High performance heterogeneous embedded computing: a review

HE Yongfu, WANG Shaojun, PENG Yu

(*Department of Automatic Test and Control, Harbin Institute of Technology, Harbin* 150080, *China*)

Abstract: As increasingly widening gap of computing demand and performance in embedded computing domain, heterogeneous computing architecture which delivers better performance as well as lower power in limited size is gaining more and more attention. At first, the heterogeneous computing model is presented. And the different tightly coupled single chip heterogeneous architectures and their application domain are introduced. Then, task partitioning methods are described. Several programming model technology are analyzed and discussed. Finally, main challenges and future perspective of High Performance Embedded Computing (HPEC) are summarized.

Key words: HPEC; heterogeneous SoCs; hardware/software partition; heterogeneous programming

1 Introduction

Computing has already become a seamless and ubiquitous part of the world [1]. High performance computing (HPC) today does not only happen in traditional application fields like climate modeling, nuclear waste simulation, and warfare modeling which based on server farms or big supercomputers[2]. Embedded applications such as mobile computing[3], airborne electronics[4], medical imaging[5] and financial services[6] increasingly demand a huge amount of computing performance. In addition, computing challenges are even more impressive when those high performance applications have to be achieved by a power sensitive and size limited device rather by traditional big and high power consumption servers, or supercomputers. For instance, Media devices like a 3G mobile phone requires 210 to 290 Giga operations per second (GOPS) of performance to handle a 100-Mbps channel in a small handheld package with a maximum power dissipation of 1W[7]. The required processing performance has already surpassed the once exotic Cray-1 supercomputer can achieve[8]. For hyper-spectral remote sensing application in space, on board space processing systems require implementing fast computations of hyper-spectral image processing algorithms on recorded visible and near-infrared spectrum whose data volume is gigabytes per flight by low weight and low power integrated components[4].

High performance embedded computing (HPEC) provides a type of computing model which embedded in equipment or an appliance to perform specific compute and data intensive applications[9]. In contrast to traditional HPC based on servers, this type of high performance computing should be carefully designed to meet stringent requirements: high performance, flexibility, low cost, low power consumption and limited size[10].

Embedded single-core General Purpose Processors (GPPs) have long been used to provide processing capacity in HPEC. The exponentially transistor densities grown along with Moore's Law and associated impractical levels of power dissipation make single-core processor facing scaling limitation. In turn, parallelism is chosen as an effective approach to continue the proportional scaling of processor performance by implementing multicore on a single chip in the past years[11]. While the "multicore era" is already facing the phenomenon of "Dark Silicon"[12] that the maximally utilization of die area is curtailed because of the failure of "Dennard Scal-

ing"[13]. It means that more than 50% transistors of a fixed-size multi-core chip will not be powered on simultaneously at future 8 nm technologies as sharp increases in power consumption. Actually, HPEC applications are now demanding much more performance than conventional multi-core processors alone can deliver[14]. Considering the power and performance constraints, parallelism in processor is not the most efficient way to address rising technology gap between requirements and performance.

Domain specific processing such as Digital Signal Processors (DSPs) (especially multi-core DSPs), General Purpose Graphics Processing Unit (GPGPUs) and Field Programmable Logic Array (FPGAs) are proved to be much more powerful in algorithm acceleration for many computing intensive applications[15-16]. However, they all have their own overheads. DSPs lack the ability to manage conventional task. GPGPUs also require GPPs to handle data throughput. FPGA is not so efficient for serial parts of an algorithm.

Heterogeneity which converges general purpose processing and domain specific processing is considered as an alternative to continue on an exponential performance scaling trajectory. Combing GPPs with powerful application specific coprocessors (e. g. DSPs, FPGAs, GPGPUs), the heterogeneous system supports control and serial task on multi-core processors and computing demanding tasks by heterogeneous parallel processing on coprocessor. The coupling style of GPPs and coprocessors is critical for the performance improvement. For a loosely coupled heterogeneous system in which processor and coprocessor are interconnected through I/O in a multi-chip way, the scalability, flexibility, power, and bandwidth may limit the further improvement . With the System on Chip(SoC) technology, multi-core GPPs and coprocessors are tightly coupled into a single hardware through high bandwidth bus on chip. This new class of heterogeneous SoC computing model delivers lower power consumption, less area and higher performance[17]. Its computing performance outperforms traditional processors only architectures by an order of magnitude or more as well as energy efficient improvement for specific application[18]. Heterogeneous architectures based on different granularity of coprocessors provide different capabilities for high performance embedded computing. DSPs and GPGPUs based heterogeneous architectures are often saw in mobile embedded computing like cell phone and digital media applications. And FPGAs provide a reconfigurable, pipelined and parallel structure computing model that can keep pace with rising performance demands. So heterogeneous architectures based on FPGAs have much more comprehensive applications. For instance, on board space processing acceleration[4], data center[19] and cloud computing[17].

In HPEC, computing architectures are increasingly turning to heterogeneity as solutions for boosting performance in the face of power constraints. However, challenges of developing these heterogeneous architectures have grown with the continued trend. The biggest barrier lies in the absence of an intuitive heterogeneous programming model.

