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

Barath Ramesh*, Asheesh Bhardwaj†, Justin Richardson*, Alan D. George* and Herman Lam*
*NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611-6200
{ramesh, richardson, george, hlam}@chrec.org
†Texas Instruments
asheeshb@ti.com

Abstract—**Power efficiency is an important aspect in today's high-performance embedded computing (HPEC) systems. Digital signal processors (DSPs) are well known for their power efficiency and are commonly employed in embedded systems. Increasing computational demands in image- and signal-processing applications in embedded systems has led to the development of multi-core DSPs with floating-point capabilities. The TMS320C6678 is an eight-core, high-performance DSP from Texas Instruments that provides 128 GFLOPS of single-precision and 32 GFLOPS of double-precision performance under 10W of power. In this paper, we optimize and evaluate the performance of the TMS320C6678 DSP using two image-processing kernels, 2D convolution and bilinear interpolation with image rotation, and two signal-processing kernels, frequency-domain finite impulse response (FDFIR) and corner turn. Our 2D convolution results show that the performance of the TMS320C6678 is comparable to a Nvidia GeForce 295 GTX GPU and 5 times better than a quad-core Intel Xeon W3520 CPU. We achieve real-time performance for bilinear interpolation with image rotation on the TMS320C6678 for high-definition (HD) image resolution. Our performance per Watt results for FDFIR shows that the TMS320C6678 is 8.2 times better than the Nvidia Tesla C2050 GPU. For corner turn, although the raw performance of the Tesla C2050 is better than the TMS320C6678, the performance per Watt of TMS320C6678 is 1.8 times better than the Tesla C2050.**

I. Introduction

Today's high-performance embedded computing (HPEC) applications include real-time, HD image processing that require significant processing capabilities at low power. The increasing computational complexity of embedded-computing applications has fueled an increasing need for more powerful embedded architectures. For many of these applications, designers have often turned towards fixed accelerators such as graphics-processing units (GPUs) and also reconfigurable accelerators such as field-programmable gate arrays (FPGAs) to meet their computational requirements. However, due to power limitations and the need for fast turn-around times with embedded-system designs, these platforms may not be viable options in many cases. An alternative solution for such HPEC applications is the use of emerging high-performance multi-core DSP devices.

In this paper, we optimize and evaluate the performance of the TMS320C6678 device using two common image-processing kernels and two common signal-processing kernels. The image processing kernels chosen are 2D convolution and bilinear interpolation with image rotation. The 2D convolution kernel is one of the most commonly used algorithms in image processing and computer vision. In a survey of state-of-the-art algorithms for object recognition, convolution ranked as the most-employed algorithm [1]. Bilinear interpolation with image rotation is a computationally and memory-intensive image transformation which is commonly used in medical-image registration. The signal processing kernels were taken from HPEC Challenge benchmark suite [2]. From this suite, the two signal-processing kernels chosen are FDFIR and corner turn. These kernels address key operations across a variety of image- and signal-processing applications.

The contribution of this paper comes from multiple levels of optimization and analysis with four prominent kernels in image and signal processing on the TI TMS320C6678 multicore DSP, as well as comparison with other processors, in terms of speed and efficiency. Optimized designs of the four kernels under study were developed for the TMS320C6678 architecture, using various levels of DSP optimization. Our results show that the performance of 2D convolution on the TMS320C6678 is comparable to a Nvidia GeForce 295 GTX and 5 times better than a Xeon W3520 CPU [3]. For bilinear interpolation with image rotation, we achieve real-time performance for both 1080p and 2048×2048 resolutions on the TMS320C6678 DSP. The power efficiency of our optimized DSP design for FDFIR is 8.2 times better than the Nvidia Tesla C2050 GPU [4]. For corner turn, although the raw performance of the Tesla C2050 is better than the TMS320C6678, the performance per Watt of TMS320C6678 is 1.8 times better than the Tesla C2050.

The reminder of the paper is organized as follows. Section II gives a brief overview of the TMS320C6678 device architecture. Section III presents an overview of the kernels that were benchmarked for this study. Section IV describes how the kernels were optimally mapped on the TMS320C6678 architecture. Section V presents the benchmarking results obtained for the kernels, and Section VI concludes the paper with key results and possible future work.

