# Optimization and Evaluation of Image- and Signal-Processing Kernels on the TI C6678 Multi-Core DSP

CHREC
NSF Center for High-Performance Reconfigurable Computing

HPEC '14

UF UNIVERSITY of FLORIDA
Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
BYU
BRIGHAM YOUNG UNIVERSITY
THE GEORGE WASHINGTON UNIVERSITY

Dr. Alan George
Professor of ECE
University of Florida

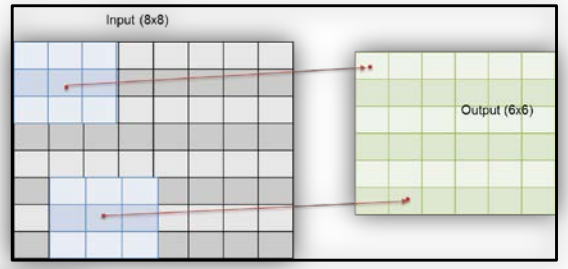
Dr. Herman Lam
Associate Professor of ECE
University of Florida

Barath Ramesh
Justin Richardson
Research Students
University of Florida

Asheesh Bhardwaj
Texas Instruments

# Outline

- TMS320C6678 Architecture Overview
- 2D Convolution
  - Kernel Overview
  - Intrinsic Optimization
  - Results and Conclusions
- Bilinear Interpolation w/ Image Rotation
  - Kernel Overview
  - DSP Design Optimization
  - DMA Optimization
  - Results and Conclusions
- HPEC Challenge Benchmark Suite
- Frequency-Domain FIR (FDFIR)
  - Kernel Overview
  - DMA Optimization
  - Results and Conclusions
- Corner Turn (CT)
  - Kernel Overview
  - Results and Conclusions
- Conclusions





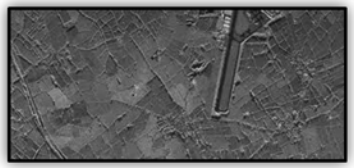
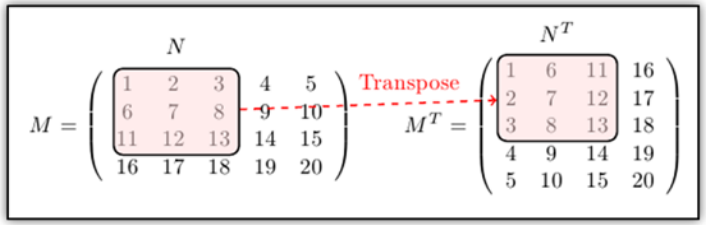


Image source:soue.org.uk



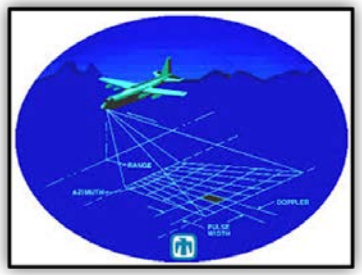
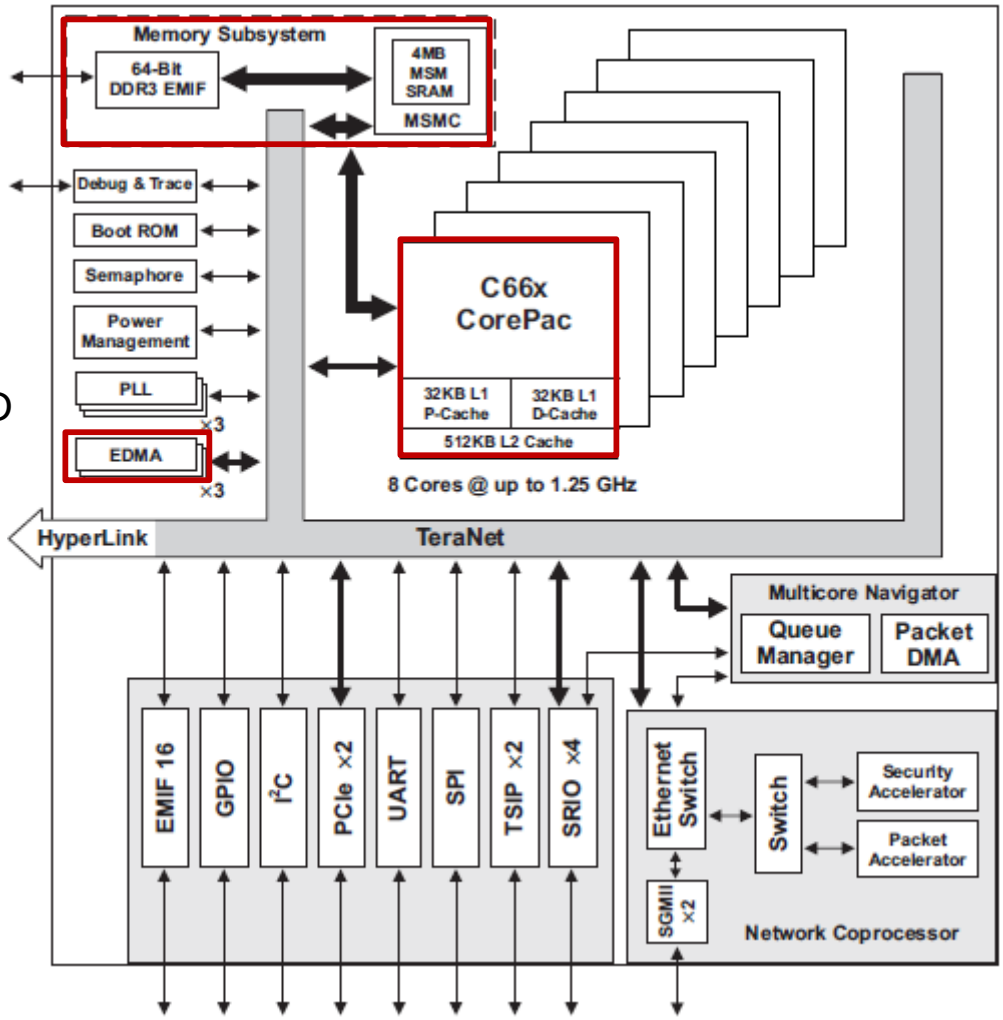



