Platform Characterization for Domain-Specific Computing

Alex Bui,[1] Kwang-Ting (Tim) Cheng,[2] Jason Cong,[3]
Luminita Vese,[4] Yi-Chu Wang,[2] Bo Yuan,[3] and Yi Zou[3]

[1]Department of Radiological Sciences, University of California, Los Angeles, CA 90095, USA

[2]Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA

[3]Department of Computer Science, University of California, Los Angeles, CA 90095, USA

[4]Department of Mathematics, University of California, Los Angeles, CA 90095, USA

Abstract— We believe that by adapting architectures to fit the requirements of a given application domain, we can significantly improve the efficiency of computation. To validate the idea for our application domain, we evaluate a wide spectrum of commodity computing platforms to quantify the potential benefits of *heterogeneity* and *customization* for the domain-specific applications. In particular, we choose medical imaging as the application domain for investigation, and study the application performance and energy efficiency across a diverse set of commodity hardware platforms, such as general-purpose multi-core CPUs, massive parallel many-core GPUs, low-power mobile CPUs and fine-grain customizable FPGAs. This study leads to a number of interesting observations that can be used to guide further development of domain-specific architectures.

I. Introduction

The goal of the Center for Domain-Specific Computing (CDSC)[10] is to develop domain-specific hardware architectures, and the software systems to greatly improve the performance and the energy efficiency of domain-specific applications. One of the application domains selected by the center is medical imaging, given its significant impact of the healthcare industry. To achieve our goal, it is essential to benchmark how well current commodity platforms perform, and identify opportunities for architectural innovations.

This paper provides a summary of our efforts to characterize the experimental platforms. We first describe several experimental platform candidates in Section II. We have two server-class platforms, one of which integrates the CPU, GPU and FPGA together; the other one combines Xeon and Atom processor in one server. Additionally, we also investigate a mobile-class platform that is based on an ARM-based SOC. We then describe the application domain and the applications used in this study in Section III. Experimental results, along with analysis and discussions are presented in Section IV.

II. Hardware Platforms

A. Server-Class Platform A: CPU+GPU+FPGA

We want to set up a server-class platform that integrates multi-core CPU and popular accelerator units (GPU and FPGA) in a tightly coupled fashion. The platform is used to demonstrate the benefit of heterogenous coprocessor acceleration and the customization capability of FPGAs.

We use the Convey HC-1ex [2] as our baseline platform. The motherboard has two PCI-e X16 slots, but there is no physical space to host a double-width GPU (e.g., GTX280) or Tesla compute card due to form-factor issues. Currently we use a PCI-express expansion box to host a Tesla compute card C1060. Fig. 1 shows the structures of the coprocessor hardware of the Convey HC-1ex. The HC-1ex uses 4 Xilinx Virtex6 LX760 as the user FPGAs. The CPU and different FPGAs access the off-chip memory using a shared memory model. The system employs an on-board crossbar to realize the interconnection. Cache coherence is also handled through the FSB protocols. Each FPGA has 16 external memory access channels. (Eight physical memory ports are connected to eight memory controllers which run at 300MHZ. The core design runs at 150MHZ. Thus, effectively the design on each FPGA is presented with 16 "logical" memory access channels through time multiplexing.) The Convey HC-1ex provides a very large bandwidth (80GB/s peak), and with 16GB capacity for coprocessor side memory. In practice, we observe that around 30% to 40% of the peak bandwidth can be easily obtained. The FPGA-side off-chip memory system is designed to better support interleaved (short) data access rather than traditional cache-line burst access.

B. Server-Class Platform B: Xeon+Atom

This platform is used to study the heterogeneity across multiple CPUs. The diagram of the platform is shown in Fig. 2.

The platform consists of one dual-core Intel® Atom™ Processor 330 and one quad-core Intel® Xeon® Processor E5450. This kind of experimental platform is considered to be a perfect heterogeneous system for evaluation [14]. The Atom and Xeon processors represent two opposite types of microarchitecture. The Xeon employs a high-performance server-class microarchitecture, while the Atom employs a low-power microarchitecture targeted for mobile devices.

The microarchitectural parameters of these two processors are shown in Table I. The Atom processor uses an in-order issue, a narrower issue width, and has a much smaller L2 cache.

