Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2011) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2011

10x10: A General-purpose Architectural Approach to Heterogeneity and Energy Efficiency

Andrew A. Chien[a]*, Allan Snavely[a], Mark Gahagan[a]

[a]*University of California, San Diego and San Diego Supercomputer Center, 9500 Gilman Drive, La Jolla, CA 92093, USA*

Abstract

Two decades of microprocessor architecture driven by quantitative 90/10 optimization has delivered an extraordinary 1000-fold improvement in microprocessor performance, enabled by transistor scaling which improved density, speed, and energy. Recent generations of technology have produced limited benefits in transistor speed and power, so as a result the industry has turned to multicore parallelism for performance scaling . Long-range studies [1, 2] indicate that radical approaches are needed in the coming decade – extreme parallelism, near-threshold voltage scaling (and resulting poor single-thread performance), and tolerance of extreme variability – are required to maximize energy efficiency and compute density. These changes create major new challenges in architecture and software.

As a result, the performance and energy-efficiency advantages of heterogeneous architectures are increasingly attractive. However, computing has lacked an optimization paradigm in which to systematically analyze, assess, and implement heterogeneous computing. We propose a new paradigm, "10x10", which clusters applications into a set of less frequent cases (ie. 10% cases), and creates customized architecture, implementation, and software solutions for each of these clusters, achieving significantly better energy efficiency and performance. We call this approach "10x10" because the approach is exemplified by optimizing ten different 10% cases, reflecting a shift away from the 90/10 optimization paradigm framed by Amdahl's law [3]. We describe the 10x10 approach, explain how it solves the major obstacles to widespread adoption of heterogeneous architectures, and present a preliminary 10x10 clustering, strawman architecture, and software tool chain approach.

Keywords: computer architecture, parallelism, heterogeneity, hybrid computing, performance, moore's law, compilers

1. Introduction

Two decades of microprocessor architecture driven by quantitative 90/10 optimization has delivered an extraordinary 1000-fold improvement in microprocessor performance, enabled by transistor scaling which improved density, speed, and energy. Recent generations of technology have produced limited benefits in transistor speed and power, so as a result the industry has turned to multicore parallelism for performance scaling. Long-range studies [1,2] indicate that radical approaches are needed in the coming decade – extreme parallelism, near-threshold voltage scaling (and resulting poor single-thread performance), and tolerance of extreme variability – are required to maximize energy efficiency and compute density. These changes create major new challenges in architecture and software. The implications of these approaches are daunting parallelism requirements (10^3 to 10^9), and extremely

* Corresponding author. *Email address*: achien@ucsd.edu.

poor single-thread performance (10x slower [17]), and extreme variability (5x or more). See [4] for a summary of these technology scaling challenges, open research problems, and possible approaches.

Through two decades of rapid performance improvement, the dominant optimization paradigm has been 90/10 [3], which focuses on 90% of the workload and on optimizing the activities of that dominant portion to optimize a general-purpose architecture. However, technology scaling and architecture trends do not favor investment of hardware resources in a general-purpose, high performance core.

We propose a new approach, 10x10, which focuses on customized design (heterogeneity) to make most efficient use of each transistor's switching energy to ameliorate the extreme energy challenges. There's a broad consensus that heterogeneity in architecture and implementation has the potential to reduce energy and increase performance [5,6], however these techniques have not made it into the mainstream because of 90/10 thinking (general-purpose core centric), and reluctance to introduce instruction set heterogeneity in an undisciplined fashion due to fears about its support cost – both functional and performance for the future. However we believe it is critical not to limit ourselves to homogeneous parallelism going forward as we need to be able to exploit all kinds of parallelism. Our focus is exploration of a new type of building block – a 10x10 architecture – which exploits the known benefits of customization for energy efficiency and performance, but does so in a fashion that enables programmability and parallel scalability.

We outline the 10x10 framework and architecture. 10x10 is a systematic approach which enables heterogeneity to be introduced and managed as an architectural element. The approach clusters applications into sets of less frequent cases (ie. 10% cases), and creates specialized architecture, implementation, and software solutions for each of these clusters, achieving significantly better energy efficiency and performance for each cluster. 10x10 is exemplified by optimizing ten different 10% cases, and it reflects a radical shift away from the 90/10 optimization paradigm framed by Amdahl and popularized in [3]. The choice of 10 is somewhat arbitrary, as the right number will be empirically determined based on a wide range of factors, including application structures, hardware design and manufacturing cost, software cost, and so on. The key to 10x10 is the partitioning of the general-purpose computing optimization problem into a reasonably small-set discrete optimization problem, unlocking the potential for energy efficiency and performance gains through customization. And, 10x10 supports stable architecture by creating a disciplined method for analyzing and optimizing for workloads which share like properties, and doing so as a subset of the larger workloads. The four critical ideas are: 1) Application Clusters and a Broad Embrace of Heterogeneity for Energy Efficiency, 2) Deep Integration of Custom Microengines to simplify Software Challenges, 3) Exploit Poor Energy Technology Scaling to reduce Heterogeneity Cost and 4) Architect and Integrate Heterogeneity to simplify Software Challenges.

