Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

Invited Paper

Heterogeneous Multi-core Architectures

Tulika Mitra[1,a)]

Received: March 23, 2015, Released: August 1, 2015

Abstract: Transistor count continues to increase for silicon devices following Moore's Law. But the failure of Dennard scaling has brought the computing community to a crossroad where power has become the major limiting factor. Thus future chips can have many cores; but only a fraction of them can be switched on at any point in time. This dark silicon era, where significant fraction of the chip real estate remains dark, has necessitated a fundamental rethinking in architectural designs. In this context, heterogeneous multi-core architectures combining functionality and performance-wise divergent mix of processing cores (CPU, GPU, special-purpose accelerators, and reconfigurable computing) offer a promising option. Heterogeneous multi-cores can potentially provide energy-efficient computation as only the cores most suitable for the current computation need to be switched on. This article presents an overview of the state-of-the-art in heterogeneous multi-core landscape.

Keywords: heterogeneous multi-core, CPU, GPU, special-purpose accelerator, reconfigurable computing

1. Introduction

The emergence of multi-cores and eventually many-cores (multiple processing cores or CPU on a single chip) has brought the computer community to a crossroad. Desktops, laptops, and smartphones have all made the irreversible transition toward multi-cores due to thermal/power constraints, reliability issues, design complexity, and so on. As transistor density continues to enjoy the exponential growth according to Moore's Law, the number of on-chip cores (but not the performance) is predicted to double every two years. However, we cannot ride this growth curve for on-chip cores in the future. The primary challenge turns out to be the increasing power density on chip that prevents all the cores to be switched on at the same time. This phenomenon, known as dark silicon, is driving the evolution of heterogeneous multi-core architectures.

Traditionally, multi-cores are just designed as just a collection of identical (possibly simple) cores. These homogeneous multi-cores are simple to design, offer easy silicon implementation, and regular software environments. Unfortunately, general-purpose emerging workloads from diverse application domains have very different resource requirements that are hard to satisfy with a set of identical cores. In contrast, there exist many evidences that heterogeneous multi-core solutions consisting of different core types can offer significant advantage in terms performance, power, area, and delay. At the same time, heterogeneous multi-cores are perfect fit for the dark silicon regime as only the cores suited for an application need to be switched on.

This article delves into heterogeneous multi-core architectures with their associated challenges and opportunities. We first present the technological, architectural, and application trends that are responsible for the advent of heterogeneity. We broadly classify heterogeneous multi-cores into performance heterogeneity, where cores with the same functionality but different power-performance characteristics are integrated together and functional heterogeneity, where cores with very different functionality are interspersed on the same die. We present performance heterogeneous multi-core architectures and the software support required to realize the full potential of such architectures. We then proceed to give an overview of different kinds of processing elements present in current functional heterogeneous multi-cores, such as graphics processing cores, special-purpose accelerators, and reconfigurable computing.

Heterogeneous multi-cores is an evolving technology at this point with myriads of challenges. This opens up interesting research problems in all aspects of the computing stack, starting from devices, circuits, architecture all the way to the software layer including operating systems, compilers, and programming paradigms. We invite the readers to explore these exciting opportunities offered by heterogeneous multi-core landscape.

2. Background

The first microprocessor was introduced in 1971 in the form of Intel 4004 — a 4-bit general purpose programmable CPU on a single chip. Since then, the microprocessor industry has witnessed unprecedented growth in performance fueled by multiple factors: Moore's Law, Dennard scaling, and micro-architectural innovations. **Moore's Law** [50] is an observation made by Gordon Moore in 1965 that the number of transistors incorporated in a chip approximately doubles every 18–24 months, resulting in an exponential increase in transistor density. As the processing speed is inversely proportional to the distance between transistors on an integrated circuit, Moore's Law implies that the speed (clock frequency) of the microprocessors will also double every 24 months. This exponential growth in processor speed contin-

[1] School of Computing, National University of Singapore 117417, Singapore
[a)] tulika@comp.nus.edu.sg

Information and Media Technologies 10(3): 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

ued unabated till about 2005. For example, the 64-bit Intel Xeon processor launched in 2005 runs at 3.8 GHz and has 169 million transistors. This is in stark contrast to Intel 4004 that had a clock frequency of 740 KHz and only 2,300 transistors.

While Moore's Law was responsible for the sustained increase in clock frequency, the processor performance improved further due to several micro-architectural innovations including processor pipeline, out-of-order execution, speculation, cache memory hierarchy. These advancements enabled the processor to execute multiple instructions per cycle by exploiting instruction-level parallelism (ILP), boosting the critical instructions-per-cycle (IPC) metric [29]. More importantly, as the ILP was extracted transparently by the underlying architecture from single-threaded programs, the software developers enjoyed the performance benefit without any additional effort. Together, the growth in clock frequency and IPC ensued the relentless gain in processor performance spanning over three decades.

Rise of Homogeneous Multi-core

The performance growth in uni-processors slowed down and came to an end due to a variety of reasons: **power wall**, **ILP wall**, and **memory wall** [57]. The primary limiting factor is the so called power wall defined by the thermal design power (TDP) — the maximum amount of power a micro-processor chip could reasonably dissipate. For almost 30 years, Moore's Law was aided by **Dennard scaling** [19] to keep the processor power within limit. According to Dennard's theory, with a linear feature size scaling ratio of $\frac{1}{\sqrt{2}}$ (one process technology generation to another), the transistor count doubles (Moore's Law), frequency increases by 40%, but power per transistor can be reduced by $\frac{1}{2}$ keeping the total chip power constant [21]. The reduction in power per transistor is achieved by scaling down the operating voltage and current proportional to the feature size scaling. Scaling down the operating voltage requires scaling down the threshold voltage (V_{th}), which was feasible while leakage power was minimal till 2005. However, at 65 nm process technology and below, leakage power contributes significantly to total chip power and lowering V_{th} results in exponential increase in leakage power. Thus V_{th} and in turn the operating voltage cannot be scaled any longer (because as operating voltage approaches V_{th}, the transistor delay increases rapidly, resulting in a drop in the clock frequency) causing the breakdown of Dennard scaling. It is no longer possible to keep the power envelope constant from generation to generation. Instead post-Dennard scaling leads to power increase by a factor of 2 per generation for the same die area. Keeping the power within TDP budget requires the chip to operate at a lower frequency than the native frequency. As dynamic power is roughly proportional to cubic frequency, keeping the frequency lower (instead of 40% increase per generation) can compensate for the increase in leakage power due to the doubling of transistor count. Indeed clock speed has stalled at roughly 4 GHz in the past decade even as transistor count has exploded following Moore's Law (e.g., 4.31 billion transistors in Intel Ivy Bridge-EX processor launched in 2014).

In post-Dennard scaling era, the abundance of transistors could have been utilized to build more complex uni-processor micro-architectures to further improve the IPC and thereby compensate the lack of increase in clock speed. But the processors had already hit the ILP wall. It was increasingly difficult to find any more parallelism in single-threaded programs [76]. The heroic attempt to uncover further ILP caused superlinear increase in processor complexity [53] and associated power consumption without linear speedup in application performance. Thus such architectural-level optimizations quickly reached the point of diminishing return. Moreover, the exponentially growing performance gap between processor and main memory (DRAM) led to the memory wall [79] where system performance is dominated by memory performance; making the processor faster (either in terms of IPC or clock speed) will not affect the execution time of an application. The ILP wall and the memory wall together precluded the introduction of any more complexity in uni-processor design.

Computing systems, at this point, made an irreversible transition towards multi-core architectures in order to effectively employ the growing number of transistors on chip [24]. A multi-core processor incorporates two or more processing elements (cores) on a single chip. As transistor density goes up from one process generation to another, individual core frequency and complexity are kept constant or even reduced to accommodate for the breakdown of Dennard scaling. Thus single-core performance remained constant or degraded. This lack of single-core performance is compensated by the presence of a large number of cores that exploit thread-level parallelism (TLP) where the threads are distributed across the cores and are executed in parallel. If the code can be parallelized appropriately, the performance of the application will be greatly improved even though each core is running at lower frequency. The first commercial dual-core processor is IBM Power 4 introduced in 2001, while the first dual-core processor for desktop platforms (Intel Pentium Extreme Edition 840) was introduced in 2005. As of now, multi-cores are prevalent in all computing systems starting from smartphones to PCs to enterprise servers.

Most multi-core architectures designed in the past decade are *homogeneous multi-core*, that is, all the cores are identical both in terms of instruction-set architecture (ISA) and the underlying micro-architecture (pipeline, cache, branch prediction configuration). Homogeneous multi-cores are easy to design and verify as we simply need to replicate the core. So it was predicted that the number of cores on chip will grow exponentially following Moore's Law. This trend has been maintained so far; for example, the latest Intel Xeon Processor E7 consists of 15 cores. However, just like the upward trajectory of single-core performance came to a halt, simply increasing core count in homogeneous multi-core will not remain profitable for very long.

