SC13 Emerging Technologies

Power Efficient, Scalable Hybrid DSP+ARM+FPGA Platform for Computer Vision Tasks

Object Detection and Tracking with TI C66x Digital Signal Processor

Jason D. Bakos, Fan Zhang, Yang Gao, Benjamin Morgan
University of South Carolina

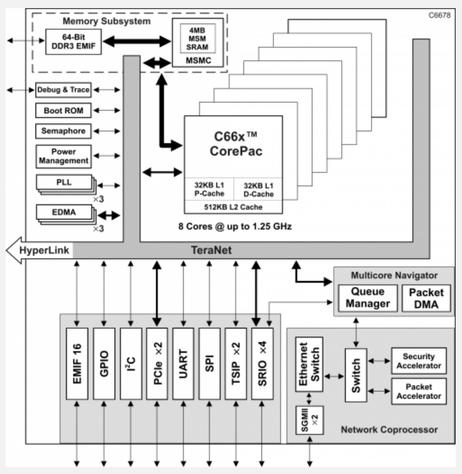
Objective: real-time object *detection* and *tracking* for **arbitrarily high resolution imaging** on a **mobile platform**

Algorithm: (1) dense optical flow, (2) cluster flow field pixels using BFS

Challenge:

- Optical flow requires, per 1920x1080 frame:
  - 180M floating-point operations,
  - 5.4 Gflops sustained for 30 fps (in addition to video, communication, and post processing overheads)
  - Need to scale for higher resolutions
- Embedded processors generally can't achieve this performance
- Embedded processor + GPU is power inefficient
- GPUs cannot run independently (no O/S)

Texas Instruments C6678 Keystone DSP:

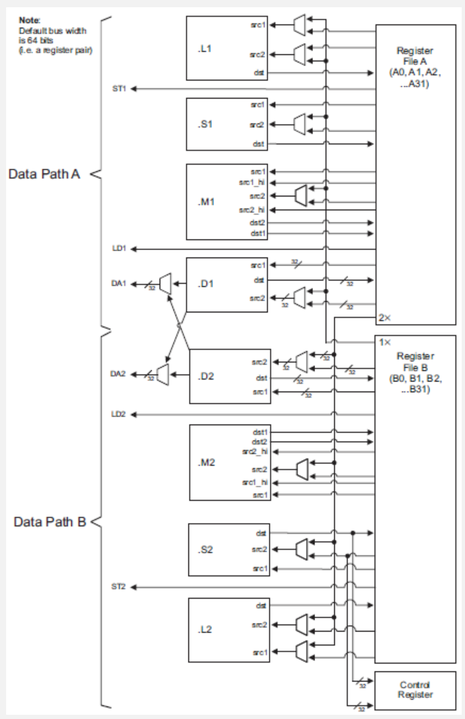


- Eight DSP cores on chip
- Each core: eight-way VLIW µarch
- Cores are logically decoupled
- 128 peak theoretical Gflops
- 128 GB/s scratchpad b/w
- 12.8 peak GB/s DDR3 b/w
- Keystone SoC includes integrated peripherals/offchip interfaces:
  - Gb ethernet, Serial RapidIO, PCIe, Hyperlink, DMA engine

DSP core design:

- Two 32x32 register files
- Eight functional units
  1. **S unit:** general arithmetic operations (2 flops/cycle)
  2. **L unit:** general arithmetic operations (2 flops/cycle)
  3. **M unit:** multiply (4 flops/cycle)
  4. **D unit:** 8-byte load/store

ILP exploited through programmer- and compiler-scheduled software pipelining for VLIW and SIMD



This material is based upon work supported by Texas Instruments and the National Science Foundation under Grant No. 0844951.