Two critic problems must be solved efficiently. First problem is task partitioning. In order to exploit heterogeneous system easily, boundaries of what to do in processor and what to do in coprocessor of an application should be defined explicitly[20]. For one appropriate partitioning of an algorithm, the nature of the algorithm, granularities of coprocessor, data transfer latency and bandwidth between the processor and coprocessor should be taken into account. The second problem is how to build compilers for development. High level compilers must be able to choose the most efficient processing core for the type of processing needed for a given application task in run time[21].

The rest of this paper is organized as follows: in Section 2, the embedded heterogeneous computing model is proposed followed with single chip heterogeneous computing model. Then different tightly coupled heterogeneous SoCs architectures and their

application domain are introduced. Section 3 firstly describes the methods for task partitioning. Then several programming model technology are analyzed and discussed. Finally, section 4 summaries the main challenges and future perspective.

2 Embedded heterogeneous computing architectures

An embedded heterogeneous computing system consists of processor (GPPs) and coprocessor (e.g. DSPs, GPGPUs, FPGAs) which are responsible for different computing tasks in one application. Processor acts as a master who is in charge of computing task schedule between processor and coprocessor. Also, processors can execute serial intrinsic parts of an algorithm. For those computing intensive and parallel intrinsic parts of an algorithm, they can be thrown into coprocessors for acceleration.

An important advantage of the heterogeneous approach is that algorithms can be executed effective on processing cores of distinctively different types that is best suited for them rather than force the entire application to a traditional GPP or to a coprocessor. In other words, it matches the computational model of the algorithm with the granularity and capabilities of the processing entity. In turn, applications can operate at minimum supply voltage and clock frequency. Hence heterogeneity provides energy efficiency and flexibility at the right granularity[22]. For example, PN-code generation runs efficiently on reconfigurable architectures like FPGA[23-24].

For the implementation of a heterogeneous computing architecture, there are two couplings models as shown in Fig.1.

Loosely coupled coprocessors access processor's memory through a bridge, adding several cycles to data accesses. The solution does suffer from communication latency problems. Because the data to be processed are stored in main memory, and must be moved to and from the coprocessor, so any acceleration realization must amortize this communication overhead. Besides, this multi-chip style's power consumption poses huge challenge for embedded application. Nowadays, with the advancement of SoC technology, GPGPU, DSPs and FPGA turn to another called tightly coupled approach with processors.

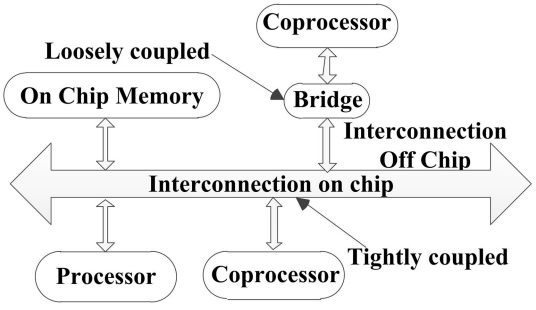


Fig. 1 Simplified overview of a heterogeneous computing architecture

Tightly coupled architecture means that processor's and coprocessors coexist on the interconnection on chip, having direct access to processor memory. This reduces the communication overhead, bringing coprocessor closer to the main processor and allows the coprocessor to take advantage of the memory. Overhead of inter-processor communication also contributes to the computation of an application. Therefore, in embedded application, high performance and tight coupling interconnect is needed to deliver low latency and high bandwidth to the GPP to balance computation and inter-processor communication. In addition, due to very high level integration of processors and coprocessors as well as much shorter wiring on the same chip, tightly coupled approach consumes considerably less power.

As a result, heterogeneous SoCs which embody a rich mix of tightly coupled GPP cores, coprocessors (e.g. DSPs, GPGPUs, and FPGAs) and a high performance interconnect is the most efficient architecture to enhance the GPP performance in embedded computing[25].

2.1 General purpose processors

GPPs work only as master in the heterogeneous computing architecture for embedded computing.

However, in the past, they were the main stream architecture of choice for a wide variety of embedded application.

GPPs have several registers, high clock rates, and complex control mechanisms. Before early 2000s, GPPs only consist of single-core CPU. Processing techniques like instruction pipelining, superscalar operation and caches are widely used to optimize instruction execution time and reduce the average latency in accessing instructions and data. Besides, single-core CPU's performance scaled with frequency in line with Moore´s Law. Unfortunately, with the scaling frequency, power dissipation escalates to impractical levels. Also, instruction parallelism suitable for superscalar issue is finite. In addition, it is difficult to design processor when pipelining past about 10-20 stages[26]. As a result, all of these problems have placed limits on single core performance scalability.

Computer architecture researchers turn to multicore architectures to meet high performance demands. Multicore processors avoid those problems by filling up a processor die with multiple, tightly on-chip coupling, relatively simpler processor cores instead of just one huge core[11]. By Symmetric multiprocessing (SMP) technology which spread multiple threads of execution across the small size cores, multicore processors can achieve higher performance with significantly lower power consumption at low clock frequency[27]. Those small size cores vary from very simple pipelines to moderately complex superscalar processors. General purpose embedded multicore products such as ARM's MPCores have already dominated the market[28]. However GPPs aimed to conventional domain applications which lead to less efficient for specific computing intensive task. And multicore processors still face the scale limitation. As a result, main stream processors today can stamp down only a limited number of processing cores on the same die to stay within the power and thermal limitation[12]. That is the reason GPPs should be augmented with specific computing elements to improve performance.