Dark Silicon Era

Esmaeilzadeh et al. in a seminal paper [21] model multi-core scaling limits by combining device scaling, single-core scaling, and multi-core scaling to predict the speedup potential for parallel workloads for the next five technology generations. The paper concludes that power and parallelism limitations will create a large gap (13 X at 8 nm technology node with ITRS projections [13] for device scaling) between achievable performance and the performance expected by Moore's Law (doubling performance every technology node). More importantly, increasing

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

core count hits the power budget after certain point. Beyond this point, more cores can be added; but these additional cores have to be switched off or significantly under-clocked to satisfy the TDP constraint. This phenomenon, where a significant fraction of the cores have to be either idle (dark) or under-clocked (dim) at any point in time, has been termed as "Dark Silicon" [49]. The study claims that power limitations will severely curtail the usable chip fraction: at 22 nm, 21% of the chip will be dark and at 8 nm, over 50% of the chip will not be utilized using ITRS device scaling projections. Architectural decisions, power/thermal management challenges, and reliability/variability challenges in the dark silicon era from hardware/software codesign perspective have been discussed in Ref. [64].

Interesting, parallelism rather than power turns out to be the primary contributing factor towards the performance gap for most workloads [21] in the dark silicon era. Very few applications have enough parallelism and still cannot realize the performance potential due to insufficient core count arising out of power limitations. Most of the contemporary and emerging applications are not perfectly parallelizable to take advantage of the abundant TLP offered by homogeneous multi-core architectures. **Amdahl's Law** [2] states that the speedup on a multi-core will be limited to $\frac{1}{S}$ where S is the serial fraction of the application. Even for an application with 99% parallel code, the remaining 1% sequential fraction limits the speedup to 100 X with an infinite number of cores. Thus most applications will have limited speedup with homogeneous multi-cores even with unlimited power budget.

Taylor [72] categorizes the potential architectural responses to the impending dark silicon era into four classes. The most obvious and pessimistic approach is the *shrinking chip*, where the chip designers simply build smaller chips. This approach not only hinders all architectural advancements, but is also problematic from the point of view of cost, revenue, power and packaging issues. The second possibility is *dim silicon* where the logic is severely under-clocked to stay below the power budget. Dim silicon techniques include multi-cores based on near-threshold voltage computing [20], larger on-chip caches rather than logic to absorb the dark silicon area, spatial dimming through dynamic voltage-frequency scaling (DVFS), and temporal dimming through computational sprinting [61] or Intel Turbo Boost [63] where the chip is allowed to exceed the power budget momentarily to provide short but substantial boost to performance followed by long period of low-power operation. "Dim silicon" is effective only for workloads with high degree of parallelism; but as lack of parallelism rather than power is the main barrier for most workloads to achieve high speedup, this approach does not have universal appeal. A more futuristic solution is *ultra low-power circuits* where transistors are built from new and emerging technology that provide better sub-threshold characteristics than CMOS devices [34] and hence can improve energy-efficiency substantially. Last but not the least, a promising approach and the one we focus on in this article to circumvent dark silicon regime is *heterogeneous multi-core* architectures.

In the dark silicon era, the silicon area becomes an exponentially cheaper resource relative to power and the architects can potentially "spend" area to "buy" energy-efficiency [72]. This is the guiding principle behind heterogeneous multi-core architectures where different types of cores co-exist on the same die. Given an application, only the cores that best fit the application can be activated leading to faster and energy-efficient computing. Heterogeneous computing architectures can be broadly classified into two categories: *performance heterogeneity* and *functional heterogeneity*.

3. Performance Heterogeneous Multi-core

Performance heterogeneous multi-core architectures consist of cores with different power-performance characteristics but all sharing the same instruction-set architecture. The difference stems from distinct micro-architectural features such as in-order core versus out-of-order core. The complex cores can provide better performance at the cost of higher power consumption, while the simpler cores exhibit low-power behavior alongside lower performance. This is also known as single-ISA heterogeneous multi-core architecture [39] or asymmetric multi-core architecture. The advantage of this approach is that the same binary executable can run on all different core types depending on the context and no additional programming effort is required. However, either the system designer or the runtime management layer has to identify the appropriate core type for each application or even for different phases within the same application. Asymmetric multi-cores can be further classified into *static asymmetric multi-core* and *dynamic asymmetric multi-core* depending on whether the mix of cores can be configured at runtime.

3.1 Static Asymmetric Multi-core

In static asymmetric multi-cores, the mix of difference core types are fixed at design time. For example, **Fig. 1** shows a static asymmetric multi-core consisting of 12 small cores and 1 big core. Examples of commercial asymmetric multi-cores include ARM big.LITTLE [26] integrating high-performance out-of-order cores with low-power in-order cores, nVidia Kal-El (brand name Tegra3) [52] consisting of four high-performance cores with one low-power core, and more recently Wearable Processing Unit (WPU) from Ineda consisting of cores with varying power-performance characteristics [33]. An instance of the ARM big.LITTLE architecture integrating quad-core ARM Cortex-A15 (big core) and quad-core ARM Cortex-A7 (small core), as shown in **Fig. 2**, appears in the Samsung Exynos 5 Octa SoC driving high-end Samsung Galaxy S4 smart-phones.

Performance heterogeneous asymmetric multi-core architec-

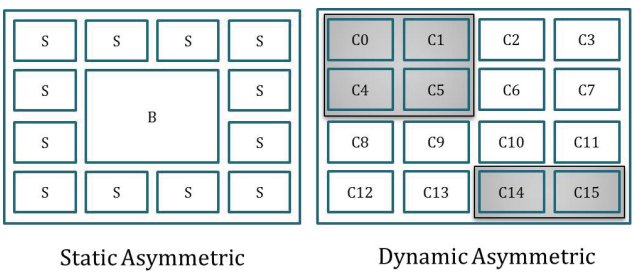

Fig. 1 Performance heterogeneity: Static and dynamic asymmetric multi-core architectures.

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

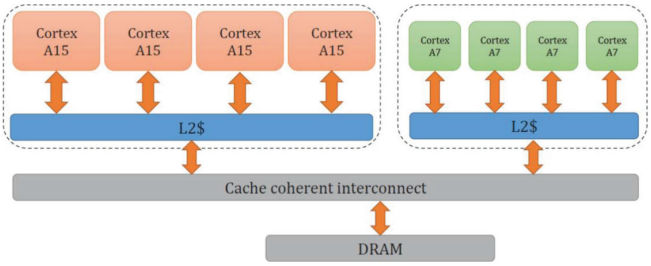

Fig. 2 ARM big.LITTLE static asymmetric multi-core.

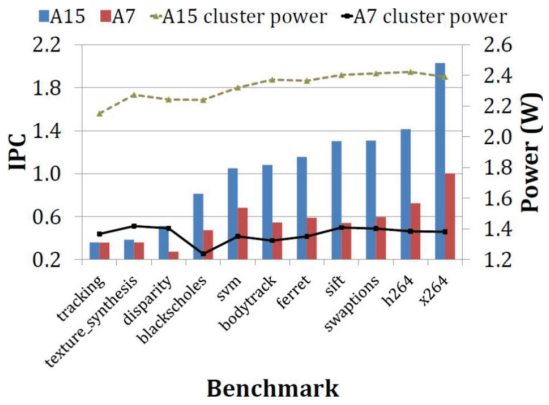

Fig. 3 Power-Performance characteristics of small, big cores.

tures are a promising solution to two related but critical problems today: performance limits due to Amdahl's Law [2] and energy efficiency of multi-core computing in the dark silicon era. Hill and Marty in a seminal paper [30] argue that heterogeneous multi-cores where the sequential code fragment can be mapped to a complex core — capable of exploiting instruction-level parallelism (ILP), for example, through out-of-order execution and hence accelerate the execution of the sequential fraction — improve the speedup of the application quite dramatically. This is because the parallel portion of the code can be accelerated through the array of simple cores offering TLP while the sequential portion can be expedited by exploiting ILP through the complex core.

The energy-efficiency advantage of heterogeneous computing is quite obvious. At any point, we simply need to turn on the core(s) that is most power-efficient for the current computing need without negatively impacting the performance. For example, in a smartphone, the low-power small core can take care of simple tasks such as email client, web browsing etc. saving energy, while the complex core has to be switched on for compute-intensive tasks such as 3D gaming, browsing flash-based websites etc. sacrificing energy. This model of computing fits in well in the dark silicon era where thermal constraints anyway restrict the fraction of cores that can be switched on at any point in time; so it is beneficial to switch on the appropriate cores for better energy efficiency. Note that apart from micro-architectural differences, these architectures offer additional design points in the power-performance trade-off curve through dynamic voltage-frequency scaling (DVFS) of the cores.