II. TMS320C6678 Architecture Overview

The TMS320C6678 is an eight-core, high-performance DSP with both fixed-point and floating-point precision capa-

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

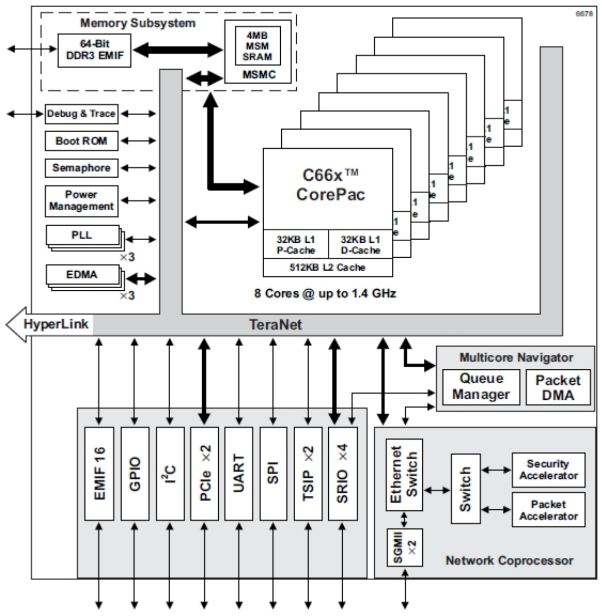

Fig. 1. TMS320C6678 device architecture [5]

bilities. As shown in the architecture diagram in Figure 1, the C66x CorePac is the heart of the device providing high-performance computing through instruction- and data-level parallelism. The EDMA3 peripheral allows for Direct Memory Access (DMA) between DDR3 and internal memory, which can be used to mask data transfer time behind computation.

A. *C66x CorePac*

The C66x CorePac is based on a Very Long Instruction Word (VLIW) architecture. VLIW architecture differs from Reduced Instruction Set Computing (RISC) or Complex Instruction Set Computing (CISC) architectures by having multiple execution units which can execute several instructions in parallel. The C66x CorePac has two identical data paths, (A and B), each with four unique functional units (M, L, S, and D). The M unit performs multiplication operations, while the L and S units handle addition, subtraction, logical, branching, and bitwise operations. The D unit is responsible for load/store and address calculations. All the functional units provide vector-processing capabilities using the SIMD instruction set included in the C66x CorePac. The SIMD instructions can operate on up to 128-bit vectors providing data-level parallelism within each core. With L, M, and S units on the two data paths, each core can perform eight single-precision or two double-precision multiply-add operations in one cycle. The TMS320C6678 also provides thread-level parallelism by scheduling application on the eight available cores.

B. *Memory hierarchy and data movement*

Each C66x CorePac is equipped with 32 KB of L1 data and program cache and 512 KB of L2 unified cache. The L1 and L2 on-chip memory can be dynamically configured as a cache, RAM or part RAM and cache. The eight cores in the device offer 256 KB of L1 and 4096 KB of L2 on-chip memory in total. There is an additional 4096 KB of internal memory shared across all eight C66x CorePacs. The device supports 64-bit DDR3 external memory operating up to 1600 MHz.

The EDMA3 peripheral enables efficient data movement between DDR3 and on-chip memory without CPU involvement. The ability to configure internal memory as part RAM and/or cache along with EDMA3 allows the developer to tune the architecture to their application or kernel needs.

III. Kernel overview

This section gives a brief overview of the kernels that were benchmarked on the TMS320C6678 device for this study. We explain the functionality and computational complexity involved in each of the kernels.

