10x10: A Case Study in Highly-Programmable and Energy-Efficient Federated Heterogeneous Architecture

Andrew A. Chien[1, 2], Tung Thanh Hoang[1], Dilip Vasudevan[1],
Yuanwei Fang[1] and Amirali Shambayati[1]
[1]Department of Computer Science, University of Chicago, Chicago, Illinois, USA
[2]Argonne National Laboratory, Chicago, Illinois, USA
E-mail: 10x10@cs.uchicago.edu

ABSTRACT

Customized architecture is widely recognized as an important approach for improved performance and energy-efficiency. To balance generality and customization benefit, researchers have proposed to federate heterogeneous micro-engines. Using the 10x10 architecture and an integrated image and vision benchmark as a case study, we explore the performance and energy benefits achievable. Results for current 32nm technology and DDR3 memory show 10x10 architecture benefits of 140x performance and 72x energy overall. Adding 3D-stacked DRAM increase benefits to 171x (performance) and 100x (energy). Finally, considering future 7nm transistor process, benefits as large as 597x (performance) and 137x energy are observed.

Keywords

Heterogeneous Architecture, Micro-architecture, Computer Architecture, 3D-stacked DRAM, Hybrid Memory Cube, SIMD, Programmability, Performance, Energy Efficiency, Accelerators, System Integration, Compute Intensive Micro-engines, Data Intensive Micro-engines.

1. INTRODUCTION

With Dennard scaling [1] over, chip level performance growth depends on both parallelism (multicore and other dimensions) and customization [2]. These techniques can increase energy efficiency, the key to continued performance growth. However, one major challenge in the path forward is to effectively balance general-purpose coverage, customization for energy efficiency, and programmability [3].

To this end, we have proposed 10x10 [3, 4], perhaps the most aggressive approach to heterogeneous architecture for general-purpose computing. 10x10 leverages hardware customization (up to 10,000x more energy-efficient than traditional processors) by splitting each core into as many as 10 customized micro-engines that individually deliver high performance and energy efficiency for a portion of the workload, and, collectively cover the general-purpose application space. Only one micro-engine (the best match to the current computation) per core is active, exploiting its customization to execute the computation many times more efficiently than a general-purpose core.

In this paper, we consider a focused workload [5] [1], embedded image processing and recognition, and use it to drive development and evaluation of a 10x10 architecture design and implementation. The architecture includes seven micro-engines ("7x7": Fast-Fourier-Transform – FFT, sort – Sort, Data-Layout-Transformation – DLT, Generalized Pattern-Matching – GenPM, Bit-Nibble-Byte – BnB, Vector Floating Point – VFP, and a simple RISC). Each micro-engine provides a small set of custom instructions, but its implementation makes a major performance difference.

We first describe the custom instructions in each micro-engine, and implement them with Synopsys tools [7]. Then we assess the architecture customization benefit of the "7x7" exemplar of the 10x10 approach using a simple, integrated benchmark and TSMC's 32nm process. Finally, we broaden the study, considering use of advanced memory systems (3D-stacked DRAM), and then by process scaling to projected 7nm FinFET technology.

Our results show that the benefits of process scaling, advanced memory technology, and customized architecture are important. However, customization unlocked by the 10x10 approach gives the largest benefits, and complements the benefits derived in other dimensions.

Specific contributions include:

- Realizing the 10x10 approach concretely with specific designs that target a focused workload, and demonstrate its potential benefits.
- Seven micro-engine designs (FFT, Sort, DLT, GenPM, BnB, VFP, and a simple RISC) and their characterization; integration into traditional ISA.
- Evaluation of system performance and energy, showing overall benefits of 140x and 72x respectively.
- Extension to consider advanced memory structures such as stacked DRAM, increases benefits, with performance gains of 168x and energy gains of 100x, and

[1]We have analyzed and characterized the hot-loops of broaden workload [6] then currently working on design of fabric-based architectures which can cover as much as hop-loops for performance and energy improvement.

Technical report, Department of CS, University of Chicago, October 2015.

.

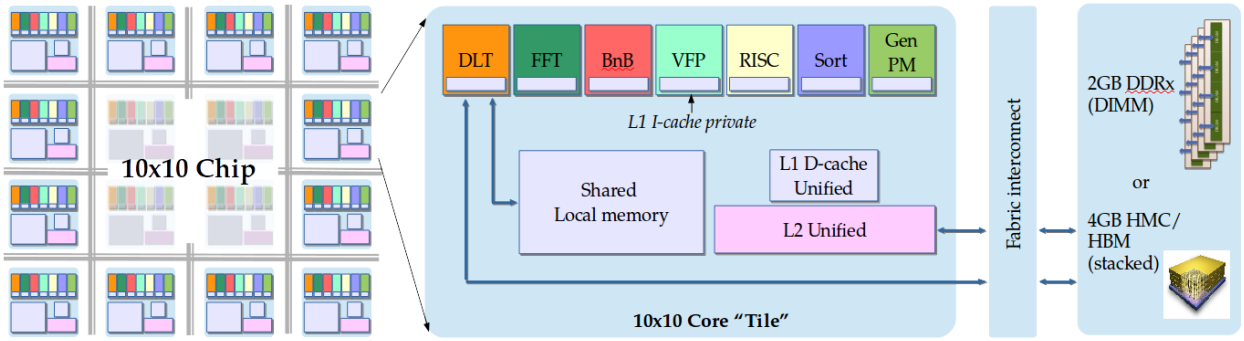


Figure 1: 10x10 Chip includes tiled 10x10 cores; each is a federation of Micro-engines.

