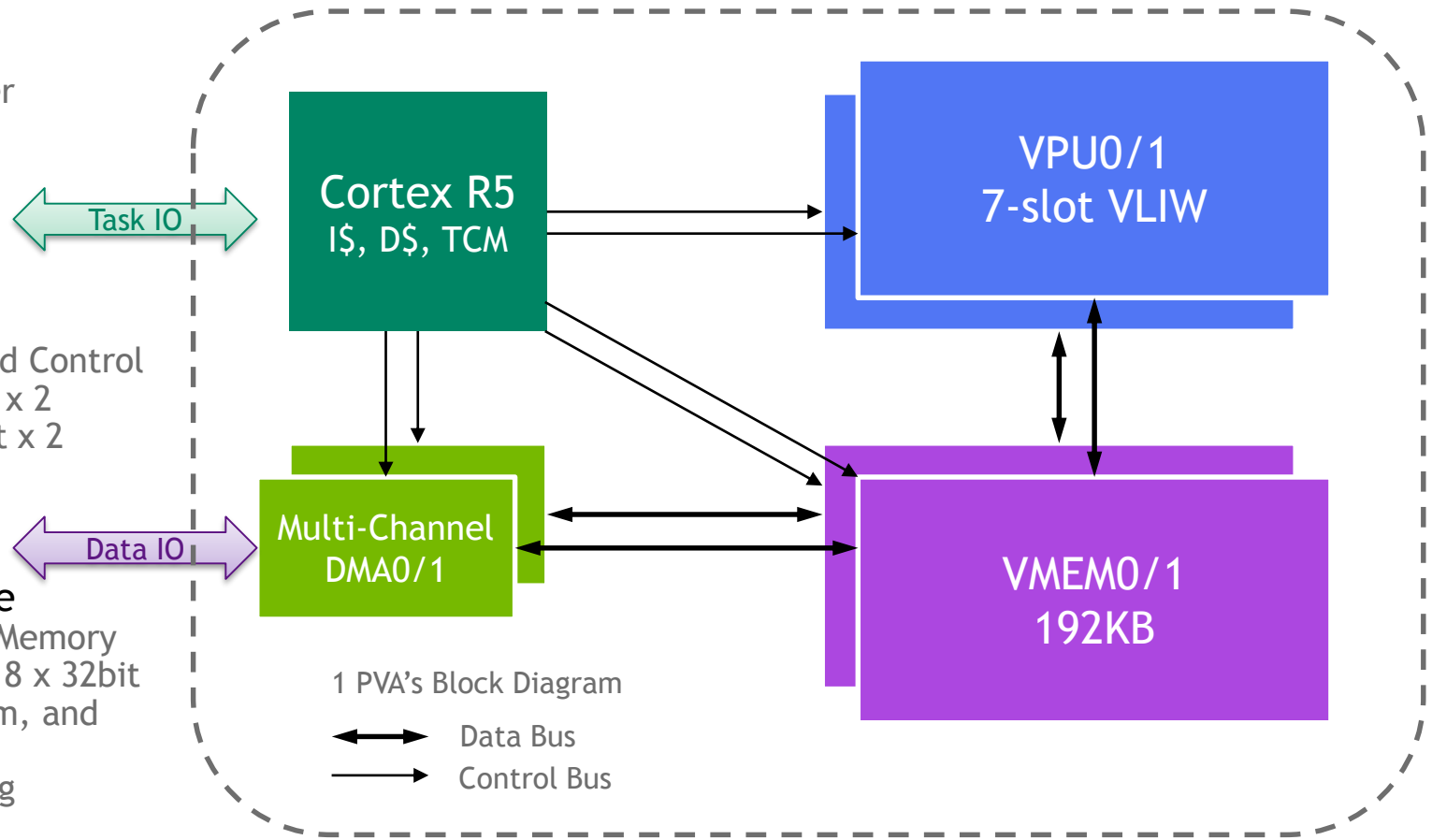
PROGRAMMABLE VISION ACCELERATOR

High-level Block Diagram

PVA x 2
- Optimized for Computer Vision Tasks

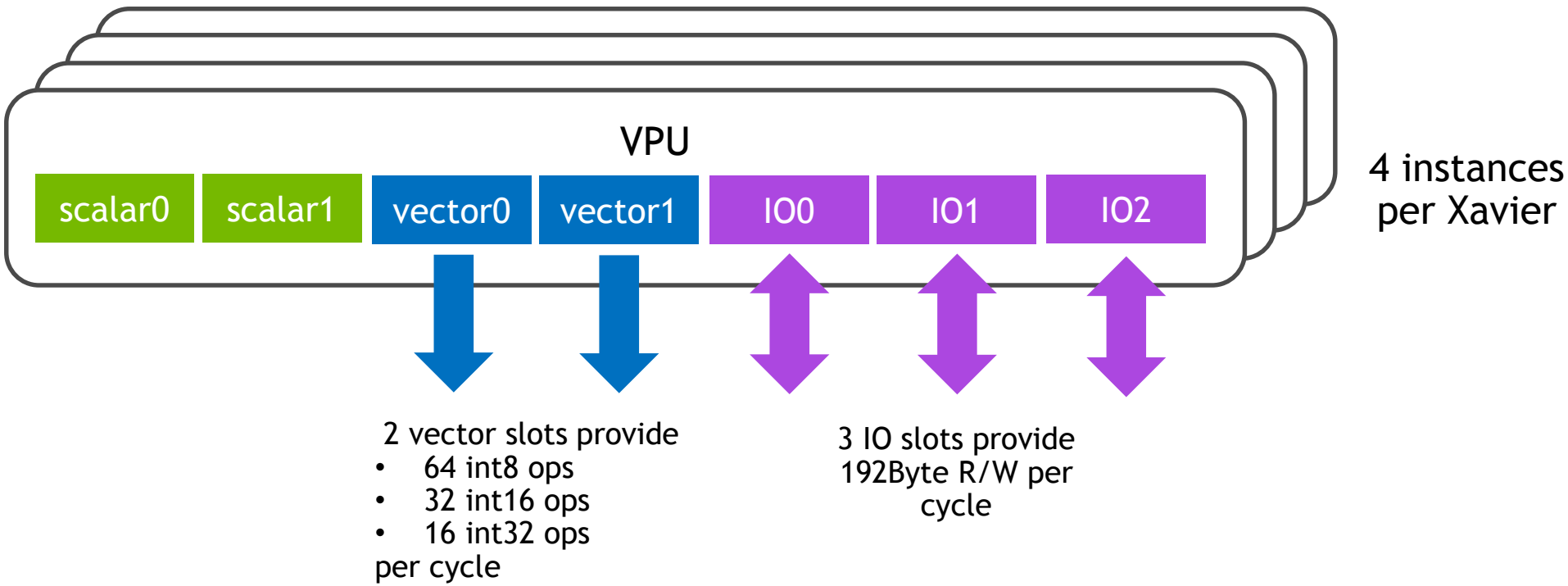
Each PVA
- Cortex R5 for Config and Control
- Vector Processing Units x 2
- DMA for Data Movement x 2

7-Slot VLIW architecture
- 2 Scalar + 2 Vector + 3 Memory
- 32 x 8bit | 16 x 16bit | 8 x 32bit
- Table Lookup, Histogram, and Vector-addressed Store
- I-cache with Prefetching
- Shared SRAM



PVA SIMD ARCHITECTURE

Wide-SIMD-Lane provides high-throughput Math and IO



PERFORMANCE MONITORS

Make sure the real performance on silicon meets our expectation

VPU activation monitor	DMA activation monitor
Kernel duration	Read transaction number
I cache miss number	Write transaction number
I cache miss penalty	Read active duration
Vector math stall number	Write active duration
…	…

PVA IN AUTONOMOUS DRIVING PIPELINE