Optical Flow:

Assume brightness of pixels are subject to constraint:

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t)$$

Apply first-order Taylor expansion:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t$$

…implies:

$$\frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t = 0 \qquad \frac{\delta I}{\delta x}\frac{\Delta x}{\Delta t} + \frac{\delta I}{\delta y}\frac{\Delta y}{\Delta t} + \frac{\delta I}{\delta t} = 0 \qquad \frac{\delta I}{\delta x}V_x + \frac{\delta I}{\delta y}V_y = -\frac{\delta I}{\delta t}$$

solve

Lucas Kanade Method:

Assume all pixels within a neighborhood of n pixels have equal V_x and V_y:

$$\begin{bmatrix} \frac{\delta I}{\delta x}(q_1) & \frac{\delta I}{\delta y}(q_1) \\ \frac{\delta I}{\delta x}(q_2) & \frac{\delta I}{\delta y}(q_2) \\ \vdots & \vdots \\ \frac{\delta I}{\delta x}(q_n) & \frac{\delta I}{\delta y}(q_n) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -\frac{\delta I}{\delta t}(q_1) \\ -\frac{\delta I}{\delta t}(q_2) \\ \vdots \\ -\frac{\delta I}{\delta t}(q_n) \end{bmatrix}$$

A $\quad v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$ $\quad$ b

Solve:

$$Av = b$$

Use LSM:

