FCUDA-HB: Hierarchical and Scalable Bus Architecture Generation on FPGAs With the FCUDA Flow

Ying Chen, Tan Nguyen, Yao Chen, Swathi T. Gurumani, *Member, IEEE*, Yun Liang, Kyle Rupnow, *Member, IEEE*, Jason Cong, *Fellow, IEEE*, Wen-Mei Hwu, *Fellow, IEEE*, and Deming Chen, *Senior Member, IEEE*

***Abstract*—Recent progress in high-level synthesis (HLS) has helped raise the abstraction level of hardware design. HLS flows reduce designer effort by allowing development in a high-level language, which improves debugging, code reuse and ability to explore different implementation options. However, although the HLS process is fast, implementation and performance analysis still require lengthy logic synthesis and physical design. For design optimization, HLS tools require design space exploration to obtain parallelism at multiple levels of granularity including parallelism within a single HLS-generated core and parallelism between multiple instances of cores. Core interconnect and external bandwidth limitations can significantly impact feasible options in the design space. With many dimensions in a design space exploration, it quickly becomes infeasible to perform full logic synthesis and physical design for each possible design point. However, generation and evaluation of communications infrastructure as part of the exploration is critical to determine the system performance. Thus, in this paper, we extend the prior multilevel granularity parallelism exploration in the FCUDA HLS flow, which takes CUDA code as design input and generates a corresponding field programmable gate array implementation. Our framework performs an initial characterization of the application design space, then analytically explores the design space considering parallelism, core interconnect, and external memory bandwidth, and selects a pareto-optimal set of designs. Our flow is completely automated to perform the exploration to characterize the analytical model, perform the exploration, select a solution, and integrate multiple instantiations of FCUDA cores via an advanced extensible interface bus interconnect. Our results demonstrate that this new FCUDA flow efficiently identifies and generates implementations with up to 5× improved system performance compared to single-level granularity parallelism (core-level optimization).**

***Index Terms*—Bus-generation, communication bus, high-level synthesis (HLS), system generation.**

I. Introduction

FAST, inexpensive, and energy efficient processing is critical throughout the wide variety of computing paradigms. In order to meet performance and efficiency objectives, multicore processors [1], [2], graphics processors [3], and field programmable gate arrays (FPGAs) are all utilized. These resources all exploit parallelism at multiple levels of granularity including instruction-level, thread-level, core-level, and memory-level parallelism.

Although these platforms have significant opportunity, there remain challenges in developing for such parallel platforms. Parallel programming languages including OpenCL [4] and CUDA [5] are increasingly used for both CPU and general public utility (GPU) development, but development for FPGA platforms is challenging and time consuming. High-level synthesis (HLS) specifically targets this issue—allowing development in high level languages to reduce design and debug effort. FCUDA [6] is one such design flow that enables development in CUDA with subsequent mapping to an FPGA. CUDA code describes parallelism at the array, thread, core, and core–cluster level; multilevel granularity parallelism synthesis (ML-GPS) exploration [7] assists in exploring design tradeoffs, while optimizing parallelism through a multilevel granularity approach.

Although the original ML-GPS can select the core design considering analytical estimates for the total number of instantiable cores, it does not consider the core interconnect, generate multicore system designs, or evaluate performance under external bandwidth limitations. These critical factors can significantly impact system design and achievable system performance; the balance of computation and communication significantly impacts delivered performance [8].

In this paper, we design systems with many cores enabled by FCUDA. Although a network-on-chip (NoC) is considered the most scalable option, NoCs on FPGAs consume resources that

Manuscript received October 12, 2015; revised January 22, 2016; accepted March 10, 2016. Date of publication April 11, 2016; date of current version November 18, 2016. This work was supported by the Research Grant for the Human-Centered Cyber-Physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research. This paper was recommended by Associate Editor J. Henkel.

Y. Chen, W.-M. Hwu, and D. Chen are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA.

T. Nguyen, S. T. Gurumani, and K. Rupnow are with Advanced Digital Sciences Center, Singapore 138632 (e-mail: k.rupnow@adsc.com.sg).

Y. Chen is with the College of Electronic Information and Optical Engineering, Nankai University, Tianjin 300350, China.

Y. Liang is with the School of Electrical Engineering and Computer Science, Peking University, Beijing 100871, China.

J. Cong is with the School of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2552821

could have been used for more computing cores, and furthermore the high amount of inter-router connections can affect routability of the design [9]. Our prior work used NoCs with the FCUDA infrastructure [10], but here we choose a bus-based interconnect to allow more core instantiations. An FPGA with hardened on-chip NoC routers [11] could make use of an NoC more efficient.

Therefore, in this paper, we extend the ML-GPS framework in the FCUDA to automatically generate advanced extensible interface (AXI)-based bus interconnects between core instantiations using one double data rate 3 (DDR3) external memory interface, and extend the design space exploration and analytical evaluation to consider memory bandwidth demand and the performance of candidate designs given total bandwidth limitations. Our new contributions are as follows.

1) Automated design space exploration considering parallelism at multiple levels of granularity as well as internal and external bandwidth limitations.
2) Automated generation of AXI-bus interconnect for systems of up to 150 cores (256 core theoretical limit).
3) Demonstration of the automated framework on both compute- and memory-bound application domains.

This new FCUDA flow, including the ML-GPS framework, automated application characterization, analytical design space exploration, and bus-based system generator is open-source.[1] The rest of this paper is organized as follows. In Section II, we discuss the original FCUDA framework and motivate exploration of parallelism at multiple levels of granularity. In Section III, we introduce system level design of multiple FCUDA cores and AXI bus-based interconnect. In Section IV we develop the analytical design space exploration models. Finally, in Section V, we present experiments, and demonstrate design space exploration, system design, and automation.

II. Background and Motivation

FCUDA originally concentrated on parallelism with a single core [6], while the follow-up ML-GPS work extended parallelism with analytical models for multiple cores [7]. In this section, we first introduce the prior framework in detail, then motivate the need for extension of the prior work to include automated generation of on-chip interconnect for system level designs, new analytical models, and design space exploration considering latency, memory bandwidth, and routability.

A. FCUDA HLS Flow

The initial FCUDA framework [6] maps core-level parallelism of CUDA kernels onto spatial parallelism of a reconfigurable fabric using MCUDA [12]. Using pragmas in the CUDA kernel to specify transformations, FCUDA translates the CUDA kernel to C-code with Xilinx Vivado HLS [13] pragma annotations that guide the HLS process.