An Example of Autonomous Pipeline on Xavier with PVA

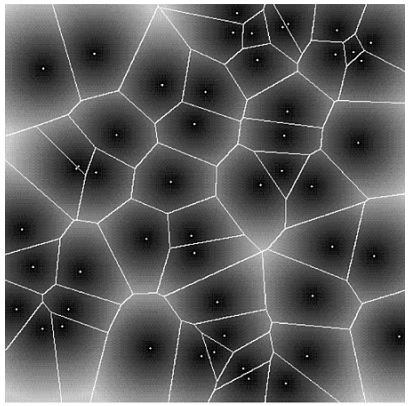
Parker	Parker ISP	Parker ISP, Pascal GPU	Pascal GPU	Pascal GPU	Pascal GPU	Pascal GPU, CPU
Xavier	Xavier ISP	Xavier ISP, PVA	DLA, PVA, Volta GPU	PVA, SOFE*, Volta GPU	PVA, Volta GPU	PVA, Volta GPU, CPU

* SOFE means Stereo and Optical Flow Engine

PVA is widely used in the pipeline to offload the non-deep-learning and integer tasks.
Then the Volta GPU has more compute budget to perform **more complex algorithms** with **higher resolution**.

EUCLIDEAN DISTANCE TRANSFORM

https://reference.wolfram.com/language/ref/DistanceTransform.html

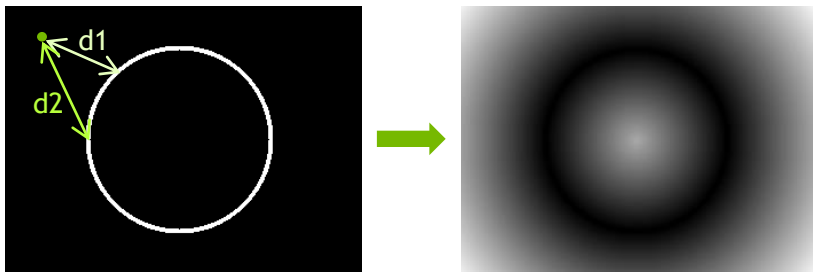


A List-Processing Approach to Compute Voronoi Diagrams and the Euclidean Distance Transform

EUCLIDEAN DISTANCE TRANSFORM (EDT)

Backgrounds

- Description (a global optimization problem)
  - $D(p) := \min\{d(p, q) \mid q \in Oc\} = \min\{d(p, q) \mid I(q) = 0\}$.