A. *2D convolution*

The 2D convolution kernel is a key component in many algorithms for object recognition, feature extraction, and image segmentation. As shown in Eq. 1, the input to the kernel consists of an image (I) of size $x \times y$ that is traversed by sliding a feature mask F (image to be matched) of smaller size $m \times n$ across all non-edge subspaces of I. Each input pixel in the window is multiplied with a constant in the feature mask and summed in order to obtain an output pixel. The size of the output image (O), after completely sliding the feature window, is $(x-m+1)\times(y-n+1)$. The 2D convolution kernel is a computation- and memory-intensive algorithm. A 1080p $(1920\times1080$ pixels) 16-bit image with a 25×25 feature mask requires $(1920-25+1)\times(1080-25+1) \approx 2$ million window calculations, requiring $2M \times 16 \times 625 \approx 20$ trillion non-sequential memory accesses, and ≈ 20 trillion calculations.

$$O[a][b] = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} I[a+i][b+j] \times F[n-i][m-j] \quad \text{where } a = 0,\ldots,x-m; b = 0,\ldots,y-n \tag{1}$$

B. *Bilinear interpolation with image rotation*

Image rotation is one of the most compute-intensive image transformations. The new pixel coordinate $(x^{'}, y^{'})$ of the rotated image pixel is obtained by performing matrix-vector multiplication as shown in Eq. 2, where (x, y) is the source pixel location, (x_o, y_o) is the point about which the image is rotated, and θ is the angle of rotation. Rotated pixel locations are constructed from the input pixel locations according to Eq. 2. Due to this mapping, rotated pixel locations often do not correspond to exact pixel locations in the input image. The output pixel values are then calculated using bilinear interpolation.

$$\begin{bmatrix} x^{'} \\ y^{'} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_o \\ y - y_o \end{bmatrix} \tag{2}$$

The kernel requires 15 arithmetic operations to obtain the transformed pixel location and six to perform bilinear interpolation. Therefore, the transform requires 21 operations per output pixel. Adding to this computational complexity is the memory-access pattern of the algorithm. Since the transformed pixel location is based on non-linear functions, input pixel values are read in a non-linear fashion.

C. FDFIR

FDFIR is an FIR filter implemented in the frequency-domain. FDFIR is a common signal-processing algorithm typically used in the front-end of applications such as synthetic-aperture radar and speech processing. The kernel consists of M FIR filters, where each FIR filter m, $m \in \{0, 1, \ldots, M-1\}$, has a set of filter coefficients $h_m[k], k \in \{0, 1, \ldots, K-1\}$. For an input vector x_m of length N, the convolved output y_m is given by Eq. 3. For a given FIR filter m, to obtain an output y_m, two Fast Fourier Transforms (FFTs), one complex vector-vector multiplication, and one Inverse FFT (IFFT) need to be performed.

$$y_m[i] = \sum_{k=0}^{K-1} x_m[i-k]h_m[k], \text{for } i = 0, 1, \ldots, N-1 \\ y_m = \mathscr{F}^{-1}\left[\mathscr{F}\{x_m\} \cdot \mathscr{F}\{h_m\}\right] \tag{3}$$

D. Corner turn

The HPEC Challenge benchmark suite defines corner turn as the change in the storage order of data in memory. In the case of multi-dimensional data such as a matrix, it refers to the transposition of elements. Eq. 4 shows the storage order of a 3×3 matrix, A, and the transposed storage order of A.

$$\begin{aligned} A &= [1 \quad 2 \quad 3; \quad 4 \quad 5 \quad 6; \quad 7 \quad 8 \quad 9] \\ A^T &= [1 \quad 4 \quad 7; \quad 2 \quad 5 \quad 8; \quad 3 \quad 6 \quad 9] \end{aligned} \tag{4}$$

IV. Mapping kernels to TMS320C6678

In this section we describe the various optimization strategies used to map the kernels on the TMS320C6678 device architecture. In our optimized design of the kernels, the L2 on-chip memory is partitioned into part RAM and part cache with the RAM section referred to as L2SRAM. The L1 on-chip data memory was configured as cache.

A. 2D convolution

For our DSP-optimized design, the 2D convolution kernel was written in C and then optimized using C66x SIMD intrinsics. The input image and feature mask are stored in DDR3. The output image, after convolving the input and feature mask, is written back to DDR3. The SIMD intrinsics, MEM8 and DOTPSU4H, allow for *quad-word* processing on 16-bit precision pixels as summarized below:

1) Pixel read: MEM8 allows unaligned loads of four bytes from memory. MEM8 can be used to read four pixels of input and feature mask of the sliding window.