Figure 3 shows the power-performance heterogeneity of heterogeneous multi-core architecture for commercial ARM big.LITTLE architecture as reported in Ref. [60]. The evaluation platform we use is the Versatile Express development platform comprising of a prototype chip with two Cortex-A15 cores and three Cortex-A7 cores at 45 nm technology. All the cores implement ARM v7A ISA. While each core has private L1 instruction and data caches, the L2 cache is shared across all the cores within a cluster. The L2 caches across clusters are kept seamlessly coherent via the cache coherent interconnect so that an application can be easily migrated from one cluster to the other. The architecture provides DVFS feature per cluster. But all the cores within a cluster should run at the same frequency level. Moreover an idle cluster can be powered down if necessary. The chip is equipped with sensors to measure frequency, voltage, power, and energy consumption of each cluster as well as performance counters.

Figure 3 plots the Instructions Per Cycle (IPC) and the average power (Watt) for benchmark applications on Cortex-A7 and Cortex-A15 cluster, respectively. In this experiment, we set the the same voltage (1.05 Volt) and frequency (1 GHz) level for the two clusters and utilize only one core at a time to run the benchmark. Note that we can only measure the power at cluster level rather than individual core level. So the power reported in this figure corresponds to the power in a cluster even though only one core is running the benchmark application, while other cores are idle. Clearly, A15 has significantly better IPC compared to A7 (average speedup of 1.86) but far worse power behavior (1.7 times more power than A7 on an average).

Software Support

Sophisticated runtime management techniques are required to leverage the unique opportunity offered by heterogeneous multi-cores towards energy-efficient computing. This includes (a) determining the core type that is most suitable for an application or the phase of an application, (b) moving the application to the appropriate core through task migration, and (c) setting the proper voltage-frequency level for the cores such that the performance requirements of the applications are satisfied at minimal energy while not exceeding the thermal design power (TDP) constraint.

The first step required in this process is an accurate power-performance estimation mechanism. As an application is running on one core type, one needs to predict its power, performance behavior on the other core types and at different voltage-frequency levels so as decide whether the application should be migrated and to where. Such a power-performance model for ARM big.LITTLE architecture has been developed in Ref. [60]. This modeling is challenging for a real architecture for various reasons. First, the big core and the small core are dramatically different, not just in the pipeline organization, but also in terms of cache hierarchy and the branch predictor. Thus given the cache miss rate or branch misprediction on one core type, we have to estimate the same for the other core type. Second, we are constrained by the performance counters available on the cores and cannot assume additional profiling information that is simply not available such as inter-instruction dependency. These challenges are overcome through a combination of static program analysis (to identify inter-instruction dependency), mechanistic modeling that builds analytical model from an understanding of the underlying architecture (such as impact of pipeline stalls due to inter-instruction dependency and resource constraints versus miss

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

events on performance), and empirical modeling that employs statistical inferencing techniques like regression to create an analytical model (for inter-core miss events estimation).

These prediction can then be employed to choose the suitable core type for an application or the phase of an application and its DVFS. For performance optimization, Craeynest et al. [73] propose a scheduling technique for asymmetric multi-cores using online performance estimation across different core types. Similarly, Koufaty et al. [38] propose a dynamic heterogeneity aware scheduler, which schedules tasks with very low memory stalls on complex cores for higher performance. But these works do not consider power issues. A study by Winter et al. [77] evaluates various scheduling and power management techniques for heterogeneous multi-cores with special considerations to the scalability of the approaches. They propose a thread scheduling algorithm called Steepest Drop, which has little overhead but does not consider frequency scaling of the cores.

A control-theory based approach that synergistically integrates multiple controllers (handling different constraints or optimization goals) to achieve energy-efficiency for multiple applications running on heterogeneous multi-core system has been proposed in Ref. [51]. However, this approach suffers from scalability issues due to centralized decision making regarding task migration and power allocation among the cores under tight TDP constraint. This scalability issue is addressed through a distributed approach based on the solid foundations of price theory from economics in Ref. [68]. The resource allocation, DVFS, task mapping, and migration are all controlled through the virtual market place, where the commodity being traded is processing unit using virtual money. The framework is realized as a collection of autonomous entities called agents, one for each task, core, cluster, and the entire chip. The performance requirement is modeled as the demand while the processing capability is modeled as the supply (depends on core type and frequency). The principle of price theory states that the market is only stable at a price equilibrium, which is the price at which the supply is equal to the demand and hence corresponds to the minimal energy consumption. Across a range of workloads, the price theory based power management framework reduces average power consumption to 2.96 W compared to 5.99 W for Linux heterogeneity-aware scheduler (which makes naive task migration decision) plus on-demand governor (for DVFS) at the same or even better performance level.

3.2 Dynamic Asymmetric Multi-core

Even though static asymmetric multi-cores are clearly positioned to accommodate software diversity (mix of ILP and TLP workload) much better than homogeneous multi-cores with promising results, they are still not the ideal solution. As the mix of simple and complex cores has to be frozen during design/fabrication time, a static asymmetric multi-core lacks the flexibility to adjust itself to the dynamic nature of workload. Any change in the applications requirements would have a big impact on the production costs. The next logical step forward to support both diverse and dynamic workload is to design dynamic asymmetric multi-cores that can, at runtime, tailor itself according to the applications [30]. Such adaptive architectures are physically fabricated as a set of simple, homogeneous cores. At runtime, two or more such simple cores can be coalesced together to create a more complex virtual core. Similarly, the simple cores participating in a complex virtual core, can be disjoined at any point of time. A canonical example is to form coalition of two 2-way out-of-order (ooo) cores to create a single 4-way ooo core. In other words, we would like to dynamically create static asymmetric multi-cores through simple reconfiguration. Figure 1 shows a dynamic asymmetric multi-core architecture consisting of 16 base cores that has been configured at runtime to create one medium core and one big core. The following section presents some of the dynamic asymmetric multi-core architectures proposed in the literature.

The Core Fusion architecture [35] presents a detailed architectural solution to support runtime core coalescing. The physical substrate comprises of identical, relatively efficient 2-issue out-of-order cores. At runtime, these cores can be fused together to create larger (eight-issue or four-issue) out-of-order cores. The proposed architecture has a reconfigurable, distributed front-end and instruction cache organization that can leverage individual core's front-end structure to feed an aggressive fused back-end, with minimal over-provisioning of individual front-ends.

Kumar et al. provides another interpretation of the core coalition in Conjoined-Core Chip Multiprocessing [40] where neighboring cores can share complex structures (e.g., floating-point units, crossbar ports, instruction caches, data caches, etc.) thereby saving significant area.

Federation [71] architecture proposed by Tarjan et al., on the other hand, enables a pair of scalar cores to act as a 2-way out-of-order core by inserting additional stages to their internal pipelines. The key insight that makes federation work is that it is possible to approximate traditional out-of-order issue with much more efficient hardware structures (e.g., replacing content addressable memory and broadcast networks with simple lookup tables). These out-of-order structures are then placed between a pair of scalar cores and in conjunction with the fetch, decode, register file, cache, and datapath of the scalar cores, one can achieve an ensemble that is competitive in performance with an out-of-order superscalar core. Federated cores are best suited for workloads that usually need high throughput but sometimes encounter sequential code fragment. Federation provides faster, more energy-efficient virtual out-of-order cores for sequential workloads without sacrificing area that would reduce thread capacity for parallel workload.

A hardware-software co-designed approach towards dynamic asymmetric multi-core design is presented in Bahurupi architecture [58]. Bahurupi is physically fabricated as a set of clusters, each containing four simple 2-way out-of-order cores. **Figure 4** shows an example of 8-core Bahurupi architecture with two clusters (C0–C3) and (C4–C7). At runtime, two or more such simple cores within a cluster can form a coalition to create a more complex virtual core. Similarly, the simple cores participating in a complex virtual core, can be disjoined at any point of time. Thus we can create diverse range of heterogeneous multi-cores on-demand through simple reconfiguration. The highlighted cores in Fig. 4 are involved in two coalitions of two (C0, C1) and four

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

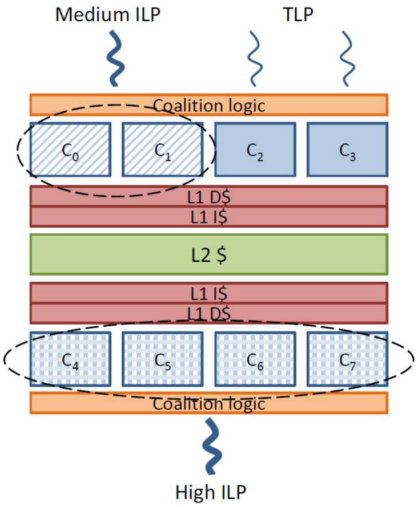

Fig. 4 Bahurupi dynamic heterogeneous multi-core.