- Application (widely used in many area, a part of DL nowadays)
  - Biomedical Image Analysis
  - ADAS (lane detection, lane keeping)
  - Neural network post processing (DriveAV pipeline)

ACCELERATING EDT

Different Solutions

- The global optimization problem is hard to be accelerated since it can't easily be cut into pieces/tiles and has multiple process elements accelerate it.
- The kernel is important because its wide application and we mainly focus on accelerating it on Xavier since it is involved in our auto driving solution.
- Three EDT algorithms are implemented and compared on Xavier (GV11B):
  - Naïve (demonstrate the principle and show the baseline)
  - Felzenszwalb Algorithm
    - Ref: "Distance Transforms of Sampled Functions"
  - Parallel Banding Algorithm
    - Ref: "Parallel Banding Algorithm to Compute Exact Distance Transform with the GPU"

NAÏVE IMPLEMENTATION

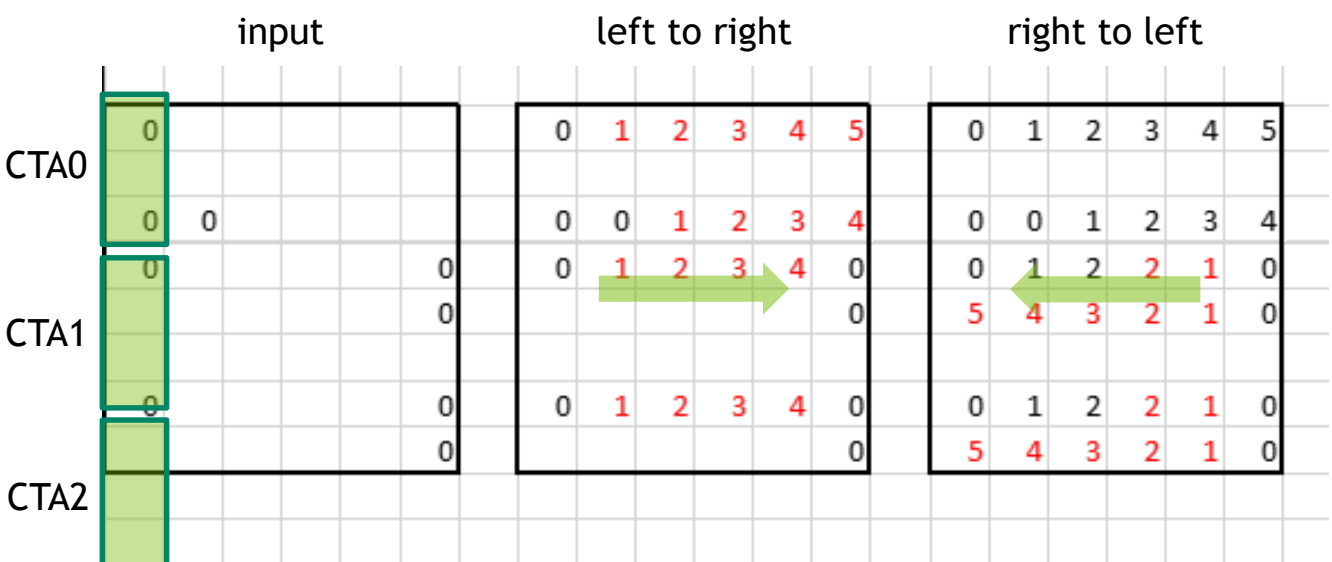
- Each result pixel' value is the shortest distance to the given target pixel set.
- Make an array to save the target pixel set, with its x and y coordinates.
- For each result pixel, calculates distance to each target pixel in the set and choose the minimal one as the value.
- If the image size is W x H = N, and the number of target pixel is n = R% x N, the total iteration number is like R% x N^2, almost O(N^2)!
- Accelerate on GPU: easy to implement and good occupancy
  - Make each thread for 1 or several output pixels
  - Load a subset of the target pixel array into shared memory

We can have a lot of CTA and thread to make the occupancy high

FELZENSWALB ALGORITHM

Horizontal Stage

- Felzenswalb is a linear time algorithm to calculate the Euclidean distance. There are 2 stages (horizontal and vertical) in the algorithm, each stage accesses every pixel once, so totally 2 x W x H = 2 x N, O(N)! LINEAR TIME!
- The idea is to make the global optimization to semi global. For example, the horizontal stage sweeps the image twice, from the left to right and the right to left, to get the minimal distance in each row (vertical distance is not considered in this stage) and save it into a buffer (hd, horizontal distance).