2) Dot product: DOTPSU4H multiplies four, signed 16-bit values by four, unsigned 16-bit values and returns the 32-bit sum. DOTPSU4H can therefore be used to perform the dot product of four input and four feature pixels read using the MEM8 SIMD intrinsic.

Figure 2 shows an excerpt of our C code for the primary computation loop with intrinsic optimizations for the 2D convolution. Intrinsics reduce the iteration count of the innermost **for** loop by four times. Also, after the compiler software pipelines the **for** loop, the iteration interval for our C code is twice that of the optimized code with intrinsics.

```
for (k = 0; k < FEATURE_SIZE; k++){
  for (l = 0; l < FEATURE_SIZE; l++){
    // Basic C
    out += imgIn[(i+k)*IMG_COLS + (j+l)]
           * featureIn[k][l];
    // SIMD intrinsic code
    imgInPix_64 = _mem8(&imgIn[(i+k)*(IMG_COLS) + j +l]);
    featureInPix_64 = _mem8(&featureIn[k*(FEATURE_SIZE)+l]);
    outInt += _dotpsu4h(imgInPix_64, featureInPix_64);
   }
 }
```

Fig. 2. C code and intrinsics for 2D convolution to get one output pixel

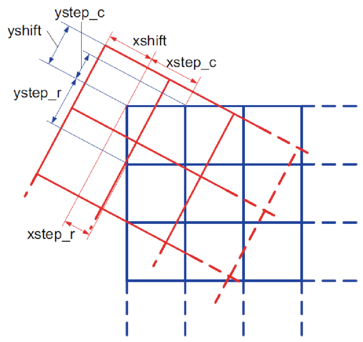

Fig. 3. Parameters involved in block-based image rotation [6]

B. Bilinear interpolation with image rotation

The naïve C code for the image rotation algorithm on the TMS320C6678 DSP depends on the cache controller's ability to keep the data paths busy. Given the complex nature of the algorithm, it is not the best way to take advantage of the DSP architecture and resources. A block-based image rotation algorithm was proposed by the authors of [6] for 64x64x64 ultrasound images. We leverage the algorithm proposed in [6] for 1080p image resolution with DMA optimization. Following a block-based approach using DMA, bilinear interpolation achieved a performance improvement of 3.4 times the naïve code for 1080p images. Figure 3 shows the parameters involved in the block-based rotation algorithm. Eq. 5 shows the relation between the parameters in Figure 3 and Eq. 2. The $xshift$ and $yshift$ parameters correspond to the first pixel value of an output block.

$$\begin{aligned} xstep\_r &= \sin\theta; \quad ystep\_r = \cos\theta \\ xstep\_c &= \cos\theta; \quad ystep\_c = -\sin\theta \end{aligned} \tag{5}$$

The following two types of optimizations were applied to improve the performance of the kernel on the DSP:

1) Intrinsic optimization: SIMD capabilities of the C66x CorePac can be used in the block-based image rotation in a couple of ways.

a) Addition: The C66x CorePac can perform four 32-bit addition operations in one cycle. The DSADD intrinsic performs an addition of two signed 32-bit values to produce two 32-bit signed results.

b) Comparison: The C66x CorePac can perform four 32-bit comparison operations in one cycle. The DCMPGT2 intrinsic performs a 4-way SIMD comparison of signed 16-bit values. The results are packed into the four least-significant bits of the return value.

```
// changes for every block
Compute: xshift & yshift;
Compute:
xy_step_c = _itoll(xstep_c,ystep_c);
xy_step_r = _itoll(xstep_r,ystep_r);
xy_start_r = _itoll(xshift,yshift);
xMaxyMax = (COLS)<<16|(ROWS);

for (row = 0; row < TARGET_ROWS; row++)
{
    xy = xy_start_r;
    for (col = 0; col < TARGET_COLS; col++)
    {
        _dcmpgt2(_itoll(xy,xMaxyMax),_itoll(0,xy))
        {
            compute input addresses and fetch input data;
            compute interpolate;
            target_pixel[row, col] = interpolate;
            xy = _dsadd(xy,xy_stepc);
        }
    }
    xy_start_r = _dsadd(xy_start_r,xy_step_r);
}
```