- Finally, considering 7nm process, performance benefits increase further to 597x and energy benefits to 137x.

The remainder of the paper is organized as follows. In Section 2, we present the 10x10 federated heterogeneous approach, and two major classes of micro-engines. Section 3 outlines the methods used for evaluation. Section 4 presents the evaluation, and Section 5 describes related work. Finally Section 6 summarizes, and points out directions for future work.

2. FEDERATED HETEROGENEOUS 10X10 ARCHITECTURE

2.1 10x10 Core

A 10x10 architecture (Figure 1) employs workload-driven co-design to customize a set of micro-engines for energy efficiency, and then federates them to achieve general-purpose coverage. Current 10x10 micro-engines are classified as *1)* compute-intensive (BnB, FFT and VFP) micro-engines and *2)* data-intensive (DLT, GenPM and Sort). The 7th micro-engine is a conventional RISC core.

A 10x10 core includes a federated set of micro-engines sharing local memory, L1 data-cache, L2-unified (Figure 1). Each micro-engine uses customized instructions, tightly integrated so as to to accelerate at fine-grain. Each micro-engine has its own general purpose and vector registers, and only one micro-engine executes at a time. Unlike traditional accelerators, the customized micro-engines are tightly coupled, sharing an L1 data cache and local memory. Switching between micro-engines is achieved with a special instruction (under software control), that transfers to a different micro-engine. All state needed is passed through the caches and local memory. The program exists as a single image in memory, but as a collection of each micro-engines specialized instruction sequences.

Table 1 shows the intrinsics for each micro-engine's custom operations. The micro-engines and their ISA are described in detail in Sections 2.2 and 2.3.

2.2 Compute-Intensive Micro-engines

2.2.1 *BnB micro-engine*

Transistor density scaling enables wide SIMD vector architectures that can increase both computation throughput and energy efficiency of streaming applications. The BnB [8] micro-engine is a fine-grain wide SIMD architecture that processes as many as 256 one-byte elements with one instruction.

The BnB micro-engine includes sixteen 32-bit scalar and sixteen 2048-bit vector registers, supporting typical vector instructions (data movement, logic, and fixed point operations (Table 1)). Furthermore, the BnB micro-engine supports 4, 8, 16, 32, 64, 128 bit element sizes, significantly more flexible than SSE (8, 16, 32 bits), and as a result offers higher performance for "short-data" (< 8 bits) computations namely encryption (AES), compression (Run Length Encoding), etc. The BnB micro-engine is often partnered with other micro-engines such as DLT and VFP to improve performance, energy efficiency and workload coverage. Finally, we are working in an compiler implementation that support compilation of OpenCL code into C code with BnB intrinsics.

2.2.2 *Fast Fourier Transform Micro-engine*

Fast Fourier Transform (FFT) is essential kernel for applications involving image, audio, and digital signal processing. It motivates design and integration of an FFT micro-engine in 10x10 to provide *programmability* and *energy-efficiency* [9]. The FFT compute kernel is an optimized custom logic and associated storage structure generated by Spiral [10] and tightly-integrated into the pipeline for high programmability.

The FFT micro-engine computes on 64 complex points (16-bit integer real and imaginary [2]) in 16 cycles, replacing a large number of compute and memory instructions with a single instruction, thereby significantly improving performance and energy efficiency. The input data of FFT micro-engine is loaded from local memory into vector registers. DLT can be used to move data efficiently from off-chip memory to local memory, and transpose data in local memory. The ISA of FFT micro-engine (Table 1) can be used to program and accelerate larger FFTs with low software overhead.

2.2.3 *Vector Floating Point Micro-engine*

Floating point numbers provide high-dynamic range, error-tolerance, and easy programming, compared to fixed point. While the cost of floating point in terms of silicon area and power is decreasing in advanced CMOS process, the VFP micro-engine enables floating point applications (e.g. inner-product) to execute with high performance on a 10x10 heterogeneous system.

[2]In [11], we have evaluated and compared relative energy of using 32-bit floating and fixed point FFT accelerators.

The VFP micro-engine is an 2048-bit wide SIMD architecture that computes 64 addition or multiplications of single precision (IEEE-754) floating point values in a single instruction, to deliver 64 to 256 GFLOPS in 32nm and 7nm processes respectively. Like other micro-engines, VFP gets operands from a 16 entry, vector register file, and is tightly coupled to enable efficient acceleration of small stretches of computation.

2.3 Data-Intensive Micro-engines

2.3.1 DLT Micro-engine

DLT micro-engine [12] optimizes data movement betwixt the main memory, cache hierarchy, vector registers and local memory. By providing a set of gather and scatter instructions, and then scheduling their constituent operations to optimize parallelism and mask latency. DLT enables all other micro-engines to be more effective. DLT's gather and scatter instructions are non-blocking, allowing data movements to be overlapped with computation, and are synchronized via fence instructions. Sequential-to-strided and strided-to-sequential data layout transformations are captured by three parameters in a DLT operation (count: number of elements that should be moved; stride: distance between two consecutive elements, normalized by data fragment size; and fragment size: element size in bytes). DLT's instructions support: data layout transformation between main memory and local memory, as well as within the local memory, and even gather/scatter data between local memory and wide vector register. Therefore, DLT offers both data movement and layout transformation efficiently.