(C4–C7) cores. In this example one parallel application runs its two threads on cores C2 and C3, one medium-ILP sequential application is scheduled to coalition (C0–C1) and one high-ILP sequential application is scheduled to coalition (C4–C7). Careful task scheduling on Bahurupi architecture [59] yields speedup ranging from 10% to 62% compared to static homogeneous and heterogeneous multi-cores across a large range of task sets.

When running in coalition mode, participating cores cooperatively execute a single thread in a distributed fashion. Basically, the cores execute basic blocks of the sequential thread in parallel and fall back to a centralized unit for synchronization and dependency resolution. Dependency comes in the form of control flow and data dependence. Bahurupi handles these with compiler support and minimal additional hardware.

A new instruction called sentinel instruction is added to the ISA, which is the first instruction of each basic block of the code. Basic blocks are constructed at compile time along with the information about live-in and live-out registers to/from each basic block. This information is encoded in the corresponding sentinel instruction, which also embeds control flow information, specifically, length of the basic block and whether it ends with a branch instruction. Thus, sentinel instructions capture both the dependency and the control flow information among the basic blocks.

Physically, all the cores share a global PC, synchronization logic, global register file, and global renaming logic. Cores participating in the coalition make use of these global structures while other cores can independently run different threads. The only communication across the cores is through the live-in and live-out register values. A broadcast mechanism is used to let the producer core send its live-out register value to any consumer core. Each core snoops the broadcast bus for live-in registers. The architecture includes a cache structure with reconfigurable banked L1 instruction and data caches where each bank is associated with a core. Cores participating in the coalition share a combined L1 instruction and data cache reconfigured from their banks.

The results presented in Ref. [58] show that, in case of integer applications, a 2-core or 4-core coalition can perform very close to a 4-way or 8-way out-of-order processor, respectively. On the other hand, for floating point applications, a 2-core or a 4-core coalition can even outperform a 4-way or 8-way true out-of-order processor. This is because Bahurupi can look far ahead in the future to exploit ILP as the compiler resolves dependencies across basic blocks.

The architectures described so far combines simple physical cores to create complex virtual cores. Unlike other solutions, Morphcore architecture [36] starts off with a traditional high performance out-of-order core and makes internal changes to allow it to be transformed into a highly threaded in-order SMT core when necessary. MorphCore modifies the internal pipeline of an out-of-order core by adding components that allow fast switching between out-of-order mode and in-order mode. The fetch stage is modified such that it can switch between 8 threaded in-order SMT core and a dual-issue out-of- order core. The decision to switch between execution modes is automatically taken care of by the hardware and not by the operating system. Generally, when the OS spawns more than two tasks (threads), the hardware switches to SMT mode and when the number of threads reduces to less than two, then the hardware switches to out-of-order mode.

Composite Cores architecture [44] has some similarity with MorphCore. The architecture allows fast switching between in-order and out-of-order execution. Essentially, there are two different pipelines connected together on the same CPU die — an out-of-order pipeline and an in-order pipeline. The connectivity between these two allows fast migration of processes from one engine to another. The two engines share the front-end of the pipeline, the branch predictor and the instruction, data caches. An extra hardware component is added to the system — a reactive PID controller that is in charge of detecting when to migrate from one pipeline to another. The online controller tries to maximize energy savings by choosing the right core configuration at runtime. The controller integrates a complex performance estimator to decide where the task will be migrated.

Other works approached the idea of dynamic asymmetric multi-core systems together with code parallelization as in the case of the Voltron processor [82]. Voltron uses multiple homogeneous cores that can be adapted for single and multi-threaded applications. The cores can operate in coupled mode when they act as a VLIW processor that will help exploit the hybrid forms of parallelism found in the code. Voltron relies on a very complex compiler that is able to exploit parallelism from the serial code, partition the code into small threads, schedule the instruction to the cores and direct the communication among between the cores. In coupled mode, the cores pass values among themselves on a specialized bus. The main challenge here is the difficulty of code parallelization.

TFlex processor [37] does not use physically shared resources among the cores and instead is dependent on a special distributed micro-architecture called Explicit Data Graph Execution (EDGE) with unique instruction-set architecture (ISA), which is configured to implement the composable lightweight processors. EDGE ISAs creates programs that are encoded as a sequence of blocks that have atomic execution semantics and these blocks are executed by different cores. The non-traditional ISA is the biggest downside of this architecture.

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

4. Functional Heterogeneous Multi-core

A large class of heterogeneous multi-cores comprise of cores with different functionality. This is fairly common in the embedded space where a multiprocessor system-on-chip (MPSoC) consists of general-purpose CPU cores, GPU cores, DSP blocks, and various hardware accelerators or IP blocks (e.g., video encoder/decoder, imaging, modem, communications such as WiFi, Bluetooth). The heterogeneity is introduced here to meet the performance demand under stringent power budget. Embedded GPUs are ubiquitous today in mobile application processors to enable not only 3D gaming but also general-purpose computing on GPU for data-parallel (DLP) compute-intensive tasks such as voice recognition, speech processing, image processing, gesture recognition, and so on. Still programmable CPU and GPU cores are not sufficient to accommodate certain demanding compute-intensive tasks at tight power budget. Hence it is necessary to include a large number of fixed-functionality hardware accelerators in the MPSoC. Finally, the need to strike a balance between flexibility and efficiency is driving the inclusion of reconfigurable computing fabric in heterogeneous MPSoC as well. In this section, we will cover GPUs, accelerators, and reconfigurable computing elements of heterogeneous multi-core systems.

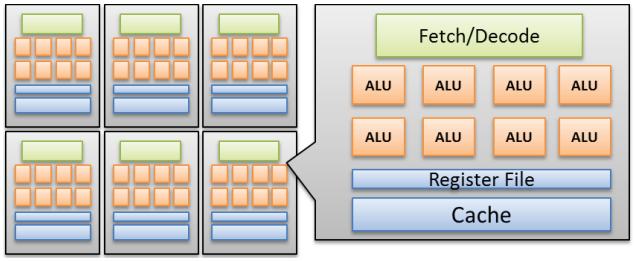

Fig. 5 An abstraction of GPU architecture.

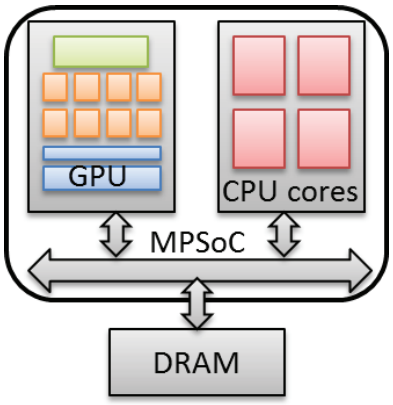

Fig. 6 Embedded GPU integrated with CPU cores on die.

4.1 Graphics Processing Unit (GPU)

A single CPU core is optimized to exploit instruction-level parallelism, while multi-cores are designed to take advantage of coarse-grained thread-level parallelism. In recent years, we have witnessed an avalanche of applications with massive amount of parallelism, for example, graphics, media processing, and signal processing applications. The parallelism in this class of applications is known as data-level parallelism where the same computation is performed on hundreds or even thousands of data elements. Massive data parallelism can only be exploited fully by deploying thousands of very simple cores on chip. This is the idea behind Graphics Processing Units (GPU). A GPU consists of thousands of cores that execute in parallel. But the biggest advantage of GPU is its power efficiency. As all the cores perform the same computation, a group of cores can share a single front end for instruction fetch and decode as shown in **Fig. 5**. As instruction fetch/decode contributes to significant power consumption in a processing core [27], this elimination of the frond-end processing leads to extremely power-efficient design of the cores that need only perform the computation. While GPUs have been originally designed for graphics and gaming, the suitability of GPUs for general-purpose data parallel applications (such as scientific computing) was quickly observed. General-purpose computing on GPUs was facilitated through CUDA programming standard [16] introduced by nVIDIA for their graphics processors. As nVIDIA graphics processors are ubiquitous in discrete GPU space, where the GPU appear as a separate device, CUDA became a de-facto standard for general-purpose programming on GPUs.