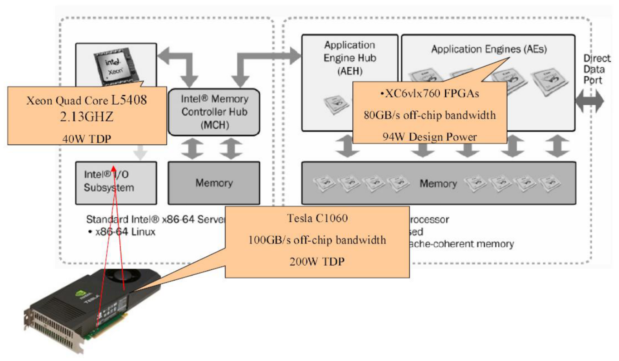

Fig. 1. Diagram of the Convey HC-1ex Hybrid Computer

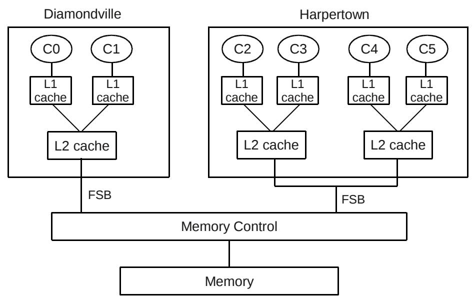

Fig. 2. Intel QuickIA Platform Diagram

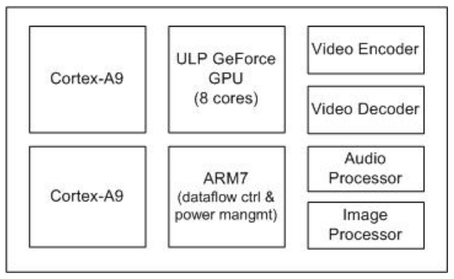

Fig. 3. Diagram of the Tegra 2 SOC

TABLE I
MICROARCHITECTURAL PARAMETERS OF THE THREE DIFFERENT PROCESSORS

Class	Xeon	Atom	ARM
Name	Xeon L5408	Atom 330	Tegra 2
Code Name	Harpertown	Diamondville	Cortex-A9
Processors	1	1	1
Cores	4	2	2
Clock Freq	2.13 GHz	1.6 GHz	1.0GHz
Issue Width	4 inst issue	2 inst issue	2 inst issue
Execution	out-of-order	in-order	out-of-order
Inst TLB	128-entry, 4-way	32-entry, full	32-entry, full
Data TLB	256-entry, 4-way	64-entry, 4-way	32-entry, full
L1 Cache	32/32KB	32/24KB	32/32KB
L2 Cache	12MB, 24-way	1MB, 8-way	1MB
FSB Freq	1066 MHz	533 MHz	N/A
Process	45nm	45nm	40nm

The FSB frequency of the Xeon is degraded to 533GHz because we need to run the front side bus at a speed that works for the Atom processor. Because of this, we use the quad-core Xeon L5408 in the Convey HC1-ex server rather than the Xeon E5450 in the Xeon+Atom platform to obtain the performance and energy of a server-class Xeon processor.

C. Mobile-Class Platform

Modern smartphones have a design specification that provides over 100GOPS workload for cellular communication, voice/audio/video processing, and graphics rendering within a 1W power budget [15]. The solution to achieving such an aggressive specification is a heterogeneous multi-core SoC where each core is highly specialized for a set of applications and running at a just-enough clock frequency for power minimization. TI's OMAP, Qualcomm's Snapdragon, Nvidia's Tegra are exemplar mobile SoCs which integrate a GPU for 3D graphics applications, a DSP for multimedia streaming applications, and a single or multi-core CPU for running operating system and general-purpose tasks. Fig. 3 shows the block diagram of Nvidia's Tegra 2 SoC, the target mobile-class platform in our study. The SoC has a dual-core ARM Cortex-A9 which runs at 1GHz. The Cortex-A9 MPcore processor implements the ARMv7 instruction set architecture which has an eight-stage pipeline and out-of-order instruction execution. It has a 32KB instruction cache and a 32KB data cache per core with both cores sharing a common 1MB L2 Cache. Other microarchitecture parameters of the processor are shown in Table I.

III. APPLICATIONS

We choose medical imaging as the primary application domain as it has become a routine tool in the diagnosis and treatment of most medical problems. Image reconstruction and medical image processing entails a large degree of computation. A typical image processing pipeline would include image reconstruction, denoising, registration and segmentation. The following subsections briefly describe each application, one by one.