Image source:sandia.gov

FIR: Finite Impulse Response

# TMS320C6678 Architecture Overview

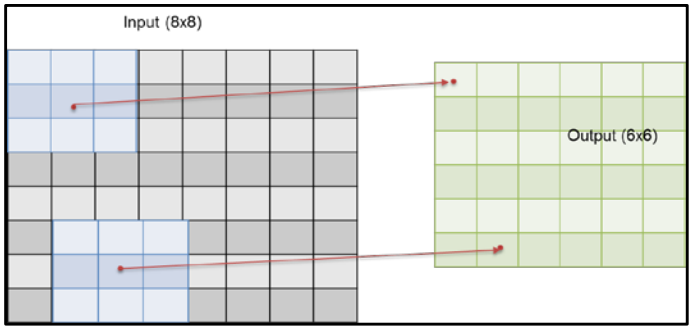
- TMS320C6678
  - 8 C66x CorePac
    - Clocked up to 1.25 GHz
    - 256KB L1P SRAM/Cache
    - 256KB L1D SRAM/Cache
    - 4MB L2 SRAM/Cache
    - Each CorePac
      - Two datapaths, 4-way SIMD for 32-bit data
      - EMIF, DDR3-64 bit with (72,64) ECC
  - 4MB MSM SRAM
    - Shared across 8 cores
  - 64-Bit DDR3 Controller
    - Up to 1600MHz
  - 3 EDMA
    - 16 independent channels
      - DSP/2 clock rate
    - 64 independent channels
      - DSP/3 clock rate




TMS320C6678 device architecture [1]

# 2D Convolution

- Used in image processing
  - Edge detection, finding a specific image within a larger image, panorama, motion detection, and stereo vision
- 2D convolution is the most commonly employed algorithm in computer vision and image processing [2]
- Example
  - Using 2D convolution one can find which parts of image have highest correlation

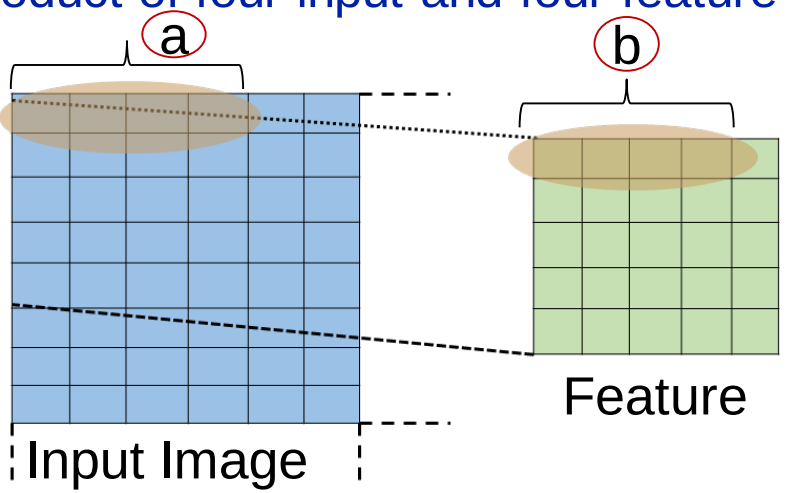



- Algorithm
  - Window slides over all possible positions of main image
    - f(window,kernel) → SAD/MAD, 2D Convolution, Correntropy etc.
  - For 1080p 16-bit image with 25×25 window
    - (1920-25+1)×(1080-25+1) ~ 2M windows
    - 2M × 16 × 625 ~ 20 trillion non-sequential memory accesses
    - 20 trillion operations to obtain resultant output image

# Intrinsic Optimization

- *Quad-Word* processing using *MEM8* and *DOTPSU4H* intrinsic on 16-bit precision pixels
  - **Pixel read:** *MEM8* for unaligned loads of eight bytes
    - *MEM8* to read four pixels of input and feature mask of sliding window
  - **Dot product:** *DOTPSU4H* multiplies four signed 16-bit values by four unsigned 16-bit values and returns 32-bit sum
    - *DOTPSU4H* to perform dot product of four input and four feature pixels read using *MEM8* intrinsic

```
a: _mem8(&imgIn)
b: _mem8(&featureIn)
a•b: _dotpsu4h(a,b)
```



# Intrinsic Optimization

- Compiler feedback after software pipelining
  - Compiler report after software pipelining second innermost ***FOR*** loop for feature size of 25x25

```
#pragma omp for
//3 nested for loops
for (l = 0; l < FEATURE_SIZE; l++){
    temp += (imgIn[(i+k)*IMG_COLS (j+l)]*featureIn[k][l]);
}
imgOut[i*OUTPUT_COLS + j] = temp;
temp = 0;
```

```
:*       Searching for software pipeline schedule
at
:*         ii = 25 Schedule found with 2
iterations in parallel
:*      Done
[!A0]  LDNW    .D2T2   *+B21(16),B4      ; |61|
<0,15>
||      MPYU2   .M1X    B4,A4,A9:A8       ; |61|
<0,15>

 [!A0]  LDNW    .D1T1   *+A28[A20],A3     ; |61|
<0,16>
||      MPYU2   .M1X    B4,A3,A5:A4       ; |61|
<0,16> :*-----------------------------------------
-----------------------------------*
```

```
#pragma omp for
//3 nested for loops
for (l = 0; l < FEATURE_SIZE-1; l += 4){
    imgInPix_64 = _mem8(&imgIn[(i+k)*(IMG_COLS) + j +l]);
    featureInPix_64 = _mem8(&featureIn[k*(FEATURE_SIZE)+l]);
    temp += _dotpsu4h(imgInPix_64, featureInPix_64);
}
imgOut[i*(OUTPUT_COLS) + j] = temp;
temp = 0;
```

```
;*      Searching for software
pipeline schedule at ...
;*         ii = 13 Schedule found
with 2 iterations in parallel

  LDNDW    .D1T1    *+A17[A16],A7:A6   ;
||DOTPSU4H .M2X    B5:B4,A9:A8,B5:B4  ;

  LDNDW    .D1T2    *+A18(24),B5:B4    ;
||DOTPSU4H .M2X    B5:B4,A5:A4,B5:B4  ;
```