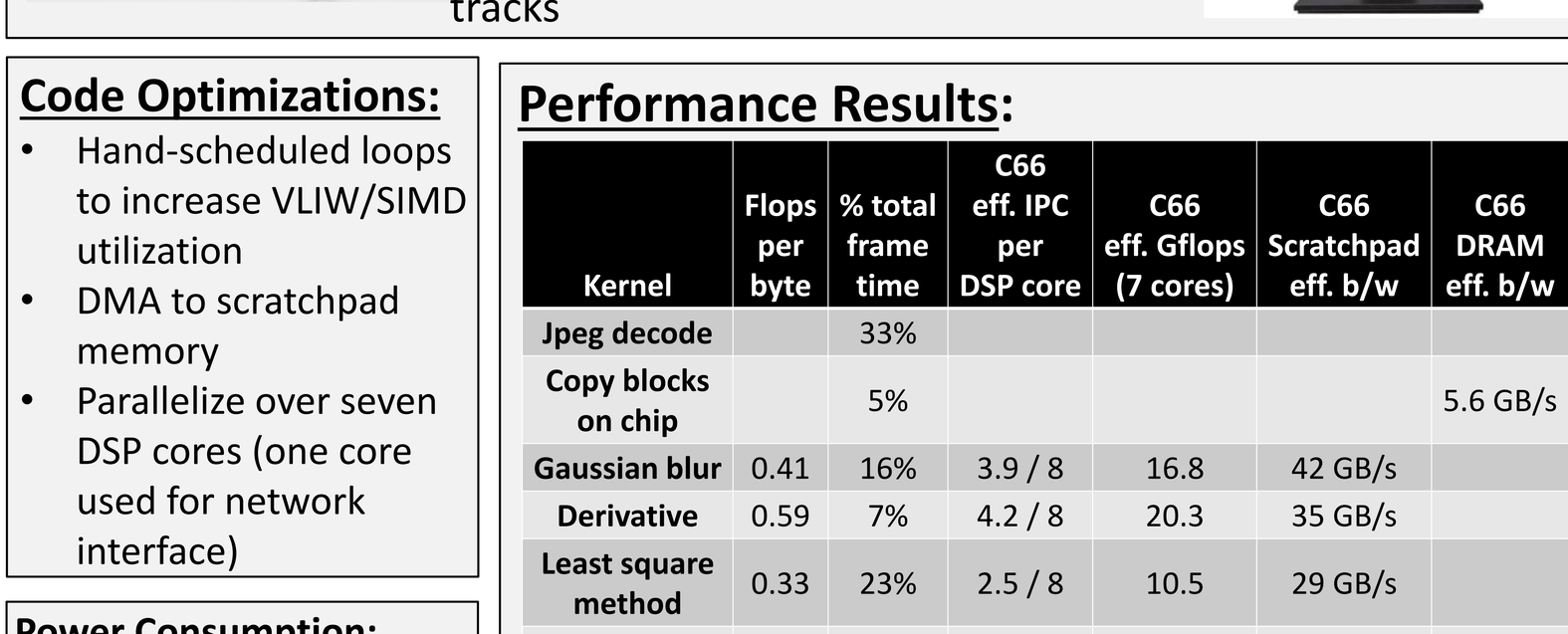
$$A^T Av = A^T b$$

$$v = (A^T A)^{-1} A^T b$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x q_i{}^2 & \sum_i I_x q_i I_y q_i \\ \sum_i I_y q_i I_x q_i & \sum_i I_y q_i{}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x q_i I_t q_i \\ -\sum_i I_y q_i I_t q_i \end{bmatrix}$$

Computational steps:

- Compute A (x and y part. derivatives)
- Compute b (t derivative part. derivative)
- Compute v



Custom Platform:

TMS320C6678: 8-core C6678 DSP EVM

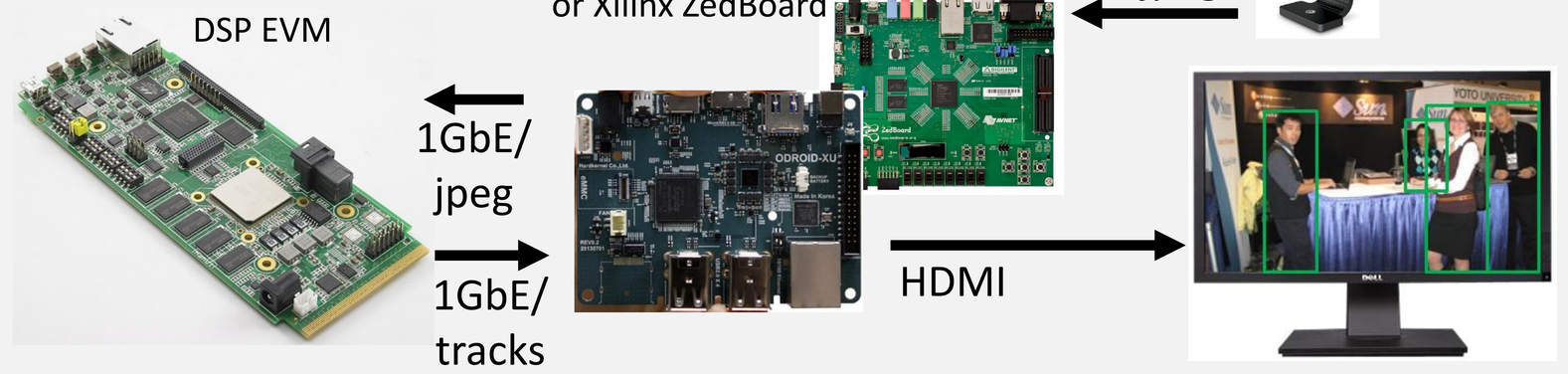
ODROID Samsung Exynos5 quad-ARM A15 or Xilinx ZedBoard

USB/ jpeg

1GbE/ jpeg

1GbE/ tracks

HDMI



Code Optimizations:

- Hand-scheduled loops to increase VLIW/SIMD utilization
- DMA to scratchpad memory
- Parallelize over seven DSP cores (one core used for network interface)

Power Consumption:

~16 Watts peak (+5 W with onboard emulator)

Performance Results:

Kernel	Flops per byte	% total frame time	C66 eff. IPC per DSP core	C66 eff. Gflops (7 cores)	C66 Scratchpad eff. b/w	C66 DRAM eff. b/w
Jpeg decode		33%				
Copy blocks on chip		5%				5.6 GB/s
Gaussian blur	0.41	16%	3.9 / 8	16.8	42 GB/s	
Derivative	0.59	7%	4.2 / 8	20.3	35 GB/s	
Least square method	0.33	23%	2.5 / 8	10.5	29 GB/s	
Copy blocks off chip		13%				5.6 GB/s
Clustering		2%				