A. Image Reconstruction: EM+TV

Computerized tomography (CT) plays a major role in modern medicine. However, the radiation associated with CT is significant. We applied compressive sensing methods for CT image reconstruction with less radiation exposure but comparable image quality. The compressive sensing implementation used by CDSC is called EM+TV, recently proposed in [16]. The reconstruction tries to recover signal (vector or images) x from measurements b where $Ax = b$. A is a $M \times N$ matrix describing the transform from the original image to measurements; M is the number of measurements, and N is the dimension of the image. The EM+TV reconstruction algorithm [16] tries to solve the non-linear optimization problem:

$$\min_x \int_\Omega |\nabla x| + \alpha \sum_{i=1}^{M} ((Ax)_i - b_i \log(Ax)_i)$$

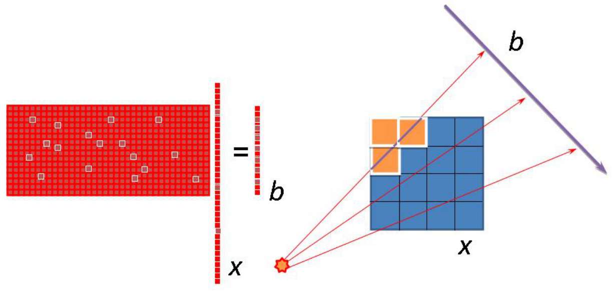

Fig. 4. Ray Tracing in Forward Projection

$$x_j \geq 0, j = 1, \cdots, N \tag{1}$$

The first term is the total variation (TV) term and the second term is the expectation maximization (EM) term. We omit the mathematical derivation details which are available in [16]. The constraint optimization problem is solved using a semi-implicit iterative scheme.

One of the major computation kernels, *EMupdate*, performs EM iterations $\tilde{x}_j^{k+1} = \frac{\sum_{i=1}^{M}(a_{ij}y_i)}{\sum_{i=1}^{M}a_{ij}}\tilde{x}_j^k$ where $y_i = (\frac{b_i}{(A\tilde{x}^k)_i})$. Inside the *EMupdate* kernel, we need to do a forward projection to obtain $A\tilde{x}^k$; perform an element-wise division to obtain y; do a backward projection to obtain A^Ty (or $\sum_{i=1}^{M}(a_{ij}y_i)$); and then obtain the updated value $\tilde{x}_j^{k+1}$ using element-wise scaling. Note that because matrix A is very large and sparse, A is never constructed explicitly. A ray-tracing based technique is used to compute the forward and backward projections. Fig. 4 illustrates the ray-tracing technique in a forward projection.

The *EMupdate* kernel involves many"random" accesses in the ray-tracing process. Load balancing is also an issue as the intersection lengths for different rays are different.

B. Image Denoising: Rician Denoise

The reconstructed image may contain certain noisy artifacts which are removed in a denoising step. Rician denoise is a TV-based (total variation) algorithm that tries to remove noises which are under Rician distributions. Assuming $f = u + n$, where u is the clear image, which is unknown; f is the observed image with Rician noise n. The problem is to recover a clear image u from a noisy image f. The following formulation can be used to resolve the noise

$$\min_{u \in BV(\Omega)} \int_\Omega |\nabla u|\, dx + \lambda \int_\Omega [\frac{u^2 + f^2}{2\sigma^2} - \log I_0(\frac{uf}{\sigma})]dx \tag{2}$$

In the equation, λ is a parameter to balance the TV term and fidelity term. Ω is the domain of the image. σ is the parameter for the Rician noise, which is given. $I_0(.)$ is the modified Bessel function of the first kind of order zero.

Our reference code then solves the minimization problem using gradient descent where the gradient

$$\frac{\partial u}{\partial t} = \nabla \cdot \frac{\nabla u}{|\nabla u|} - \frac{\lambda}{\sigma^2}u + \frac{\lambda}{\sigma^2}\frac{I_1(\frac{uf}{\sigma^2})}{I_0(\frac{uf}{\sigma^2})}f \tag{3}$$

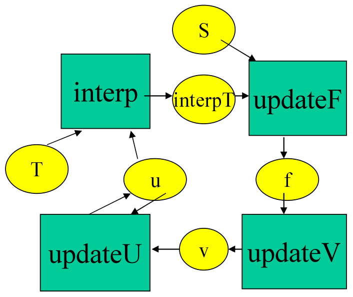

Fig. 5. Dataflow Between Procedures

The major computation involved in the gradient computation is the curvature term $\nabla \cdot \frac{\nabla u}{|\nabla u|}$, where a finite difference stencil computation kernel is used in actual numeric computation. More mathematical details are available in [12].