Single program multiple data CUDA kernels concisely describe the behavior of groups of threads organized in thread-blocks. An entire CUDA kernel is composed of one or more thread-blocks, each of which has many threads, typically in power of 2. In the CUDA computation model, thread-blocks execute independently and only synchronize through off-chip memory between kernel invocations.

[1]Available at http://dchen.ece.illinois.edu/tools.html.

Fig. 1. Sequential task synchronization.

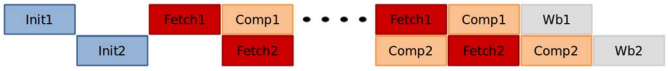


Fig. 2. Ping-pong task synchronization.

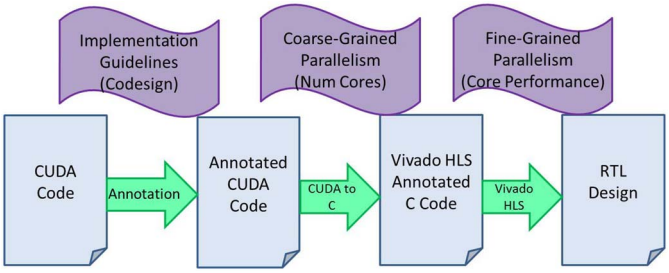


Fig. 3. FCUDA flow.

Starting from a CUDA kernel, a programmer decomposes the kernel into compute and data-transfer tasks [14]. The tasks can be scheduled sequentially or overlapped using a ping-pong buffer as shown in Figs. 1 and 2.

The programmer also adds pragma annotations to denote loop unrolling, parallelization, and thread groupings. This annotated CUDA code is translated to thread loops in C-code with Xilinx Vivado HLS annotations. A thread block in the CUDA implementation is computed by one FCUDA core; parallelism can be explored through parallelisation optimizations within a single core as well as through multiple instantiations of FCUDA cores. After translation, the C-code of the kernel core is synthesized with Vivado HLS to generate a register transfer level (RTL) implementation of the kernel, which can then be synthesized to an FPGA implementation using Vivado [15] logic synthesis. An overview of the original FCUDA flow is shown in Fig. 3.

Each FCUDA core executes threads in its thread loop sequentially, and uses private on-chip block RAMs (BRAMS) and registers for local memory, and external memory accesses to fetch data into local buffers. Different combinations of fine-grained optimizations produce cores with variation in core area, latency (in cycles), achievable frequency, and power consumption.

B. FCUDA Design Space Exploration

For each possible core design, the resource use impacts the number of instantiable cores. However, the large number of core designs (many combinations of FCUDA pragmas) have sometimes significant variance in both core area and latency. These nonlinear variations form a large design space that must be explored to find the best combination of core design and number of instantiable cores. Furthermore, evaluation of the area and execution latency for each design requires expensive logic synthesis and RTL simulation processes.

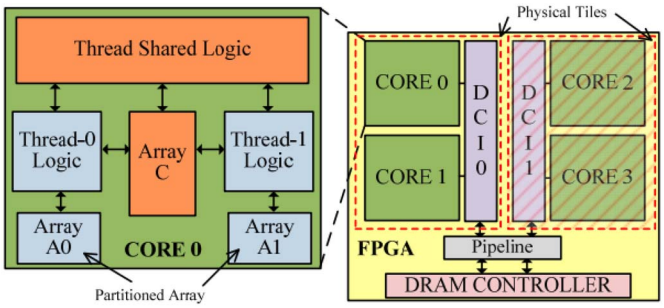


Fig. 4. Thread, core, core–cluster, memory granularities [7].

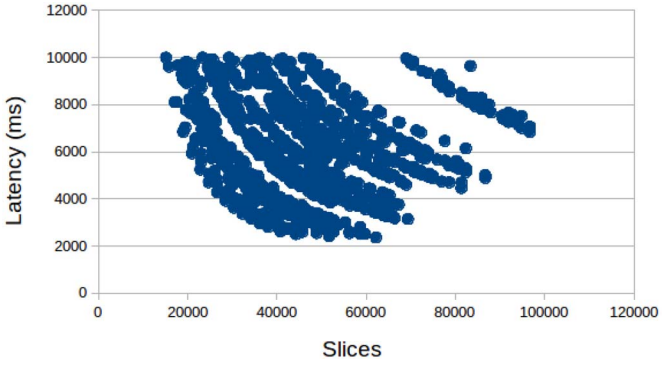


Fig. 5. Design space of CP.

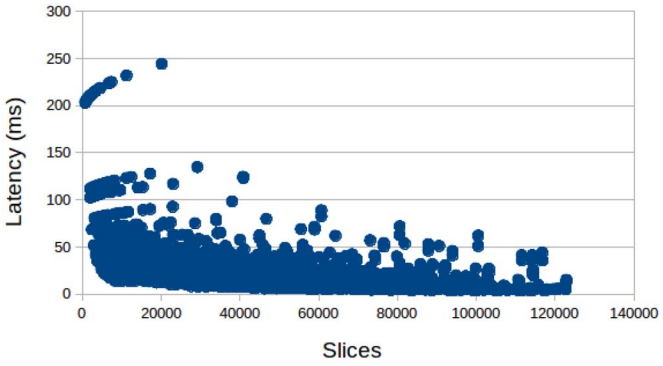


Fig. 6. Design space of FWT2.

In our original design space exploration ML-GPS framework [7], we used profiling information to generate application-specific core models to estimate core area and frequency. Thus, we enabled analytical exploration of the large design space with the cost of only a few logic syntheses and RTL simulations in order to generate profile information for the core model. This shows substantial performance improvement compared to the original FCUDA framework, but retained several weaknesses, specifically:

1) performance estimates for multicore designs idealize parallelization between cores;
2) no modeling of the area or performance effects of core interconnect or external memory access;
3) no multicore system design with interconnect.

In this paper, we address these issues to specifically consider the core interconnect during design space exploration with improved analytical models, exploration of both number of instantiable cores, and the total bandwidth demand of cores, and automated generation of system level designs with multiple cores and interconnect. In addition, we update the analytical models to use Xilinx's Virtex 7 series of FPGAs.

In the ML-GPS framework [7], we cluster cores that share a single data communication interface (DCI) and placement constraints (Fig. 4). Multiple core–clusters are instantiated, with the multiple DCIs connected with an additional shared interconnect. The DCI module is responsible for arbitration between cores for off-chip storage access. Sharing a single DCI module among all cores may result in long intercommunication wires, affecting achievable frequency. The DCI is required for communication, but we should minimize resource use to reduce impact on the number of instantiable cores. Thus, for this paper, we select a simple on-chip bus architecture based on AXI which is hierarchical and scalable and we explore the hierarchy of bus interconnects in order to minimize resource use and impact on achievable frequency.