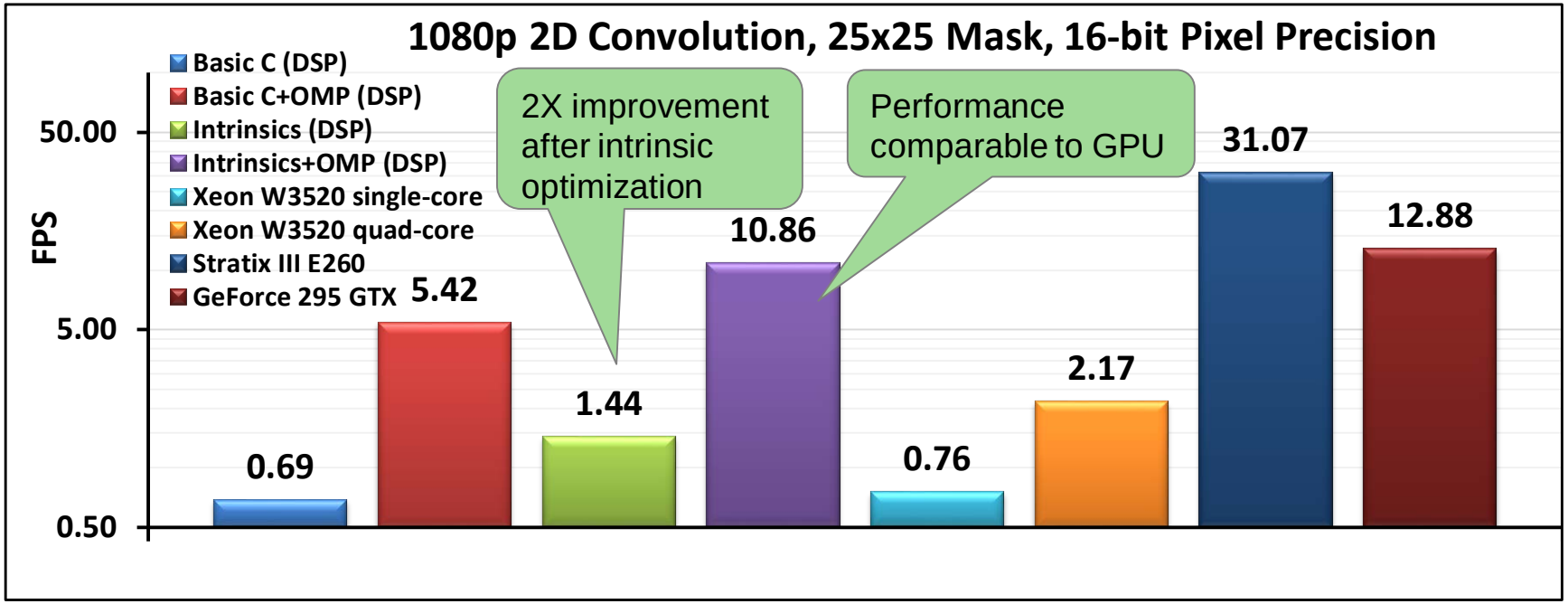
CHREC NSF Center for High-Performance Reconfigurable Computing

OpenMP used to parallelize across all 8 cores



UF UNIVERSITY of FLORIDA · Virginia Tech · BYU BRIGHAM YOUNG UNIVERSITY · THE GEORGE WASHINGTON UNIVERSITY

# Results and Conclusions



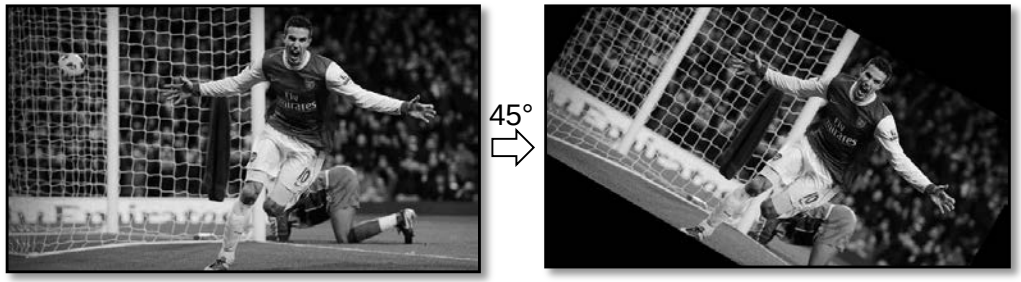

- TMS320C6678 DSP consumes only 10 Watts of power and achieves comparable performance to GPU at 144.5 Watts
- 2D convolution is an embarrassingly parallel algorithm, real-time performance can be achieved by adding more DSP devices
- DMA optimization should further improve performance, future exploration

FPGA, CPU, and GPU results from [3]

# Bilinear Interpolation w/ Image Rotation

- Used in medical-image registration
  - Typically in C-Arm X-ray machines where patient table cannot be rotated during procedures
- Algorithm
  - Kernel requires high memory bandwidth and large amounts of parallelizable computation
    - 15 arithmetic operations to obtain transformed pixel location and six to perform bilinear interpolation, total of 21 operations/pixel
  - Reads of pixel values will not be contiguous nor in strides
    - (x'y') – new pixel location; (x,y) – current pixel location; ($x_0$,$y_0$) – point of rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_o \\ y - y_o \end{bmatrix}$$



