Lucas Kanade Optical Flow Estimation on the TI C66x Digital Signal Processor

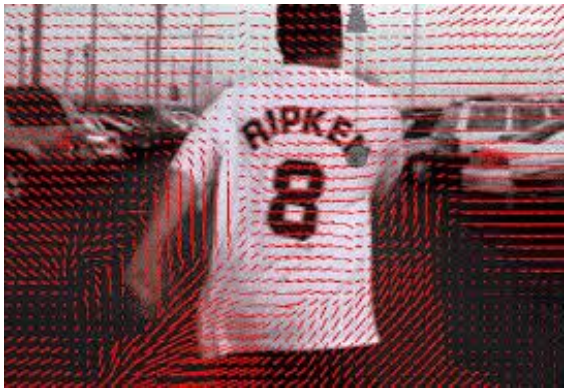
Fan Zhang, Yang Gao and Jason D. Bakos

2014 IEEE High Performance Extreme Computing Conference(HPEC '14)

UNIVERSITY OF SOUTH CAROLINA

What is Optical Flow

- **Evaluates the pixel movement in video stream**
  - Represented as a dense 2D fields
- **Many applications need to apply real-time optical flow**
  - Robotic vision
  - Augmented reality
- **Computationally intensive**



TI C66x Multicore DSP

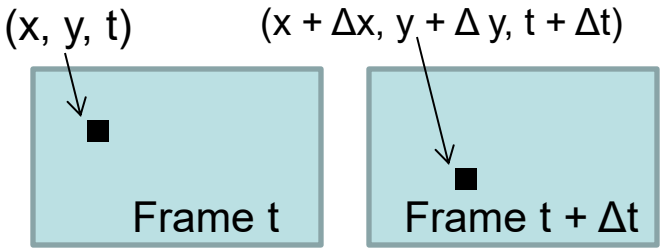
- **Unique architectural features**
  - Eight cores
  - Statically scheduled VLIW/SIMD instructions
  - 2 register files and 8 functional units per core

	Tesla K20X GPU	Intel i7 Ivy Bridge	TI C6678 Keystone	NVIDIA Tegra K1
	Server GPU	Server CPU	**Embedded**	Embedded
Peak single precision throughput	3.95 Tflops	448 Gflops	**160 Gflops**	327 Gflops
Peak Power	225 W	77 W	**<10 W**	<20 W
DRAM bandwidth	250 GB/s	25.6 GB/s	**12.8 GB/s**	17.1 GB/s

Why using DSP?

- Low power consumption
- High performance floating point arithmetic
- Strong potential for computer vision tasks
  - 1D vs 2D (signal processing vs image processing)

Gradient-based Optical Flow Solver



- Optical flow evaluation

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

- First-order approximation

$$f(x, y, t) = f(x, y, t) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \frac{\partial f}{\partial t}\Delta t$$

Gradient of pixel in x, y and t direction, known

$$\frac{\partial f}{\partial x}v_x + \frac{\partial f}{\partial y}v_y = -\frac{\partial f}{\partial t}$$

Optical flow, unknown

One formula with two unknowns

Lucas Kanade Method

If we assume the pixel adjacent to the center has the same optical flow as the center

x-1,y-1	**x,y-1**	**x+1,y-1**
x-1,y	**x,y**	**x+1,y**
x-1,y+1	**x,y+1**	**x+1,y+1**

$$\begin{cases} \dfrac{\partial f(x-1,y-1)}{\partial x}v_x + \dfrac{\partial f(x-1,y-1)}{\partial y}v_y = -\dfrac{\partial f(x-1,y-1)}{\partial t} \\ \cdots \\ \dfrac{\partial f(x+1,y+1)}{\partial x}v_x + \dfrac{\partial f(x+1,y+1)}{\partial y}v_y = -\dfrac{\partial f(x+1,y+1)}{\partial t} \end{cases}$$