2.2 Heterogeneous socs based on dsps

Digital Signal Processors (DSPs) are typically weak at traditional general purpose applications while designed specifically to rapidly perform arithmetic multiply accumulate operation in one clock cycle. By contrast, most GPPs require multiple cycles to perform a multiplication. It makes DSPs suitable for signal processing because most signal processing algorithms can be map to standard vector and matrix operations. And multiply accumulates are common in these operations.

For Texas Instruments TMS320C64x family, the DSP core processor has two multipliers for a 32-bit result and six Arithmetic Logic Units (ALUs). The DSP core can produce four 16-bit Multiply-Accumulates (MACs) per cycle for up to 3600 million MACs per second (MMACS), or eight 8-bit MACs for up to 8800 MMACS[29].

To scale performance, high-end DSPs often adopt multiprocessor system designed with arrays of DSPs. For instance, TMS320C66x features with four 1.0 GHz or 1.2 GHz C66x Fixed/Floating-Point DSP Cores. Each core delivers 38.4 GMACS for Fixed Point or 19.2 GFlops for Floating Point at 1.2 GHz operating frequency [30]. However, DSPs are still processors in essence that have to face "Dark Silicon" phenomenon.

DSPs are widely used in mobile computing application in which GPP tightly coupled with DSPs serve as cell phone processor to perform baseband operations, including both communication and multimedia operations. The Texas Instruments (TI) OMAP architecture has several implementations. The OMAP 3525 has two CPUs: an ARM Cortex-A8 and a TMS320C64x digital signal processor (DSP)[31]. The ARM acts as a master, and the DSP acts as slave performs signal processing operations.

With the right mix of processing elements, TI's new scalable KeyStone multicore SoCs architecture provides the efficient blend of general processing and

specific DSP processing, as shown in Fig. 2. KeyStone consists of high performance four ARM Cortex™-A15 MPCore processors and eight latest TMS320C66x DSP cores. TeraNet is a multilevel interconnection which delivers over two terabits per second of concurrent data throughput of the on-chip data flow within the KeyStone multicore architecture. It enables every processing element operating near full capacity at all times. This enable KeyStone II SoCs to deliver very high performance for various computing demanding applications, such as medical imaging, mission critical radar, test and measurement[32].

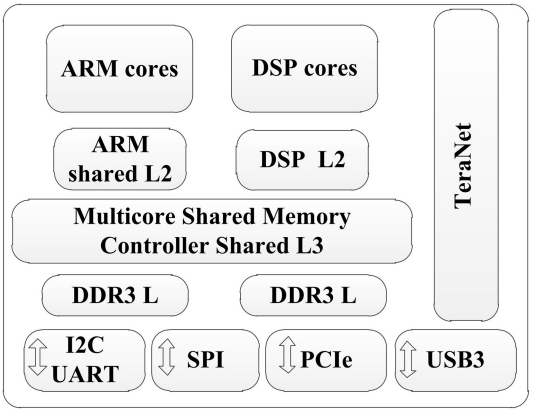


Fig. 2 Keystone multicore SoC architecture

2.3 Heterogeneous socs based on gpgpus

Unlike powerful single-thread performance and fast clock speed of GPPs, the processing element for Graphics Processing Unit (GPU) is relatively simple. It consists of a single fetch unit and eight scalar units. Each instruction is fetched and executed in parallel on all eight scalar units. Much more processing elements can be packed per die as the area of each is relatively small.

In embedded application, GPU is widely used in mobile media computing specifically for graphics applications that have a large degree of data parallelism. For General Purpose computing with GPU, the algorithm is broken up into thousands of threads, which are mapped to the available computational units. Each of these processing elements produces a vector wide of output data every clock cycle. GPGPUs tend to perform efficiently on algorithms where the ratio of computation task to I/O is very high. As the host processor must provide data over interconnection to the GPU. And I/O bandwidth is limited. So for those data without a high degree of calculations to be done, GPU will become data starved[33].

Mobile SoC providers are beginning to integrate GPUs and GPP into one chip for the field of portable high performance consumer electronics. Tightly coupled approach greatly reduces the CPU-to-GPU latency. One prime example is NVIDIA's Tegra 3, which combines an ARM Cortex-A9 with NVIDIA's ultra-low power GEForce GPU which supports 1080p video. By offloading much of the computation to the GPU, Tegra 3 can improve performance significantly[34]. Another device is the PICA200 which embedded GPU for the Digital Media application. The PICA200 can operate up to 40 million triangles per second or 800 million pixels per second, while consuming only 0.5 – 1.0 mW/MHz[35].

2.4 Heterogeneous socs based on fpgas

FPGAs enable designers to achieve performance similar to traditional Application Specific Integrated Circuit (ASIC) with lower design costs and quicker time-to-market. The FPGA architecture consists of a large number of programmable logic elements, SRAM memories and multiplier blocks, digital signal processing (DSP), serial transceivers, memory controllers, and advanced I/O functions on a VLSI chip.