In addition, there are two key technical challenges that must be addressed by the software tool chain. First, the micro-engines represent additional back-end code targets for the software tool chain. Second, multiple micro-engines requires that the software partition the application, coordinate across use of different micro-engines (1 or at most a few at a time due to power limits), and do so intelligently. We outline simple approaches to these problems, and argue that they are tractable in a 10x10 architecture because of the deep integration of the micro-engines.

The five major technical contributions of the paper are: 1) Definition of 10x10 application workload clustering, 2) Definition of 10x10 architecture, 3) A sample 10x10 application clustering based on well-known benchmark suites, 4) A sample 10x10 architecture consisting of seven specialized micro-engines, 5) A software approach which leverages the existing software tool chains and renders 10x10 practicable in contemporary software environments.

The remainder of the paper is organized as follows. In Section 2, we motivate the heterogeneous architecture approach, explaining how historical obstacles are addressed by the 10x10 approach. Section 3 describes the 10x10 paradigm and how it enables a new approach to architectural management of heterogeneity. Section 4 shows how existing application benchmark suites can be clustered to form a basis for 10x10, and then applies them to develop a 10x10 architecture with seven micro-engines. Section 5 addresses the software challenges for 10x10, making the case that existing software tool chains can be leveraged for programming 10x10 architectures. Section 6 discusses related work, and Section 7 summarizes and discusses future research directions.

2. The Problem: Making the Benefits of Heterogeneity Accessible

The performance and energy-efficiency advantages of customized hardware implementations (termed accelerators, custom-computing, or ASICs) are widely recognized in the computer architecture community. In the

context of media processing, graphics, or signal processing customized systems can deliver energy-delay product improvements of 100-fold or more compared to general-purpose microprocessors. Representative examples of the hundreds of such systems include: low-level accelerators for H.264 and MPEG , vision and graphics accelerators [7,8,9], base band processing engines [10], and high level application customized architectures [11,12,13], and dynamically configurable systems [6,14].

Heterogeneous approaches which make major architectural change to achieve large performance gains [6,7,8] narrow their applicability, achieving these benefits to a few application areas with little benefit to the rest. Accelerators have had inefficient coupling with the CPU, increasing overhead -- complicating partitioning and scheduling. Nearly all of these accelerators have been implemented as discrete chips or add-on boards, accruing a significant cost. Discrete implementation also creates a software problem – of scarce deployment – limiting the motivation for software tool chains to support them. Finally, many accelerators developed have poor high level programmability – targeting low-level library developers, "embedding" the accelerators. In summary, the energy efficiency advantages of heterogeneity are widely acknowledged, few of these "custom" techniques have made it into large-scale deployment, and none have become the mainstream because of four critical challenges to heterogeneity: 1) Limited Breadth of Applicability: performance improvement or increased energy efficiency delivered on only a small part of the application space, 2) Inefficient Coupling: runtime cost of data movement, synchronization, format translation between computing elements, 3) Cost: the Si area/transistors and power budget, as weighed against the benefits and 4) Software (programmability, management, and maintenance): Tool chain and programming complexity; locality, synchronization, and long-term costs

3. The 10x10 Framework

Some heterogeneity exists in today's microprocessors (floating point, vector/multimedia units, etc.). However, this the lack of a systematic and integrated approach to heterogeneous computing has resulted in little progress towards mainstream adoption of heterogeneity as a dominant paradigm. However today the energy efficiency gap has become so large (as much as 500x [15,16,17]) that the incentive to solve these problems in a systematic way is tremendous. The 10x10 framework and architecture represents a systematic approach to overcoming all four critical obstacles to heterogeneity. By creating a structured approach to understanding how to introduce such heterogeneity and of understanding of its benefits and costs, the 10x10 framework creates a disciplined approach to introducing heterogeneity as an architectural element. It unlocks the potential for energy efficiency and performance gains through customization by creating a disciplined method for analyzing and optimizing for workloads which share like properties, and doing so as a subset of the larger workloads. The critical ideas include:

3.1. Application Clusters and a Broad Embrace of Heterogeneity for Energy Efficiency

The 10x10 framework begins with a broad view of the application space, departing from key traditional approaches to thinking about computer systems design. Rather than focus on the "common case" (the central assumption of 90/10 optimization [3]), 10x10 takes a broad view, recognizing that it is essential to expose the distinctive characteristics of application phases, to enable customization for more efficient execution. We will undertake a broad application analysis to identify 10 "application clusters" (each on average 10% of the workload). Working with the architects, we will negotiate these groupings to enable both programmability and major potential gains in energy efficiency and performance. Ten application clusters implies that a 10x10 architecture may incorporate as many as 10 custom micro-engines which collectively enable dramatically greater energy efficiency. We believe that such a broad embrace of heterogeneity will overcome "limited breadth of applicability", and expose both large (10x for particular clusters) and broad benefits (coverage of a large application space), creating a large enough overall benefit to enable the adoption of heterogeneity.