C. Image Registration: Fluid Registration

Image registration tries to find a transformation function of the coordinate system of one image study into the coordinate system of another image study, in order to better align the two image studies and capture the progressive development of the illness (e.g., tumors). Fluid registration regularizes the deformation using a fluid PDE equation, and it allows registrations of large deformations. Fluid regularizers ensure that the transform function is smooth. The two images are S and T. The deformation field is termed u. In each iteration, we first perform linear interpolation based on the deformation field:

$$\tilde{T}(x,t) = T(x - u(x,t)) \tag{4}$$

We obtain the force field using the derivative of an L2 Sum of Square Difference (SSD) metric:

$$f(x, u(x,t)) = -[\tilde{T}(x,t) - S(x)]\nabla\tilde{T}(x,t) \tag{5}$$

Instantaneous velocity $v(x,t)$ can be obtained by solving the fluid PDE:

$$\mu\Delta v(x,t) + (\mu + \lambda)\nabla div\ v(x,t) = f(x, u(x,t)) \tag{6}$$

In our implementation, we simply use a Gaussian convolution as in [11, 17].

$$v(x,t) \approx G_\sigma * f(x, u(x,t)) \tag{7}$$

We use the recursive Gaussian IIR proposed by Alvarez and Mazorra [4]. This IIR only needs two MADD operations per dimension. After that, we obtain an updated deformation field by solving the PDE $du(x,t)/dt = v(x,t) - v(x,t)\nabla u(x,t)$, using an explicit Euler scheme:

$$R(x,t^i) = (v(x,t^i) - v(x,t^i)\nabla u(x,t^i)) \tag{8}$$

$$u(x,t^{i+1}) = u(x,t^i) + (t^{i+1} - t^i)R(x,t^i) \tag{9}$$

The advancement of timestep needs to be bounded so that $(t^{i+1} - t^i)\ max\|R(x,t^i)\|_2$ does not exceed the maximum displacement allowed in one iteration. More mathematical derivations can be seen in [17].

The dataflow of this application is shown in Fig. 5. The application involves stencil computation in two functions (*updateU* and *updateF*), and random access in one function (*interp*). The *updateV* function involves an in-place sweeping of the 3D data array in all six directions. The FPGA implementation of the algorithm is presented in [8].

D. Image Segmentation: Active Contours

Image segmentation tries to find and segment an object of interest. For example, if we have a medical image that contains a tumor, image segmentation can be used to segment the tumor, and we can then perform volumetric assessment of the tumor size. We use Chan-Vese active contours method [5], which is a level-set-based technique for image segmentation. The method tries to minimize

$$\min F_2(c_1, c_2, \varphi) = \int_\Omega (I_0(\overrightarrow{x}) - c_1)^2 (1 - H(\varphi)) d\overrightarrow{x} + \int_\Omega (I_0(\overrightarrow{x}) - c_2)^2 H(\varphi) d\overrightarrow{x} + \beta \int_\Omega |\nabla H(\varphi)| d\overrightarrow{x} \quad (10)$$

where φ is the level-set function and $I_0(.)$ is the input image. $H(.)$ is the Heaviside step function:

$$H(z) = \begin{cases} 1 & if \ \ z \geq 0 \\ 0 & if \ \ z < 0 \end{cases} \quad (11)$$

c_1 and c_2 are the average intensities of the two phases (regions) segmented by the level-set function.

$$c_1(\varphi) = \frac{\int_\Omega I_0(\overrightarrow{x})(1 - H(\varphi(t, \overrightarrow{x})))d\overrightarrow{x}}{\int_\Omega (1 - H(\varphi(t, \overrightarrow{x})))d\overrightarrow{x}} \quad (12)$$

$$c_2(\varphi) = \frac{\int_\Omega I_0(\overrightarrow{x}) H(\varphi(t, \overrightarrow{x}))d\overrightarrow{x}}{\int_\Omega H(\varphi(t, \overrightarrow{x}))d\overrightarrow{x}} \quad (13)$$

The optimization problem is again solved using gradient descent where the gradient of φ is:

$$\frac{\partial \varphi}{\partial t} = \delta(\varphi)[\beta \nabla \cdot (\frac{\nabla \varphi}{|\nabla \varphi|}) + (I_0 - c_1)^2 - (I_0 - c_2)^2] \quad (14)$$

where $\delta(.)$ is the Dirac Delta function.