Discrete GPUs, while powerful, suffer from high communication cost with the CPU. The GPU device memory is distinct from CPU and it requires DMA to transfer data between the two. Recently, GPUs are being integrated with CPUs on the same die for MPSoCs. The CPU and GPU can then share a unified memory address space making the different computing elements compatible with each other from programmer's perspective and eliminating the costly memory copy operations from one address space to another. In the desktop environment, AMD accelerated processing unit (APU) [6] combines multi-core CPU with full-featured GPU as an SoC; similarly, Intel Ivy Bridge [18] integrated multi-core CPU with power-performance optimized graphics processing units. Unlike the discrete GPU space that is dominated by nVIDIA, the mobile GPU landscape is quite fragmented with many possible candidates. Apart from Radeon GPU in AMD APU and IRIS in Intel SoC, the smartphone and tablet platforms have recently featured many programmable embedded GPUs [67] including Imagination PowerVR, ARM Mali, Qualcomm Adreno, Vivante ScalarMorphic, nVIDIA Tegra, and others.

Embedded GPUs are quite different from desktop GPUs along multiple dimensions. First, as mentioned before, CPU and GPU share a unified memory address space. A system bus connects CPU, GPU, and the memory controller together as shown in **Fig. 6**. The CPU and GPU need to share the memory bandwidth and thus sharing the memory bandwidth effectively among the processing elements remain a key challenge. At the same time, shared and coherent memory across CPU and GPU enable low latency data transfer. Secondly, power rather than performance is the primary consideration in designing an embedded GPU. For example, while running an FFT application, embedded GeForce ULP GPU in Tegra consumes only 4 Watt, while high-end GeForce 8800 GPU consumes about 480 Watt [8]. Third, chip power budget being the critical resource in the dark silicon era, it is important to allocate the power budget appropriately and dynamically between the CPU and GPU. For example, **Fig. 7**

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

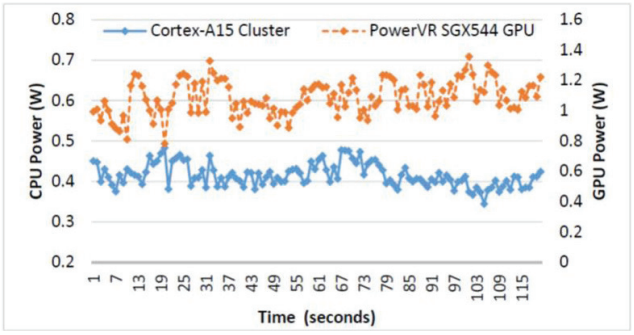

Fig. 7 CPU-GPU power behavior for mobile 3D games.

shows the power behavior of the CPU (Cortex-A15 cluster only) and the GPU on the Exynos 5 Octa MPSoC running a popular Android game "Asphalt 7: Heat" over 2-minute lifetime. Clearly, both the CPU and the GPU contribute equally to the power consumption. AMD APU [6] and Intel Ivy Bridge [18] take care of this power allocation in firmware. ARM introduced operating system level control-theoretic framework to intelligently allocate the power budget among the processing cores [3]. Quality-of-Service (QoS) aware CPU-GPU power management for mobile gaming applications has been explored in Refs. [55], [56]. Finally, given the fragmented nature of the embedded GPU market, it is imperative to use a common and open programming standard for both graphics and general-purpose computing on GPUs. OpenGL [78] for graphics and OpenCL [69] for general-purpose computing are two such standards that are portable across different mobile GPU platforms. OpenCL for mobile GPUs is still in its infancy making widespread appearance only in 2014. Programming, power/thermal management and memory bandwidth allocation all remain open problems for integrated CPU-GPU SoC devices.

4.2 Special-purpose Accelerators

General-purpose homogeneous and heterogeneous multi-cores as discussed in the previous section are far more cost-effective compared to an application-specific integrated circuit (ASIC) accelerator custom designed for a specific functionality (e.g., video encoding). The software programmability of general-purpose processors enables the same architecture to be reused across a large class of applications, thereby effectively amortizing the enormous non-recurring engineering (NRE) cost. However, general-purpose processors also greatly lack the performance and energy-efficiency of ASICs. For example, 3G mobile phone receiver requires 35–40 giga operations per second (GOPS) at roughly 1W budget, which is impossible to achieve without custom designed ASIC accelerator [17]. Thus current system-on-chips (SoCs) include a number of special-purpose accelerators. Shao et al. [66] analyzed die photos from three generations of Apple's SoCs: A6 (iPhone 5), A7 (iPhone 5S) and A8 (iPhone 6) to show that consistently more than half of the die area is dedicated to application-specific hardware accelerators and estimate the presence of around 29 accelerators in A8 SoC. The ITRS roadmap predicts hundreds to thousands of customized accelerators by 2022 [13].

Hameed et al. [27] attempt to understand the sources inefficiencies in general-purpose cores through a case-study of the H.264 video encoding application. They observe that ASIC implementation is 500x more energy efficient compared to a four-core homogeneous multi-core. The difference comes from the overhead to support programmability. The basic operations performed for most applications are extremely low energy. But processing an instruction in a general-purpose core involves significant additional overheads — instruction fetch, register fetch, data fetch, control, and pipeline registers — that dominate the overall power. These overheads are completely eliminated in ASIC accelerators at the price of zero flexibility and exceedingly high development cost.

4.2.1 Design Methodologies

There are two possible directions to achieve ASIC-like performance and energy-efficiency with processor-like application development cost. The first option it to devise design methodologies that will simplify the creation of customized ASIC accelerators. A powerful technique to accomplish this goal is high-level synthesis (HLS) [45] that allows the designer to specify an abstract high-level behavioral specification of the accelerator for automatically generating a register-transfer level (RTL) representation that realizes the abstract specification and is ready to be synthesized in hardware. After almost two decades of research, HLS tools have finally reached the level of maturity where they can potentially be adopted by industry in the design flow. The interested reader can refer to Ref. [46] for an extensive overview of state-of-the-art HLS tools.

The conservation cores approach [74] focuses only on energy efficiency rather than performance and targets automated hardware synthesis of irregular code with little parallelism and/or poor memory behavior. GreenDroid mobile application processor [25] uses hundred or so automatically generated, highly specialized, energy-efficient conservation cores to dramatically improve the energy-efficiency of general-purpose mobile applications.

While HLS tools enable automated design of individual accelerators, future accelerator-rich architectures will call for comprehensive design-space exploration techniques to evaluate the potential benefits and trade-offs of putting together general-purpose cores and hundreds of accelerators together. Aladdin [65] is a pre-RTL power-performance accelerator simulator offering such quick and effective design space exploration option. For mobile platforms, GemDroid [9] is a comprehensive simulation infrastructure for architects to conduct holistic evaluation of SoCs comprising of cores, accelerators, and system software.

4.2.2 Processor Customization

A second promising solution to achieve ASIC-like power-performance behavior without completely sacrificing programmability is through processor customization [31], [32]. A baseline processor core, is configured and extended to match the application. Typically, different components of the micro-architecture, such as register file, cache, functional units etc., are configurable according to the needs of the applications. More interestingly, the instruction-set of the processor can be extended with application-specific custom instructions. These custom instructions capture frequently executed computational patterns within an application or application domain. These patterns are then synthesized as hardwired custom functional

Information and Media Technologies 10(3): 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

units and added to the data path of the processor to substantially accelerate the execution of the application. Finally, the application-development tool chain (e.g., the compiler) needs to be updated to take advantage of these new custom instructions. The custom instructions require incremental modification and not complete re-design of the processor to be included. Thus once the configuration parameters and the extensions have been decided, an application-specific instruction-set processor (ASIP) can be synthesized within hours along with the updated software development tools through an automated process. Besides, as the ASIP remains software programmable, it can be reused across different applications — albeit with potentially reduced performance. Examples of commercial customizable processors include Tensilica Xtensa core [43] and Stretch software configurable processor [4]

The primary challenge in processor customization is to automate the selection of the appropriate custom instructions and the configuration parameters for an application [31]. Earlier research focused on identifying small but repetitive computational patterns [5], [12], [15], [81] that can easily fit as additional functional units in the processor datapath. These patterns have limited number of source and destination operands (just like basic processor instructions) and do not include memory accesses. Recent research reveals that bridging the gap between ASIC and general-purpose core can only be realized through aggressive customization where a single custom instruction can cover hundreds of simple operations through wide SIMD datapath and custom storage [27]. QsCORE [75] offers such broad-scale specialization capability by seeking to offload complete functions, typically containing 1000 s of instructions, onto Quasi-specific (Qs) coprocessor cores. Unlike ASICs, a single QsCORE can support multiple similar computations. An automated toolchain synthesizes QsCORES by leveraging similar code patterns within/across applications. The toolchain also performs extensive design space exploration to consider the tradeoff between the computational power of individual QsCORES and the area requirement, while maximizing energy savings.