3.2. Deep Integration of Custom Micro-engines

The deep integration of custom micro-engines into the cores gives them equal high-speed access to the memory hierarchy and enables them to collaborate efficiently (see Figure 2). Data is shared at the highest level (nominally L1 cache). The micro-engines are aggressively power-gated when not in use. In general only one micro-engine will be active at a time, but we may diverge from this dogmatic view slightly. Data sharing between micro-engines is

then normally temporal (a micro-engine finds in the cache and memory data generated by a previous micro-engine). This deep integration via cache reduces the overhead for data sharing and synchronization, enabling micro-engines switching at fine-granularity – perhaps as few as 100 operations. Further, the 10 micro-engines can be targeted and designed to complement each other – for example one might choose to build separate short-data type engines for media and signal processing workloads, or if it makes sense in terms of cost and energy, separate them and customize. Of course, deep integration is feasible only if one is contemplating a microprocessor designed for energy efficiency (exactly the ambition of DARPA's UHPC program [26]). To achieve the most efficient coupling across micro-engines, we will use the same data representations across micro-engines (a constraint not generally respected for accelerators). This is our initial design point for 10x10 architectures, but we expect it to evolve as compelling micro-engine customization opportunities emerge (and software matures to manage increased complexity).

3.3. Exploit Poor Energy Scaling (Dark Silicon) to reduce Heterogeneity Cost

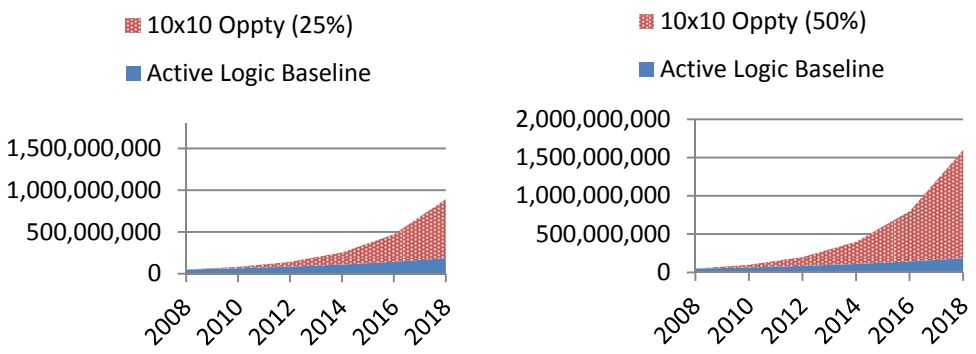

Figure 1. Scaling will increase transistor budget by 2x per generation, but modest energy scaledown (1.3x per generation) may limit the ability to exploit transistors for logic. There may be as much as 5x excess logic transistors in a 25/75 chip and 8x in a 50/50 chip for 10x10 to exploit.

The 10x10 proposal which increases the number of logic transistors dramatically might appear wasteful, but a close look at microprocessor design trends suggests otherwise. Recent CMOS technology scaling and projections continue to yield 2x transistor density, but limited energy scaling per generation. For several generations, core design has been limited by power [4,19]. Recent projections (see Figure 1) indicate that logic transistors on future microprocessors will be limited primarily by energy. With a modest allocation of chip area to logic (mostly inactive), there is indeed room for 10x10 micro-engines. For a 50% logic/50% cache (by area) microprocessor , the projections show a 10x10 transistor opportunity of over 8-fold. In a more conservative 25/75 microprocessor, the opportunity is over 4-fold. While the area is not free, using it for 10x10 micro-engines instead of additional cache (the energy feasible alternative) is likely to yield greater performance. The obvious alternative, adding more traditional cores, is *not* energy feasible. Further, if 10x10 micro-engines deliver on higher performance, say 10x faster, a 10x10 system might deliver the same performance with one-tenth the parallelism – alleviating by an order-of-magnitude the parallelism challenge all systems face. Additionally, when heterogeneous task-level and/or data-pipeline parallelism is available, 10x10 may deploy multiple micro-engines and with much higher efficiency.

3.4. Architect and Integrate Heterogeneity to simplify Software Challenges

10x10 introduces heterogeneity in an integrated fashion, and therefore under the control of a microprocessor designer. This ensures that the software requirements are architected systematically and heterogeneity carefully managed within a single design (which micro-engines, make sure they work well together) and across multiple generations (evolution and compatibility). Deep integration eases the difficulties of scheduling/partitioning. At the macro-scale, architectural management increases the software ecosystem's ability to support the heterogeneity and exploit its benefits. For example, one might reasonably expect continuity in individual micro-engines from generation to generation, and our scheme will provide simple guidance on which micro-engines are likely to perform best for different specific application phases.