2.3.2 GenPM Micro-engine

GenPM micro-engine is designed to accelerate FSM-based applications such as dictionary-based decoding, deep packet inspection, Bioinformatics and JSON/XML processing. The critical parameters for the GenPM architecture are vector length, local memory parallelism, and DFA-steps. Vector instructions are used to implement multiple DFAs processing one input stream. DLT loads DFA tables from main memory into the local memory with contiguous layout in the address space with 256 entries (for every possible input) per state, and loads the input stream from main memory into double-buffered vector registers. To begin processing a stream against a set of DFAs, a programmer initializes a vector register with the base addresses of a set of DFAs. At each step, the matching unit generates a set of next state addresses. The matching unit implements parallel DFA state sequence and acceptance testing. It can advance a number of DFAs forward 1, 8, or 16 steps. In addition to implementing multi-step DFA sequence, it checks against acceptance states, flagging those DFAs that have accepted the input string. For detailed ISA, micro-architecture, and system integration refer to [13].

2.3.3 Sort Micro-engine

Sort is a fundamental computational building block in many applications. The SORT micro-engine contains 16 customized vector lanes accelerating sort operation. Each lane has a 4-way merge network with a buffer that holds data from the head of each input array. Each sort lane takes 4 sorted arrays and merges them into one array. The sort for any array is based on merge-sort, a recursive use of the 4-way merge network, and 16 sort lanes parallelize the merge-sort process. The sort instruction takes 4 operands as source address (local memory), destination address (local memory), number of arrays, and each array's size. Two 2048-bit vector registers contain a set of pointers for the 64 input arrays and two 32-bit general-purpose registers store number and size of arrays. Each cycle, each lane commits the the smallest element in the buffer to local memory and a replacement element from an input array is added to the buffer. The maximum throughput of a sort lane is one element/cycle. After the merge process for all sort lanes finishes, DLT transfers the sorted data from local memory to main memory.

2.4 RISC Micro-engine and Local memory

RISC Micro-engine: the RISC core is a conventional 6-stage pipeline, used for computation not well supported by other micro-engines. For example, the histogram equalization application benefits little from other micro-engines, so we run it on the RISC core.

Local memory: The local memory provides 64 word (2048 bits) parallel access to all the micro-engines with vector register files, enabling them to load/ store 64 words with a single instruction. Load/store is between local memory and vector registers. The local memory of a 10x10 core is shared by all the micro-engines, and is 1MB. As described above, the DLT micro-engine can efficiently move data between the main memory and local memory, or within the local memory. Typically, first DLT efficiently stages input data from main memory into local memory, and then the appropriate micro-engine running the application will operate on this data in local memory with low-latency, low-energy, and highly-parallel access. Once the computation is done, DLT again transfers the updated output to the main memory.

3. METHODOLOGY

3.1 Simulation Systems

We have developed a cycle-accurate simulator of the 10x10 micro-engines and the RISC processor, a 16x2048-bit vector register file and 1MB, 64-bank, local memory with single cycle access. The micro-engine simulations are generated by the Synopsys tools from a LISA language ISA description [7]. We extended the Synopsys model, integrating MARSim [14] to simulate a two-level cache hierarchy modeled on the Intel Atom.

On-chip memories have performance and power models based on Cacti [15] for 32nm bulk-CMOS process and extrapolated for 7nm FinFET [16]. Logic components (core, register files and micro-engines) are synthesized from LISA generated RTL for cell libraries; we then extract delay and power. Our designs achieve 1Ghz in TSMC-based 32nm bulk-CMOS and 4Ghz for the project 7nm FinFET process.

We enhanced DRAMSim2 [17], adding online simulation capability, and used it to support modeling of both DDR3 and 3D-stacked DRAM called Hybrid Memory Cube (HMC) that provide 10 Gbps and 128 GBps bandwidth respectively. The DDR models are validated by Micron, and 3D-stacked DRAM models are based on public HMC specifications [18] and Cacti-3DD [19]. A 10x10 system configuration is summarized in Table 2. In addition, the 10x10 design flow is shown in Figure 2.

Table 1: 10x10 Micro-engines Instruction Set Architecture.