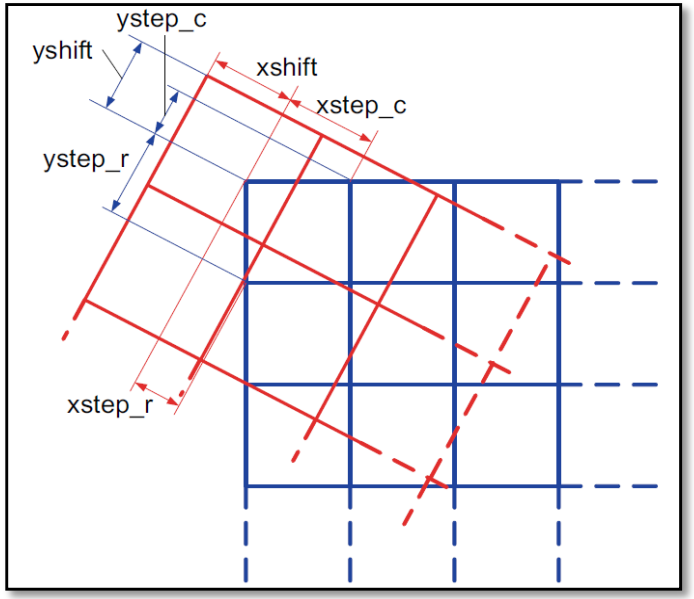
# DSP Design Optimization

- TMS320C6678 design
  - Naïve C-code depends on DSP's cache controller
  - Block-based DMA approach to improve performance
- Block-based image rotation [4]

```
Compute: xshift & yshift; //changes for every block
Compute:
xstep_c & ystep_c; //change in src x,y as we step across target cols
xstep_r & ystep_r; //change in src x,y as we step down target rows
xstart_r = xshift; ystart_r = yshift;
for (row = 0; row < TARGET_ROWS; row++)
{
    x = xstart_r; y = ystart_r;
    for (col = 0; col < TARGET_COLS; col++)
    {
        compute input addresses and fetch input data;
        compute coeffs;
        target_pixel[row, col] = interpolate;
        x += xstep_c; y += ystep_c;
}
        xstart_r += xstep_r; ystart_r += ystep_r;
}
```

Pseudo code for rotating block of image [4]

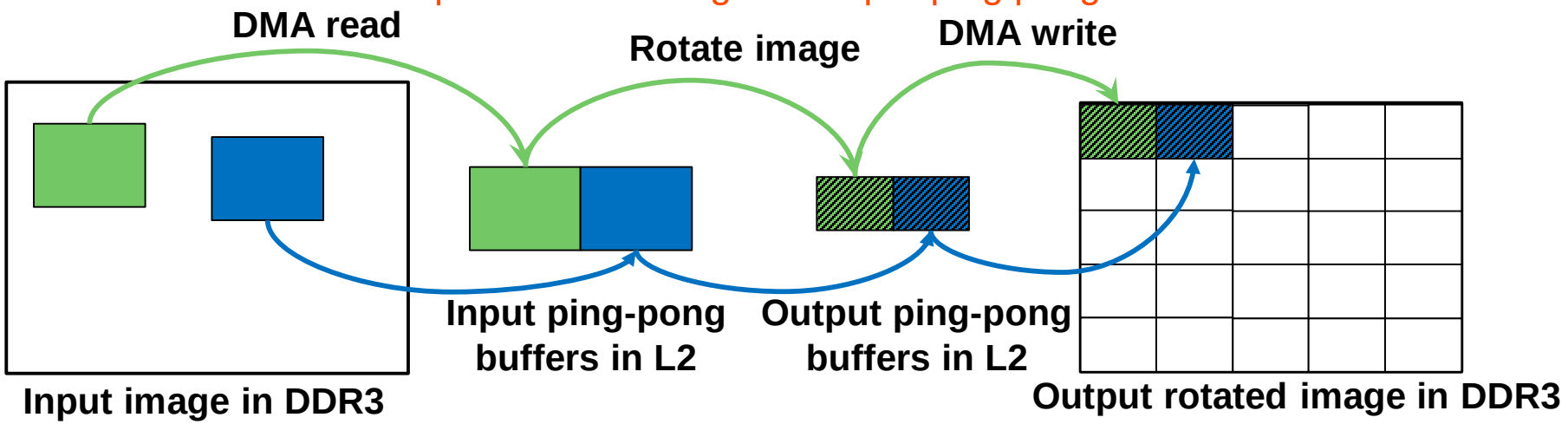
$$xstep\_r = \sin\theta; \quad ystep\_r = \cos\theta$$
$$xstep\_c = \cos\theta; \quad ystep\_c = -\sin\theta$$