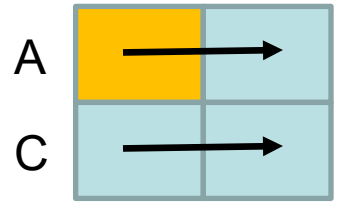
Let $A = \begin{bmatrix} \dfrac{\partial f(x-1,y-1)}{\partial x}, \dfrac{\partial f(x-1,y-1)}{\partial y} \\ \cdots \\ \dfrac{\partial f(x+1,y+1)}{\partial x}, \dfrac{\partial f(x+1,y+1)}{\partial y} \end{bmatrix}$ $b = \begin{bmatrix} \dfrac{\partial f(x-1,y-1)}{\partial t} \\ \cdots \\ \dfrac{\partial f(x+1,y+1)}{\partial t} \end{bmatrix}$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = (A^T A)^{-1} A^T b = \begin{bmatrix} \sum \dfrac{\partial f}{\partial x}^2 & \sum \dfrac{\partial f}{\partial x}\dfrac{\partial f}{\partial y} \\ \sum \dfrac{\partial f}{\partial x}\dfrac{\partial f}{\partial y} & \sum \dfrac{\partial f}{\partial y}^2 \end{bmatrix} \begin{bmatrix} -\sum \dfrac{\partial f}{\partial x}\dfrac{\partial f}{\partial t} \\ -\sum \dfrac{\partial f}{\partial y}\dfrac{\partial f}{\partial t} \end{bmatrix}$$

Least Square Method

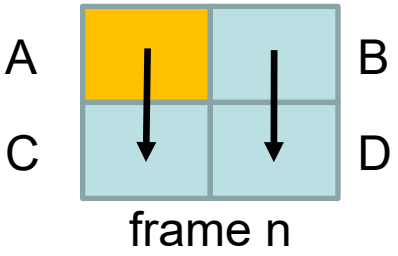
Image Derivative Computation

$$\frac{\partial f}{\partial x} =$$



(A – B + C – D) / 2

$$\frac{\partial f}{\partial y} =$$



(A – C + B – D) / 2

$$\frac{\partial f}{\partial t} =$$

A - B

Lucas Kanade Method

- Input: Frame n, frame n+1, window size w
- Output: Optical flow field F

- For each pixel x, y
- Compute x,y,t derivative matrices Dx, Dy, Dt for its neighbor window
- Compute optical flow by the least square method

Derivative Computation (Example)

Image n

10	10	10	10
10	30	10	10
10	10	10	10
10	10	10	10

Image n + 1

10	10	10	10
10	10	10	10
10	10	30	10
10	10	10	10

-10	10	0
-10	10	0
0	0	0

$$\frac{\partial f}{\partial x}$$

-10	-10	0
10	10	0
0	0	0

$$\frac{\partial f}{\partial y}$$

0	0	0
0	20	0
0	0	-20

$$\frac{\partial f}{\partial t}$$

Least Square Method Example (Neighbor Window Size = 3)

$\frac{\partial f}{\partial x}$

-10	10	0
-10	10	0
0	0	0

$\frac{\partial f}{\partial y}$

-10	-10	0
10	10	0
0	0	0

$\frac{\partial f}{\partial t}$

0	0	0
0	20	0
0	0	-20

$$\sum_W \left(\frac{\partial f}{\partial x}\right)^2 = 400$$

$$\sum_W \left(\frac{\partial f}{\partial y}\frac{\partial f}{\partial t}\right) = 200$$

$$\sum_W \left(\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\right) = 0$$

$$\sum_W \left(\frac{\partial f}{\partial y}\right)^2 = 400$$

$$\sum_W \left(\frac{\partial f}{\partial x}\frac{\partial f}{\partial t}\right) = 200$$

$$\begin{bmatrix} Vx \\ Vy \end{bmatrix} = \begin{bmatrix} 400 & 0 \\ 0 & 400 \end{bmatrix}^{-1} \begin{bmatrix} -200 \\ -200 \end{bmatrix}$$