Fig. 4. Pseudo-code for rotating an image block with intrinsic optimization

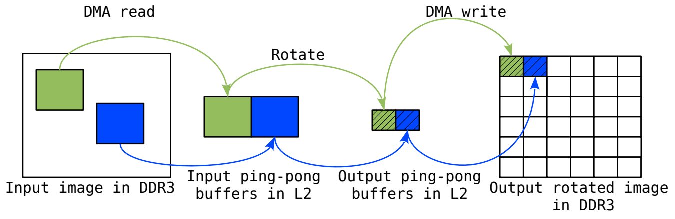
DMA read
Rotate
DMA write
Input image in DDR3
Input ping-pong buffers in L2
Output ping-pong buffers in L2
Output rotated image in DDR3

Fig. 5. Software architecture for image rotation

Pixel coordinates can be packed into a single 64-bit value using the ITOLL intrinsic. ITOLL builds a register pair by reinterpreting two 32-bit unsigned values, enabling the use of SIMD intrinsics as shown in Figure 4.

2) DMA optimization: Due to the non-linear, function-based reads in the image rotation, it is impossible to read the exact block of pixels from input image. In order to ensure that all input pixel values are available, a $2K \times 2K$ block of the input image values is read into L2SRAM for rotating an image of block of size $K \times K$. The first transformed pixel location of an output block determines the starting location of the input block to be read. Thus, we read four times the output-pixel count per block to ensure that all pixel values required for the transform are available in L2SRAM. Then, for each pixel location in the output block, the mapping is calculated based on Eq. 2, followed by bilinear interpolation. The above method is employed for all output blocks to obtain a rotated image of the input. In order to mask the DMA transfer time, L2SRAM is split into four buffers, two for input-block read and two for output-block write. Figure 5 shows the ping-pong strategy used to mask DMA transfer time behind computation.

C. FDFIR

Figure 6 shows the software architecture of the FDFIR benchmark. Complex filter inputs x_m and corresponding zero-padded filter coefficients h_m are stored in external DDR3 memory. Outputs from each filter y_m are written back to DDR3 external memory. Six buffers are allocated in L2SRAM: two input buffers to hold x_m and h_m; two buffers to write back data after performing an N-point FFT on the first two buffers; a buffer to store results after complex vector-vector multiplication; and a buffer to store output y_m obtained by performing IFFT on the data in the previous buffer. EDMA3 was used to efficiently transfer data between DDR3 and L2SRAM. The following steps were used to maximize performance by overlapping DMA transfers and computation:

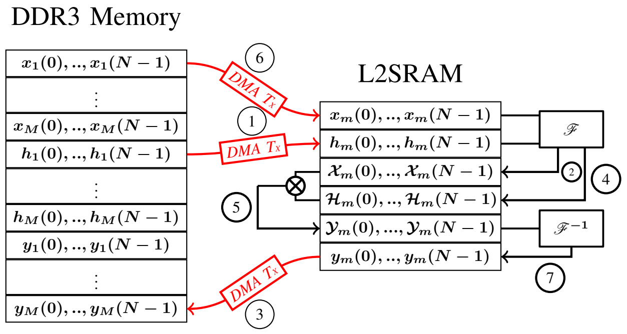

Fig. 6. FDFIR software architecture

1) Initiate DMA transfer to read filter coefficient h_m from DDR3 to input buffer in L2SRAM
2) Perform N-point FFT on x_m and write result X_m to buffer in L2SRAM
3) Initiate DMA transfer to write result y_m obtained in previous iteration from L2SRAM to DDR3
4) Perform N-point FFT on h_m and write output H_m to buffer in L2SRAM
5) Perform complex, vector-vector multiplication of X_m and H_m and write result Y_m to buffer in L2SRAM
6) Initiate DMA transfer to read filter input x_m required in the next iteration
7) Perform IFFT of Y_m and write result y_m to buffer in L2SRAM

Using this approach, the time required for every DMA transfer is masked by an FFT operation. FFT and IFFT operations are performed using optimized code with intrinsics available in TI's C66x DSPLIB [7].