As examples of the size and variation in the potential system design space, we show the entire design space for two benchmarks, CP and FWT2, comparing total latency and area (slices) in Figs. 5 and 6. Each application has more than 2000 unique combinations of core–cluster, core, thread, and array parameters in the design space. If each of these design points is fully evaluated with logic synthesis, place and route, and RTL simulation, evaluation would require over two months of execution time for each benchmark.

The design space contains many complex tradeoffs. Fig. 7 demonstrates that different configurations (C0 to C10) are optimal in terms of cycles, clock frequency, total parallel threads, and execution latency for the benchmark matrix multiplication. With automatic multiple instantiation, all configurations have high resource utilization (over 75% of device slices). Configuration C0 corresponds to the FCUDA configuration without design space exploration. This simple demonstration shows the need for design space exploration due to the variable impact of parallelism extraction on design objectives.

III. AXI Bus-Based FCUDA System

This paper has two main extensions from prior work in CUDA-based HIS: 1) automated generation of AXI-based bus interconnect for multiple FCUDA core instantiations and 2) extension of the profiling and design space exploration to improve accuracy and include communication hierarchy limitations in the exploration.

First, we will discuss the design, implementation, and automated optimizations for the bus-based interconnect, followed by the extension of the analytical models and design space exploration in Section IV.

A. Bus Interconnection for FCUDA

On-chip interconnect is required for systems with multiple cores sharing access to external memories. In the CUDA computation model, it is not necessary to support communication between cores, as no output of a core can be directly the

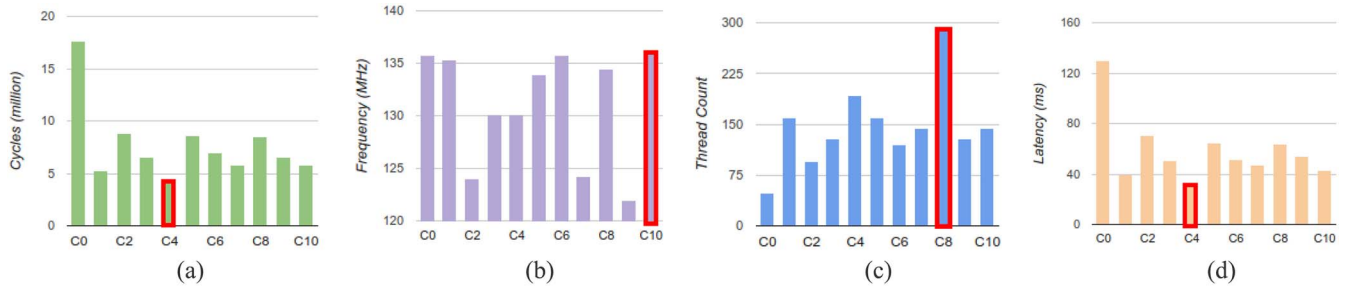


Fig. 7. Performance attributes of MM design space configurations. (a) Execution cycles. (b) Clock frequency. (c) Hardware concurrency. (d) Execution latency.

input of another core. Although input data can be shared among cores for some data access patterns, more complex interconnect will use FPGA resources, limiting the number of instantiable cores. Our prior FCUDA-NoC work demonstrated effective NoCs with support for data sharing, but with limitations on total network size due to routability [10]; our FCUDA-SoC work demonstrates use of bus-based interconnect on SoC architectures, but without modeling and design space exploration [16]. Thus, we concentrate on bus-based interconnect to minimize resource use and effects on routability. Bus interconnect size scales with the number of components connected to the bus; thus, we decompose large networks into a hierarchy of buses to decrease impact on area and achievable frequency. This tree of buses can be considered a small tree-based NoC, but we use only two-level hierarchies.

The FCUDA flow uses Xilinx Vivado HLS, targeting Xilinx FPGAs (in our case, a Virtex-7), and the Vivado HLS generated FCUDA cores implement AXI-compatible communication interfaces. Therefore, we use the AMBA AXI4 protocol. Although Vivado-generated interfaces are AXI-compatible, we need to make several changes in the CUDA to C compilation in the original FCUDA framework in order to support a system level design with multiple FCUDA cores.

1) Decentralized Control: With multiple core instantiations, instead of a single core that represents the top-level CUDA kernel, we must now have multiple cores that can independently operate on portions of the kernels' computation workload. Thus, we must add core and thread identifiers (similar to CUDA thread and block identifiers) for each of the synthesized hardware computing core. In addition, we must add code that uses those identifiers to compute workload distribution; in CUDA, this is handled by the GPU driver and hardware support for queues of thread-blocks. In FCUDA, we must implement these workload distribution techniques in order to decentralize workload distribution.

2) Memory Mapped Interface: In the original FCUDA core interface, each input or output array or scalar parameter has independent memory ports. Although this is possible in our system, it would require bus connections for each interface of the core, which would increase area overhead of the system as well as effecting the achievable frequency. Thus, all function arguments are mapped into a single external port with appropriate offsets, and Vivado HLS pragmas are used to direct generation of AXI interfaces.

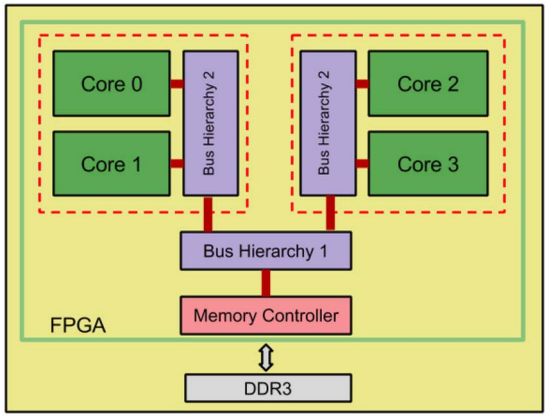


Fig. 8. Bus system designed for FCUDA.

B. FCUDA With AXI-Based Bus

Our AXI-based bus must efficiently share access to high speed external memory interfaces, such as DDR3 on our target Xilinx Virtex-7 FPGAs. In order to share these resources efficiently, the bus-design must arbitrate between multiple cores. The cores follow the AXI protocol and assume that a requested transaction will eventually complete; thus, arbitration must include sufficient buffering to prevent dropped transactions.