Intrinsic	Instruction	Operands	Functional description
Bit-nibble-Byte (BnB) micro-engine			
bnb_shuffle(vec vDst, vec vSrc, int size)	SHUFFLE	V1 V2 R1	Do shuffle of between two 2048-bit vectors (*size* is element size to shuffle)
bnb_rotate(vec vDst, int size, int count)	VROTATE	V1 R1 R2	Rotate each element of *size* in *vDst* for *count* times
bnb_vxor(vec vDst, vec vSrc)	VXOR	V1 V2	Do vector bitwise XOR between *vDst*, *vSrc* and store result in *vDst*
bnb_vand(vec vDst, vec vSrc)	VAND	V1 V2	Do vector bitwise AND between *vDst*, *vSrc* and store result in *vDst*
bnb_vor(vec vDst, vec vSrc)	VOR	V1 V2	Do vector bitwise OR between *vDst*, *vSrc* and store result in *vDst*
bnb_vmul(vec vDst, vec vSrc)	VMUL	V1 V2	Do 64 integer vector multiplication of src and dst and store result in *vDst*
bnb_vadd(vec vDst, vec vSrc)	VADD	V1 V2	Do 64 integer vector addition of src and dst and store result in *vDst*
bnb_vsub(vec vDst, vec vSrc)	VSUB	V1 V2	Do 64 integer vector subtraction of src and dst and store result in *vDst*
bnb_shift(vec vDst, vec vSrc, int size)	VSHIFT	V1 V2 R1	Shift right each element of *size* in *vDst* for *count* times
bnb_vcntlz(vec vDst, vec vSrc)	VCNTLZ	V1 V2	Count the number of leading zeros in src and store the count in *vDst* vector
bnb_svxor (vec vDst, int src, int size)	SVXOR	V1 V2 R1	Do XOR of *src* over each element of *vDst* vector of size 32-bit
int bnb_rvsmov(vec vSrc, int index)	RVSMOV	V1 V2 R1	Get 32-bit value indexed by *index* from vector *vSrc*
bnb_vrsmov(vec vDst, int src, int index)	VRSMOV	V1 V2	Move 32-bit segment indexed by *index* of vector register *vDst* to the register *src*
bnb_lm2vec(vec vDst, int *lmA)	LM2VEC	R1 R2	Move 2048-bit data from the local memory to vector register
bnb_vec2lm(, vec vSrc)	VEC2LM	R1 R2	Move 2048-bit data from vector register to the local memory
Fast Fourier Transform (FFT) micro-engine			
fft_sp64_fxp16 (vec vSrc, vec vTdf64 vec vTdm, vec vDst, int opt)	FFTSP64FXP16	V1 V2 V3 V4 R1	Do FFT compute for 64 complex values, each complex consists of two 16-bit integer values
fft_ldlm2v(vec vDst, int* srcA)	LDLM2V	V1 R1	Load 2048-bit from local memory (*srcA*) to *vDst* vector register
fft_stv2lm(int* dstA, vec vSrc)	STV2LM	R1 V1	Store 2048-bit vector register (*vSrc*) to local memory (*dstA*)
fft_bvmm2sm(int* mmA, int* lmA, int nByte)	BVMM2LM	R1 R2 R3	Moving nByte data from main memory (*mmA*) to local memory (*lmA*)
fft_bvmm2sm(int* lmA, int* mmA, int nByte)	BVLM2MM	R1 R2 R3	Moving nByte data from local memory (*lmA*) to main memory (*mmA*)
Vector Floating-Point (VFP) micro-engine			
vfp_fvec2lm(float* lmA, vec vSrc)	FVEC2LM	V1 R1	Function is similar to LM2VEC except it belongs to VFP
vfp_flm2vec(vec vDst, float* lmA)	FLM2VEC	V1 V1	Function is similar to VEC2LM except it belongs to VFP
vfp_vec fvset(vec vDst, float src)	FRVMOV	V1 R1	Set all 32-bit segment of *vDst* vector by floating value *src*
vfp_float frvsmov(vec vSrc, int index)	FRVSMOV	V1 R1	Function is similar to RVSMOV except it returns a floating value
vfp_fvsmov(vec vDst, float src, int index)	FVRSMOV	V1 R1 R2	Function is similar to VRSMOV except it takes floating input *src*
vfp_vmulf(vec vDst, vec vSrc1)	FVMUL	V1 V2	SIMD 64-multiplication instruction for floating-point operands
vfp_vaddf(vec vDst, vec vSrc1)	FVADD	V1 V2	SIMD 64-addition instruction for floating-point operands
Data Layout Transformation (DLT) micro-engine			
int dlt_form_descriptor(int count, int stride, int fsize)	FORMDESC	R1 R2 R3	Packs descriptors (number_of_elems, stride, elem_size) into one 32-bit register for DLT
dlt_gather(ptr* dstA, ptr* srcA, int desc)	GATHER	R1 R2 R3	Gather data described by *desc* from *srcA* to *dstA*
dlt_scatter(ptr* dstA, ptr* srcA,int desc)	SCATTER	R1 R2 R3	Scatter data described by *desc* from *srcA* to *dstA*
dlt_vgather(dstVec, ptr* lmA,int desc)	VGATHER	V1 R1 R2	Gathers data described by *desc* in local memory (*desc*) to vector register (*vDst*)
dlt_vscatter(ptr* lmA, vec vSrc,int desc)	VSCATTER	R1 V1 R2	Scatters data described by *desc* from vector register (*desc*) to local memory (*lmA*)
dlt_flush(ptr* addr1,ptr* addr2)	FLUSH	R1 R2	Flush and invalidate all the cache lines within address range defined by (*addr1, addr2*)
dlt_fence()	FENCE		Avoid execution of ALL memory instructions of DLT or RISC
dlt_gather_fence()	GATHERFENCE		Avoid execution of memory instructions of DLT or RISC (except for gathers), until all previous gathers are executed
dlt_scatter_fence()	SCATTERFENCE		Avoid execution of memory instructions of DLT or RISC (except for scatters), until all previous gathers are executed
General Pattern Matching (GenPM) micro-engine			
gpm_gmvsnext(int pos, int step, int mode)	GMVNEXT	R1 R2 R3	Parallel execute parallel FSMs with multiple steps
gpm_gmvsacc (int dfa)	GMVSACC	R1 R2	Check the acceptance status of GenPM lanes
int gpm_gmvscnt(int dfa)	GMVSCNT	R1 R2	Get the position of first pattern matching
gpm_gmstrideback()	GMSTRIDEBACK		Recover the status before match, which enables coarse or fine-grained matching
Sorting (Sort) micro-engine			
sort_ldlm2vr(vec vDst, ptr *lmA)	LDLM2VE	V1 R1	Load 2048-bit from local memory (*lmA*) to vector register *vDst*
sort_parallel_merge (vec vSrc, vec vDst, int nlanes, int nstreams)	PARMERGE	V1 V2 R1 R2	Use *nlanes* lanes to sort *nstreams* in parallel. Here, source and destination addresses of streams are stored in *vSrc* and *vDst*