D. Corner turn

The corner turn benchmark in the HPEC Challenge benchmark suite is equivalent to matrix transposition. We leveraged the block-based DMA approach proposed in [8]. Figure 7 shows the ping-pong strategy employed for block-based matrix transpose. EDMA3 allows for efficient block transfers between the DDR3 and L2SRAM. Optimized code with intrinsics available in TI's C66x DSPLIB is used for transposing a matrix block in L2SRAM. Although matrix transpose can be performed using only EDMA3, the employed scheme allows for the use of the fast internal memory and the DSP's logical units to transpose the matrix.

V. Results

All kernels were evaluated using a TMDXEVM6678LE evaluation module (EVM). This EVM has a TMS320C6678 device clocked at 1 GHz and equipped with 512 MB of DDR3 memory. After the design of the kernels was optimized for a single DSP core, OpenMP was used to parallelize the kernels across all eight cores of the device. We developed basic C code for the image-processing kernels and leveraged optimized

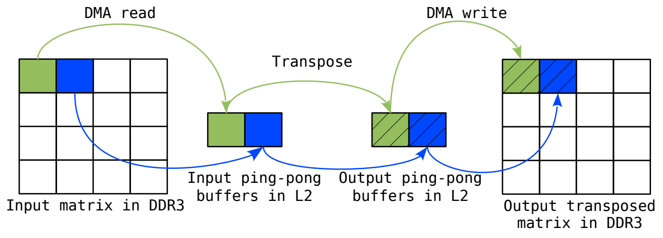
DMA read
Transpose
DMA write
Input matrix in DDR3
Input ping-pong buffers in L2
Output ping-pong buffers in L2
Output transposed matrix in DDR3

Fig. 7. Corner Turn software architecture

intrinsic-code available from TI's C66x DSPLIB for the signal-processing kernels. Various levels of DSP optimization were performed and are denoted as follows:

1) ***Basic C***: Plain C code ported onto single core of TMS320C6678 device
2) ***Basic C+OMP***: Basic C code parallelized across eight cores of TMS320C6678 device using OpenMP
3) ***Intrinsics***: Performance on single core of TMS320C6678 with intrinsic optimizations
4) ***Intrinsics+DMA***: Performance on single core of TMS320C6678 with intrinsic and DMA optimizations
5) ***Intrinsics+DMA+OMP***: Intrinsics+DMA design parallelized across eight cores of TMS320C6678 using OpenMP

A. 2D convolution

The performance of the 2D convolution kernel for HD image resolution was studied on an FPGA using highly optimized circuit architecture by Fowers et al. in [3]. To the best of our knowledge, [3] is the best performing implementation of 2D convolution on an FPGA. Figure 8 shows the performance comparison between the TMS320C6678 DSP, an Nvidia GeForce 295 GTX GPU, an Intel Xeon W3520 CPU (single- and quad-core), and an Altera Stratix III E260 FPGA in frames per second (FPS). The FPGA, GPU, and CPU results were obtained from [3] with input image of size 1920×1080, feature mask of size 25×25, and 16-bit fixed-point precision for pixels. Results show that performance of the DSP is 5 times better than the CPU and comparable to the GPU. As expected, the intrinsic-based code leads up to 2 times better performance over our basic C code on the DSP. The TMS320C6678 DSP consumes only 10W of power and achieves comparable performance to that of a GPU at 144.5W, making it a good choice for power-efficient applications that involve 2D convolution. Although the FPGA outperforms the DSP, one has to consider that the design effort and time to implement an optimized design on the DSP is much lower than that on an FPGA. Also, 2D convolution is an embarrassingly parallel algorithm and, thus, real-time performance for 2D convolution can be achieved by adding more DSP devices.