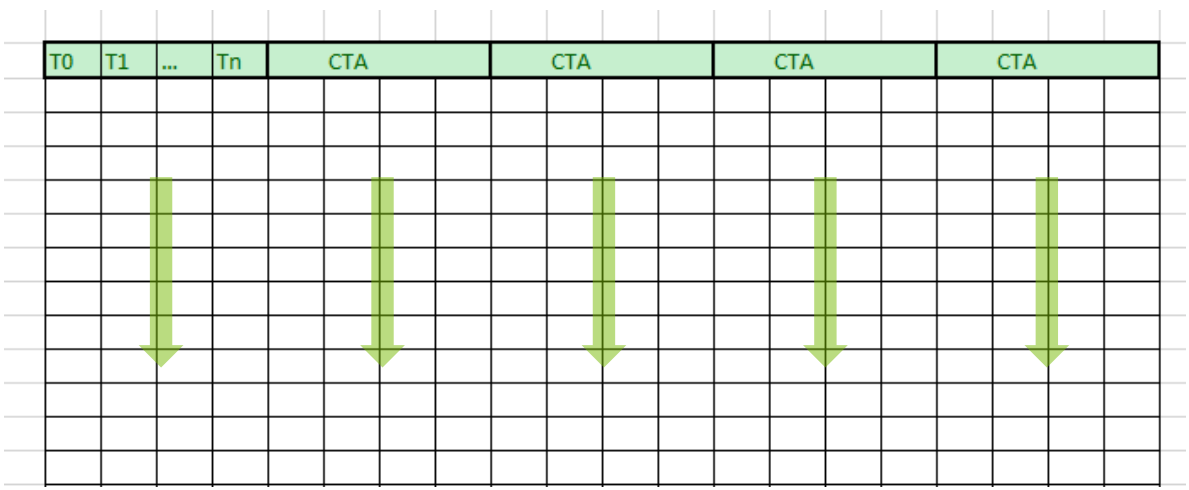
We can have totally H threads reside in M CTAs
The occupancy/utilization is a problem when Processing the small image.

If there is no target pixel in a row, set all the distances larger than W, means invalid.

FELZENSWALB ALGORITHM

Vertical Stage

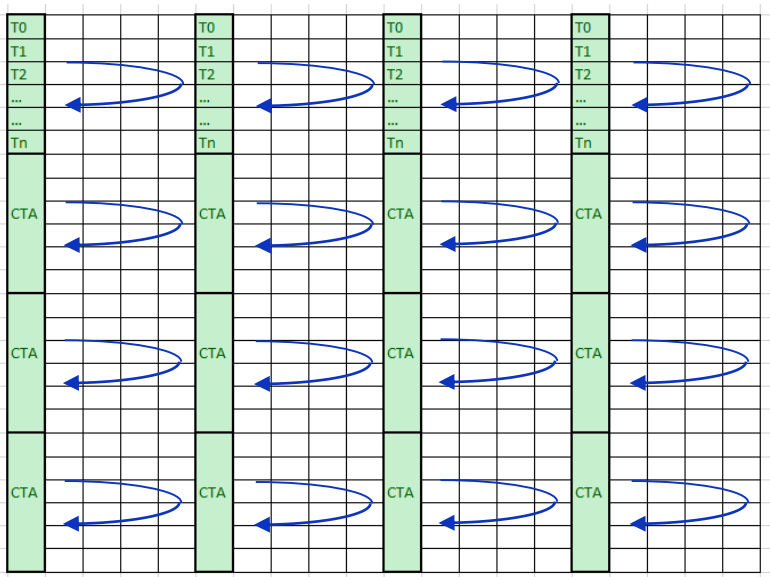
- When implementing the vertical stage on GPU, we scan the horizontal buffer from top to bottom. Make each thread process 1 column. The threads still need to be grouped by several CTAs.
- The issue here is we have limited data parallelism and not enough active warp to hide the latency, especially when the image size is small. And the utilization of the GPU also needs to be considered.
- The good point is the complexity of the algorithm is largely reduced so we can see a non-trivial speedup even if the image is not big.

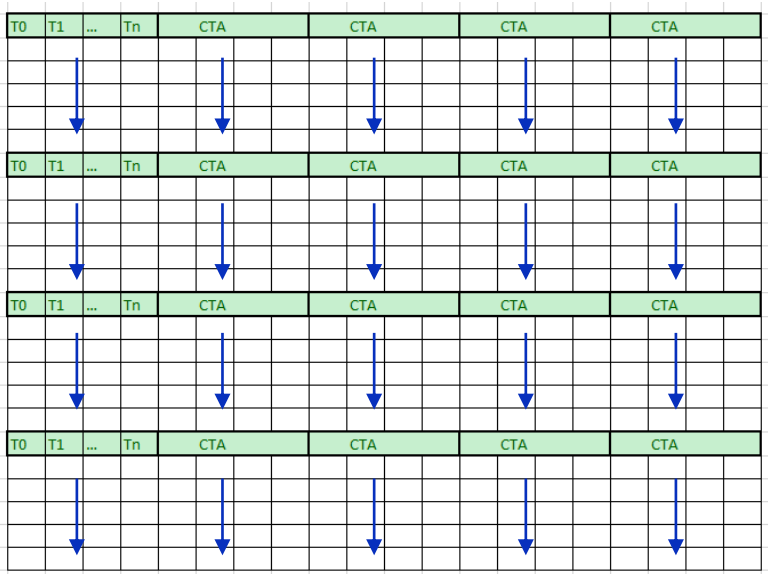


PARALLEL BANDING ALGORITHM

PBA

- The math principle of PBA is equivalent to the Felzenswalb algorithm so the complexity is O(N). PBA is designed to maximum the data parallelism, which targets to be accelerated on GPU (or other many-PE machine).
- For each stage, PBA split the image/hd into multiple band, and has more CTAs to process each band. The utilization and occupancy increase but need extra stages to merge the result of each band (since band is only the local optimal, needs to make it global). So we may have more kernels.