FPGAs are relatively new to high-performance computing, but have compelling advantages. Firstly, FPGA architecture provides the flexibility to create a specific pipeline and parallel hardware with massive array of application specific custom logic that enable both instruction and data-level parallelism. As a result, the latency for processing a given data stream is much lower than on GPGPUs and CPUs. This can be critical for real time applications, such as financial

trading algorithms acceleration. Secondly, FPGAs have superior GFLOPS/W capability and this can be a key advantage in applications where matrix manipulations, filters, transforms, and DSP operations dominated and environment constrained, such as avionics. The peak performance of the Virtex-7 980XT FPGA provided by Xilinx is estimated around 987 GFLOPS for floating-point operations[36]. Altera's Stratix-series FPGAs offer over 1 TFLOPS of floating point DSP performance, which greatly exceeds the performance of any ARM-based processor and only outperformed by high-end GPUs[37]. And the frequency of FPGAs typically operates between 100 - 300 MHz consuming tens of watts. While multicores' CPUs execute between 2 - 3 GHz that tend to consume power in hundreds of watts. It means that for a given power budget, the FPGA can typically perform far more computations than a GPGPU and CPUs. Thirdly, reconfigurable ability of FPGAs offers the flexibility and adaptivity needed for future scalable applications.

Due to the highly parallel reconfigurable architecture and peak float performance coupled with lower power consumption compared to CPUs and GPGPU, the acceleration of various applications using FPGAs can obtain one or even two orders of magnitude and high performance-to-power efficiency[38] In addition, FPGA provide high bandwidth, low latency interfaces to both the main processor and system memory meeting challenging interface demands of HPC applications.

FPGAs are suitable for use as computing devices in HPC market includes high-performance servers and clusters as well as high-performance embedded computers.

One main drawback to use FPGAs is the difficulty in programming them. The traditional way to program FPGAs has been through the use of hardware description languages (HDLs) such as Verilog and VHDL languages which require technical abilities and take too many efforts.

Heterogeneous SoCs based on FPGA take advantages of the benefits of FPGA while make up its weakness of serial processing ability. Altera SoCs integrate an ARM-based hard processor system (HPS) consisting of a dual-core ARM © Cortex™-A9 MPCore processor, peripherals, and memory interfaces with the FPGA fabric using high-bandwidth interconnection. This tightly integration provides over 100-Gbps peak bandwidth with integrated data coherency[39]. The Zynq ©-7000 family is based on the Xilinx All Programmable SoC architecture as shown in Fig.3. It enables implementation of custom logic in the 28 nm Xilinx programmable logic (PL) and custom software in a feature-rich dual-core ARM © Cortex™-A9 based processing system (PS). It allows for the realization of unique and differentiated system functions. The integration of the PS with the PL allows levels of performance that two-chip solutions cannot match due to their limited I/O bandwidth, latency, and power budgets[40].

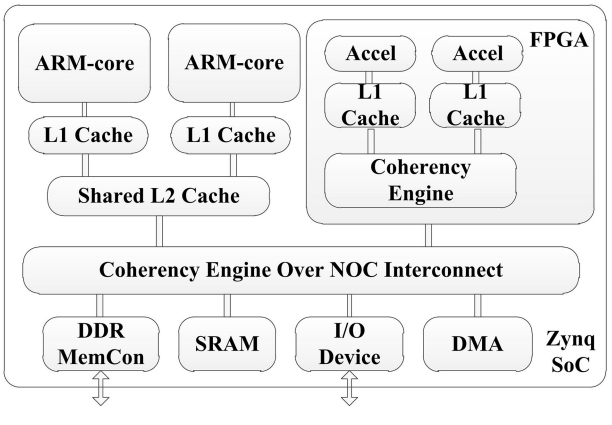


Fig. 3 Zynq SoCs architecture

In the next phase of cloud computing, there is an emerging trend of incorporating embedded purpose-built computing elements to supplement general purpose servers and optimally address the specialized processing requirements for compute intensive application, such as time-sensitive financial analysis and data center. FPGA based heterogeneous SoCs architecture with the right mix of processing elements provide the efficient blend of processor and FPGA processing for optimizing cloud applications, per-

forming better and more economically than cloud powering servers[17].

3 Heterogeneous programming

Full benefit of heterogeneous computing architectures will not be harnessed until the software tool fully embraces heterogeneous programming. The biggest barrier comes from the absence of an intuitive programming model that can match the computational model of the algorithm with the granularity and capabilities of the processing entity. Firstly, tasks have to be explicitly partitioned for the individual processors and coprocessors. Secondly, high level compilers must choose the most efficient processing core for the type of processing needed for a given application task in run time.

3.1 Hardware/software partitioning

Hardware/software partitioning considers an application as comprising a set of regions and maps each region to either processor component or coprocessor components to optimize performance[20].

In general, most applications start with an all software sequential implementation. The speedup of sequential programs is governed to a large extent by the well-known Amdahl's Law. Amdahl's Law states that partitioning yield substantial results only when mapping those regions accounting for a large percentage of execution to coprocessor for maximize acceleration. Ideally, for a 10x speedup, those regions accounting for at least 90 percent of an application's execution time must be mapped to coprocessor.