For an efficient bus design, we must consider buffering, arbitration, area consumption, and achievable frequency in order to ensure that the bus is not the limiting factor in system performance. Fig. 8 shows an example bus-based system, with four cores organized in two tiles of two cores each.

1) Bus Hierarchy: An AXI interconnect module can support up to 16 connected cores, but a single level interconnect with 16 cores has significantly higher area and lower achievable frequency than a two-level hierarchy decomposed into four clusters of four cores each. Furthermore, many of our benchmarks could fit significantly more than 16 cores, necessitating at least one extra level of hierarchy. In our implementation, we explore potential two-level hierarchy decompositions to find the best combination of first and second-level hierarchy sizes for all particular system sizes. For a particular number of cores, we explore possible options for cores per cluster (CR) and total clusters (CL), where due to the port limitations of an AXI-bus, neither CR nor CL can be greater than 16. For simplicity, we only consider options where the cores are divided evenly among clusters.

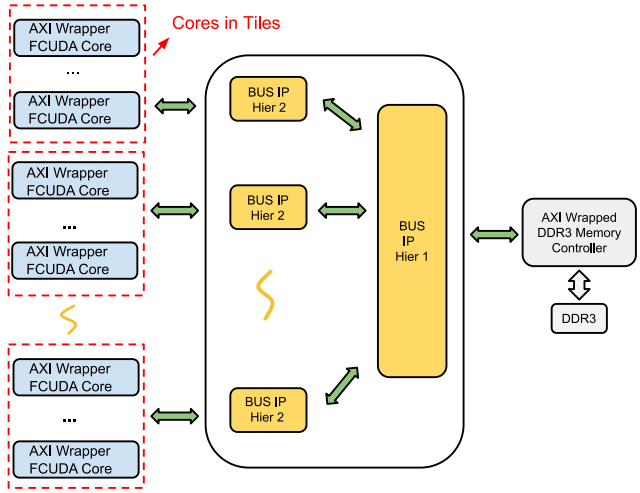


Fig. 9. Bus system hierarchies of FCUDA.

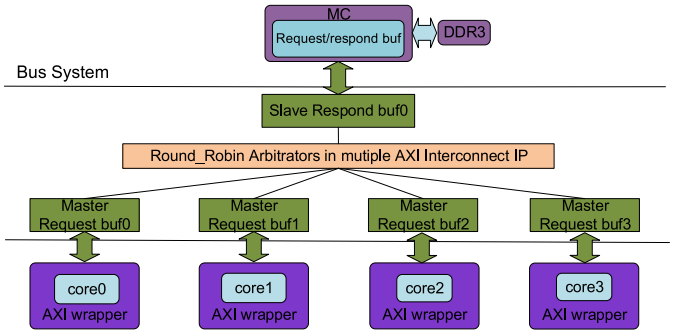


Fig. 10. Buffers for master and slaves inside bus.

There is no benchmark which can instantiate more than 160 cores in a single design in our experiments due to either exceeding board resources or limitations in place-and-route. Furthermore, due to the effects of long routing paths, our analytical model will select designs with additional loop unrolling and internal core parallelism instead of more cores once those routing paths are too expensive. Thus, although we could consider multilevel hierarchies (or a full binary tree), this paper concentrates on two-level buses. The hierarchical bus system is shown in Fig. 9.

2) Transaction Buffering: Ideally, all FCUDA cores will be active at the same time, with each core generating a stream of load and store requests. In most systems, a large number of cores are executing simultaneously for coarse-grained parallelism. However, the shared bus can only process one transaction at a time, and each core only process a single (burst) request at a time. Thus, we can simply add first-in first outs (FIFOs) at each bus port to buffer unprocessed requests and responses as shown in Fig. 10.

In the example four-core system, each core has an associated request buffer, and arbitration logic shares access to the memory controller. Similarly, a slave response buffer holds memory controller responses that must arbitrate for bus access. Total internal bandwidth can be computed with transaction size, cycle time, and transactions per cycle.

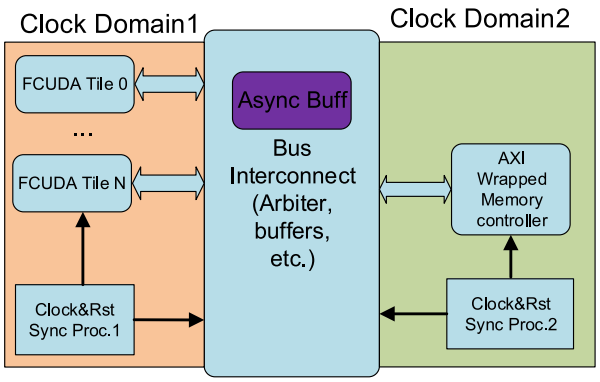


Fig. 11. Clock domains in bus system.

3) Transaction Arbitration: We use round-robin transaction arbitration for two main reasons. First, round-robin is a simple arbitration algorithm and mechanism that can be implemented with low area overhead and little effect on the achievable frequency of the module. Next, in the FCUDA model, all the cores will start execution at roughly the same time, and the kernel is considered complete when the last core finishes execution. Furthermore, each core is only processing at most one memory transaction at a time. Thus, a simple method to evenly share access among cores will yield uniform core performance with fair, low variation average access latency.

C. System Optimizations

After composing FCUDA cores and the AXI-based bus interconnect, we perform additional system-level optimization to improve performance and area of system implementations.

1) Placement Constraints: In a large system with many components, adding placement constraints for core clusters can effectively reduce long wires in the final placement and routing and thus improve achievable frequency. Thus, we use Xilinx's Advanced Silicon Modular Block architecture [17], and assign Pblock placement constraints for core clusters together with their shared AXI Interconnect IP. These placement constraints achieve improved clustering and result in improved achievable frequency.

2) Multiclock Domain Setting: After placement constraints, achievable frequency is improved, but the clock and reset networks remain long wires. Furthermore, because the location of the DDR3 memory controller is fixed, the connections between the memory controller and the bus hierarchies may be long. The memory controller generated by Vivado IP Integrator provides a fixed clock output as application interface clock, but the clock can only be used as a reference clock for communication between the memory controller and applications; it cannot be used as the clock for the hierarchy of buses and cores. In this paper, we always have a single core type, and thus do not consider multiple cores with different clocks. We divide the system level design into two clock domains, one for the memory controller and one for the FCUDA cores as shown in Fig. 11.