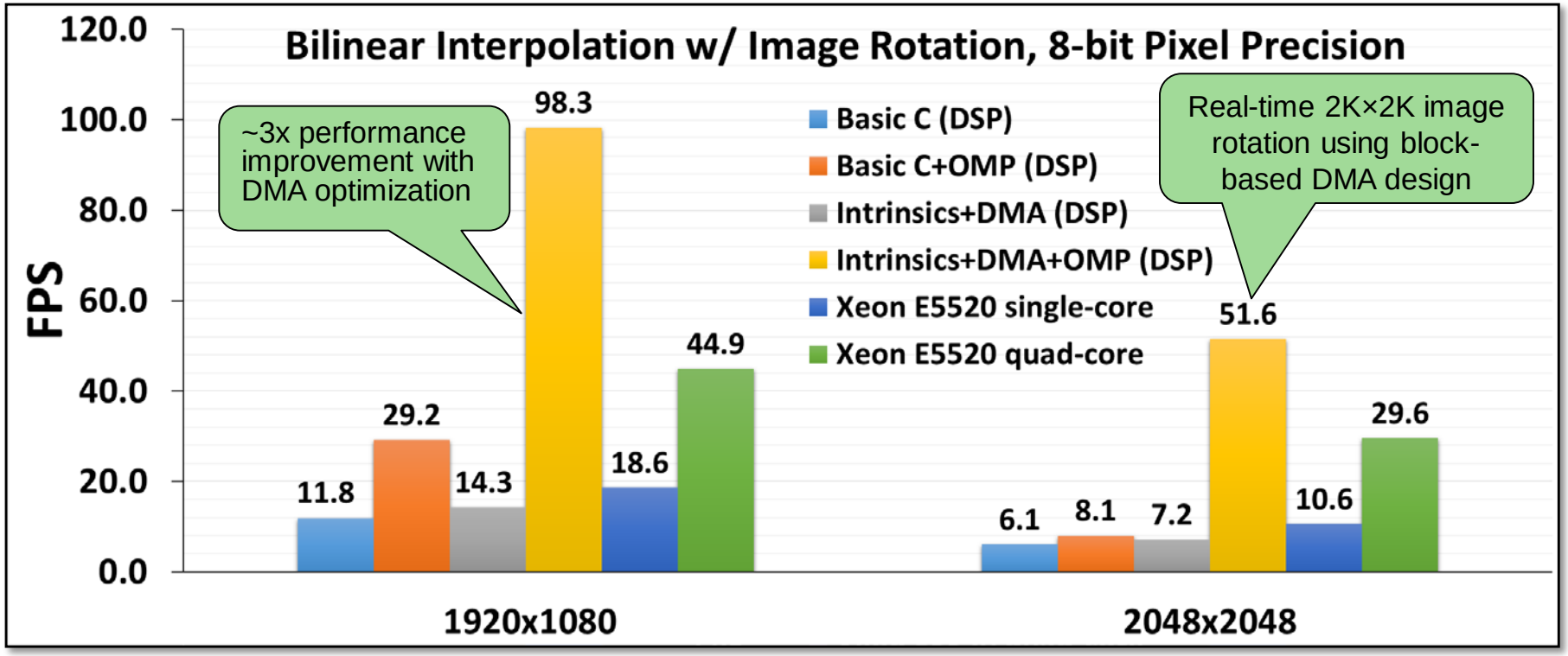

Block-based image rotation parameters [4]

# DMA Optimization

- Double buffering to mask data transfer time behind computation
  - Read $2K \times 2K$ image block to output rotated $K \times K$ image block
    - Starting location of block to be read given by xy_start_r
    - Reading 4 times the output-pixel count ensures that all pixels required for rotation are available in internal L2SRAM memory
  - Two input ping/pong buffers and two output ping/pong buffers in L2SRAM
  - Step 1: Calculate starting address of block to be read
  - Step 2: Initiate DMA transfer of $2K \times 2K$ input block to input ping/pong buffer
  - Step 3: Initiate DMA transfer of $K \times K$ block from output ping/pong buffer to DDR3
  - Step 4: Apply image rotation and bilinear interpolation on input ping/pong buffer and write $K \times K$ output block of image to output ping/pong buffer



# Results and Conclusions




- Scalable block-based design to rotate an image
- Ability to pre-fetch required block of data makes DSP a good choice for spatial transformations

# HPEC Challenge Benchmark Suite

- Kernels address key operations across variety of DoD image- and signal- processing applications [5]
  - Time-Domain FIR
  - Frequency-Domain FIR
  - Corner Turn
  - QR factorization
  - Constant False alarm rate detection
  - Graph Optimization via Genetic Algorithm
  - Database Operations
- Low-power, multi-core DSP architecture of TMS320C6678 well suited for high-performance embedded applications
  - Kernels benchmarked on TMS320C6678
    - Frequency-Domain FIR
    - Corner Turn

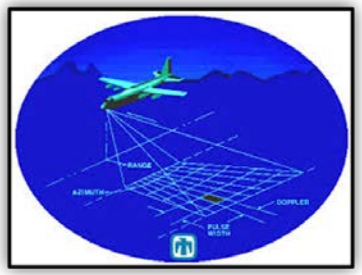

Image source:http://www.omgwiki.org/hpec/files/hpec-challenge/

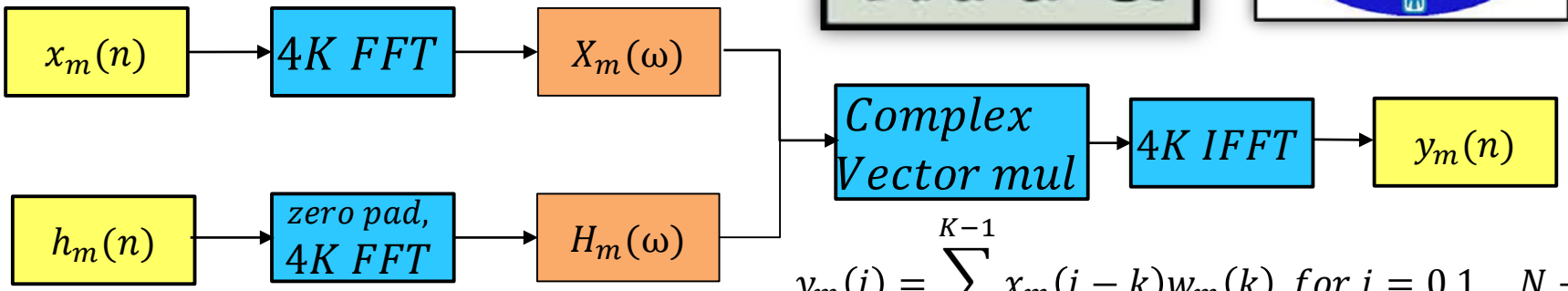


DoD: Department of Defense

# Frequency-Domain FIR (FDFIR)

- Used in front end of signal-processing applications
  - Synthetic Aperture Radar (SAR)
  - Speech-signal processing
- Algorithm

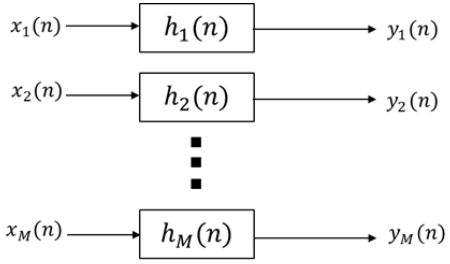



Dataset 1, single-precision floating-point

| Parameter | Description | Value |
|---|---|---|
| M | Number of filters | 64 |
| N | Length of input | 4096 |
| K | Number of filter coeffs. | 128 |
| W | Workload (MFLOPS) | 41.68 |