Finding the optimal hardware/software partitioning is a NP-hard problem in part because of the large number of possible partitions[41]. Several issues such as granularity of critical region, execution models and algorithm nature make the problem of partitioning of sequential programs quite challenging.

Finer granularity involves arithmetic operations or statements may expose better partitions at the expense of a more complex partitioning problem and more difficult estimation challenges. While coarser granularities that involve basic blocks, loops, or entire functions are much more efficient as coarser granularity simplifies the partitioning problem by reducing the number of possible partitions. It also enables more accurate early estimations of a region's performance, size and power[20].

Execution mode of coprocessor and processor includes overlapping or mutually exclusivity. In the overlapping model, the processor activates a coprocessor and then continues to execute concurrently with it. In the mutually exclusive model, the processor waits idly until the coprocessor finishes, then processor resumes execution. Overlapping can improve overall performance, but mutual exclusivity simplifies implementation by eliminating issues related to memory contention, cache coherency, and synchronization. When the processor and coprocessor cycles are closer to be equal, overlapping may improve performance up to a limit of 2 times[20].

Furthermore, for some applications originally written in software, part of its regions may perform well in coprocessor like DSPs or GPUs but not be suitable for hardware based coprocessor (e.g. FPGAs) implementation. For instance, recursive function calls, pointer based data structures, or dynamic memory allocation of application regions may not be easy to implement in FPGAs[42]. In order to support a wider range of program constructs and behavior, new synthesis techniques have been developed[43].Or those regions must be stay in GPPs of a heterogeneous computing architecture.

3.2 Heterogeneous programming model

Programming model is an efficient way that enables programmers to abstract the logic of applications and map it to the hardware platform[44]. A programming model must not only hide heterogeneity of the underlying processing elements, communication mechanisms, the storage elements and I/O blocks, but also expose the type of high-level parallelism[21]. Many programming model technology has been re-

searched recently. Runtime heterogeneous execution has been supported by several tools such as OpenMP, CUDA and OpenCL. Some research also presents automatic parallelization and design space exploration to improve runtime heterogeneous execution. Besides, high level synthesize is the basic technology for heterogeneous programming model.

3.2.1 High Level Synthesize

High level synthesize is a kind of programming model technology which follows the flow of acceleration from the initial C, C++ or System C description of functionality at a high level of abstraction to the final device specific Verilog or VHDL register transfer level (RTL) description of a hardware implementation (targeting an FPGA) that matched the data-flow structure of the program. Commercial tools include ROCCC[45], Xilinx Vivado HLS[46]. In the same way, graphical programming models provide digital signal processing algorithm designers with a natural way of specifying an application. Commercial examples include MATLAB Simulink[47], National Instruments LabVIEW[48].

High level synthesize make the optimization of computing architecture based on the program, rather than optimizing the program based on the computer. However, high level synthesize does not covers runtime support for parallel execution of heterogeneous tasks on heterogeneous computing architectures.

3.2.2 Runtime Heterogeneous Execution

Plenty of heterogeneous programming models have been conducted at task level. OpenMP expresses parallelism using a set of compiler directives called #pragma. OpenMP is supported on heterogeneous Cell processor[49]. NVIDIA's Unified Device Architecture (CUDA) is a heterogeneous computing environment in C and C++, aims for the development of CPU+GPU architecture[50]. CUDA requires the programmer to be aware of the underlying architecture in writing special code for performing parallel processing. Open Computing Language (OpenCL) is a standard multi core programming model. It enable acceleration of task parallel or data parallel computing targeting heterogeneous computing environment consisting of the host CPU and any attached OpenCL "devices" (CPUs, GPUs, and FPGA) in a high-level language such as C[51,52]. An OpenCL application comprises a host program and a set of kernels. In OpenCL, parallelism is declared by the programmer. Parallel threads are instances of computational kernels and used to express data parallelism. Task parallelism is accomplished by using of queues and events. With which, the coordination of the coarse grained control flow is also achieved[53]. However, OpenCL still exposes too much low-level details such as explicit platform, context management, kernel and data transfer management, making it complicated to use by non-experts. The ideal heterogeneous programming model is enable programmers to focus on developing their algorithm rather than focusing on the tedious details of underlying architecture.

Besides, a major weakness of OpenMP, CUDA and OpenCL approaches is the lack of fully support for automatic parallelization. Task mapping and scheduling schemes are performed manually; therefore the speedup achieved is largely dependent on the experiences of programmers.

3.2.3 Automatic Parallelization

FPM[54] is a flexible programming model for heterogeneous multiprocessors, which is composed of a front-end source-to-source compiler, an out-of-order scheduler and an adaptive mapping scheme. Compiler translates annotated programs to internal functions for parallel execution. Scheduler checks the data dependencies, renames the parameters, and issues the tasks automatically when the tasks are ready.

It can automatically identify the parallel region and eliminate the data dependencies with renaming techniques. Using simple annotations (# Pragma) in the sequential program that indicates which parts of the code will be run in parallel. Programmers are no long concerned on dealing with the task mapping and distribution any more.