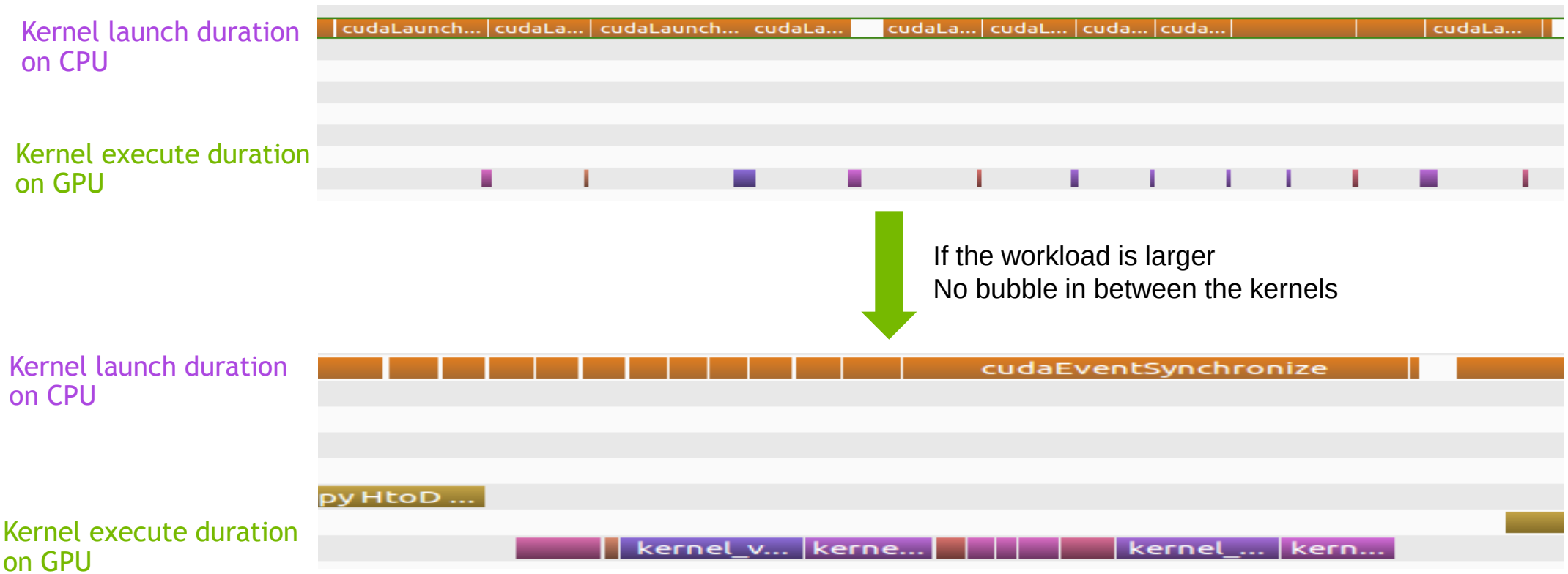




CUDA KERNEL LAUNCH DURATION

small image, fast kernel

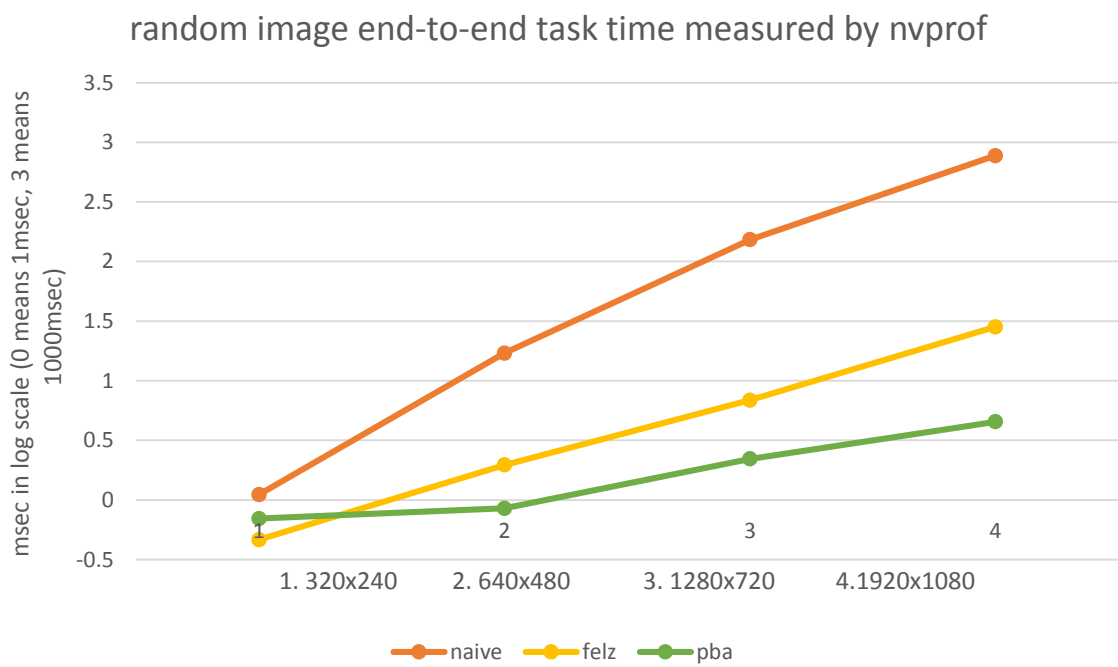
- Hundreds of CUDA cores enable the PBA to process an image in a short time, with nearly a dozen of kernels. Each kernel is short especially when the image size is small. CPU launches the kernels asynchronized but sequentially. So if the average kernel launch time is T, and if the total kernel time is less than 12T, it can be a kernel launch bound.



PERFORMANCE COMPARE

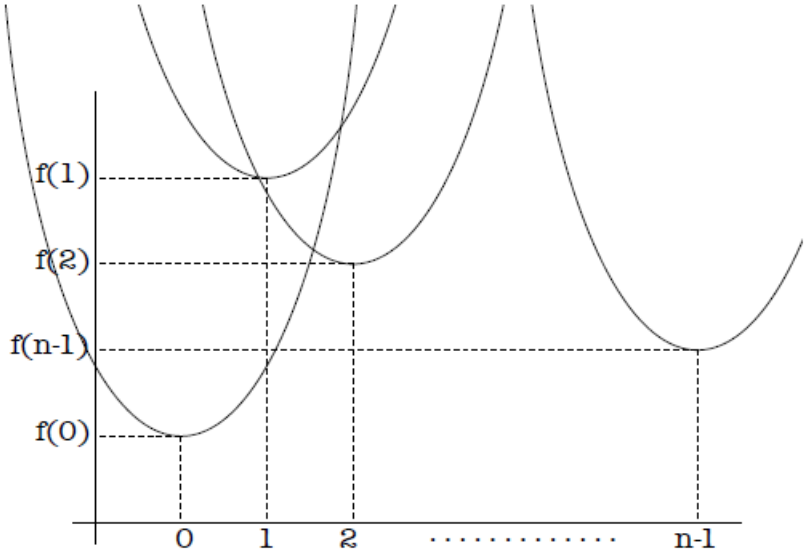
- First we compare the end-to-end task times of 3 kernels to process the same input image, range from 320x240 to 1920x1080. The data pattern is random and the target pixel density is 2%.
- The plot is in the log10 scale since the time increases in a non-linear way.

The PBA shows a perf regress when process the small size input. But we can find the trend to be faster than Felz if it can be non-kernel launch bound.



The baseline perf is sensitive to the total number of target pixel while the other 2 are not. So we can conclude averages speedups:
Felz: 15x to the baseline
PBA: 65x to the baseline