Asynchronous FIFOs handle cross-domain buffering between the memory controller and FCUDA core clock

domains. With separate clock domains, we operate the memory controller at a higher frequency independent of the cores' achievable frequency, thus increasing total system bandwidth.

D. System Automation

We automate FCUDA compilation, bus generation, and system design generation into a unified tool chain. The tool chain flow includes front-end CUDA to C compilation, Vivado HLS synthesis, batch script mode system set up in Vivado, and launching of synthesis (including P&R) or system simulation. With a given number of FCUDA cores and core clusters as input, the bus generation first instantiates cores within a cluster and connect them to one shared AXI Interconnect IP (bus level 2). The tool chain iterates this process for the given number of core clusters. Then, all the AXI Interconnect IPs are wiring to another AXI Interconnect IP (bus level 1). Finally, the first-level AXI Interconnect is connected to the memory controller, enabling the communication between FCUDA cores and the memory. We also combine the design space exploration into the tool chain which considers resource usage, latency, and memory bandwidth of the bus system. The optimal settings will be selected by the tool chain based on the models and algorithms we present in Section IV.

IV. Design Space Exploration

Exploration of the multilevel granularity space is based on analytical models to estimate resource consumption, achievable frequency, and latency in cycles. The analytical models are built by first performing synthesis and full implementation of selected test cases and then using linear regression to estimate the relative importance of variables. In this paper, since we have automated the system generation, we are able to collect area and performance estimates based on full-system implementation results rather than the linearly scaled estimates of the prior design space exploration [7]. Finally, using the analytical model, we perform a design space exploration and select a set of candidate design points that form a pareto-optimal set of system designs that the user can select from.

In this section, we will first present the analytical models for estimating system performance and resource usage, then discuss the automated model training procedure. Finally, we will present the analytical design space exploration algorithm.

A. Analytical Modeling

Accurate and efficient analytical modeling is critical to the development of design space exploration [18]. Without analytical models, we would need to actually perform synthesis (including P&R) and simulation to evaluate candidate core and system designs. Thus, the best result is discovered based on exhaustive enumeration or sampling, instead of understanding the relation between parameters and system results.

The relation between HLS synthesis parameters and resulting system characteristics is application dependent; the effects of loop unrolling, memory partitioning, and parallelism (among others) all depend on the code being transformed. Thus, for each application, we sample a set of designs and use linear regression to determine the values of analytical model coefficients. We will now present the analytical models we use for resource consumption, frequency, and latency.

1) Resource Usage: The resource consumption of a system design is the combination of: number of cores per cluster (CR), threads per core (TH), array-partitioning per core (AP), and number of core–clusters (CL), plus overheads for AXI bus IPs (IP_BUS), bus hierarchy (Bus_Hierarchy), and a constant overhead including the area of the memory controller and asynchronous FIFOs. For this model, we use six fit coefficients, R_0–R_5. The model and coefficients are shown in (1) and (2). R_{Cluster} is the resource consumption of a single core–cluster which depends on the number of cores in one cluster (CR), unrolling (TH), and memory partitioning (AP) degrees as well as the bus resource, whereas R is the total resource consumption of the design containing CL core–clusters, the bus resource, and additional resource due to memory controller, reset logic, system IPs, etc. (R_5). Note that each of the core–cluster has a bus IP that supports the communication of the cores in a cluster with the memory controller at the next hierarchy level. The size of a bus IP is a function of the number of enabled AXI slave ports (from 1 to 16 ports) either connecting to FCUDA cores of a cluster [IP_BUS(CR)] or connecting clusters of cores at the next hierarchy level [IP_BUS(CL)]. In systems with only one level of hierarchy, additional overhead for multiple levels of hierarchy evaluates to 0. Together, the resource model includes variables for the primary factors that can affect the area of a core, the area of communications infrastructure, and the system area (that includes multiple instantiation of cores and communications). This model currently only supports 1 or 2 levels of hierarchy, but could easily be extended to multilevel

$$R = R_{\text{Cluster}} \times \text{CL} + (\text{Bus\_Hierarchy} - 1) \times \text{IP\_BUS(CL)} + R_5 \quad (1)$$

$$R_{\text{Cluster}} = R_0 + R_1 \times \text{CR} + R_2 \times \text{CR} \times \text{TH} + R_3 \times \text{CR} \times \text{AP} + R_4 \times \text{TH} \times \text{AP} + \text{IP\_BUS(CR)}. \quad (2)$$

We construct an independent resource model for each type of FPGA resource (lookup table, flip-flop, BRAM, and digital signal processor) so that we can effectively estimate which resource is the bottleneck with multiple instantiations. We additionally verify routability using the existing routability analytical model [9]. The resource model can be used to determine a maximum number of instantiable cores, but later analysis in memory latency may determine that a smaller number of cores fully utilizes memory bandwidth. In this case, we may select either: 1) a core design with different memory use or 2) fewer cores at the system level in order to improve achievable frequency.

2) Achievable Frequency: The frequency (clock period) model, aims to capture frequency degradation due to wire routing within the core–cluster, as well as organization of core–clusters. HLS-generated RTL is often pipelined for a nominal clock period defined by the user. However, the actual clock period of the placed netlist is often degraded (i.e., lengthened) due to wire delays introduced during P&R. Thus, we

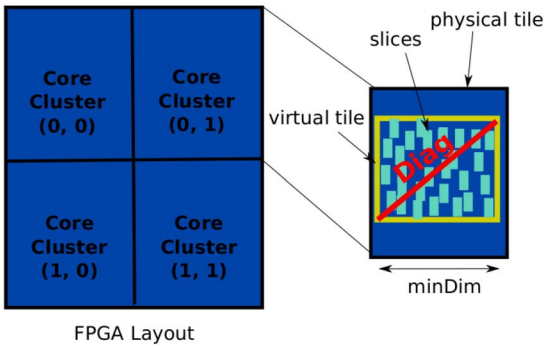


Fig. 12. Physical tiles, virtual tiles, and model parameters.

also model the system level placement. First, we consider the total number of core–clusters in the system-level design. The number of core clusters is used to divide the entire FPGA into a set of physical tiles. When there are multiple possible options, we explore different options for dividing the FPGA into equal tiles. Within those physical tiles, we find a virtual tile: the minimum subregion of a tile with a valid placement of the core–cluster. In Fig. 12, we show an example FPGA with four physical tiles, and the virtual tile within a physical tile that is the minimum placement region.