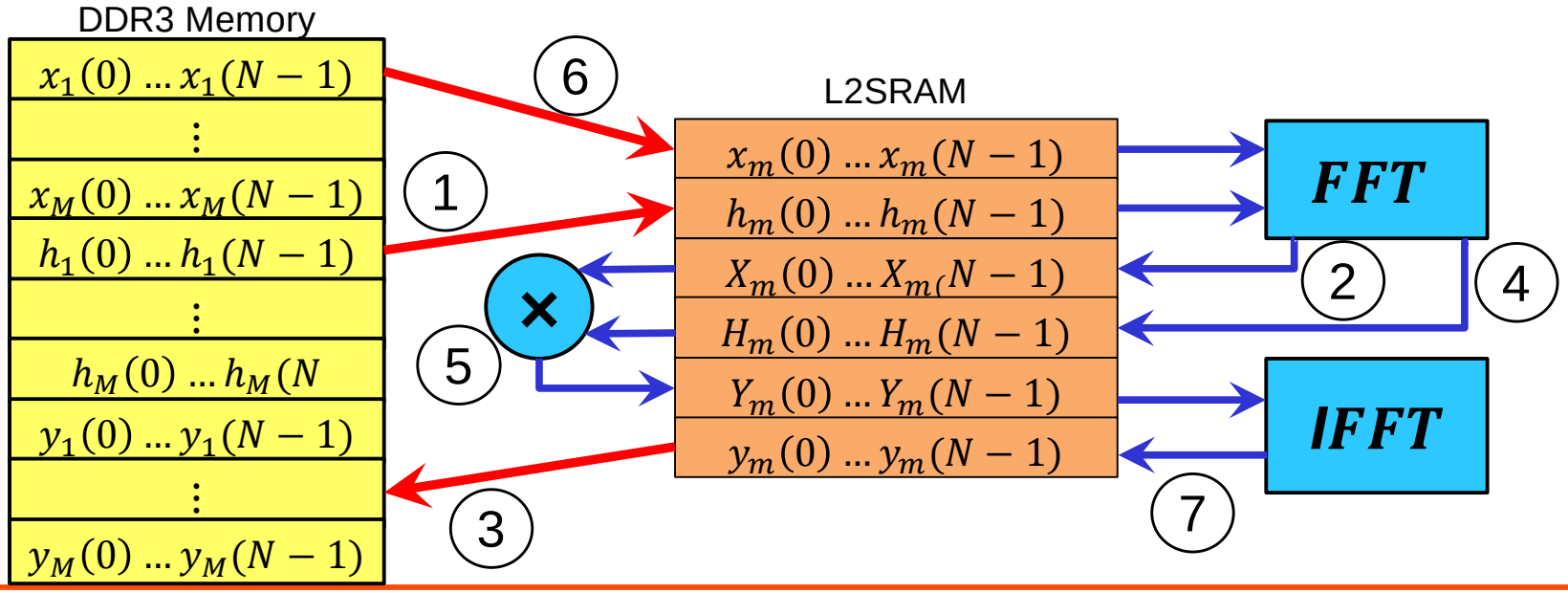
$$y_m(i) = \sum_{k=0}^{K-1} x_m(i-k)w_m(k), for\ i = 0,1 \ldots N-1$$