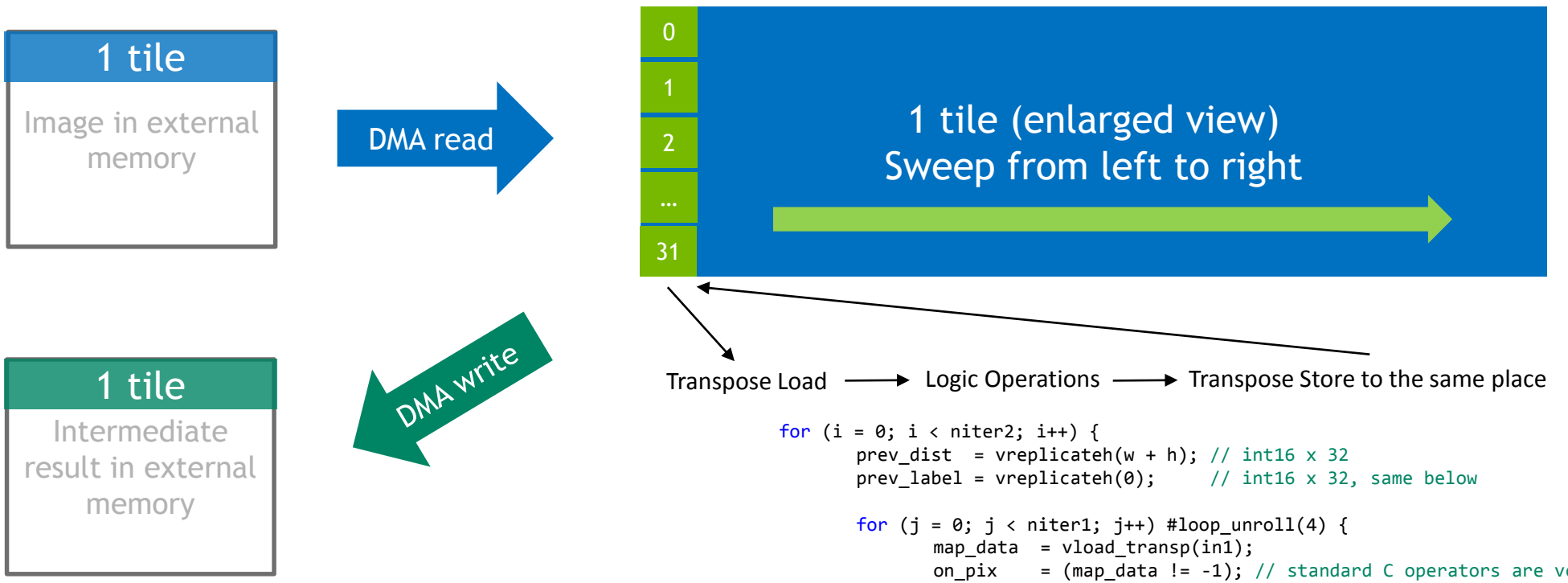
USING PVA TO ACCELERATE EDT



From paper "Distance Transforms of Sampled Functions

ACCELERATING EDT ON PVA

Using 1 VPU to elaborate the process

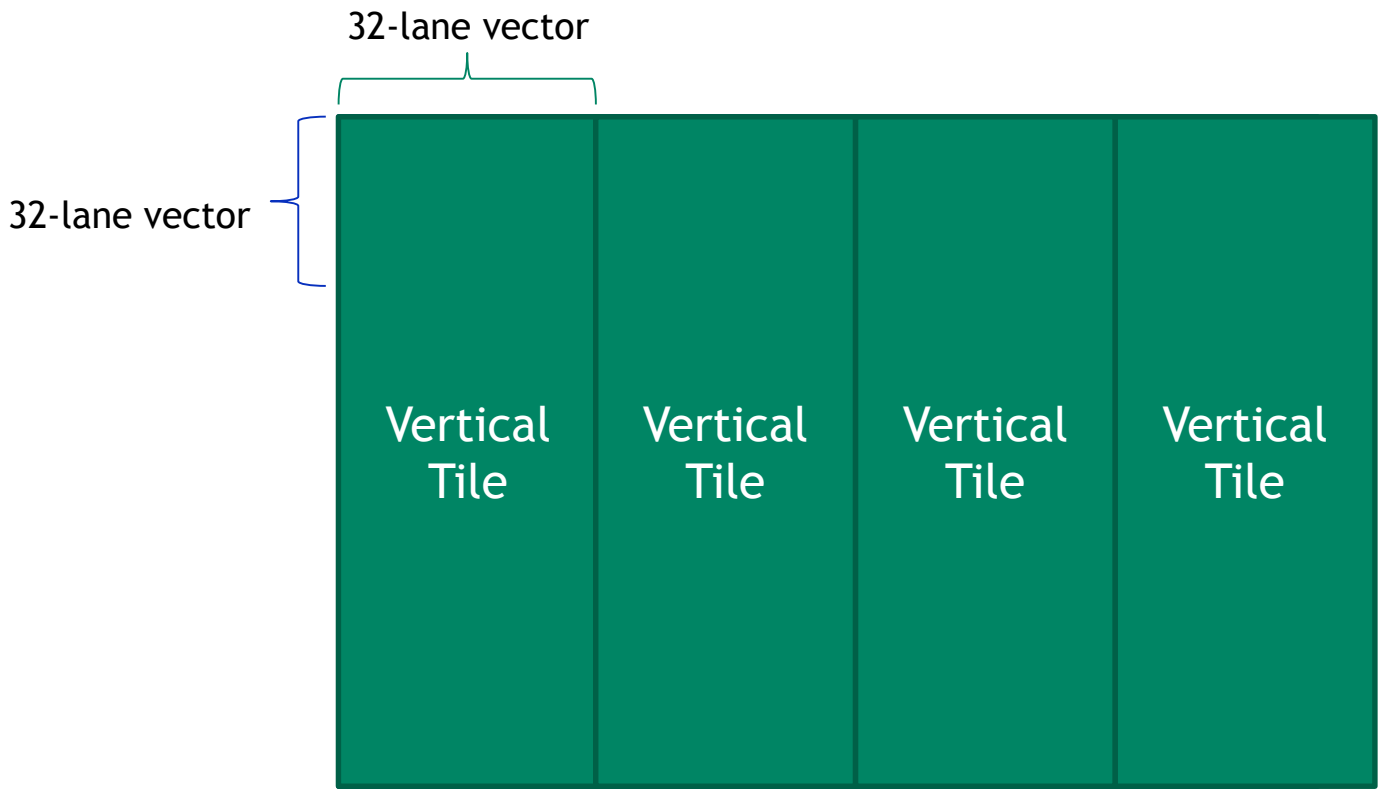



```
for (i = 0; i < niter2; i++) {
        prev_dist  = vreplicateh(w + h); // int16 x 32
        prev_label = vreplicateh(0);     // int16 x 32, same below

        for (j = 0; j < niter1; j++) #loop_unroll(4) {
                map_data  = vload_transp(in1);
                on_pix    = (map_data != -1); // standard C operators are vectorized
                prev_dist = vmux(on_pix, const_zero, prev_dist + 1);

                prev_label = vmux(on_pix, map_data, prev_label);
                hd         = vshiftor(prev_label, replicateh(12), prev_dist);
                vstore_transp(hd, out1);
        }
}
```