Some other work[55] presents OmpSs programming model. OmpSs is a task dataflow programming

model that includes very fine-grained task heterogeneous execution support as well as data and task dependency management. It considers each accelerator (e.g. a GPU, a FPGA) as a single execution unit, which can efficiently execute specialized pieces of code. As a result, begin with C code annotated with OmpSs directives; the presented OmpSs ecosystem can map the execution of certain tasks to SMPs and a type of hardware accelerator. Combining OmpSs programming model with Zynq ecosystem, ARM elf executable and the bit-stream containing the accelerators for the FPGA tasks specified in the source code are generated automatically and transparently.

3.2.4 Design Space Exploration

Automatic Parallelization technology eases programming by allowing the programmer to implicitly define tasks, task mapping and even synchronization. Compiler and the runtime system are responsible to insert communication and synchronization barriers as well as to map tasks to processing core. However, if the parallelization directive entered by the programmers is not correct, wrong code will be generated.

Many design Frameworks (e.g. MAMPSx[56]) are presented to simplify the design and programming of heterogeneous computing architectures through automated design space exploration (DSE). DSE takes the application specifications, the architecture model and scheduling and communication model of heterogeneous computing architectures as input to analyze and predict the worst case performance bound and generate a large number of architectures that could be ported to target heterogeneous computing platform in a short amount of time. Thus it offers a choice to the programmers between performance, cost and power[57].

4 Challenge and future perspective

Currently, in the field of high performance embedded computing, many heterogeneous architectures have exploited right engines (e.g. DSPs, GPUs, FPGAs) to accommodate digital signal processing, graphics, real-time processing, and general compute processing tasks delivering performance, power, and cost benefits.

Heterogeneous multiprocessing using reconfigurable, pipeline and parallel structure processing engines (e.g. FPGA) targeting specific task will become a mainstream choice of future, scalable computer architectures in the face of the impending threat of dark silicon.

Heterogeneous multiprocessing provides designers with software, hardware, interconnect, power, security, and I/O programmability. With the most advanced TSMC 16nm Fin-FET process technology and second-generation SSI (stacked silicon interconnect) 3D IC technology, a "More than Moore" system with more than double the capacity and a 50% bandwidth advantage versus programmable logic device today will be implemented.

However, heterogeneous multiprocessing still presents following challenges.

1) Optimizing memory access and providing low-latency, coherent communications with adequate bandwidth for heterogeneous processing.

2) Developing programming model that enable high level abstraction while automatic partitioning the task and optimizing the mapping of system-level tasks to all available resources at runtime.

Acknowledgmeng

This project is supported by National Natural Science Foundation of China (Grant No. 50305035)

References

[1] FISHER J A, FARABOSCHI P, YOUNG C. Embedded computing: a VLIW approach to architecture, compilers and tools [M]. Elsevier, 2005.

[2] MASCO J. 7 Bad Weather. Times of Security: Ethnographies of Fear, Protest and the Future [M]. Routledge, 2013.

[3] JALIER C, LATTARD D, JERRAYA A A, et al. Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem [C]. Proceedings of the

Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2010: 184-189.

[4] PLAZA A, DU Q, CHANG Y L, et al. High performance computing for hyperspectral remote sensing [J]. Journal of Selected Topics in Applied Earth Observations and Remote Sensing, IEEE, 2011, 4(3): 528-544.

[5] CONG J, SARKAR V, REINMAN G, et al. Customizable domain-specific computing [J]. Design & Test of Computers, IEEE, 2011, 28(2): 6-15.

[6] LOCKWOOD J W, GUPTE A, MEHTA N, et al. A low-latency library in FPGA hardware for high-frequency trading (HFT) [J]. High-Performance Interconnects, Symposium on, High-Performance Interconnects, Symposium, IEEE, 2012: 9-16.

[7] VAN BERKEL C H. Multi-core for mobile phones, Proceedings of the Conference on Design, Automation and Test in Europe[J].European Design and Automation Association, 2009: 1260-1265.

[8] RAJOVIC N, CARPENTER P M and GELADO I, et al. Supercomputing with commodity CPUs: are mobile SoCs ready for HPC? [C]. Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2013: 40.

[9] MARTINEZ D R, VAI M M and BOND R A. High performance embedded computing handbook [M]. A systems perspective, CRC Press, 2008.

[10] WOLF W. High-performance embedded computing: architectures, applications, and methodologies [M]. Morgan Kaufmann, 2010.

[11] OLUKOTUN K, HAMMOND L, LAUDON J. Chip multiprocessor architecture: techniques to improve throughput and latency [J]. Synthesis Lectures on Computer Architecture, 2007, 2(1): 1-145.

[12] ESMAEILZADEH H, BLEM E, ST AMANT R, et al. Dark silicon and the end of multicore scaling, 38th Annual International Symposium on Computer Architecture (ISCA), IEEE, 2011: 365-376.

[13] CHUNG E, BURGER D, BUTTS M, et al. Reconfigurable computing in the era of post-silicon scaling[C]. 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2013.

[14] QAYUM M A, SIDDIQUE N A, HAQUE M A, et al. Future of multiprocessors: International Conferen ce on Heterogeneous Chip Multiprocessors, Informatics, Electronics & Vision (ICIEV), IEEE, 2012: 372-376.