Optimize Lucas Kanade Method on DSP

- **Derivative computation**
  - SIMD arithmetic
  - Data interleaving

- **Least square method**
  - Loop unrolling
  - SIMD load & arithmetic

Derivative Computation

10	10	10	10
10	30	10	10
10	10	10	10
10	10	10	10

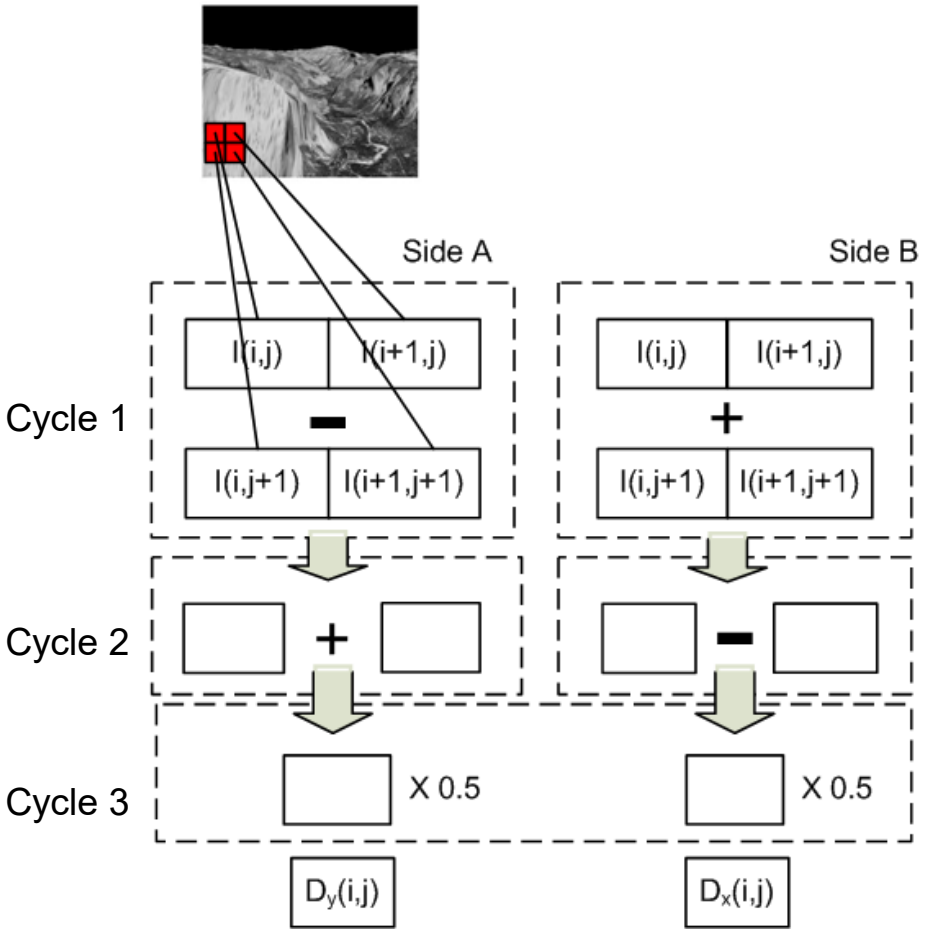
Derivative Computation (Dx, Dy)