There are two key technical challenges that must be addressed by the software tool chain to enable 10x10. First, the micro-engines represent additional back-end code targets for the software tool chain. This represents additional complexity, but because these micro-engines will be carefully architected for code generation, we believe that the effort to support these micro-engines will be within current complexity for the varied platforms (e.g. embedded, ARM variations, etc.) that many compilers already generate code for, and that this form of support is more realistic than the open models required by FPGA-based models [6,20]. Second, multiple micro-engines requires that the software partition the application, coordinate across use of different micro-engines (1 or at most a few at a time due to power limits), and do so intelligently. We will research simple approaches to these problems, and believe they are tractable in a 10x10 architecture framework because of the deep integration of the micro-engines.

4. Applying 10x10: an Initial Application Clustering and Strawman Architecture

It has long been recognized that application workloads have a broad variety of characteristics, and as a result, the processor benchmarking has been based on diverse collections of benchmark programs [21]. To develop and exercise the 10x10 paradigm, we use the industry-standard SPEC benchmark suite [22], the NAS parallel benchmarks [18], and the DARPA UHPC challenge applications [26], and map them into the Berkeley Motifs [23], a suite of computation types recently assembled as the basis for multicore software and hardware research and building Phil Collela's Seven Dwarves [24].

The SPEC benchmarks have been used to measure microprocessor performance for nearly twenty years. The most recent suite, SPEC2006, has two distinct components - integer and floating point. SPECint, is a collection of C/C++ programming benchmarks that test integer computations and data structure analysis. Within this suite, benchmarks range from simple decision-class games such as Go and Chess, to branch-and-bound computations, all the way to video and image compression. The SPECfp benchmarks focus on floating point calculations over a large data sets, adding Fortran applications. Many applications in the SPECfp suite are scientific modeling experiments, including biology and physics problems that rely on grid computations and N-body problems. The SPEC2006 benchmarks model the broad variety of possible use. In the case of 10x10, these programs give us a sample of the diversity of applications to develop clusters for micro-engines.

NAS parallel benchmarks consist of parallel kernel benchmarks and three simulated application benchmarks. Together they mimic the computation and data movement characteristics of large scale computational fluid dynamics applications [18]. The DARPA UHPC challenge applications reflect compelling decision support, signal processing, intelligent and semantic reasoning, graph applications, and traditional HPC applications [18].

We map these three sets of benchmarks onto a well-developed taxonomy of computational structures called the Berkeley motifs. There are 13 motifs in all, capturing classes of programs that range from dense/sparse linear algebra, to combinational logic and finite state machines. On top of the Berkeley motifs, we have added an additional motif, image and media processing. The recent increase in rich media applications has been so great that nearly all processors now have media extensions or accelerators. So, we add image/media processing as yet another application cluster that we would like to be able to work with in a 10x10 system.

Our general approach is to use static and dynamic analysis to classify and cluster application benchmarks and motifs based on their computational structure [25]. We use the benchmarks and motifs to collectively capture the range of calculations that applications do, and specific benchmarks to capture the opportunity to customize for greater efficiency and performance in a micro-engine. By nature this is an iterative process. One defines (nominally 10) different buckets, analyzes benchmarks and motifs, places them into these buckets (clusters), checks for coverage, designs a micro-engine for each cluster customized to achieve high efficiency, evaluates the resulting efficiency. The iterative step is to recluster, reimplement, and if necessary broaden the set of applications studied. A longer term goal is to find a covering set of such structures for all applications (idioms [25]). Our preliminary study involved a manual analysis of the application benchmarks in the SPEC suite, the NAS parallel benchmarks, and the UHPC challenge applications [26]. The resulting clusters are outlined in Table 1.

Table 1.Mapping Benchmark Applications into 10x10 Application Clusters

SPEC2006 Benchmark	NAS Benchmark	UHPC Challenge	Motifs/Application Clusters
milc, zeusmp, tonto, GemsFDTD	MG	Multi-scale Physics	Dense Linear Algebra
catcusADM, soplex, lbm	CG, BT, SP, LU	Implicit Solvers	Sparse Linear Algebra
povray, sphinx3	FT	SAR/DSP	Spectral Methods
omnetpp, gromacs, namd		NBODY	N-body Methods
dealII, calculix, wrf	UA	Adaptive Mesh Refinement	Structured Grids
bwaves, leslie3d		Adaptive Mesh Refinement	Unstructured Grids
gobmk, sjeng	EP	Decision Class	Map Reduce
		Data Mining	Combinational Logic
mcf, gcc, xalancmbk		Dynamic Graphs	Graph Traversal
			Dynamic Programming
astar, hmmer	DC, DT	Data Mining	Branch-n-Bound
		Dynamic Graphs	Graphical Models
Gcc, xalancmbk		Decision Class	Finite-state Machines (FSM's)
bzip2, h264ref			Image and Media Processing