[15] GUO C, FU H, LUK W. A fully-pipelined expectation-maximization engine for Gaussian Mixture Models [C]. International Conference on Field-Programmable Technology (FPT), 2012: 182-189.

[16] ZHANG F, ZHANG Y, BAKOS J D. Accelerating frequent itemset mining on graphics processing units [J]. The Journal of Supercomputing, 2013, 66(1): 94-117.

[17] CHUNG E S, MILDER P A, HOE J C, et al. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs[C]. 43th Annual International Symposium on Microarchitecture, IEEE Computer Society, 2010: 225-236.

[18] KAELI D, AKODES D. The convergence of HPC and embedded systems in our heterogeneous computing future[C]. 29th International Conference on Computer Design (ICCD), IEEE, 2011: 9-11.

[19] BYMA S, STEFFAN J G, BANNAZADEH H, et al. FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack[C]. 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2014: 109-116.

[20] HAUCK S, DEHON A. Reconfigurable computing: the theory and practice of FPGA-based computation [M]. Morgan Kaufmann, 2010.

[21] MAZO J C, LEUPERS R. Programming Heterogeneous MPSoCs[M]. Springer International, 2013.

[22] SMIT G J M, KOKKELER A B J, WOLKOTTE P T, et al. Multi-core architectures and streaming applications, Proceedings of the international workshop on System level interconnect prediction. ACM, 2008: 35-42.

[23] BHAT G M, MUSTAFA M, PARAH S A, et al. Field programmable gate array (FPGA) implementation of novel complex PN code generator based data scrambler and descrambler[J]. International Journal of Science and Technology, 2010, 4(1): 125-135.

[24] SHAMI M A, HEMANI A. Address generation scheme for a coarse grain reconfigurable architecture [C]. International Conference on Application-Specific Systems, Architectures and Processors (ASAP),

IEEE, 2011: 17-24.

[25] WATKINS M A, ALBONESI D H, REMAP A. reconfigurable architecture for chip skmultiprocessors [J]. IEEE micro, 2011, 31(1): 65-77.

[26] VRANESIC Z, ZAKY S, MANJIKIAN N. Computer organization and embedded systems [M]. McGraw-Hill, 2012.

[27] GARIBOTTI R, OST L, BUSSEUIL R, et al. Simultaneous multithreading support in embedded distributed memory MPSoCs[C]. Proceedings of the 50th Annual Design Automation Conference. ACM, 2013: 83.

[28] SHIKANO H, ITO M, ONOUCHI M, et al. Heterogeneous multi-core architecture that enables 54x AAC-LC stereo encoding[J]. IEEE Journal of Solid-State Circuits, 2008, 43(4): 902-910.

[29] TMS320C6452 Digital Signal Processor [N/ OL] http://www.ti.com/lit/ds/symlink/tms320c6452.pdf. 2012.

[30] TMS320C6670 Multicore Fixed and Floating-Point System on Chip[N/ OL]. http://www.ti.com/product/tms320c6670&DCMP = c66hw_110411 &HQS = Other%2BPR%2Bc66hw-pr-6670pf.2014.

[31] OMAP3530 and OMAP3525 Applications Processors [N/ OL]. http://www. ti. com. cn/cn/lit/ds/symlink/omap3530.pdf.2013.

[32] Multicore DSP+ARM KeyStone II System-on-Chip [N/ OL]. http://www.ti.com.cn/cn/lit/ds/symlink/66ak2h06.pdf.2013.

[33] LEE V W, KIM C, CHHUGANI J, et al. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU [J]. ACM SIGARCH Computer Architecture News, ACM, 2010, 38(3): 451-460.

[34] NVIDIA's Tegra 3 [N/ OL]. http://www. nvidia. com/object/tegra-3-processor.html.2014.

[35] PICA200[N/ OL]. http://people.csail.mit.edu/kapu/EG_08/Mobile3D_EG08.pdf.2008.

[36] VANDERBAUWHEDE W, BENKRID K. High performance computing using FPGAs [J/OL]. WP (Xilinx): WP375 (v1.0) September, 2010, 10: 105. http:// http://link.springer.com/book/10.1007/978-1-4614-1791-0? no-access=true

[37] SUAREZ H, ZHANG Y R. FPGA implementation of a software-defined radar processor [J]. SPIE Defense, Security, and Sensing. International Society for Optics and Photonics, 2013: 871403-871403-10.

[38] PRATAS F, ORIATO D, PELL O, et al. Accelerating the Computation of Induced Dipoles for Molecular Mechanics with Dataflow Engines [C]. 21st Annual International Symposium on Field - Programmable Custom Computing Machines (FCCM), IEEE, 2013: 177-180.

[39] PENDLUM J, LEESER M, CHOWDHURY K. Reducing Processing Latency with a Heterogeneous FPGA-Processor Framework[C]. 22nd Annual International Symposium on Field Programmable Custom Computing Machines (FCCM), IEEE, 2014: 17-20.

[40] ZYNQ 7000,"Zynq-7000 all programmable soc overview, advance product specification-ds190(v1.2) [J/OL]. available on: http://www. xilinx. com/support/documentation-/data sheets/-ds190-Zynq-7000-Over-view. pdf," August, 2012.