4.2.3 Approximate Accelerators

The conventional approaches to hardware acceleration strive to maintain the accuracy of the computation and provide the exact solution. An interesting aspect of the emerging media and artificial intelligence applications, which dominate the consumer electronics domain, is that the algorithms involved either produce lossy results to save space and performance (such as lossy image, video, audio encoding applications) or inexact solutions simply because it is infeasible to compute the exact solution (as in data mining, machine learning, etc.) This approximate nature of the computation can be exploited to build accelerators that can trade accuracy of computation for gains in both performance and energy. The neural acceleration approach [22] proposes a low-power reconfigurable accelerator called a Neural Processing Unit (NPU). The NPU can perform the computation for a range of neural network topologies. An algorithmic transformation converts each approximable region of code to a neural model, which is processed by the NPU to generate results with acceptable accuracy but at substantially reduced energy cost.

4.3 Reconfigurable Computing

Traditional fixed-function ASIC-based accelerators discussed in the previous section offer high efficiency but limited or zero flexibility. At the other end of the spectrum, general-purpose processors provide full flexibility through software programmability but orders of magnitude difference in performance and energy compared to ASICs. Reconfigurable computing [14] fills this gap between hardware and software with far superior performance potential compared to programmable cores while maintaining higher-level of flexibility than ASICs. We will briefly describe two broad classes of reconfigurable computing fabric that have been introduced as part of heterogeneous multiprocessor system-on-chip devices alongside general purpose cores, GPUs, and fixed-function accelerators.

4.3.1 Field-Programmable Gate Arrays (FPGAs)

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated semiconductor devices that can be electrically reprogrammed to create almost any digital circuit/system within seconds [42]. FPGAs contain an array of computation elements, called configurable logic blocks, whose functionality can be changed instantly through multiple programmable configuration bits. These logic blocks, in turn, are connected through a set of programmable routing resources. A digital circuit can be fabricated on FPGAs by appropriately setting the configuration bits of the logic blocks for the appropriate functionality and connecting these blocks together through reconfigurable routing. This comes at the cost of area, power, and delay: an FPGA requires approximately 20 to 35 times more area than ASIC, has roughly 3–4 times slower performance than ASIC and consumes about 10 times as much dynamic power [41]. Still, FPGAs strike a compelling compromise between ASICs and general-purpose processors.

Recently, there has been considerable interest to bring general-purpose cores and FPGAs together on a single SoC enabling close coupling (high bandwidth) between the two. The Zynq platform from Xilinx [62] and SoC-FPGA platform from Altera [1] are examples of such heterogeneous platforms in the current embedded market. Such platforms typically integrate an application processor such as dual-core Cortex A9 from ARM, with a highly efficient reconfigurable FPGA fabric. These heterogeneous platforms are becoming more and more complex and will integrate more and more processors and logic elements. The new Xilinx UltraScale+ MPSoC [80] contains 4 ARM-Cortex-A53 cores running at up to 1.3 GHz, a Mali embedded GPU, up to 1 M logic elements and 3 K DSPs (see **Fig. 8**). The processors offer the opportunity to implement the applications with frequently changing specifications or standards, while the FPGA fabric allows to accelerate critical components of the system for real-time responsiveness.

A complex SoC platform such as Xilinx UltraScale+ MPSoC presents daunting challenges from programming point of view. Mapping compute-intensive kernels onto the reconfigurable fabric itself is a daunting proposition. Fortunately, there currently exist “C-to-gate” tools such as Xilinx Vivado synthesis tool [23] that greatly relieves the burden of the programmers. The bigger obstacle to the acceptance of such heterogeneous platforms is the divergent programming strategy for each component: CPU, GPU, and

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

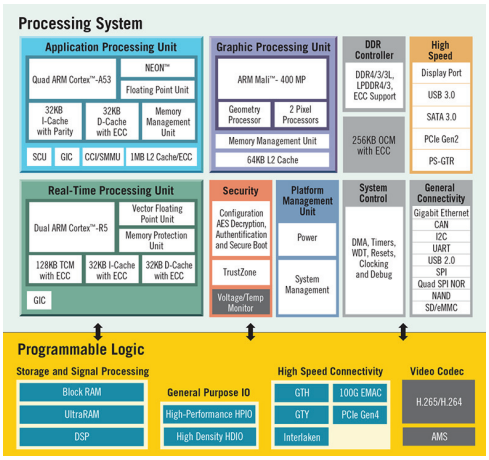

Fig. 8 Xilinx UltraScale+ MPSoC [source: Xilinx].

FPGAs. The emergence of open parallel programming standards for heterogeneous computing systems such as OpenCL [69] is an excellent development in the right direction. OpenCL programs are portable across CPU, GPU, and FPGAs. There have been some advances in compilation and runtime support for OpenCL to different kinds of computing cores. But this remains a rich area of research with the ultimate goal being transparent partitioning and mapping of a complete application on the MPSoC utilizing all its resources starting from a single high-level specification (such as one in OpenCL). The relative merits of GPU, FPGAs, and special-function accelerators (custom logic) have been investigated in Ref. [11]. The study concludes that GPUs and FPGAs with their programmability are competitive with respect to custom logic when the available parallelism is relatively high. When off-chip bandwidth is the major limitation, flexible cores such as GPUs and FPGAs can keep up with custom logic in terms of performance; but custom logic still maintains significant power advantage.

4.3.2 Coarse-Grained Reconfigurable Arrays (CGRAs)

Coarse-Grained Reconfigurable Arrays (CGRAs) [10] are promising alternative between ASICs and FPGAs. As mentioned before, FPGAs provide high flexibility, but may suffer from low efficiency compared to ASICs [41]. This is due to the fine bit-level granularity of reconfiguration opportunities offered by FPGAs that results in lower performance, higher energy consumption, and longer reconfiguration penalty. In contrast, CGRAs, as the name suggests, comprise of coarse-grained functional units (FUs) connected via typically a mesh-like interconnect as shown in **Fig. 9**. The functional units are capable of performing arithmetic/logic operations and can be reconfigured on a per cycle basis by writing to a control (context) register associated with each functional unit. The functional units can exchange data among themselves through the interconnect. As many functional units work in parallel, CGRAs can easily accelerate compute-intensive loop executions by exploiting instruction-level parallelism. The primary challenge lies with the compiler that needs to map and schedule the instructions on the FUs as well as take care of the routing of data among the FUs through the interconnect [7], [28], [48], [54]. A representative example of CGRA architecture is ADRES [47] that tightly couples a VLIW (very-long instruction word) processor with coarse-grained reconfigurable matrix. A variation of ADRES architecture has been introduced commercially as Samsung Reconfigurable Processor (SRP) [70] as part of the mobile application processor system-on-chip. The SRP consists of sixteen FUs, one or two register files, four load/store units, scratch pad memory (SPM), an instruction cache for VLIW mode and configuration memory for CGRA. High-performance and energy-efficiency of the SRP has been demonstrated for multimedia applications, 3D graphics, and software-defined radio.

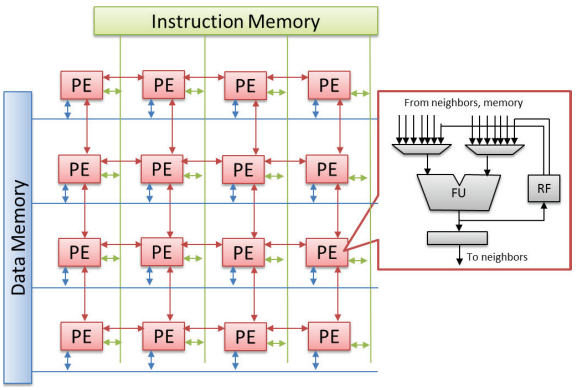

Fig. 9 CGRA Architecture.

5. Conclusions

The technology trends has ushered in the dark silicon era where power rather than the transistor count is the limiting factor and thus only a fraction of the chip can be powered on at any point in time. Heterogeneous computing has emerged in response to this development where area can be traded for energy-efficiency. A heterogeneous multi-core integrates a diverse mix of processing cores including general-purpose programmable cores, graphics cores, special-purpose accelerators in custom logic, and even reconfigurable fabric on a single chip. The cores most suited for an application need to be switched on in heterogeneous computing platforms leading to low-power, high-performance computing. In this article, we detailed the technology trends driving heterogeneous multi-cores, provided an overview of functional heterogeneous and performance heterogeneous systems, as well as discussed the obstacles to fully realize the potential of heterogeneity. Research in heterogeneous multi-core is currently at a nascent stage. But heterogeneity offers exciting opportunities and challenges for all layers of the computing stack starting from devices, circuits, and architecture to operating system, compiler, and programming layers.