The clock period within a cluster is affected by wire-length, which we model by the diagonal length of a virtual tile containing the core cluster (Diag), and routability in the physical tile modeled by the slice utilization of a cluster in the physical tile (Util). In addition, the amount of routing necessary is affected by array partitioning (AP) and threads per core (TH). The clock period estimation model and fit coefficients are shown in (3). In this model we incorporate the effects of parallelism granularities (core-design) and layout information; we assume that the design and interconnection on a cluster level are the primary effect on frequency, and that wires between clusters can be appropriately pipelined

$$\text{Period} = P_0 + P_1 \times \text{Diag} + P_2 \times \text{Util} + P_3 \times \text{AP} + P_4 \times \text{TH}. \quad (3)$$

To minimize Diag, we enforce that placement constraints are minimal, given core–cluster area requirements and minimal region perimeter. Thus, core clusters in a virtual tile are near-square rectangular areas; Diag is computed as in (4), where minDim is the minimum dimension of a physical tile in slices, and R_{slice} is the slice counter of core–cluster logic

$$\text{Diag}^2 = \begin{cases} 2 \times R_{\text{slice}} : & \text{if } R_{\text{slice}} \leq \text{minDim}^2 \\ \text{minDim}^2 + \left(\frac{R_{\text{slice}}}{\text{minDim}}\right)^2 : & \text{if } R_{\text{slice}} > \text{minDim}^2. \end{cases} \quad (4)$$

Util in (3) represents the slice utilization rate of the physical tile by the core–cluster slices. Note that in general the physical tile is larger than the virtual tile: this models that when a placement region can be relaxed (reducing utilization), routability improves and thus clock period also improves. Parameters R_{slice} (hence Diag) and Util in (3) are calculated using the resource model. Hence, parameter Diag incorporates the core–cluster resource area and layout information

Fig. 13. Computation-bound ping-pong.

Fig. 14. Memory transfer-bound ping-pong.

while Util incorporates the routing flexibility into the period model, i.e., whether the core–cluster logic can be routed into one physical tile.

3) System Latency Model: In the original FCUDA design space exploration models, latency was computed simply as core latency (per thread block) multiplied by the number of thread blocks, divided by the number of cores in the system. However, the original model idealizes parallelization efficiency, ignores communication latency overheads, and opportunity to overlap communication and computation.

As discussed in Section II, applications may overlap compute and communication using ping-pong buffers to pipeline processing of blocks of data. Fig. 2 depicts a perfectly overlapped system, where communication and computation latency exactly match. However, in practice, either the computation or communication will be the dominant factor in total latency, as shown in Figs. 13 and 14. Our FCUDA compiler offers ping-pong double buffering option to overlap computation and communication for better performance. Therefore, in this paper, we enable this option to study the impact of communication as well as computation tasks on the overall performance.

Thus, we refine the idealized computation latency model

$$\text{Latency} = \max\{\text{cmp\_latency}, \text{trn\_latency}\} + \text{overhead}. \quad (5)$$

In (5), cmp_latency is the sum of computation latency for all thread blocks. trn_latency is the sum of latency for all memory transfer overlapped with computation. In this paper, we assume that ping-pong buffers are always used. If ping-pong buffers are not used, the latency is simply the sum of computation and transfer latencies. overhead is the sum of latencies of initialization, nonoverlapped fetch or writeback, and any delays between processing. In practice, overhead is a very small factor compared to computation and memory transfer, which can be safely ignored.

a) Computation latency: Since all the blocks are executed as a for loop and all cores execute the same code, total computation latency can be estimated using

$$\text{cmp\_latency(TH, AP, CR, CL)} = \text{Cyc} \times \frac{N_{\text{block}}}{\text{CR} \times \text{CL}} \times \text{Period}. \quad (6)$$

In (6), N_{block} represents the total number of kernel thread-blocks, Cyc is the number of execution cycles required for one thread-block and Period is the shortest clock period the system can meet. As discussed earlier, Period is affected by all

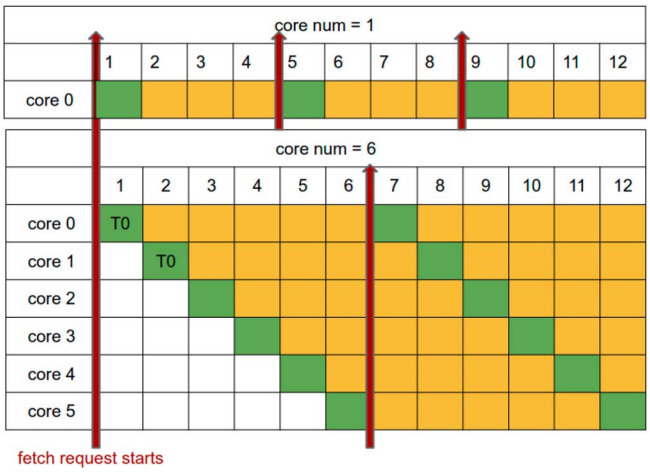


Fig. 15. Bus utilization of one core versus six cores.

the design space dimensions and is estimated through our estimation model for achievable frequency. As shown earlier, CR and CL are the pair of core number and core–cluster parameters. However, Cyc is affected by the TH and AP parameters, and is gathered from Vivado HLS reports. This means that although other parameters will be fit as normal, we still require an HLS invocation to determine the latency in cycles. For this reason, even with the analytical model, it may still be prohibitively expensive to try all combinations of parameters. Thus, in Section IV-C1, we will discuss a binary search method used with the analytical model.

b) Transfer latency: Transfer latency is the latency of the data preparation stage for the next stage of computation in FCUDA. For some benchmarks, this only contains data fetching from off-chip DDR3, while for some others, this may also contain writing intermediate results back to DDR3.

Moving data between DDR3 and computation logic in FCUDA cores via our bus communication architecture is a pipeline with multiple stages, as shown in Fig. 10. Intermediate stages are either buffers or arbitration logic, including FCUDA AXI wrappers' output buffer, BUS interfaces, BUS arbitration logic, clock domain synchronization, the memory controller, and buffers inside DDR3. The memory frequency for the system depends on the longest stage in the pipeline.

Fig. 15 illustrates the bus utilization of a system with one core versus a system with multiple cores. Stages in green denote the start of a memory request, whereas orange stages denote time-steps where the bus is not utilized (for that core). With a single core, the bus is under-utilized, as the core spends time waiting for a single request to return. In contrast, multiple cores can initiate requests in subsequent cycles to fully utilize the bus. However, with many cores, full utilization of the bus with round-robin bus arbitration can increase average access latency compared to the original under-utilized bus. The overall utilization is related to both the number of cores and workload per-core which is affected by core design. Therefore, it is important to choose a number of cores that balances average transfer latency and the total number of transfers.