*** Intrinsic functions without specific returned types refer as *void*.

Table 2: 10x10 Configuration Parameters

Parameter	Value
Core type	In-order, 5-stage pipeline
ISA	MIPS-like, mixed 32/16b ISA
Register files	16 x 32-bit, 16 x 2048-bit registers
Local Memory	64 banks, 1MB total capacity
Cache hierarchy	L1-I: 32KB, 2-cycle latency
	L1-D: 24KB, 2-cycle latency
	Shared L2: 512KB, 10-cycle latency
Main memory	2GB, 4-rank, 16-device DDR3 or
	4GB, 4-rank, 8-device 3D-stack DRAM
Processes	32nm bulk-CMOS or 7nm FinFET

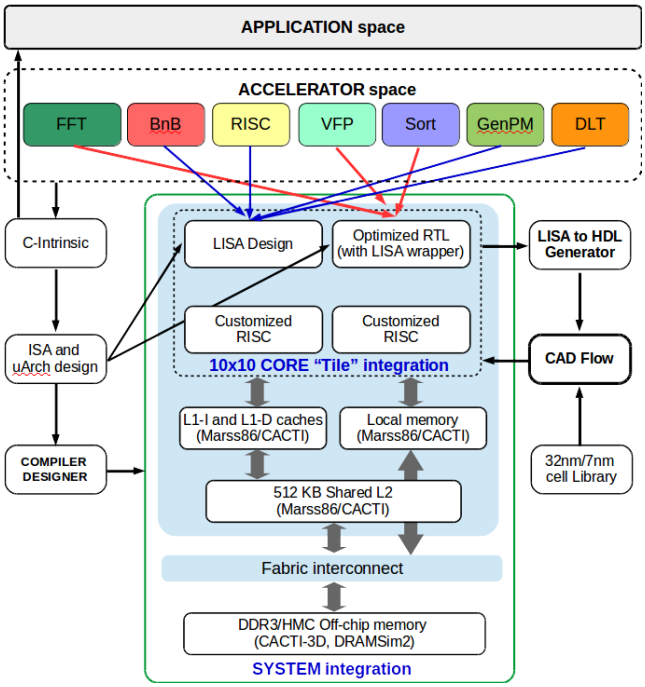


Figure 2: System design flow of the 10x10 heterogeneous architecture which supports cycle-accurate modeling for performance and power.

3.2 Evaluated Benchmark

As workload, we use an ANSI-C unified benchmark, composed of a sequence of fundamental computations widely-used in embedded computer vision and video applications; now a major driver for every type of mobile and embedded computing. We modified the source code of the fundamental computations which are derived from PERFECT benchmark [5] to make use of accelerator C-intrinsic functions described in the Table 1. It includes several phases:

- *Conditioning*: 1D-FFT (4K points) and 2D-FFT (4Kx4K points) models image conditioning.
- *Processing*: 2D-Convolution (640x480 image), DWT (640x480 image), Histogram (640x480 image) and Outer Product (4Mx4M matrix) models image processing tasks (noise filtering, compression etc.)
- *Recognition*: Sort (64 streams, each stream contains 1024 elements) and General Pattern Matching (640x480 characters) models high-level of pattern recognition and analysis (image ranking, recognition, tracking etc.).

By integrating these kernels into a unified benchmark, we can study the efficient composition of micro-engines (and computational kernels) in the 10x10 architecture. Each kernel was written as a C program, and then optimized with the use of intrinsics. For 2D-convolution, DWT, and VFP, compiler-based automatic exploitation is possible, but at the current state of software maturity gives lower performance. The mapping of the unified benchmark onto the accelerator set of 10x10 architecture is shown in Figure 3.

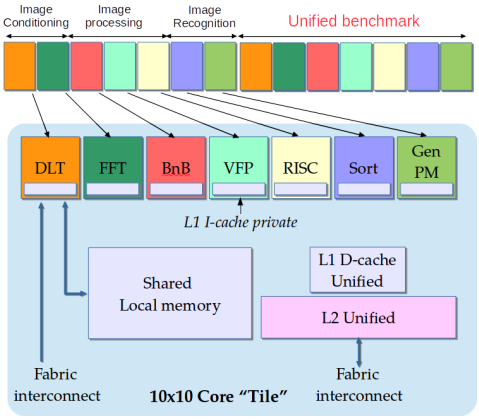


Figure 3: Mapping of the unified benchmark phases onto 10x10 micro-engines.