This application mainly involves stencil computations, the computation of c_1 and c_2 needs an additional reduction step.

Most of the codes discussed in this section are available in [3]. A summary of the application characteristic is shown in Table II.

IV. Application Performance and Energy Efficiency

A. CPU vs. GPU vs. FPGA

The CPU code is parallelized using OpenMP. The GPU kernel is implemented using Nvidia CUDA; and our FPGA kernel is described using hardware-oriented C codes which are further synthesized into Verilog RTL using AutoESL tool [9] version 2011.1. Tables III, IV and V show the experimental results.

In Tables III and IV, we can see that both the GPU and FPGA offer substantial speedup compared to the single-thread CPU version. However, different applications will prefer different accelerators. In the case of *denoise* and *segmentation*, because the dynamic range of the data values are large, we need to use floating point computations, which are better suited for the GPU. Moreover, the data accesses in these two applications are quite regular, therefore the data coalescing and data reuse can be done easily. Thus, the GPU is the better accelerator for those two applications. For the other two applications, we performed the fixed-point computation for the FPGA implementation. In the FPGA *registration* implementation, we also explored inter-module streaming (overlapped tiling) to conserve bandwidth [8]. For the *reconstruction* application, because the data accesses in the ray-tracing are random, the GPU implementation will use a lot of uncoalesced data accesses with a much lower off-chip bandwidth. The Convey FPGA system uses an interleaved (banked) off-chip memory (similar to the on-chip shared memory in GPU) that excels in random accesses. Because of these facts, the FPGA delivers a better performance for these two applications. Note that so far we only implemented one kernel of the *reconstruction* (the *EMupdate* part) [6], while the *TVupdate* is done by the multi-core CPU for the data presented in Tables IV and V.

Note that we try to use reasonable optimizations for each platform. At one point, the GPU-based segmentation is more than 100X faster than the single-threaded CPU. We later discovered that there is a *powf(a,1.5)* invocation that slows down the CPU code considerably. Replacing that with *sqrtf(a*a*a)* speeds up the CPU code by 4X to 5X. On the other hand, GPU has the special function unit (SFU) that can compute transcendental functions efficiently. Computations involving subnormal floating points are also very slow on the CPU. We have to use double-precision code for the *denoise* application for the 256^3 dataset, because double-precision code is around 2X faster than the single-precision code.

Table V shows the estimated energy used by the different platforms. Because of the difficulty in measuring the component power in real-time, we use the thermal design power (TDP) to approximate the full-load power consumption. We then multiply TDP with the multi-threaded execution time to estimate the energy consumption. The TDP of the Quad-core Xeon L5408 2.13GHZ is 40W. The TDP of the Tesla C1060 is 200W. We use the Xilinx xPower tool to estimate the power of the FPGA design, and each FPGA design reports around 23.5W, and 94W in total for the four user FPGAs. We can see that using GPU or FPGA accelerators can be up to 4X more energy efficient for the four benchmark applications. When the accelerator is not presented with a good bandwidth (e.g., *reconstruction* on the GPU), its energy efficiency may be worse than a generic multi-core processor.

TABLE II
APPLICATION CHARACTERISTICS

	Reconstruction	Denoise	Registration	Segmentation
Data access pattern	Random in *EMupdate* Stencil in *TVupdate*	Stencil	Random in *interp*; Stencil/sweeping in other parts	Stencil
Major computation	FP(mul,add)	FP(mul, add,div,sqrt)	FP(mul, add)	FP(mul,add,div,sqrt)

TABLE III
PERFORMANCE OF THE APPLICATIONS ON CPU (QUAD-CORE XEON L5408 2.13GHZ)

	Image	Reconstruction	Denoise	Registration	Segmentation
Num. of Iterations		$EMTV_{iternum} = 100$ $EM_{iternum} = 3$	15	500	150
Single thread	$128 * 128 * 128$	2163s	1.55s	212.4s	14.7s
	$256 * 256 * 256$	20376s	16.3s(DP)	2289s	110.3s
4 threads	$128 * 128 * 128$	629s	0.501s	85.3s	4.71s
	$256 * 256 * 256$	5931s	6.41s(DP)	739.7s	35.1s