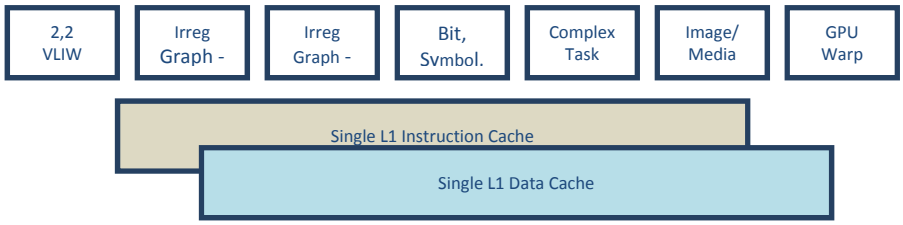

Figure 2. Straw-man 10x10 Architecture integrates the customized micro-engines above a shared memory hierarchy. These seven micro-engines are based on SPEC and 13 Motifs analysis

4.1. Mapping Application Clusters to Micro-engines

Table 2. 10x10 Application Clusters and 10x10 Micro-engines

10x10 Application Cluster	Micro-engine
Dense Linear Algebra, Spectral Methods, Structured Grids, Map Reduce	GPU Warp
Sparse Linear Algebra, Unstructured Grids	Irregular Graph - Local
Spectral Methods, N-body Methods, Unstructured Grids	Irregular Graph - Global
Combinational Logic, Finite-state Machines (FSM's)	Bit/Symbol
Graph Traversal, Dynamic Programming, Branch-n-Bound, Graphical Models	Complex Task
Finite-state Machines (FSM's)	Bit/Symbol
Image and Media Processing, Finite-state Machines (FSM's)	Image/Media

The critical idea for 10x10 design slices is then to enable customization specific data types, operations, and computational structures as embodied an application's use of an algorithm, a phase, datatypes, or even a single loop or procedure. To achieve that, in contrast to the general pressure on traditional microprocessor cores to be 90/10 efficient, the 10x10 slices should be narrow enough to enable 10x better energy efficiency in each domain. For the cluster's we've derived, a range of implementation optimizations in the micro-engines is possible. For image, media,

signal processing – fixed point and special format operations. For irregular matrix, graph-based, and n-body problems – transformation of memory access to match memory system and fast data synchronization and task scheduling. For data mining and sensor data processing - bit, symbol, character level operations. For decision class and streaming data sensor problems – micro-engine can support efficient task management, synchronization, and scheduling. For traditional HPC and regular parallelism a micro-engine which support fine-grained regular parallelism is attractive, such as a GPU WARP. The table above captures a preliminary set of application clusters and corresponding 10x10 micro-engines. While the optimized design of the micro-engine implementations is the subject of future work, we sketch several of the micro-engines to convey the opportunity for customization and concomitant benefit in energy efficiency and performance.

Irregular Graph - Local micro-engine. The impact of long memory latencies and the block memory structure cause irregular, dynamic graph applications to execute poorly on sequential and parallel machines – using memory bandwidth and the processing pipe inefficiently. This class of applications includes sparse matrices, graphs, social networks, semantic networks, etc. Source and library optimizations can produce 16-32x speedups by reordering references based on application or library level information [27], and systems such as Convey [28,29] can deliver 50x improvements based on runtime prefetching and buffering, based on reference analysis of the source code. Energy efficiency can be improved by generating the references efficiently, ordering/organizing them to match the memory system, and lightweight data synchronization triggers when sufficient memory values for efficient computation. Some potential directions include decoupled access and execute architectures, and lightweight threading and synchronization architectures [30,31] for dramatically higher performance.

Complex Task micro-engine. For decision class and high-level semantic analysis to reach actionable information, a distinctive characteristic of these applications is tree walking and search, pruning, semantic network maintenance, and many types of analysis. A critical bottleneck and cost is task management bothl within cores and across a large-scale machine. Extremely simple cores with lightweight threads and lightweight synchronization (Cyclops [31,32]) as well as specialized hardware structures customized for A* or appropriate search task queueing and prioritization coupled with the task scheduling mechanisms increase efficiency. Eliminating core complexity and queueing and synchronization cost should produce an energy-efficiency improvement of 10x [31] .

Image/Media micro-engine. Media and image processors spend significant time manipulating short data values (8, 16, or another # of bits) in packed representations, and with data-type specific operations – thresholding, scaling, shifting, etc. Special operations dealing with these datatypes, packed representations, compression, and short vector parallelism can increase efficiency significantly [33,34]. Several major elements of traditional microprocessors are not required – floating point – ops and register files and instructions, complex instruction, system management instructions, and sophisticated dynamic parallelism detection (ooo or superscalar). Smaller instruction sets (16 bit encoding), smaller instruction caches, and register files bring further gains in energy efficiency compared to a traditional 64-bit integer/FP superscalar core. Key operations such as small building-block FFT's might receive direct hardware support. A range of published examples suggest that 10-50x greater energy efficiency gains are easily accessible [15,16]. Overheads may be further reduced by 3-4x with macro-instructions which combine sets of operations in common computational idioms – reducing instruction fetch, decode, and issue overhead.