4. EVALUATION

Figure 4 shows the performance benefit for 10x10's customized architecture, compared to a RISC baseline and Intel Atom. Each *Blue Bar* represents the relative performance of a benchmark phase on one or more micro-engines which is normalized to the RISC baseline. FFT1D, FFT2D, SORT, and GenPM all achieve remarkable ($> 580x$) improvement; even more remarkable that SORT and GenPM achieve this on data-intensive computations. Effective use of the 64-bank, on-chip local memory is key. The geometric mean of 10x10 improvement versus Baseline (*Orange Bar*) and versus Intel Atom (*Green Bar*) are 140x and 90x respectively. It is evident that maximum performance benefit achieved by 10x10 through radical customization is better than could be expected from conventional multi-core approaches.

Figure 5 shows the system energy benefit for 10x10's customized architecture compared to a RISC baseline and Intel Atom. Energy improvements of $18 - 919x$ are achieved (shown by *Blue Bars* against the RISC baseline) and $38 - 4612x$ versus Intel Atom. Overall energy benefit is 72x and 195x versus the RISC baseline and Intel Atom respectively. The geometric mean of energy benefit versus Intel Atom is higher than that versus the RISC baseline; this is because the baseline RISC is a Synopsys simulation and conservative compared to the Intel Atom board level measurement.

We next explore the advantages of 10x10 architecture in the context of advanced memory systems and future transistor processes. Figure 6a, shows that adding 3D-stacked HMC memory gives little performance benefit – 10x10 per-

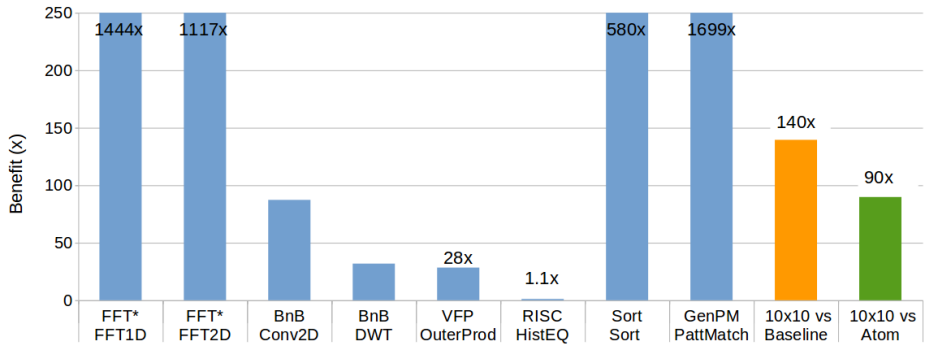


Figure 4: 10x10 Relative System Performance (32nm, DDR3)

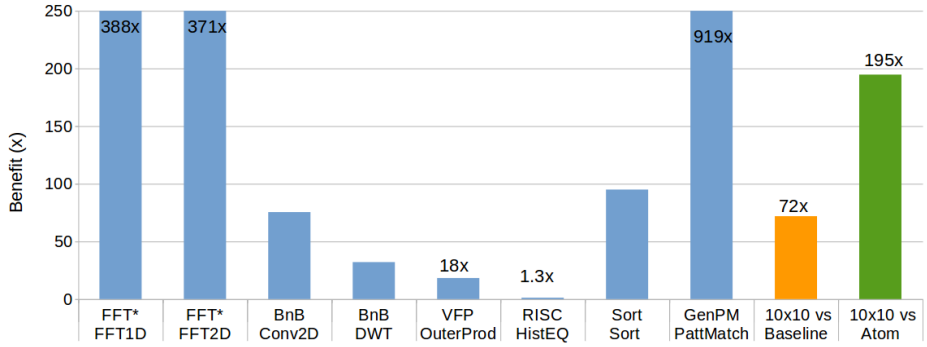
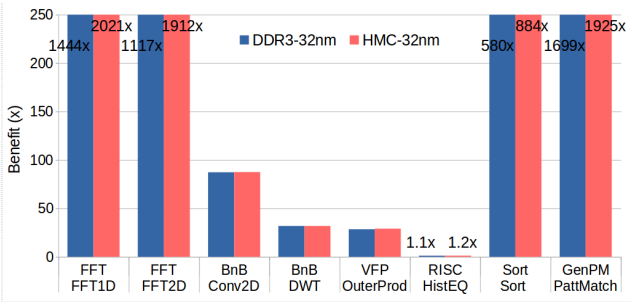
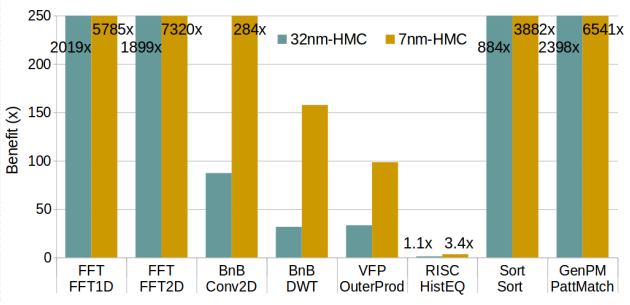


Figure 5: 10x10 Relative System Energy (32nm, DDR3)

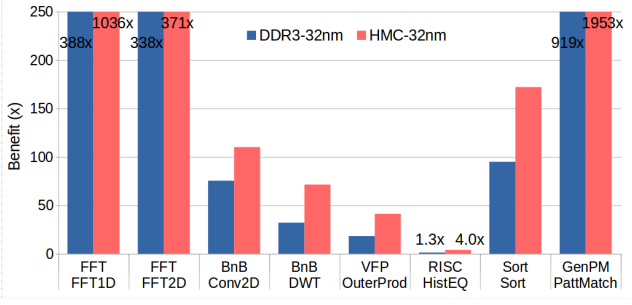


(a) 10x10 Relative Performance (32nm, DDR & HMC).