-10	10	0
-10	10	0
0	0	0

-10	-10	0
10	10	0
0	0	0

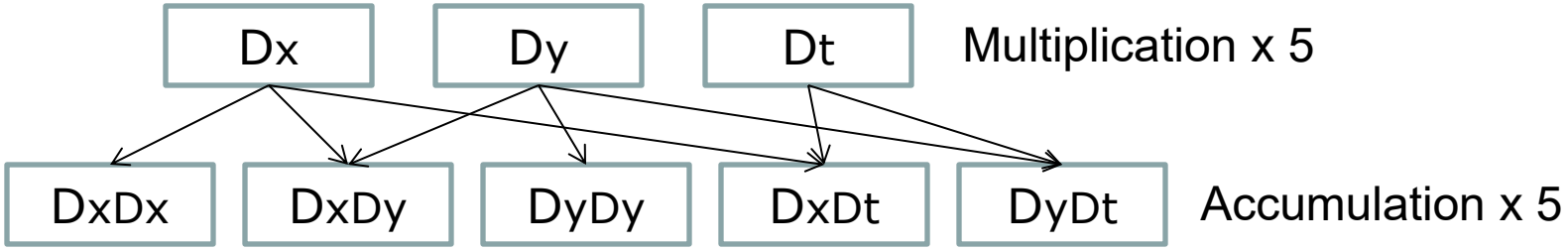
Interleave

-10	-10	10	-10	0	0
-10	10	10	10	0	0
0	0	0	0	0	0

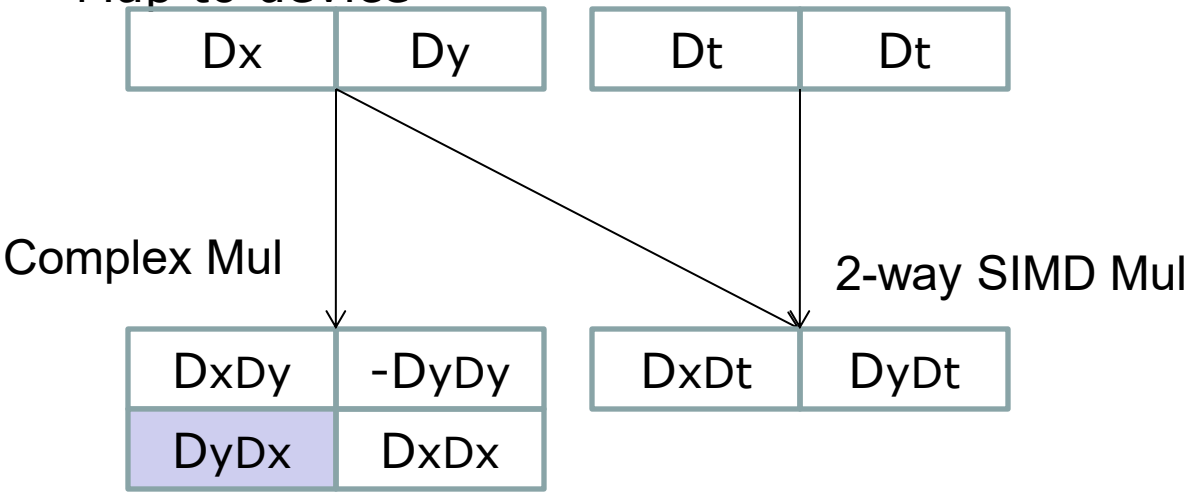


Least Square Method

- Required computations



- Map to device



(a+bj)(c+dj) = (ac-bd) + (ad+bc)j

Least Square Method

- Unrolled 2x
  - Group up loads into SIMD operation

Load DxDy Load Dt Complex MUL SIMD MUL SIMD ADD

DxDy

-10	-10	10	-10	0	0
-10	10	10	10	0	0
0	0	0	0	0	0

Dt

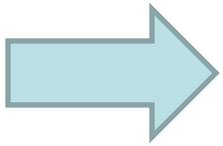
0	0	0
0	20	0
0	0	-20

(Dx,Dy,Dt)

-10	-10	0

(Dx,Dy,Dt)

10	-10	0



(DxDx,DxDy,DyDy,DxDt,DyDt)

100	100	100	0	0

(DxDx,DxDy,DyDy,DxDt,DyDt)