Based on the previous analysis, the transfer latency is affected by the total number of cores in the design, which in turn depends on number of cores per cluster (CR) and number of core clusters (CL). Thus, it can be computed using (7). trn_cycle is the latency in clock cycles of the transfer task per thread block. It is estimated by using the model described in (8). The coefficient R_0 represents constant overhead during memory transfer. R_1 and R_2 factor the impact of number of cores per cluster (second-level hierarchy) and number of clusters (first-level hierarchy), respectively, on data communication. Finally, the total number of cores in the design is weighted by R_3

$$\text{trn\_latency}(\text{CR}, \text{CL}) = \text{trn\_cycle} \times \frac{N_{\text{block}}}{\text{CR} \times \text{CL}} \times \text{Period} \quad (7)$$

$$\text{trn\_cycle}(\text{CR}, \text{CL}) = R_0 + R_1 \times \text{CR} + R_2 \times \text{CL} + R_3 \times \text{CR} \times \text{CL}. \quad (8)$$

B. Application Profiling

To determine the fit parameters for the analytical models discussed in Section IV-A, we perform several sets of profiling measurements. For each application, we perform synthesis, place and route for 20 selected combinations of system level designs including different parameter settings for unrolling, pipelining, number of cores per cluster, and number of core–clusters. In addition, we perform short simulations for 14 different number of cores per cluster and core–clusters to characterize the scaling of transfer latency. For each combination, we perform FCUDA HLS, Vivado HLS, Vivado logic synthesis and implementation (for area and frequency results). When collecting area data after implementation, we also count the extra resources instantiated by Vivado to perform routing [19]. It ensures the routability of all the design candidates produced by our analytical models. The results of all combinations are used to perform linear regression to compute all model coefficients for the subsequent design space exploration.

C. Analytical Design Space Exploration

Given trained analytical models, we perform design space exploration of possible design points. However, because the latency in cycles for any combination of parameters not originally profiled is determined by executing Vivado HLS, it is still expensive to exhaustively try all possible combinations. Thus, we will use a binary search technique to determine the most attractive set of configurations to try, and then use Vivado HLS to find the performance of those designs (in term of clock cycles). In experiments, the binary search technique finds the parameter setting that minimizes latency within 1% of the optimal determined through exhaustive search.[2]

1) Efficient Design Space Search: Before we present our design space search algorithm, we first discuss our observation of the impact of HLS parameters on the performance of an input CUDA kernel. Fig. 16 depicts the MM kernel latency for different unroll and array-partition pairs (TH, AP). For each unroll and array-partition pair (TH, AP), we use $\text{CR}_{\text{TH}}^{\text{AP}}$ and $\text{CL}_{\text{TH}}^{\text{AP}}$ to denote the core and core–cluster values that minimize system latency in (5). We can observe [Fig. 16(a)] that

[2]The exhaustive search runs Vivado HLS for all parameter combinations, but uses the analytical model for area and frequency estimates.

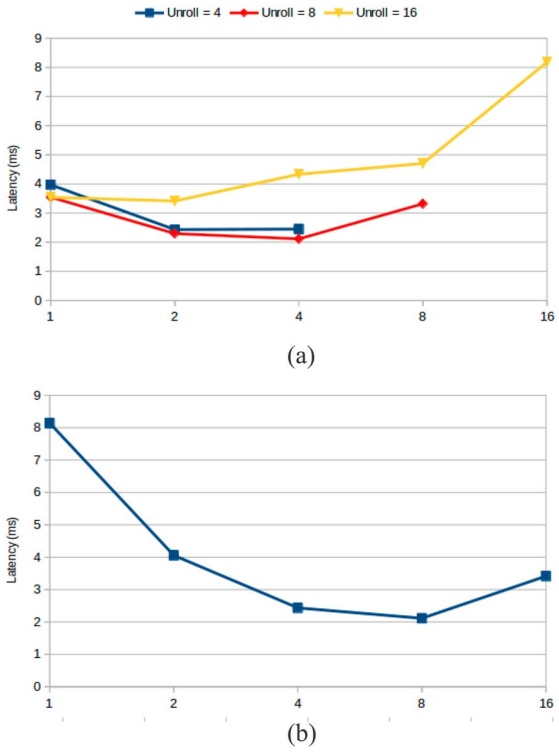


Fig. 16. Unroll and array partition effects on latency. (a) Array partition degree. (b) Unroll factor.

the value of execution latency as a function of array partition degree for a fixed unroll factor decreases monotonically until a global optimal point, after which it increases monotonically. With a certain unroll value, the core behaves like a multi-issue core and computation is done in parallel. However, without memory partitioning, limited memory ports would become the bottleneck to support the level of parallelized computation, so increasing array partitioning increases thread parallelism and reduce computation latency. However, after significant array partitioning, latency in cycles does not decrease anymore, yet higher connectivity hurts achievable frequency.

Unroll factor [Fig. 16(b)] also has a similar impact on latency. Intuitively, more unrolling exploits more computation parallelism. However, after sufficient unrolling, it may not be beneficial due to array access bottlenecks, reduced achievable frequency, and additional area overhead. These observations have been verified on other benchmarks as well.

Instead of using exhaustive exploration, we use a binary search heuristic (Algorithm 1) leveraged by the two key observations as described above: 1) for a fixed unroll degree, latency curve is convex with respect to array partitioning and 2) the latency curve is convex with respect to unroll degree under the best partitioning degree. For each point p in unroll-partition space, the binary search estimates the latency of p and $p+1$ by calling HLS to get the cycle values followed by latency computation based on (5). Because the two main core-design parameters are convex and can be determined independently, we can perform a binary search on each dimension. First, we perform a binary search on the array partitioning parameter to determine the optimal partitioning degree AP between 1 and an application-specific maximum in log|AP| steps. Then, at

Algorithm 1 Design Space Search

```
1:  SEARCH(1);
2:      ▷ Search sequence: unroll followed by array partition
3:  function SEARCH(dim)
4:      if search all the dimensions then
5:          Let (TH, AP) be unroll and array partition pair;
6:          lat = Lat(TH, AP, CR_TH^AP, CL_TH^AP);
7:          UPDATELATENCY(lat, TH, AP);
8:          return lat;
9:      end if
10:     Let space[] be the design space of dimension dim;
11:     BINARY_SEARCH(dim);
12:     return curr_best;
13: end function
```

that array partitioning, we again perform a binary search on the unrolling parameter to find its optimal value in log|TH| steps. Thus, the overall complexity of our binary search is log|TH| × log|AP|, where TH and AP represent the design dimensions of unrolling and array partitioning, respectively.