$$y_m = F^{-1}\{[F(x_m)\} \bullet [F(h_m)]\}$$

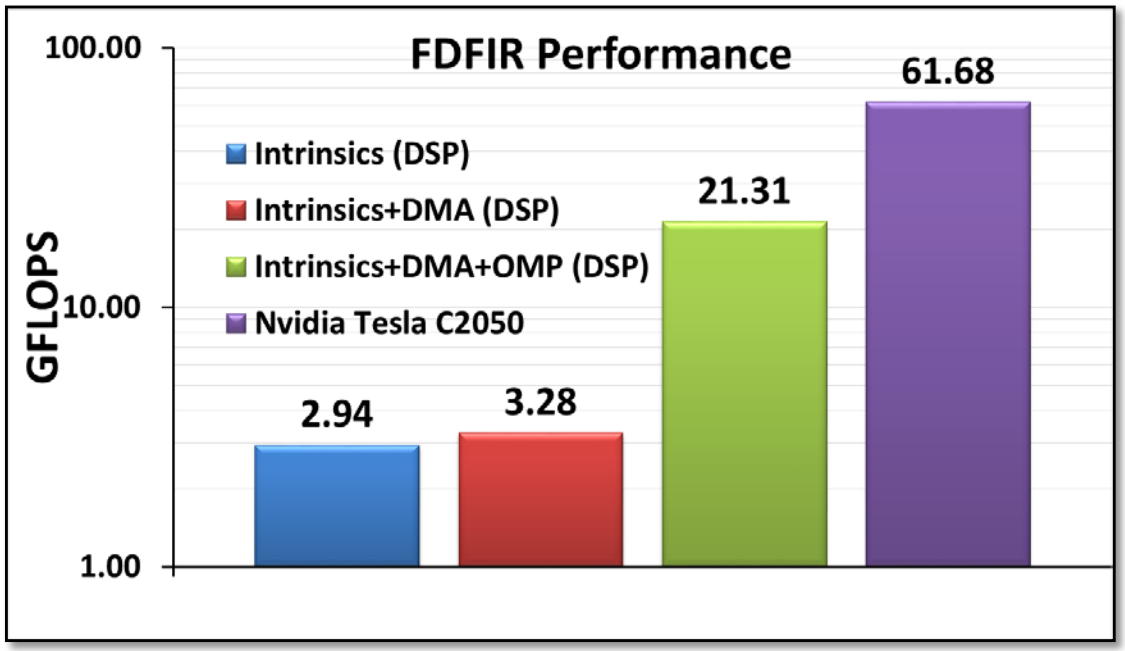


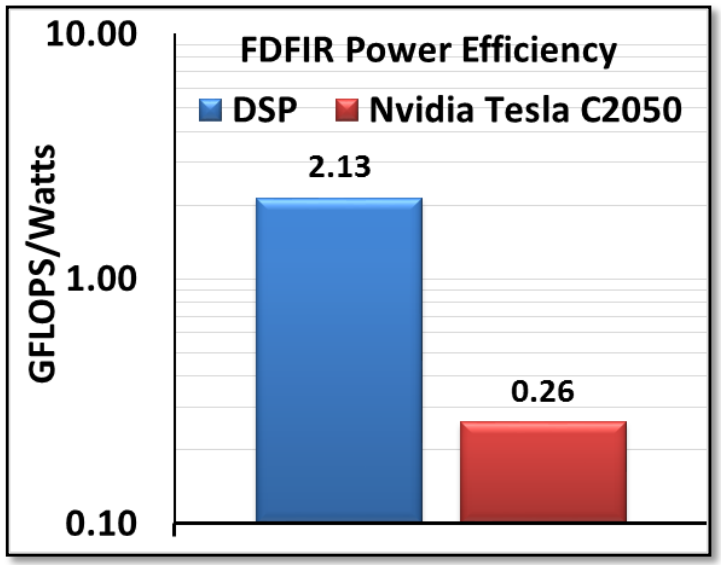
# DMA Optimization

- Hiding DMA transfers behind FFT
  - Step1: Initiate DMA transfer of filter coeffs. from DDR3 to L2SRAM filter coeff. buffer
  - Step2: Perform FFT of data in L2SRAM filter input buffer to obtain $X_m(\omega)$
  - Step3: Initiate DMA transfer of result vector from L2SRAM buffer to DDR3
  - Step4: Perform FFT of data in L2SRAM filter coeff buffer to obtain $H_m(\omega)$
  - Step5: Perform complex vector-vector multiplication of $X_m(\omega)$ and $H_m(\omega)$
  - Step6: Initiate DMA transfer for filter inputs from DDR3 to L2SRAM buffer
  - Step 7: Perform IFFT of data obtained from step5 to obtain result vector



# Results and Conclusions




DSP = 10 Watts; GPU = 238 Watts

- Both DSP and GPU are manufactured in 40nm technology making it fair comparison in terms of power efficiency
- Although GPU outperforms DSP by 2.9 times, power efficiency of our optimized DSP design is 8.2 times better than that of GPU
- TMS320C6678 DSP is a good choice in terms of power efficiency for algorithms with FFT/IFFT as their major computational load

CHREC NSF Center for High-Performance Reconfigurable Computing

Nvidia Tesla C2050 results from [6]



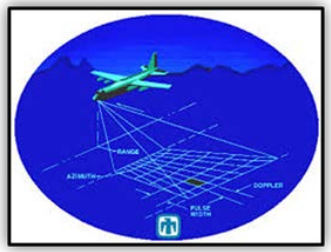
UF UNIVERSITY of FLORIDA Virginia Tech BYU BRIGHAM YOUNG UNIVERSITY THE GEORGE WASHINGTON UNIVERSITY

# Corner Turn (CT)

- Used in managing multi-dimensional data
  - 2D-FFT
  - Synthetic Aperture Radar

$$M = \begin{pmatrix} \overset{N}{\overbrace{\begin{matrix}1 & 2 & 3\\ 6 & 7 & 8\\ 11 & 12 & 13\end{matrix}}} & \begin{matrix}4 & 5\\ 9 & 10\\ 14 & 15\end{matrix}\\ \begin{matrix}16 & 17 & 18\end{matrix} & \begin{matrix}19 & 20\end{matrix}\end{pmatrix} \xrightarrow{\text{Transpose}} M^T = \begin{pmatrix} \overset{N^T}{\overbrace{\begin{matrix}1 & 6 & 11\\ 2 & 7 & 12\\ 3 & 8 & 13\end{matrix}}} & \begin{matrix}16\\ 17\\ 18\end{matrix}\\ \begin{matrix}4 & 9 & 14\end{matrix} & \begin{matrix}19\end{matrix}\\ \begin{matrix}5 & 10 & 15\end{matrix} & \begin{matrix}20\end{matrix}\end{pmatrix}$$

- Algorithm
  - For HPEC suite, it is matrix transposition (2D data)
    - Matrix elements stored in row-major format
    - Input matrix transposed and stored in row-major format
    - 750x5000 matrix, single-precision floating-point



- DSP design optimization [7]
  - Breaking input matrix into smaller sub-matrices and transpose sub-matrices to obtain final transposed matrix
  - Using optimized intrinsic code from C66x DSPLIB to transpose sub-matrix
  - DMA optimization using double buffering to improve performance

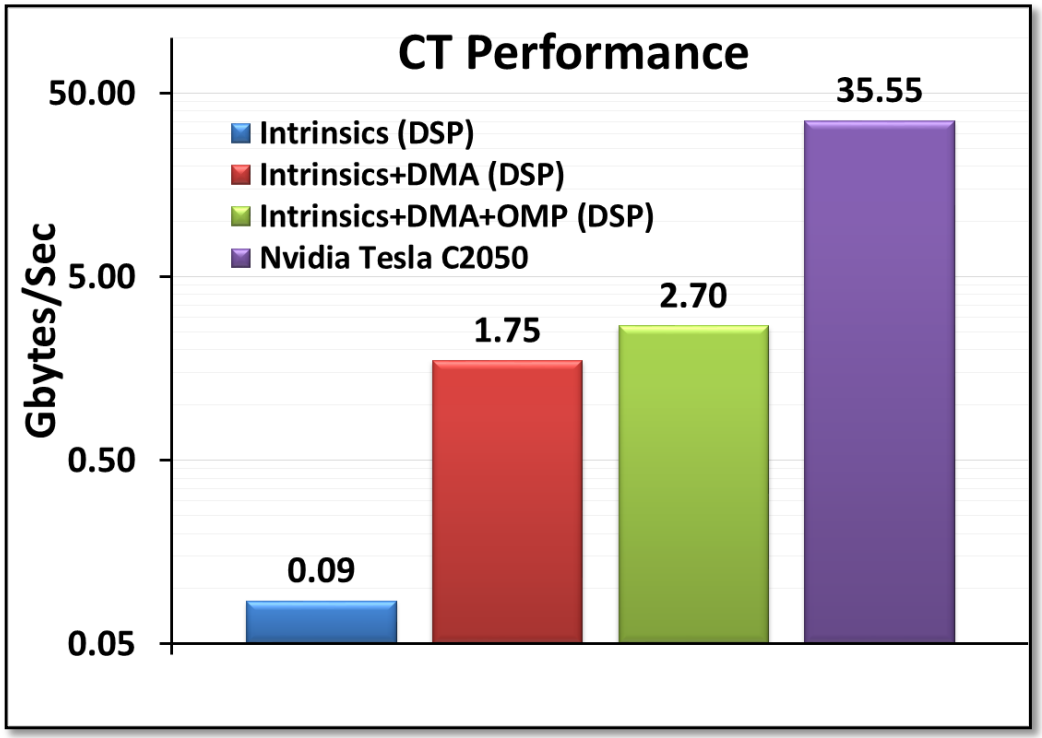
$$A = \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}$$ Memory storage of A is