ACCELERATING EDT ON PVA

Full Frame View

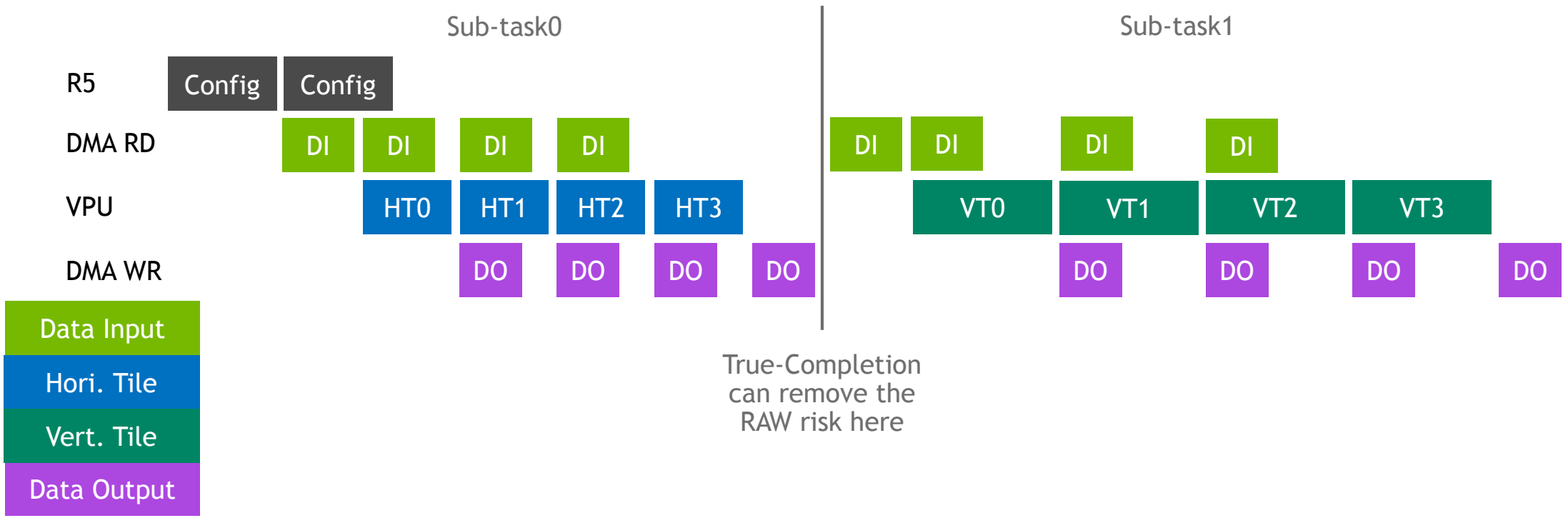
- We need to DMA in entire row in the horizontal stage and entire column in the vertical stage.



ACCELERATING EDT ON PVA

Pipelining the tasks

- We can pipeline the computation of each tiles and overlap the DMA transferring with the computing to keep the VPU working continuously.
- DMA, agen, zero-overhead loop, etc help to reduce the control overhead, close to **SOL**!



PERFORMANCE AND LIMITATION

The numbers are given by 1 VPU

- Performance
  - The VPU performance is ~330usec for a 320x240 image, while the performance on the GPU is 300usec.
  - If we have 4 instances of VPU batch 4 frames, the average DT process time is ~83us.
- Limitations
  - The image data type should be uint16. The size limitation on PVA is 960x960 due to the VMEM size bound. The larger size/uint32 date type can also be processed but will show a perf regress, since the parallelism goes down.

CONCLUSION AND FUTURE WORK

- Conclusion
  - For the high-resolution image, the GPU PBA algorithm is preferred to leverage all the compute resource of the GPU in Xavier; when the image is smaller, Felzenswalb algorithm shows the advantage of it simplicity.
  - PVA performs well and can offload some tasks from the GPU, even the global optimization problem like EDT.
- Future work
  - PVA is a new engine. We need to continue exploring use cases that can be offloaded to the PVA to increase the overall system efficiency.
  - Build a better software ecosystem that allows the programmer to easily implement their GPU pipelines on PVA, allowing the GPU to be freed up for deep-learning related tasks.


THANK YOU
email: vbao@nvidia.com
NVIDIA