Bit/Symbol micro-engine. Bit-manipulation applications have long been a target of FPGA based customization and special instructions/operations, and delivered efficiency improvements of 10-100x. We will design a customized micro-engine which exploits bit-level manipulation across multiple operations, dramatically increasing energy efficiency and performance.

Scalar and GPU Warp micro-engine. These are likely to be conventional designs, developed over years by the research community and industry to efficiently execute scalar and GPU-parallel codes with high performance and energy efficiency. Because the 10x10 cores (each with 10 micro-engines) will be tiled in a full system, the analogous piece of a GPU would be a single "WARP"or core.

Several themes emerge for customizing and conserving energy in the micro-engines. First is the use of restricted operations and data types matched to domain (simpler, and avoid composite constructions, match precision to need). This allows simplification of the datapath – smaller register files (size and width) and ALU (operations and width), discarding in some cases large quantities of logic and storage (e.g. the entire floating point or SSE part of processor). In addition, a smaller set of instructions allows us to use shorter instruction formats, simpler decoding, smaller instruction buffer or cache, and so on. Second are customized operations crafted to implement common behaviors efficiently, matching them to the hardware (e.g. irregular reference). Third, while not discussed here, we will analyze expression trees and code traces for key applications to design special "multi-operation" instructions which

enable execution with higher energy efficiency by eliminating instruction execution overhead, reducing the number of register file accesses, and reducing pipeline overhead (see [35]). Good examples include block and string symbol operations. Finally, it is useful to combine instructions that involve things beyond computation or adding lightweight primitives (data synchronization, task queueing) which reduce overheads and eliminate instructions.

5. Software for 10x10 Architectures

As compiler technology has advanced over several decades, micro-architectures have grown in complexity (issue, pipeline, memory), but compilers struggled to span this gap, managing the underlying hardware complexity to deliver performance. To achieve success, architecture features had to be carefully designed, and compilers exploited detailed understanding of the hardware. Nonetheless, in many cases, automatic, seamless spanning of this gap has proven challenging. Recently, many successful research and commercial approaches have taken specialized approaches – employing specialized libraries hand-crafted by architecture and algorithm experts [36] and semi-automatic techniques that intelligently explore the performance space, empirically selecting the best-performing approach from a parameterized implementation [37].

The emerging heterogeneity as evidenced in GPU's and SoC accelerators (media, crypto, etc.) is beyond the reach of these techniques, requiring new *explicit* approaches to managing both the use of heterogeneous accelerators, including explicit code and data partitioning as well as customized algorithm selection and expression. Examples of explicit code and data partitioning frameworks include CUDA and OpenCL [38,39] where programmers explicitly label procedures for execution on the CPU or GPU, and can further subdivide and place computations on compute threads within the GPU. Data can be partitioned and placed in the separate memories, and because these systems have historically not included a shared memory, the placement of computation implied a placement (and movement) of the data. Explicit control is required because the choices are all performance critical. 10x10 software can exploit all of these maturing techniques (library, auto-tuning, and explicit management) to exploit heterogeneous hardware for better energy efficiency and performance. In fact, we expect all of these techniques will be both more successful and require less effort on 10x10 architectures because the deep integration of 10x10 micro-engines into a shared memory hierarchy eliminates the performance sensitivity to data partitioning – there isn't any. In addition, because our 10x10 micro-engines will be architected with the goal of being efficiently targetable, we are optimistic that many aspects of custom algorithm selection and expression will not be required. While these arguments suggest that 10x10 software will require no greater effort than existing and emerging heterogeneous platforms, we are optimistic that in the future the productive programming effort for 10x10 systems can be much lower. First, we expect that programming effort for partitioning can be gradually automated as compiler technology improves. Second, we believe that advanced software techniques such as those embodied in Transmeta, Java VM, Debik VM, CLR, OpenCL, etc.) will mature in the future to enabling a much more open view of software portability which reaches beyond todays ABI compatibility, and enables flexible, portable, energy efficient software.

6. Related Work

Systems-on-Chip (SoC's) which integrate multiple computational engines are an idea close to 10x10, but distinguished by the software and hardware design process. Compute engines on an SoC are typically glued together and interact via a shared off-chip DRAM, producing a loosely-coupled architecture. Programming of SoC's is typically done separately for each of the compute engines (and a very explicit view of the hardware specifics) and then the software glued together at a high level. As a result, SoC's have not developed the integrated software approach that 10x10 represents, and the type of architectural stability and evolution that enables development of a unified software tool set and characterizes mainstream microprocessors. Further, because SoC's continue to be driven towards flexibility late in the design (late binding, tied to performance, royalties, etc.), we suspect that they will not develop the sort of architecture and software stability which is the objective of 10x10.