TABLE IV
PERFORMANCE OF THE PIPELINE ON GPU OR FPGAS; NUMBERS SHOWN IN BRACKETS ARE THE SPEEDUP COMPARED WITH THE SINGLE-THREAD RUNTIME IN TABLE III

	Image	Reconstruction	Denoise	Registration	Segmentation
GPU	$128 * 128 * 128$	363s(6.0X)	0.0578s(26.8X)	10.9s(19.5X)	0.675s(21.8X)
	$256 * 256 * 256$	3625s(5.6X)	0.426s(38.3X)	86.3s(26.5X)	3.790s (29.1X)
FPGAs	$128 * 128 * 128$	178s(EM)+57s(TV)(9.2X)	0.119s(13.0X)	9.8s(21.7X)	1.57s(9.4X)
	$256 * 256 * 256$	1826s(EM)+460s(TV)(8.9X)	0.948s(17.2X)	76.1s(30.1X)	12.52s(8.8X)

TABLE V
ENERGY COMPARISON BETWEEN CPU, GPU AND FPGAS,; NUMBERS SHOWN IN BRACKETS ARE THE ENERGY SAVINGS COMPARED WITH THE XEON CPU

	Image	Energy			
		Reconstruction	Denoise	Registration	Segmentation
Xeon CPU	$128 * 128 * 128$	25160J	20.0J	3412J	188.4J
	$256 * 256 * 256$	2.37E5J	256.4J	29588J	1404J
GPU	$128 * 128 * 128$	72600J(0.35X)	11.6J(1.72X)	2180J (1.57X)	135J (1.40X)
	$256 * 256 * 256$	7.25E5J(0.33X)	85.2J(3.01X)	17260J (1.71X)	758J (1.85X)
FPGAs	$128 * 128 * 128$	19012J(1.33X)	11.2J(1.83X)	921J(3.70X)	147.6J(1.28X)
	$256 * 256 * 256$	1.90E5J(1.25X)	89.1J(2.87X)	7153J(4.1X)	1177J(1.19X)

TABLE VI
PERFORMANCE OF THE PIPELINE ON CPU: XEON, ATOM AND ARM(128^3 DATASET)

	Processor	Reconstruction	Denoise	Registration	Segmentation
Num. of Iterations		$EMTV_{iternum} = 100$ $EM_{iternum} = 3$	15	500	150
Single thread	Atom	15023s	8.02s	818.8s	53.3s
	ARM	12921s	10.2s	1005s	69.3s
4 threads	Atom	5638s	2.75s	298.7s	18.3s
2 threads	ARM	6569s	6.4s	561.6s	37.4s

TABLE VII
ENERGY COMPARISON BETWEEN XEON, ATOM AND ARM(128^3 DATASET); NUMBERS SHOWN IN BRACKETS ARE THE ENERGY SAVINGS COMPARED WITH THE XEON CPU

	Energy			
	Reconstruction	Denoise	Registration	Segmentation
Xeon	25160J	20.0J	3412J	188.4J
Atom	45014J (0.56X)	22J(0.91X)	2389J (1.43X)	146.4J (1.29X)
ARM	4815J(5.23X)	4.7J(4.26X)	411.7J(8.29X)	27.4J(6.87X)

B. Xeon vs. Atom

Surprisingly, we see that while the Atom processor has a much lower power consumption, its computing energy is not substantially better for the four domain-specific applications. In particular, for the *reconstruction* application, the IPC for Xeon is quite high (the working set fits the L2 cache of Xeon but not Atom). For the *denoise* and *segmentation* application, floating point square root and divisions drag down the performance of Atom significantly.

Note, because the memory footprint of a big dataset would not fit our Tegra 2 development board, only 128^3 data points are shown in Tables VI and VII. The TDP of the Dual-core Atom 330 is 8W. [1]

C. ARM-Based SOC

As the cross-compiler used by the SOC does not support OpenMP, we instead run multiple processes with a reduced workload to simulate the multi-threading behavior. The available parallelism is slightly larger using this scheme. The codes are compiled using android NDK using *softfp* ABI. The single-thread performance of ARM is only slightly worse than Atom. But it achieves consistent gain in energy than the latter. ARM uses RISC instructions natively, while Atom converts CISC x86 instructions into RISC microcodes. We can see that ARM-based system has a quite good balance between computation unit and memory systems, and they deliver remarkable energy savings compared with desktop systems. According to ARM webpage [1], dual-core A9 at 800MHZ consumes 0.5W in TSMC 40nm technology, and 1.9W at 2GHz. Our Tegra 2 runs at 1GHz, and we use linear interpolation to get an estimate of 0.733W.[2] Note the marketing power consumption of whole Tegra 2 SOC is less than 0.5W [7]. We use our estimated power number to compute the energy numbers in Table VII.