100	-100	100	0	0

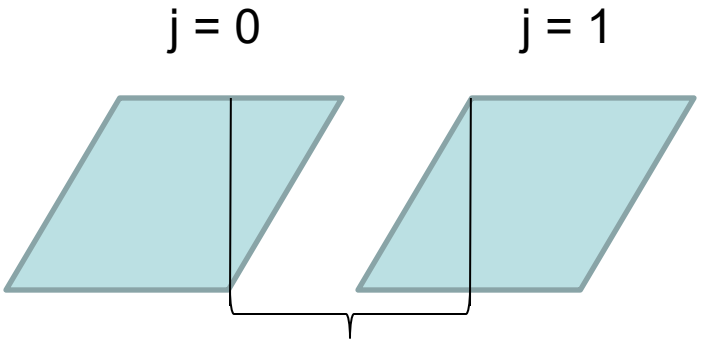
+

200
0
200
0
0

Loop Flattening

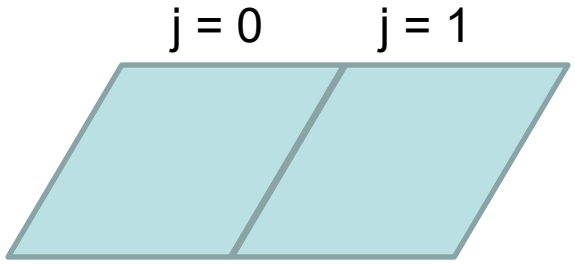
- Software Pipelining
  - TI's compiler technique to allow independent loop iterations be executed in parallel
  - The consecutive iterations are packed and executed together so that the arithmetic functional units are fully utilized

```
for (i = 0; i < m; ++i)
  for (j = 0; j < n; ++j)
```