B. Bilinear interpolation with image rotation

Figure 9 shows the performance of the DSP for various levels of optimization. The DSP-optimized design uses 8-bit fixed-point precision for pixel values. Cosine/sine tables for a step size of 1° were stored in the shared-memory space with 16-bit, fixed-point, signed precision. The performance of the DSP for various optimizations was tested using an 8-bit

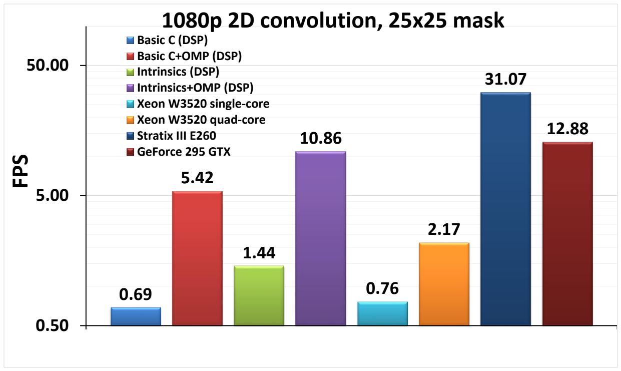

Fig. 8. Performance of 2D convolution

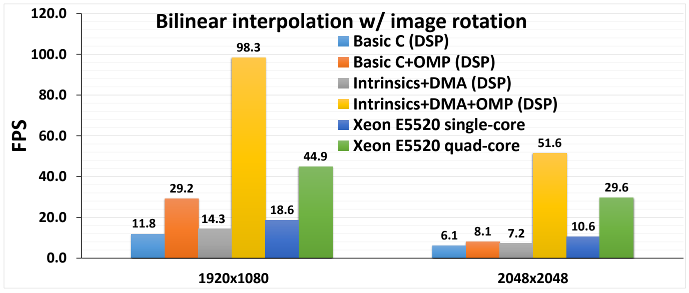

Fig. 9. Performance of bilinear interpolation with image rotation

gray-scale image of HD (1080p) and 2048×2048 resolutions. Results show that real-time performance (98.3 FPS) can be achieved on the DSP using intrinsics+DMA+OMP optimization. The non-linear memory access pattern of the kernel makes it inefficient to implement on cache-based system. The ability to pre-fetch the block of image required for computation using EDMA3 on the DSP makes the device a good choice for kernels involving spatial transformations.

C. FDFIR

As shown in Figure 10, we compare our DSP-optimized design for FDFIR with the Nvidia Tesla C2050 GPU result obtained from [4]. In [4], the authors implement the complete HPEC Challenge benchmark suite on an Nvidia Tesla C2050 GPU and claim to achieve better performance than all previously published results using GPUs. Both the DSP and the Nvidia Tesla C2050 GPU are manufactured using 40nm process technology, thus making it a reasonable comparison in terms of power efficiency. FDFIR results shown in Figure 10 are for Set 1 [2] of the HPEC Challenge benchmark suite. The parameters used are $M = 64, N = 4096, K = 128$, and single-precision complex data. The performance of FDFIR on all eight cores of the DSP is 21.31 GFLOPS. Considering that the DSP consumes 10W of power, the performance per Watt for the DSP is calculated to be 2.13 GFLOPS/W. The GPU's performance for FDFIR is 61.68 GFLOPS, with a power consumption of 238W, so the performance per Watt of the GPU is calculated to be 0.26 GFLOPS/W. Thus, although the performance of the GPU is 2.9 times better than the DSP, the power efficiency of our optimized DSP design is 8.2 times better than that of the GPU. It can be noted from Figure 10 that intrinsics+DMA based design leads to 11% performance increase over an intrinsics-only based code. The performance