Custom computing has been synonymous with discrete FPGA's, and a sizable community have explored the performance advantages of customization for over a decade [6]. Because FPGA's excel in unstructured bit-level manipulation provide the capability to hard-wire computational structure (data flow, control flow), and scale up computation (many ALU's), there are numerous applications where an FPGA implementation can outperform a traditional microprocessor. However, FPGA-based solutions have failed to achieve widespread adoption, much less

displace mainstream microprocessors, because of their inherent speed and energy disadvantages (when compared to custom hard-wired design) and the difficulty of application development of FPGA-based solutions. Notably, key former advocates are questioning the capability of FPGA-based systems to "ever catch up".

The rise of heterogeneity is evidenced in the increasing popularity of hybrid systems which incorporate new kinds of accelerators, increasing peak performance per watt significantly at a cost in programming effort. Examples include IBM's CELL accelerator [40], Graphics processing units programmed with CUDA and OpenCL [38,39], and most recently, integrated CPU/GPU systems on the same chip [41,42] Research studies have explored more integrated configurable logic, exploring pipeline integration for compute intensive loops [43], customized cores for different workloads [44], and even tuning energy efficiency by "compiling" the hot routines into configurable logic [45]. These approaches all reflect the same energy-scaling/speed challenges that 10x10 addresses, but do not take a systematic architectural approach tied to custom hard-wired VLSI design. Recently, as the need for focus on energy-efficiency has become clear, there have been studies on the cost of general purpose architecture which compare to accelerator/ASIC [15], and extreme new approaches for microprocessor implementation which achieve 30x energy improvement [16]. These efforts are complementary to the 10x10 approach, and their results support the progress of developing 10x10 architectures.

7. Summary and Future Work

As described extensively in [4], the properties of CMOS technology scaling have changed dramatically, and the architecture and software of computing systems must respond to the new realities of energy efficiency as a critical goal, slower transistors, and a continuing bounty of transistor density. These changes are stressing existing software paradigms by demanding dramatically higher parallelism, creating an opportunity for heterogeneity in architecture – customization for greater performance and energy efficiency. 10x10 is a new paradigm which allows architects and system designers to systematically study and cluster application structure, and map it into customized architecture in a structured fashion. This new paradigm is required because the traditional, 90x10 paradigm obscures this structure – by design – in its focus on general-purpose architecture. This allows customization to be exploited systematically, and managed to stability from an architectural perspective. The 10x10 paradigm gives rise to a new type of architecture, in which multiple customized micro-engines are both higher performance and more energy efficient. In addition, the systematic management of heterogeneity that the 10x10 paradigm provides enables existing software tool chains to be leveraged. We have outlined one application of the 10x10 paradigm to the SPEC and Berkeley Motif benchmarks which gives rise to a seven micro-engine 10x10 architecture.

We are only at the beginning for 10x10. Numerous future research challenges remain, including many questions about application clustering and micro-engines. What is the right balance of generality and customization? How do we optimize performance, energy efficiency, and of course the ability to leverage existing software tool chains? By doing real implementation studies, what are the critical implementation costs (sharing of the L1 caches, tile size stretch?, ability to rapidly power on/off) and do these mitigate some of the 10x10 benefits? Of course there are chip-level issues of memory hierarchy and parallel coordination – how do we partition/fuse memory system and compute system customization? Should there be special support for coordination across the micro-engines on a chip? How much overall energy efficiency and performance opportunity is exposed by 10x10, and how much of it is tapped by the software tool chain? At what granularity can we execute efficiently (energy and code path/time) across micro-engines? How do we partition/schedule to exploit the heterogeneity? These are just a few, and if anything, each question we answer raises more interesting questions for future 10x10 paradigm studies and 10x10 architectures.

Acknowledgements

This work was supported in part by NSF Grant OCI-1057921 and inspired by the Exascale studies chartered by Dr. Bill Harrod of DARPA. We acknowledge colleagues at Intel who have improved our understanding of these issues through many thoughtful discussions.

References

1. DARPA Exascale Hardware Study, Exascale Computing Study: Technology Challenges in Achieving an Exascale Systems. http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf, 2008.

2. Kaxiras and Martonosi, “Computer Architecture Techniques for Power Efficiency”, Morgan-Claypool, 2008.

3. J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Prentice-Hall, 5th Edition, 2006.

4. Shekhar Y. Borkar and Andrew A. Chien, The Future of Microprocessors: Technology Scaling: Changing Technology Landscape Shapes the Next 20 Years of Microprocessor Architecture, Communication of the Association for Computing Machinery, to appear, Spring 2011.