Acknowledgments This work was partially supported by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115, Huawei International Pte. Ltd. research grant, and CSR research funding.

References

[1] Altera: User customizable ARM-based SoC (2014).
[2] Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities, *Proc. April 18-20, 1967, Spring Joint Computer Conference*, pp.483–485, ACM (1967).
[3] ARM: ARM intelligent power allocation (2014).
[4] Arnold, J.M.: S5: The architecture and development flow of a software

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

configurable processor, *Proc. 2005 IEEE International Conference on Field-Programmable Technology, 2005*, pp.121–128. IEEE (2005).

[5] Atasu, K., Pozzi, L. and Ienne, P.: Automatic application-specific instruction-set extensions under microarchitectural constraints, *International Journal of Parallel Programming*, Vol.31, No.6, pp.411–428 (2003).

[6] Bouvier, D., Cohen, B., Fry, W., Godey, S. and Mantor, M.: AMD APU SoC: "Kabini", *IEEE Micro*, p.1 (2014).

[7] Chen, L. and Mitra, T.: Graph minor approach for application mapping on CGRAs, *ACM Trans. Reconfigurable Technology and Systems* (*TRETS*), Vol.7, No.3, p.21 (2014).

[8] Cheng, K.-T. and Wang, Y.-C.: Using mobile GPU for general-purpose computing – A case study of face recognition on smartphones, *2011 International Symposium on VLSI Design, Automation and Test* (*VLSI-DAT*), pp.1–4, IEEE (2011).

[9] Nachiappan, N.C., Yedlapalli, P., Soundararajan, N., Kandemir, M.T., Sivasubramaniam, A. and Das, C.R.: GemDroid: A framework to evaluate mobile platforms, *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, pp.355–366, ACM (2014).

[10] Choi, K.: Coarse-grained reconfigurable array: Architecture and application mapping, *IPSJ Trans. System LSI Design Methodology*, Vol.4, pp.31–46 (2011).

[11] Chung, E.S., Milder, P.A., Hoe, J.C. and Mai, K.: Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?, *Proc. 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp.225–236, IEEE Computer Society (2010).

[12] Clark, N., Zhong, H. and Mahlke, S.: Processor acceleration through automated instruction set customization, *Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture*, p.129, IEEE Computer Society (2003).

[13] International Roadmap Committee, et al.: International technology roadmap for semiconductors, 2011 edition, *Semiconductor Industry Association*, available from ⟨http://www.itrs.net/Links/2011ITRS/2011Chapters/2011ExecSum.pdf⟩ (2011).

[14] Compton, K. and Hauck, S.: Reconfigurable computing: A survey of systems and software, *ACM Computing Surveys* (*csuR*), Vol.34, No.2, pp.171–210 (2002).

[15] Cong, J., Fan, Y., Han, G. and Zhang, Z.: Application-specific instruction generation for configurable processor architectures, *Proc. 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, pp.183–189, ACM (2004).

[16] Cook, S.: *CUDA programming: A developer's guide to parallel computing with GPUs*, Newnes (2012).

[17] Dally, W.J., Balfour, J.D., Black-Schaffer, D., Chen, J., Harting, R.C., Parikh, V., Park, J. and Sheffield, D.: Efficient embedded computing, *IEEE Computer*, Vol.41, No.7, pp.27–32 (2008).

[18] Damaraju, S., George, V., Jahagirdar, S., Khondker, T., Milstrey, R., Sarkar, S., Siers, S., Stolero, I. and Subbiah, A.: A 22nm IA multi-CPU and GPU system-on-chip, *2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers* (*ISSCC*), pp.56–57, IEEE (2012).

[19] Dennard, R.H., Gaensslen, F.H., Rideout, V.L., Bassous, E. and LeBlanc, A.R.: Design of Ion-implanted MOSFET's with very small physical dimensions, *IEEE Journal of Solid-State Circuits*, Vol.9, No.5, pp.256–268 (1974).

[20] Dreslinski, R.G., Wieckowski, M., Blaauw, D., Sylvester, D. and Mudge, T.: Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits, *Proc. IEEE*, Vol.98, No.2, pp.253–266 (2010).

[21] Esmaeilzadeh, H., Blem, E., St Amant, R., Sankaralingam, K. and Burger, D.: Dark silicon and the end of multicore scaling, *2011 38th Annual International Symposium on Computer Architecture* (*ISCA*), pp.365–376, IEEE (2011).

[22] Esmaeilzadeh, H., Sampson, A., Ceze, L. and Burger, D.: Neural acceleration for general-purpose approximate programs, *Proc. 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.449–460, IEEE Computer Society (2012).

[23] Tom Feist: Vivado design suite, *White Paper* (2012).

[24] Geer, D.: Chip makers turn to multicore processors, *Computer*, Vol.38, No.5, pp.11–13 (2005).

[25] Goulding-Hotta, N., Sampson, J., Swanson, S., Taylor, M.B., Venkatesh, G., Garcia, S., Auricchio, J., Huang, P.-C., Arora, M. and Nath, S., et al.: The GreenDroid mobile application processor: An architecture for silicon's dark future, *IEEE Micro*, Vol.31, No.2, pp.86–95 (2011).

[26] Greenhalgh, P.: Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7, *ARM White paper* (2011).

[27] Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B.C., Richardson, S., Kozyrakis, C. and Horowitz, M.: Understanding sources of ineffciency in general-purpose chips, *Comm. ACM*, Vol.54, No.10, pp.85–93 (2011).

[28] Hamzeh, M., Shrivastava, A. and Vrudhula, S.: EPIMap: Using epimorphism to map applications on CGRAs, *Proc. 49th Annual Design Automation Conference*, pp.1284–1291, ACM (2012).

[29] Hennessy, J.L. and Patterson, D.A.: *Computer architecture: A quantitative approach*, Elsevier (2012).

[30] Hill, M.D. and Marty, M.R.: Amdahl's law in the multicore era, *Computer*, Vol.7, pp.33–38 (2008).

[31] Ienne, P. and Leupers, R.: *Customizable embedded processors: Design technologies and applications*, Academic Press (2006).

[32] Imai, M., Takeuchi, Y., Sakanushi, K. and Ishiura, N.: Advantage and possibility of application-domain specific instruction-set processor (ASIP), *Information and Media Technologies*, Vol.5, No.4, pp.1064–1081 (2010).

[33] Ineda Systems: Hierarchical Computing (2014).

[34] Ionescu, A.M., De Michielis, L., Dagtekin, N., Salvatore, G., Cao, J., Rusu, A. and Bartsch, S.: Ultra low power: Emerging devices and their benefits for integrated circuits, *2011 IEEE International Electron Devices Meeting* (*IEDM*), pp.16.1.1–16.1.4, IEEE (2011).

[35] Ipek, E., Kirman, M., Kirman, N. and Martinez, J.F.: Core fusion: Accommodating software diversity in chip multiprocessors, *ACM SIGARCH Computer Architecture News*, Vol.35, pp.186–197, ACM (2007).

[36] Khubaib, K., Suleman, M.A., Hashemi, M., Wilkerson, C. and Patt, Y.N.: Morphcore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP, *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (*MICRO*), pp.305–316, IEEE (2012).

[37] Kim, C., Sethumadhavan, S., Govindan, M.S., Ranganathan, N., Gulati, D., Burger, D. and Keckler, S.W.: Composable lightweight processors, *40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007. MICRO 2007*, pp.381–394, IEEE (2007).

[38] Koufaty, D., Reddy, D. and Hahn, S.: Bias scheduling in heterogeneous multi-core architectures, *Proc. 5th European Conference on Computer Systems*, pp.125–138, ACM (2010).

[39] Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P. and Tullsen, D.M.: Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction, *Proc. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*, pp.81–92, IEEE (2003).

[40] Kumar, R., Jouppi, N.P. and Tullsen, D.M.: Conjoined-core chip multiprocessing, *Proc. 37th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.195–206, IEEE Computer Society (2004).

[41] Kuon, I. and Rose, J.: Measuring the gap between FPGAs and ASICs, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.26, No.2, pp.203–215 (2007).

[42] Kuon, I., Tessier, R. and Rose, J.: FPGA architecture: Survey and challenges, *Foundations and Trends in Electronic Design Automation*, Vol.2, No.2, pp.135–253 (2008).

[43] Leibson, S.: *Designing SOCs with Configured Cores: Unleashing the Tensilica Xtensa and Diamond Cores*, Academic Press (2006).