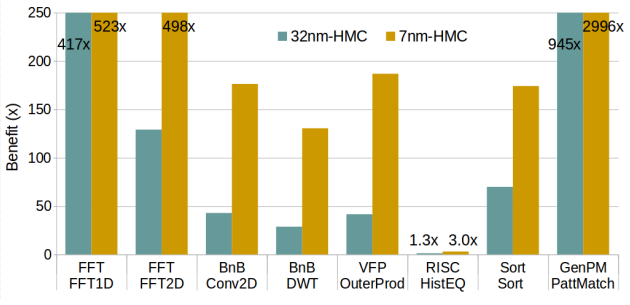


(b) 10x10 Relative Performance (32nm & 7nm, HMC).

Figure 6: 10x10 Performance benefit varying memory and process.



(a) 10x10 Energy benefit (32nm, DDR & HMC).



(b) 10x10 Energy benefit (32nm & 7nm, HMC).

Figure 7: 10x10 Energy benefit varying memory and process.

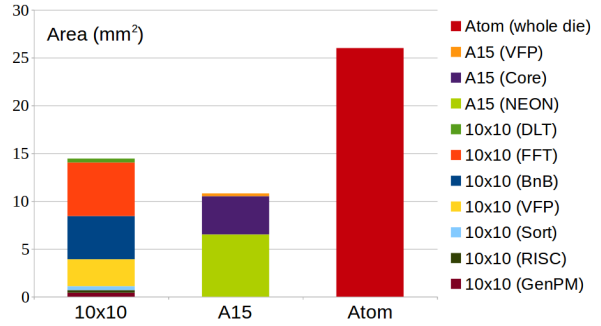


Figure 8: Area of a 10x10 core (7 micro-engines, unoptimized) and several commercial designs.

formance uses the memory system so efficiently that the improved memory bandwidth by using HMC only increases 10x10's benefit from 140x to 171x compared to DRAM in some memory-limited benchmarks such as FFT1D, FFT2D. Figure 6b show improved transistor process gives a much greater improvement, growing benefit from 140x to 597x when using the same advanced HMC memory system. This is because the 7nm process enables faster 10x10 micro-engines (4Ghz), contributing a broad performance improvement for all phases of the unified benchmark.

Considering 10x10 energy benefits in the context of advanced memory systems and future transistor processes tells a different story. Figures 7a shows that 3D-stacked HMC improves energy efficiency overall from 72x to 100x with individual phases improving to as much as 100-2000x because the HMC memory interfaces have much lower static power compared to conventional DRAM. And Figure 7b shows that 10x10 with 3D-stacked HMC yields even greater energy benefit in 7nm, growing from 72x to 137x (and individual phases improving to as much as 500-3000x). Here the much shorter runtime enabled by 10x10 and the faster 7nm process compound both logic and DRAM energy reductions.

A common initial concern is that employing several micro-engines might increase Si area. Figure 8 plots 10x10 core, Cortex A15 Quad-core, and Intel Atom in comparable 32nm processes. Both 10x10 and the A15 include L1 caches; the Intel Atom includes the full die (L1, L2 caches, memory controllers, pad, etc.). While the 10x10 core area is significant at 12.9 mm^2, it's a completely unoptimized academic design. Despite that it's already comparable to the quad-core Cortex-A15 (10.8 mm^2) and the core part of the Intel Atom die (26 mm^2). We believe that a commercial quality design for these seven micro-engines would be substantially smaller (2-4x).

5. RELATED WORK

While CMOS transistor density continues to scale, the end of Dennard scaling has created a "power wall" challenge. The architecture field's response has been the widespread adoption of multi-core parallelism, GPU's, and customized accelerators. The concomitant effect is that most of the time, most of the billions of transistors on a modern processor chip are not simultaneously activate –termed *Dark silicon* [2]. Therefore, the system performance is not well scaled with the increasing number of cores per CPU/GPU. Although customized accelerators can boost system performance much better than many-core system [20], it may incur low programmability due to inefficient architecture, ISA design and software interface which require extreme fine-grained application analysis and powerful HW/SW co-design (with sufficient compiler support).

This opportunity is the focus of 10x10, and the broader heterogeneous computing community [21]. Hardware customization –heterogeneous systems– has been identified as a key approach to continue scaling performance in the exascale computing era [3].

A comprehensive survey of the voluminous work on heterogeneous systems is not possible in a brief related work section, but we describe a number of landmark approaches here. Researchers have explored heterogeneous systems composed of lightweight, specialized (customized) simple cores [22], applying them to specific workloads to find fine-grained customization opportunities. Others have pursued application-specific FPGA [23] or ASIC [24, 25] accelerators that exploit much coarser-grained customization opportunities. 10x10 falls into middle-ground, using a coarser-grained, more radical customization than in systems like [22], but integrating the customization tightly into the ISA – a single core – to enable flexible use. Finally, others have pursued the design of highly optimized "networks on chip" for more efficient coarse-grain, loose composition of accelerators [26].