5. Proceedings of the ACM/IEEE International Symposium on Computer Architecture, 1995 through 2010, numerous papers.

6. IEEE Workshop on Field-programmable Custom Computing Machines, 2000-2010.

7. Imagination Technologies, Power VR Technology Overview, http://www.imgtec.com/, 2009.

8. Fuchs, et. al. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories, SIGGRAPH 1989.

9. A. Zein, et. al. Performance Evaluation of the NVIDIA GeForce 8800 GTX GPU for Machine Learning. (ICCS '08), 2008.

10. Robert Broderson. 2004. A Conversation with Teresa Meng. Queue 2, 1 (March 2004), 14-21.

11. J. Makino, et. al. GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing, ACM/IEEE Conference on Supercomputing, 2007.

12. Shaw, et. al. Anton, a Special-Purpose Machine for Molecular Dynamics Simulation, ISCA 2007, San Diego, CA, 2007.

13. D. Donofrio, et. al. Energy-Efficient Computing for Extreme-Scale Science, IEEE Computer, November 2009 (vol. 42 no. 11), pp. 62-71.

14. SRC: Delivering programmer-friendly reconfigurable processor-based computers. http://www.srccomp.com/

15. R. Hameed, et. al. Understanding sources of inefficiency in general-purpose chips, ISCA 2010.

16. James Balfour, "Efficient Embedded Computing" . Stanford University Ph.D. Thesis, 2010.

17. Mudge, et. al. Near-Threshold Computing: Reclaiming Moore’s Law Through Energy Efficient Integrated Circuits, Proceedings IEEE | Vol. 98,No. 2, February 2010.

18. D. H. Bailey,et. al. NAS parallel benchmark results. ACM/IEEE conference on Supercomputing (Supercomputing '92), 1992.

19. Chang, et. al. Practical Strategies for Power-Efficient Computing Technologies, Proceedings IEEE, Vol 98, No. 2, Feb 2010.

20. Gokhale, et. al. “Building and Using a Highly Parallel Programmable Logic Array”, IEEE Computer, Volume 24, Issue 1, January 1991.

21. R. Jain, The Art of Computer Systems Performance Analysis, Wiley, New York, 1991.

22. Standard Performance Evaluation Corporation (SPEC), http://www.spec.org/

23. Asanovic, et. al. The Landscape of Parallel Computing Research: A View from Berkeley, UCB/EECS-2006-183, Dec 2006. (Motifs)

24. Phillip Colella, Defining Software Requirements for Scientific Computing, 2004 (the Seven Dwarves)

25. Olschanowsky, C., Mitesh Miswani, Laura Carrington, Allan Snavely, PIR: PMaC's Idiom Recognizer, PSTI 2010.

26. DARPA UHPC - https://www.fbo.gov/download/3d7/3d7cfa30b2b1a93332a444047dea52d8/UHPC_BAA_final_3-2-10_post_(2).pdf

27. Bader, et. al., “Scalable Graph Exploration on Multicore Processors”, in Proceedings of the ACM/IEEE paper on Supercomputing, 2010.

28. Snavely, et. al. “Modeling the effect of acceleration hardware: A study of Gather-scatter on the Convey”, submitted for publication, 2010.

29. “The Convey HC-1 Computer: Architecture Overview”, Convey Computer, November 2008.

30. J. Smith, “Decoupled Access/Execute Computer Architectures”, ACM Transactions on Computer Systems, 1984, pp. 112—119.

31. Cuvillo, et. al. Toward a Software Infrastructure for the Cyclops-64 Cellular Architecture, (HPCS'06), 2006.

32. The Cray XMT Supercomputer, www.cray.com, 2007.

33. The Streaming SIMD Extension. http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions

34. Advanced Vector Extensions http://en.wikipedia.org/wiki/Advanced_Vector_Extensions

35. Hauck and Ebeling, et. al. “Totem: Domain-Specific Reconfigurable Logic”, submitted for journal publication, 2010.

36. S. Taylor, Intel Integrated Performance Primitives - How to Optimize Software Applications Using Intel IPP, Intel Press, April 2004.

37. Auto-tuning

38. Nvidia Corporation. CUDA Programming Guide Version 2.0, June 2008.

39. The Khronos Group. OpenCL, the Open Standard for Heterogeneous Parallel Programming, February 2009.

40. J. Kahle, et. al. Introduction to the Cell Multiprocessor, IBM Journal of R&D, Vol. 49, No. 4/5, July/Sept. 2005, pp. 589-604.

41. Intel Sandy Bridge - http://en.wikipedia.org/wiki/Sandy_Bridge_(microarchitecture)

42. AMD Fusion http://en.wikipedia.org/wiki/AMD_Fusion

43. M. Budiu, et. al. Spatial computation. ASPLOS-XI, 2004, pgs 14-26.

44. G. Venkatesh, et. al. Conservation cores: reducing the energy of mature computations. ASPLOS ’10, (March 2010), 205-218.

45. N. Goulding, et. al. GreenDroid: A Mobile Application Processor for Silicon’s Dark Future, Proceedings of HotChips, 2010.