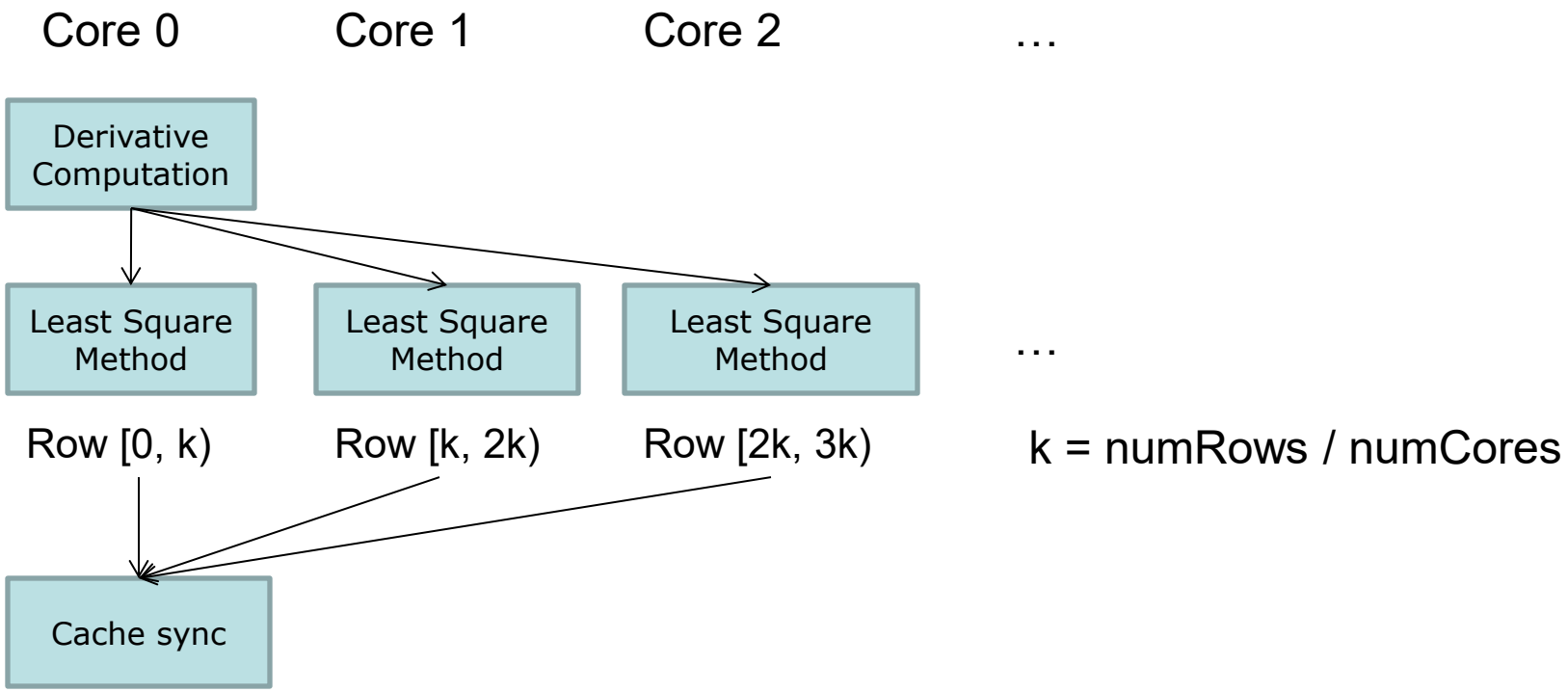


Pipeline prologue/epilogue overhead

```
for (k = 0; k < m * n; ++k)
  …
  Update i, j;
```



Multicore Utilization



Accuracy Improvement

- **Cannot catch movement larger than window size**
  - Gaussian Pyramid
    - A coarse to fine strategy for optical flow computation
    - Catches large movements

- **First order approximation may not be accurate**
  - Iterative refinement
    - Iteratively process and offset pixels until the computed optical flow is converged
    - Introduce data random access

Experiment Setup

Platform	#Cores	Implementation	Power Measurement
TI C6678 DSP	8	Our Implementation	TI GPIO-USB Module
ARM Cortex A9	2	Our Implementation	YOKOGAWA WT500 Power Analyzer
Intel i7-2600	4	Our Implementation	Intel RAPL
Tesla K20 GPU	2688	OpenCV	NVIDIA SMI

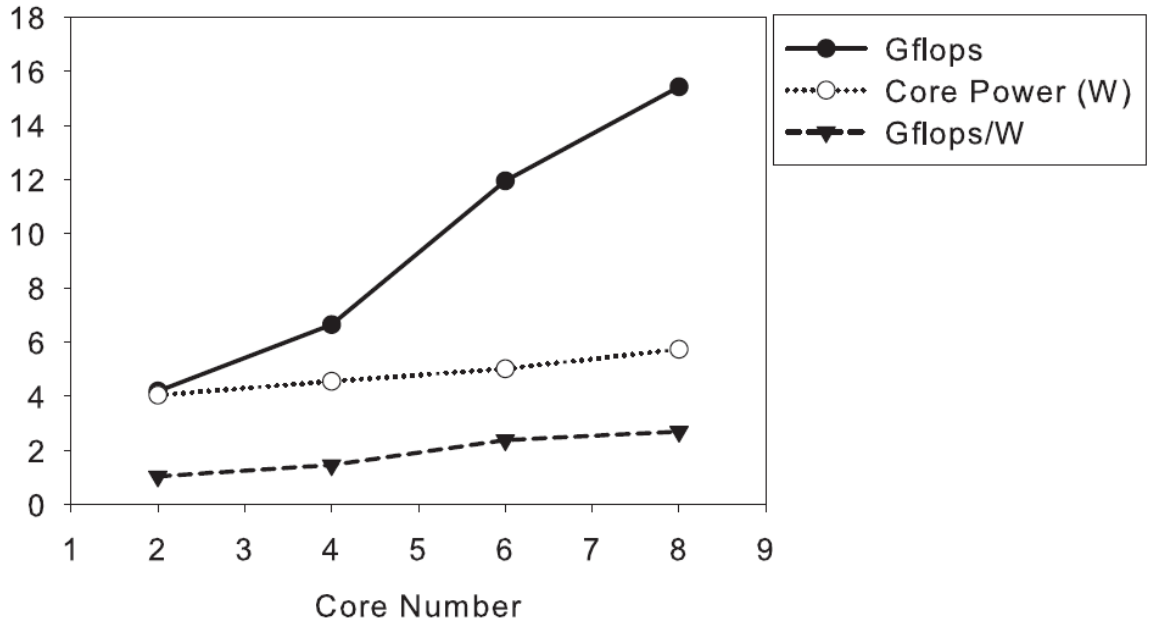
Results and Analysis

- Highest Performance
  - Tesla K20
- Lowest Power
  - Cortex A9
- Best Power Efficiency
  - TI C66x DSP