6. SUMMARY AND FUTURE WORK

We have presented a concrete 10x10 design, and evaluated it with a unified visual processing benchmark. Our results show that its federated, heterogeneous approach can combine large improvements in performance and energy-efficiency with fine-grained composition of customization, and therefore programmability. We are encouraged that the 10x10 approach is a promising avenue for further research.

Interesting future directions include *1*) exploring additional micro-engines based on new workload studies; for example, micro-engines based on the hot-loop analysis of the chosen workloads, *2*) explore 10x10 multicore architecture regarding to on-chip interconnection, simulation and compiler support.

7. ACKNOWLEDGEMENT

Funding of this work was provided in part by the Defense Advanced Research Projects Agency under award HR0011-13-2-0014 and the NSF under award NSF OCI-10-57921. We would like to thank: the Spiral team whose FFT generator was used for the 64-point accelerator blocks; Calvin Deutschbein and Henry Hoffmann at UChicago for contributions to FFT micro-engine; Ananta Tiwari and Laura Carrington at SDSC/UCSD for contributions to Atom board measurement. Finally, we thank Synopsis for generous university program support.

References

[1] R. Dennard, F. Gaensslen, H.-N. YU, V. Leo Rideovt, E. Bassous, and A. R. Leblanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 38–50, 2007.

[2] H. Esmaeilzadeh, E. Blem, R. St.Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of

multicore scaling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 365–376, 2011.

[3] S. Borkar and A. A. Chien, "The future of microprocessors," *ACM Communication*, vol. 54, pp. 67–77, May 2011.

[4] A. A. Chien, A. Snavely, and M. Gahagan, "10x10: A general-purpose architectural approach to heterogeneity and energy efficiency," vol. 4, pp. 1987–1996, 2011.

[5] Pacific Northwest National Laboratory (PNL), "PERFECT benchmark." Online, `http://hpc.pnl.gov/PERFECT/`.

[6] A. Guha, Y. Zhang, R. ur Rasool, and A. A. Chien, "Systematic evaluation of workload clustering for extremely energy-efficient architectures," *SIGARCH Comput. Archit. News*, vol. 41, pp. 22–29, May 2013.

[7] Synopsys, "Processor Designer." Online, `http://www.synopsys.com/IP/ProcessorIP/asip/processor-designer/Pages/default.aspx`.

[8] D. V. and A. C. Andrew, "BnB: Bit-nibble-Byte microengine for accelerating low-level bit operations," in *Proceedings of the Great Lakes Symposium on VLSI (GSVLSI)*, 2015.

[9] H. T. Tung, A. Shambayati, C. Deutschbein, H. Hoffmann, and A. Chien, "Performance and energy limits of a processor-integrated fft accelerator," in *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Sept 2014.

[10] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer generation of hardware for linear digital signal processing transforms," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, pp. 15:1–15:33, Apr 2012.

[11] H. T. Tung, A. Shambayati, F. Yuanwei, H. Hoffmann, and A. Chien, "Does arithmetic logic dominate data movement ? A systematic comparison of energy-efficiency for fft accelerators," in *Proceedings of the IEEE Application-specific Systems, Architectures and Processors (ASAP)*, July 2015 (to appear).

[12] H. T. Tung, A. Shambayati, , and A. A. Chien, "A Data Layout Transformation (DLT) accelerator: Architectural support for data movement optimization in accelerated systems," in *Department of Computer Science Technical Report, University of Chicago*, March (also submitted) 2015.

[13] F. Yuanwei, R. Raihan, V. Dilip, and A. A. Chien, "Generalized pattern matching micro-engine," in *Workshop on Architectures and Systems for Big Data (ASBD), joined with the International Symposium on Computer Architecture (ISCA)*, 2014.

[14] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A full system simulator for multicore x86 CPUs," in *Proceedings of the Design Automation Conference (DAC)*, pp. 1050–1055, June 2011.

[15] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," tech. rep., HP Laboratories, 2009.

[16] USC-SPORT, "DARPA Empower project: 7nm FinFET cell library." Download link `http://sportlab.usc.edu/downloads/packages/`, 2014.

[17] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, pp. 16–19, Jan 2011.

[18] HMC-Consortium, "Hybrid Memory Cube specification v1.1," tech. rep., 2014.

[19] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. Brockman, and N. Jouppi, "Cacti-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 33–38, Mar 2012.

[20] A. Guha, Y. Zhang, R. ur Rasool, and A. A. Chien, "Calibrating the relationship between hardware customization and energy efficiency," tech. rep., July 2013.

[21] C. Cascaval, S. Chatterjee, H. Franke, K. Gildea, and P. Pattnaik, "A taxonomy of accelerator architectures and their programming models," *IBM Journal of Research and Development*, vol. 54, pp. 5:1–5:10, Sept 2010.

[22] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pp. 205–218, 2010.

[23] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "Dyser: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, pp. 38–51, Sept 2012.

[24] M. David, B. Brendan, R. Richard, C. Fergal, B. Cormac, and D. David, "Myriad 2: Eye of the computational vision storm," in *Proceedings of Hot Chips*, 2014.

[25] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. Al Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini, "Exploring architectural heterogeneity in intelligent vision systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, Feb 2015.

[26] A. Bakhoda, J. Kim, and T. M. Aamodt, "Designing on-chip networks for throughput accelerators," *ACM Transaction on Architecture and Code Optimization (TACO)*, vol. 10, pp. 21:1–21:35, Sept. 2008.