V. Conclusions and Future Work

In this paper we benchmark a wide spectrum of platforms, from mobile-class to server-class platforms. We compare the performance as well as energy consumption (approximated by computing the product of TDP and execution time). Different applications prefer different platforms, thus validating the benefits of heterogeneity.

Currently, the power numbers are estimated through the TDP. They should be replaced by more dedicated real-time measurements for more accurate energy results.

[1]According to [13], the measured average power of this Atom in search applications is 3.2W, much smaller than the 8W TDP.

[2]We are not able to find the official TDP number for the Tegra 2 chip. The actual TDP should be even larger than the power consumption of ARM cores, because SOC integrates many other application processors. Also the ARM-based Tegra 2 is a more recent product, which uses a newer technology as well. Thus the actual energy efficiency gap between Atom and ARM could be smaller than the numbers we reported in the Table VII.

Acknowledgements

This research is supported by the Center for Domain-Specific Computing (CDSC) which is funded by the NSF Expedition in Computing Award CCF-0926127. This research is conducted in a renovated laboratory funded by NSF Grant No. 0963183, awarded under the American Recovery and Reinvestment Act of 2009 (ARRA).

We greatly appreciate the research gift and equipment donations from Intel, and helpful discussions with Ganapati Srinivasa, Ravishankar Iyer, and Mishali Naik from Intel about the Intel QuickIA system. We also thank Ming Yan, Igor Yanovsky , and Pascal Getreuer for sharing their medical image processing source codes for this study.

References

[1] ARM Cortex A9, `http://www.arm.com/products/processors/cortex-a/cortex-a9.php`.

[2] Convey HC-1ex, `http://www.conveycomputer.com/`.

[3] `http://code.google.com/p/cdsc-image-processing-pipeline/downloads/list`.

[4] L. Alvarez and L. Mazorra. Signal and image restoration using shock filters and anisotropic diffusion. *SIAM J. Numer. Anal.*, 31:590–605, April 1994.

[5] T. Chan and L. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266 –277, Feb. 2001.

[6] J. Chen, J. Cong, M. Yan, and Y. Zou. FPGA accelerated 3D reconstruction using compressive sensing. In *Proc. FPGA*, 2012.

[7] P. Clarke. Nvidia launches Tegra 2 processor, `http://www.eetimes.com/electronics-news/4086778/Nvidia-launches-Tegra-2-processor`.

[8] J. Cong, M. Huang, and Y. Zou. Accelerating fluid registration algorithm on multi-FPGA platforms. In *Proc. FPL*, 2011.

[9] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for FPGAs: From prototyping to deployment. *TCAD*, 30(4):473 –491, april 2011.

[10] J. Cong, G. Reinman, A. Bui, and V. Sarkar. Customizable domain-specific computing. *IEEE Design Test of Computers*, 28(2):6 –15, march-april 2011.

[11] E. D'Agostino, F. Maes, D. Vandermeulen, and P. Suetens. A viscous fluid model for multimodal non-rigid image registration using mutual information. In *Proc. MICCAI*, pages 541–548, 2002.

[12] P. Getreuer, M. Tong, and L. A. Vese. A variational model for the restoration of mr images corrupted by blur and rician noise. In *Proc. ISVC*, pages 686–698, 2011.

[13] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *Proc. ISCA*, pages 314–325, 2010.

[14] D. A. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proc. Eurosys*, pages 125–138, 2010.

[15] C. H. K. van Berkel. Multi-core for mobile phones. In *Proc. DATE*.

[16] M. Yan and L. A. Vese. Expectation maximization and total variation-based model for computed tomography reconstruction from undersampled data. In *Proc. SPIE Conference on Medical Imaging: Physics of Medical Imaging*, 2011.

[17] I. Yanovsky, A. D. Leow, S. Lee, S. J. Osher, and P. M. Thompson. Comparing registration methods for mapping brain change using tensor-based morphometry. *Medical Image Analysis*, 13(5):679–700, October 2009.