[44] Lukefahr, A., Padmanabha, S., Das, R., Sleiman, F.M., Dreslinski, R., Wenisch, T.F. and Mahlke, S.: Composite cores: Pushing heterogeneity into a core, *Proc. 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.317–328, IEEE Computer Society (2012).

[45] McFarland, M.C., Parker, A.C. and Camposano, R.: The high-level synthesis of digital systems, *Proc. IEEE*, Vol.78, No.2, pp.301–318 (1990).

[46] Meeus, W., Van Beeck, K., Goedemé, T., Meel, J. and Stroobandt, D.: An overview of today's high-level synthesis tools, *Design Automation for Embedded Systems*, Vol.16, No.3, pp.31–51 (2012).

[47] Mei, B., Vernalde, S., Verkest, D., De Man, H. and Lauwereins, R.: Adres: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix, *Field Programmable Logic and Application*, pp.61–70, Springer (2003).

[48] Mei, B., Vernalde, S., Verkest, D., De Man, H. and Lauwereins, R.: Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling, *IEE Proc. Computers and Digital Techniques*, Vol.150, pp.255–61, IET (2003).

[49] Merritt, R.: ARM CTO: Power surge could create dark silicon, *EE Times* (2009).

[50] Moore, G.E., et al.: Cramming more components onto Integrated Circuits (1965).

[51] Muthukaruppan, T.S., Pricopi, M., Venkataramani, V., Mitra, T. and Vishin, S.: Hierarchical power management for asymmetric multi-core in dark silicon era, *Proc. 50th Annual Design Automation Conference*, p.174, ACM (2013).

[52] nVidia: Variable SMP — A Multi-Core CPU Architecture for Low Power and High Performance (2011).

Information and Media Technologies 10(3) : 383–394 (2015)
reprinted from: IPSJ Transactions on System LSI Design Methodology 8: 51–62 (2015)

[53] Palacharla, S., Jouppi, N.P. and Smith, J.E.: *Complexity-effective superscalar processors*, Vol.25, ACM (1997).
[54] Park, H., Fan, K., Mahlke, S.A., Oh, T., Kim, H. and Kim, H.-S.: Edge-centric modulo scheduling for coarse-grained reconfigurable architectures, *Proc. 17th International Conference on Parallel Architectures and Compilation Techniques*, pp.166–176, ACM (2008).
[55] Pathania, A., Irimiea, A.E., Prakash, A. and Mitra, T.: Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs, *2015 52nd ACM/EDAC/IEEE Design Automation Conference* (*DAC*), IEEE (2015).
[56] Pathania, A., Jiao, Q., Prakash, A. and Mitra, T.: Integrated CPU-GPU power management for 3D mobile games, *2014 51st ACM/EDAC/IEEE Design Automation Conference* (*DAC*), pp.1–6, IEEE (2014).
[57] Patterson, D.A.: Future of computer architecture, *Berkeley EECS Annual Research Symposium* (*BEARS*), College of Engineering, UC Berkeley, US (2006).
[58] Pricopi, M. and Mitra, T.: Bahurupi: A polymorphic heterogeneous multi-core architecture, *ACM Trans. Architecture and Code Optimization* (*TACO*), Vol.8, No.4, p.22 (2012).
[59] Pricopi, M. and Mitra, T.: Task scheduling on adaptive multi-core, *IEEE Trans. Comput.*, Vol.63, No.10, pp.2590–2603 (2014).
[60] Pricopi, M., Muthukaruppan, T.S., Venkataramani, V., Mitra, T. and Vishin, S.: Power-performance modeling on asymmetric multi-cores, *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (*CASES*), pp.1–10, IEEE (2013).
[61] Raghavan, A., Luo, Y., Chandawalla, A., Papaefthymiou, M., Pipe, K.P., Wenisch, T.F. and Martin, M.M.K.: Computational sprinting, *2012 IEEE 18th International Symposium on High Performance Computer Architecture* (*HPCA*), pp.1–12, IEEE (2012).
[62] Rajagopalan, V., Boppana, V., Dutta, S., Taylor, B. and Wittig, R.: Xilinx Zynq-7000 EPP–An Extensible Processing Platform Family, *23rd Hot Chips Symposium*, pp.1352–1357 (2011).
[63] Rotem, E., Naveh, A., Ananthakrishnan, A., Rajwan, D. and Weissmann, E.: Power-management architecture of the Intel microarchitecture code-named Sandy Bridge, *IEEE Micro*, Vol.2, pp.20–27 (2012).
[64] Shafique, M., Garg, S., Mitra, T., Parameswaran, S. and Henkel, J.: Dark silicon as a challenge for hardware/software co-design: Invited special session paper, *Proc. 2014 International Conference on Hardware/Software Codesign and System Synthesis*, p.13, ACM (2014).
[65] Shao, Y.S., Reagen, B., Wei, G.-Y. and Brooks, D.: Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures, *2014 ACM/IEEE 41st International Symposium on Computer Architecture* (*ISCA*), pp.97–108, IEEE (2014).
[66] Shao, Y.S., Xi, S., Srinivasan, V., Wei, G.-Y. and Brooks, D.: Toward Cache-Friendly Hardware Accelerators, *Proc. Sensors to Cloud Architectures Workshop* (*SCAW*), *in conjuction with HPCA 2015* (2015).
[67] Shebanow, M.: An evolution of mobile graphics, *Keynote talk at High Performance Graphics* (2013).
[68] Muthukaruppan, T.S., Pathania, A. and Mitra, T.: Price theory based power management for heterogeneous multi-cores, *Proc. 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.161–176, ACM (2014).
[69] Stone, J.E., Gohara, D. and Shi, G.: OpenCL: A parallel programming standard for heterogeneous computing systems, *Computing in Science & Engineering*, Vol.12, No.1-3, pp.66–73 (2010).
[70] Suh, D., Kwon, K., Kim, S., Ryu, S. and Kim, J.: Design space exploration and implementation of a high performance and low area Coarse Grained Reconfigurable Processor, *2012 International Conference on Field-Programmable Technology* (*FPT*), pp.67–70, IEEE (2012).
[71] Tarjan, D., Boyer, M. and Skadron, K.: Federation: Repurposing scalar cores for out-of-order instruction issue, *Proc. 45th Annual Design Automation Conference*, pp.772–775, ACM (2008).
[72] Taylor, M.B.: Is dark silicon useful?: Harnessing the four horsemen of the coming dark silicon apocalypse, *Proc. 49th Annual Design Automation Conference*, pp.1131–1136, ACM (2012).
[73] Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P. and Emer, J.: Scheduling heterogeneous multi-cores through performance impact estimation (pie), *ACM SIGARCH Computer Architecture News*, Vol.40, No.3, pp.213–224 (2012).
[74] Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S. and Taylor, M.B.: Conservation cores: Reducing the energy of mature computations, *ACM SIGARCH Computer Architecture News*, Vol.38, pp.205–218, ACM (2010).
[75] Venkatesh, G., Sampson, J., Goulding-Hotta, N., Venkata, S.K., Taylor, M.B. and Swanson, S.: QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores, *Proc. 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.163–174, ACM (2011).
[76] Wall, D.W.: *Limits of instruction-level parallelism*, Vol.19. ACM (1991).
[77] Winter, J.A., Albonesi, D.H. and Shoemaker, C.A.: Scalable thread scheduling and global power management for heterogeneous many-core architectures, *Proc. 19th International Conference on Parallel Architectures and Compilation Techniques*, pp.29–40, ACM (2010).
[78] Woo, M., Neider, J., Davis, T., Shreiner, D., et al.: OpenGL programming guide (1999).
[79] Wulf, W.A. and McKee, S.A.: Hitting the memory wall: Implications of the obvious, *ACM SIGARCH Computer Architecture News*, Vol.23, No.1, pp.20–24 (1995).
[80] Xilinx: Xilinx UltraScale MPSoC Architecture (2015).
[81] Yu, P. and Mitra, T.: Scalable custom instructions identification for instruction-set extensible processors, *Proc. 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp.69–78, ACM (2004).
[82] Zhong, H., Lieberman, S.A. and Mahlke, S.A.: Extending multicore architectures to exploit hybrid parallelism in single-thread applications, *IEEE 13th International Symposium on High Performance Computer Architecture, 2007.* (*HPCA 2007*), pp.25–36, IEEE (2007).

Tulika Mitra is a Professor of Computer Science at School of Computing, National University of Singapore (NUS). She received her Ph.D. in Computer Science from the State University of New York at Stony Brook in 2000. Her research interests span various aspects of the design automation of embedded real-time systems with particular emphasis on application-specific processors, software timing analysis/optimizations, heterogeneous multi-cores, and energy-aware computing.

(Invited by Editor-in-Chief: *Hiroyuki Tomiyama*)