[41] LI H, LIU W, HAN H. Graph Reduction Algorithm for Hardware/Software Partitioning, Control [C]. International Conference on Automation and Systems Engineering (CASE), IEEE, 2011: 1-4.

[42] BACON D F, RABBAH R, SHUKLA S. FPGA Programming for the Masses [J]. Communications of the ACM, 2013, 56(4): 56-63.

[43] COOLE J, STITT G. Traversal Caches: A Framework for FPGA Acceleration of Pointer Data Structures [J]. International Journal of Reconfigurable Computing, 2011, Article ID 652620.

[44] FERNANDEZ-ALONSO E, CASTELLS-RUFAS D, JOVEN J, et al. Survey of NoC and Programming Models Proposals for MPSoC [J]. International Journal of Computer Science Issues, 2012, 9(2): 22-32.

[45] VILLARREAL J, PARK A, NAJJAR W, et al. Designing modular hardware accelerators in C with ROCCC 2.0 [C]. 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2010: 127-134.

[46] WINTERSTEIN F, BAYLISS S, CONSTANTINIDES G A. High-level synthesis of dynamic data structures: A case study using Vivado HLS [C]. International Conference on Field - Programmable Technology (FPT), IEEE, 2013: 362-365.

[47] ZOSS R, HABEGGER A, BANDI V, et al. Comparing signal processing hardware - synthesis methods based on the Matlab tool-chain [C]. 6th IEEE International Symposium on Electronic Design, Test and Application (DELTA), IEEE, 2011.

[48] RAVINDRAN K, GHOSAL A, LIMAYE R, et al. Tools for deploying dataflow models on FPGA targets [C]. Conference on Design and Architectures for Signal and Image Processing (DASIP), IEEE, 2012: 1-2.

[49] FERRER R, PLANAS J, BELLENS P, et al. Optimizing the exploitation of multicore processors and GPUs with OpenMP and OpenCL, Languages and Compilers for Parallel Computing [M]. Springer Berlin Heidelberg, 2011: 215-229.

[50] TOPA T, KARWOWSKI A, NOGA A. Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wire-grid models [J]. Antennas and Wireless Propagation Letters, IEEE, 2011, 10: 342-345.

[51] SINGH D P, CZAJKOWSKI T S, LING A. Harnessing the power of FPGAs using altera's OpenCL compiler, Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays [J]. (FPGA). ACM, 2013: 5-6.

[52] STONE J E, GOHARA D, SHI G. OpenCL: A parallel programming standard for heterogeneous computing systems [J]. Computing in science & engineering, 2010, 12(3): 66.

[53] CZAJKOWSKI T S, AYDONAT U, DENISENKO D, et al. From OpenCL to high-performance hardware on FPGAs [C]. 22nd International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2012: 531-534.

[54] WANG C, LI X, ZHANG J, et al. FPM: A flexible programming model for MPSoC on FPGA [C]. 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE, 2012: 477-484.

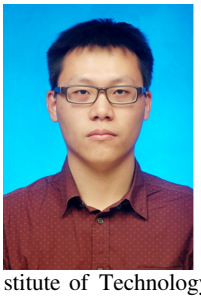
[55] FILGUERAS A, GIL E, JIMENEZ-GONZALEZ D, et al. Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays [C]. FPGA '14. ACM, 2014: 137-146.

[56] FERNANDO S, SIYOUM F, HE Y, et al. MAMPSx: A design framework for rapid synthesis of predictable heterogeneous MPSoCs, International Symposium on Rapid System Prototyping (RSP) [C]. IEEE, 2013: 136-142.

[57] CORRE Y, DIGUET J P, HELLER D, et al. A framework for high-level synthesis of heterogeneous MPSoC [C]. Proceedings of the great lakes symposium on VLSI. ACM, 2012: 283-286.

Authors' Biographies

PENG Yu, born in 1973, received B. Sc, M. Sc. and PhD degrees all from Harbin Institute of Technology, in 1996, 1998 and 2004, respectively. Now, he is professor in the Department of Automatic Test and Control, School of Electrical Engineering and Automation, Harbin Institute of Technology, China. He is also the Associated Dean of School of Electrical Engineering and Automation. His research interests include automatic test technology, intelligent test data processing, WSNs, prognostics and system health management, reconfigurable computing, etc.



HE Yongfu, born in 1988, received B. Sc and M.Sc degrees from Central South University (CSU), in 2010 and 2013 respectively. He is currently a Ph.D candidate in the Department of Automatic Test and Control, School of Electrical Engineering and Automation, Harbin Institute of Technology (HIT), China. His research interests include reconfigurable computing and intelligent test data processing, etc.

WANG Shaojun, born in 1982, received B. Sc, M. Sc. and PhD degrees all from Harbin Institute of Technology (HIT), in 2005, 2007 and 2012, respectively. Now he is an assistant professor in the Department of Automatic Test and Control, School of Electrical Engineering and Automation, Harbin Institute of Technology (HIT), China. His research interests include time series analysis and prediction, and automatic test technology, intelligent test data processing, reconfigurable computing, etc.

E-mail: wangsj@hit.edu.cn