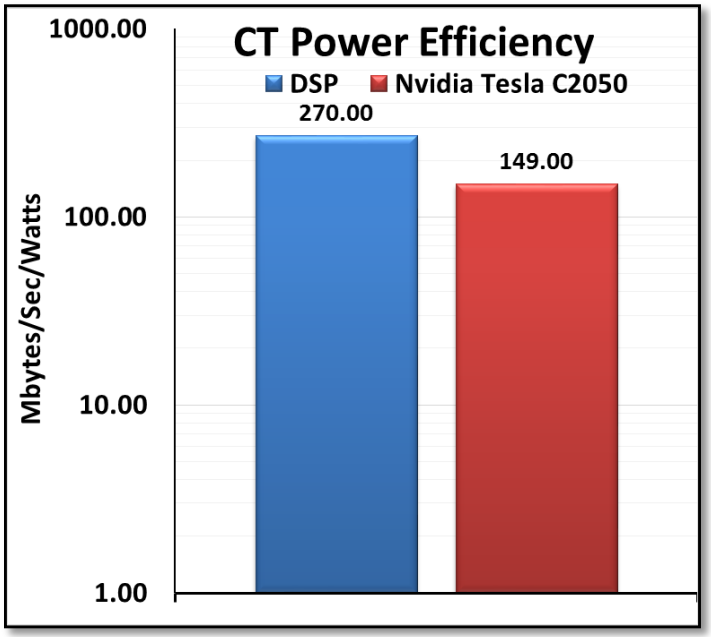
$$A = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9]$$

$$B = \begin{bmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{bmatrix} = A^T$$ Memory storage of B is

$$B = [1 \quad 4 \quad 7 \quad 2 \quad 5 \quad 8 \quad 3 \quad 6 \quad 9]$$

# Results and Conclusions




DSP = 10 Watts; GPU = 238 Watts

- GPU's higher external memory bandwidth over DSP leads to significantly better performance of CT on GPU
- Performance of CT is memory bound, increasing the number of DSP cores does not yield better performance

Nvidia Tesla C2050 results from [6]

# Conclusions

- 2D Convolution
  - Performance of TMS320C6678 comparable to Nvidia GeForce 295 GTX
  - 2D convolution is an embarrassingly parallel algorithm, real-time performance can be achieved by adding more DSP devices
- Bilinear Interpolation w/ Image Rotation
  - Real-time performance for 1920x1080 and 2048x2048 image resolutions on TMS320C6678
  - Ability to pre-fetch block using EDMA was exploited to gain better performance
- Frequency-Domain FIR
  - Power efficiency of TMS320C6678 8.2 times better than Nvidia Tesla C2050
- Corner Turn
  - Performance of kernel limited by external memory bandwidth
  - Power efficiency of TMS320C6678 1.8 times better than Nvidia Tesla C2050
- Summary
  - Optimized and evaluated performance of TMS320C6678 using four commonly used signal- and image-processing kernels
  - Compared performance against GPU, FPGA, and CPU
  - Results show that TMS320C6678 viable solution for today's HPEC applications vs. available traditional accelerator options

# References


[1] Tms320c6678 multicore fixed and floating-point digital signal processor, data manual. http://www.ti.com/lit/ds/sprs691c/sprs691c.pdf

[2] F. Iandola, D. Sheffield, M. Anderson, P. Phothilimthana, and K. Keutzer, "Communication-minimizing 2d convolution in gpu registers," in Image Processing (ICIP), 2013 20th IEEE International Conference, Sept 2013, pp. 2116–2120.

[3] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in International Symposium on Field Programmable Gate Arrays, ser. FPGA '12, Feb 2012, pp. 47–56.

[4] Implementation of affine warp using ti dsp. http://www.ti.com/lit/an/sprabc5/sprabc5.pdf

[5] Hpec challenge suite. http://www.omgwiki.org/hpec/files/hpec-challenge/

[6] S. Mu, C. Wang, M. Liu, D. Li, M. Zhu, X. Chen, X. Xie, and Y. Deng, "Evaluating the potential of graphics processors for high performance embedded computing," in Design, Automation Test in Europe Conference Exhibition (DATE), 2011, March 2011, pp. 1–6.

[7] D. Wang and M. Ali, "Synthetic aperture radar on low power multicore digital signal processor," in High Performance Extreme Computing (HPEC), 2012 IEEE Conference on, Sept 2012, pp. 1–6.

# Appendix

# Bilinear Interpolation w/ Image Rotation Intrinsic Optimization

- Processing x and y co-ordinates in parallel
  - **Co-ordinate packing:** *ITOLL* builds register pair by reinterpreting two 32-bit unsigned values
    - Used for packing pixel co-ordinates (x,y) and rotation parameters (xstep_c,ystep_c), and (xstep_r,ystep_r)
      - xystep_c = _itoll(xstep_c,ystep_c)
  - **Co-ordinate increments:** *DSADD* performs addition of two signed 32-bit values to produce two 32-bit signed results
    - Calculating new rotated pixel location with (xstep_c,ystep_c)
      - xy = _dsadd(xy,xystep_c)
  - **Out-of-bounds comparison:** *DCMPGT2* performs 4-way SIMD comparison of signed 16-bit values
    - Verify if newly calculated pixel location falls within image resolution
      - _dmpgt2(_itoll(x'y',xMaxyMax),_itoll(0,x'y'))

# DMA Optimization for CT

- Block-based DMA scheme [7]
  - Step1: Initiate DMA transfer for block of sub-matrix from DDR3 to L2SRAM input ping/pong buffer
  - Step2: Initiate DMA transfer for result from L2SRAM output ping/pong buffer to DDR3
  - Step3: Perform transpose on block of sub-matrix in L2SRAM input ping/pong buffer and write to L2SRAM output ping/pong buffer