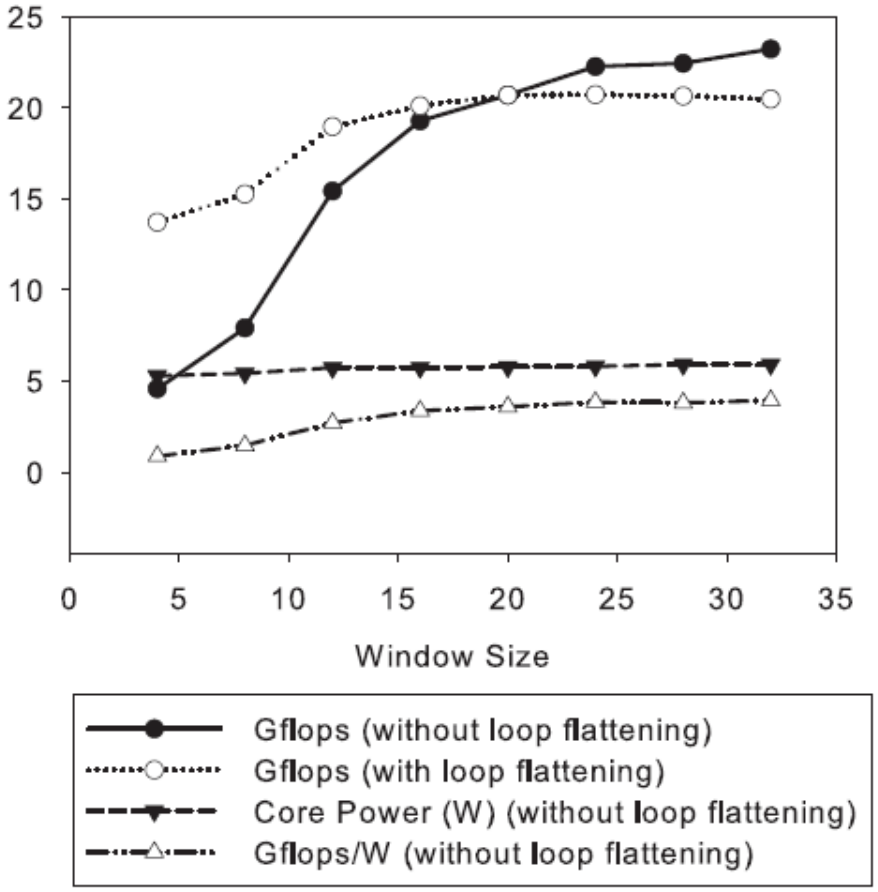
Platform	C66x	CortexA9	Intel i7-2600	K20
Actual Gflops/ Peak Gflops	**12%**	7%	4%	3%
Gflops	15.4	0.7	17.1	**108.6**
Power (W)	5.7	**4.8**	52.5	79.0
Gflops/W	**2.69**	0.2	0.3	1.4

Results and Analysis

- We achieve linear scalability on multi-cores
- The power efficiency of the DSP is low when its cores are partially utilized
  - Static power consumption



Results and Analysis



- Performance are related with window size
  - Software pipeline performance
- Loop flattening is able to improve performance significantly on small window size

Conclusion

- First research work on DPS accelerated Lucas Kanade method
- Achieve higher energy efficiency and device utilization than GPU and CPU

Q & A

Kernels of Pyramidal Lucas Kanade Method

- Gaussian Blur — 28 flop/pixel
- Derivative Computation — 8 flop/pixel
- Least Square Method — 10^5 flop/pixel
- Flow Field Bilinear Interpolation — 3 flop/pixel