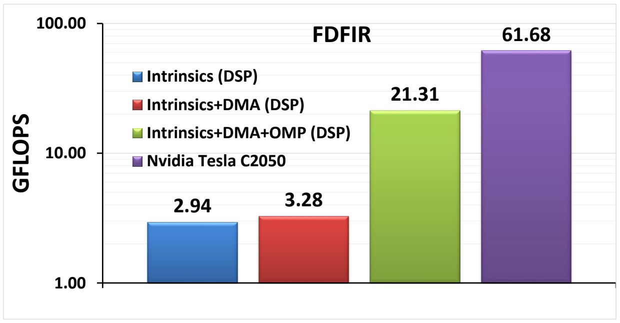

Fig. 10. Performance of FDFIR

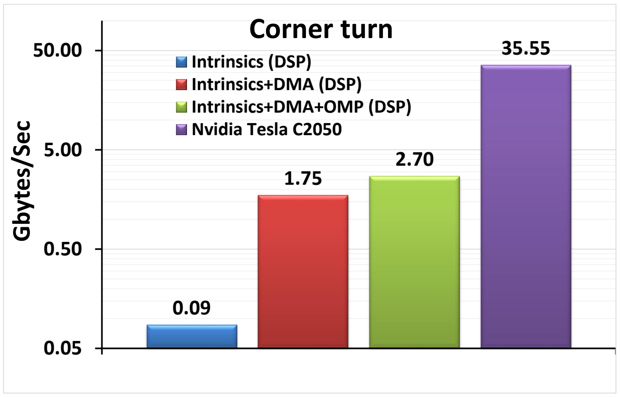

Fig. 11. Performance of corner turn

of FDFIR is dominated by FFT/IFFT and our results show that the TMS320C6678 DSP is a good choice in terms of power efficiency for algorithms with FFT/IFFT as their major computational load.

D. Corner turn

The corner turn results shown in Figure 11 are for Set 2 of the HPEC Challenge benchmark suite, a 750×5000 matrix of single-precision, floating-point data. The Nvidia Tesla C2050 GPU result for corner turn was obtained from [4]. Results show that using DMA improves the performance of corner turn by 20 times on the DSP (intrinsics vs. intrinsics+DMA). The performance of the DSP for corner turn on all eight cores is 2.70 Gbytes/Sec, thus the performance per Watt is calculated to be 270 Mbytes/Sec/W. The performance of the GPU is 35.55 Gbytes/Sec and hence the performance per Watt is calculated to be 149 Mbytes/Sec/W. Corner turn is a memory-intensive operation requiring high external-memory bandwidth. The GPU's higher external bandwidth over the DSP leads to a significantly better performance of the kernel on the GPU. But in terms of performance per Watt, the DSP is 1.8 times better than the GPU. The performance of corner turn is dominated by I/O and so increasing the number of DSP cores does not yield better performance once the DDR3 memory bandwidth is saturated. Hence, moving to an eight-core design using OpenMP on the DSP leads to a speedup of only 1.54 times over a single core.

VI. CONCLUSIONS

With growing computational demands of signal- and image-processing applications, power efficiency has become an important factor in HPEC systems. DSPs are commonly employed in low-power embedded systems. But the advent of multi-core DSPs, such as the TMS320C6678, enables DSPs to compete better in the HPEC domain. In this paper, we have optimized and evaluated the performance of the TMS320C6678 DSP device using four commonly used image- and signal-processing kernels. We compared our results for the TI TMS320C6678 with optimized CPU, GPU, and FPGA implementations.

Our 2D convolution results show that the performance of the TMS320C6678 DSP is comparable to that of a Nvidia GeForce 295 GTX GPU and 5 times better than a quad-core Xeon W3520 CPU with better power efficiency. There is potential for additional performance improvement using DMA for 2D convolution, which the authors plan to explore in future work. For bilinear interpolation with image rotation, we achieve real-time performance for both 1080p and 2048×2048 resolutions on the DSP. The power efficiency of the optimized DSP design for FDFIR is 8.2 times better than the Nvidia Tesla C2050 GPU. For corner turn, the power efficiency of the optimized DSP design is 1.8 times better than Tesla C2050 GPU. In summary, current-generation, multi-core DSPs, such as TI's TMS320C6678, can provide a viable solution for today's HPEC applications verse existing, traditional accelerator options.

ACKNOWLEDGMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022.

REFERENCES

[1] 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 on*, Sept 2013, pp. 2116–2120.

[2] Hpec challenge suite. [Online]. Available: http://www.omgwiki.org/hpec/files/hpec-challenge/

[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] 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.

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

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

[7] Tms320c64x+ dsp little-endian dsp library programmers reference. [Online]. Available: http://www.ti.com/lit/ug/sprueb8b/sprueb8b.pdf

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