V. Experiments

In this section, we evaluate the resource consumption of the AXI bus-based architecture, the accuracy and effectiveness of analytical models and design space search algorithm with enhanced memory latency model together with the performance advantage of multilevel granularity parallelism exploration.

We use our extended FCUDA system tool on a set of benchmarks from the CUDA software development kit [20] and Rodinia [21]. In this paper, we concentrate on integer benchmarks due to smaller per-core resource use, and therefore more system cores in the design space, and more complex core interconnect. Thus, all benchmarks are either originally integer benchmarks or converted to be integer benchmarks. Table I summarizes the benchmarks including number of thread blocks and number of processing data in the CUDA implementation and description of the kernel functions. One thread block is an execution unit of a single FCUDA core. Therefore, the number of thread blocks represents the amount of execution units that a design needs to process by evenly distributing them to FCUDA cores.

For each application, we perform an initial set of experiments for collecting profiling data to build the analytical model. Then, based on the analytical model, we explore the design space and select a design that maximizes performance. The selected design is placed and routed using Xilinx Vivado 2014.4, targeting a Xilinx VC709 with Virtex-7 690T FPGA. We report area, achievable frequency, and latency of designs. We now present our experiments and results in detail.

A. Resource Utilization of Bus Back-End

First, we evaluate the resource usage of our AXI-bus based interconnect. We measure the resource consumption of the AXI Interconnect IP core with between 1 and 16 ports. Table II shows the resource utilization of a single hierarchy of AXI-bus IP as a percentage of the Virtex 7 690T device.

TABLE I
BENCHMARKS DESCRIPTION

Application (Kernel Name)	Thread Blocks	Data Set Size	Description
Matrix Multiply (mm)	4096	1024x1024 array	Computes multiplication results of two matrices, a standard and common benchmark for parallel computing.
Fast Walsh Transform (fwt1)	8192	8MB Vector	Walsh-Hadamart transform, a generalized Fourier transformation used in various engineering applications.
Fast Walsh Transform (fwt2)	8192	8MB Vector	Same as above.
Discreet Wavelet Transform (dwt)	4096	262144 points	Compute 1D partial wavelet decomposition using Haar basis.
Coulombic Potential (cp)	32768	128x128 grid, 1024 atoms	Computation of electrostatic potential in a volume containing charged atoms.
HotSpot	4096	512x512 grid	A widely used tool to estimate processor temperature based on an architectural floorplan and simulated power measurements.
LavaMD	4096	4096 boxes, 128 particles/box	Calculation of particle potential and relocation due to mutual forces between particles within a large 3D space.
PathFinder	4096	100000x100 grid	Dynamic programming used on 2-D grid to find a path with smallest accumulated weights.

TABLE II
UTILIZATION OF AXI-BUS WITH DIFFERENT PORT NUMBERS

	LUT	Flip-Flop	BRAM	DSP
1	0.06%	0.07%	0.10%	0%
2	0.24%	0.23%	0.30%	0%
3	0.33%	0.30%	0.34%	0%
4	0.40%	0.37%	0.37%	0%
8	0.79%	0.64%	0.51%	0%
16	1.56%	1.18%	0.78%	0%

Resource usage of the single hierarchy of the bus IP scales linearly between 2 and 16 ports. Our design space exploration will explore system designs with two levels of bus IP hierarchy, with modeling of interconnect within a core cluster and between core clusters as in (1).

B. Analytical Model Accuracy

Next, we evaluate the accuracy of the analytical model as well as the performance benefits of design space exploration using our analytical model. For each benchmark, the design space consists of roughly 2000 combinations of thread, array, core, and core–cluster parameters. The analytical model is generated by performing Vivado HLS followed by synthesis, place and route for 20 selected combinations of HLS settings to determine analytical model parameters as described in Section IV-B.

Since most of the applications are either memory bound or computation bound, we choose CP and FWT2 as representative cases because CP is a computation bound application and FWT2 is a memory bound application. Fig. 17(a) and (c) depicts the entire design space for CP and FWT2 kernels, and Fig. 17(b) and (d) shows the subset of design points close to the optimal configuration. The design point selected by the analytical exploration is marked in red. We compare the optimal solution selected by the analytical model using our binary search algorithm to the best solution based on exhaustive search and find that the two solutions differ by only 1%. We then perform synthesis and place and route of the selected optimal solution for the two benchmarks and compare the actual implementation results with the estimations reported by our analytical model. On an average, we find that the resource estimation of our model is within 10%, and the frequency estimation is within 20% when compared to the results from the Vivado implementation. Although 10% and 20% seem relatively high error percentages, our analytical model can accurately capture the different trends when we explore different design parameters. That explains why our analytical model-driven solution is only 1% away from the optimal solution based on exhaustive search. This 1% shows that our analytical model in tandem with the binary search algorithm is able to search for the best solution in the design space effectively and efficiently.

TABLE III
NUM. CORES COMPARISON

Benchmark	ML-GPS Best Num. Cores		ML-GPS Max Num. Cores	
	32-bit	16-bit	32-bit	16-bit
mm	112	112	128	128
fwt1	135	150	135	150
fwt2	105	120	112	126
dwt	128	128	135	135
cp	16	35	16	35
hotspot	80	90	84	96
lavamd	80	117	81	117
pathfinder	112	112	128	130

The analytical model explores different core design options, the interconnect organization, and the total number of cores. In Table III, we show the number of cores selected by our analytical exploration, and the maximum number of cores of the same core design. For several benchmarks, we can see a gap between the number of cores selected and the maximum number of cores of the same configuration. This demonstrates that the analytical model determines that the memory bandwidth saturates, and that the benefit of additional cores does not justify the frequency degradation.

C. Improved ML-GPS Effectiveness

As mentioned earlier, we extend the original ML-GPS [7] in this paper to include our bus system and an automated bus-based system generation, improved analytical model, and a model for external bandwidth. Since we demonstrated the low area requirements for our bus-based system in Section V-A and the accuracy of our analytical model in Section V-B, next we demonstrate the effectiveness of the design space exploration of our enhanced ML-GPS. Additionally, we will also compare the performances between ML-GPS and SL-GPS [6][3]; we

[3]Single-level granularity parallelism synthesis.

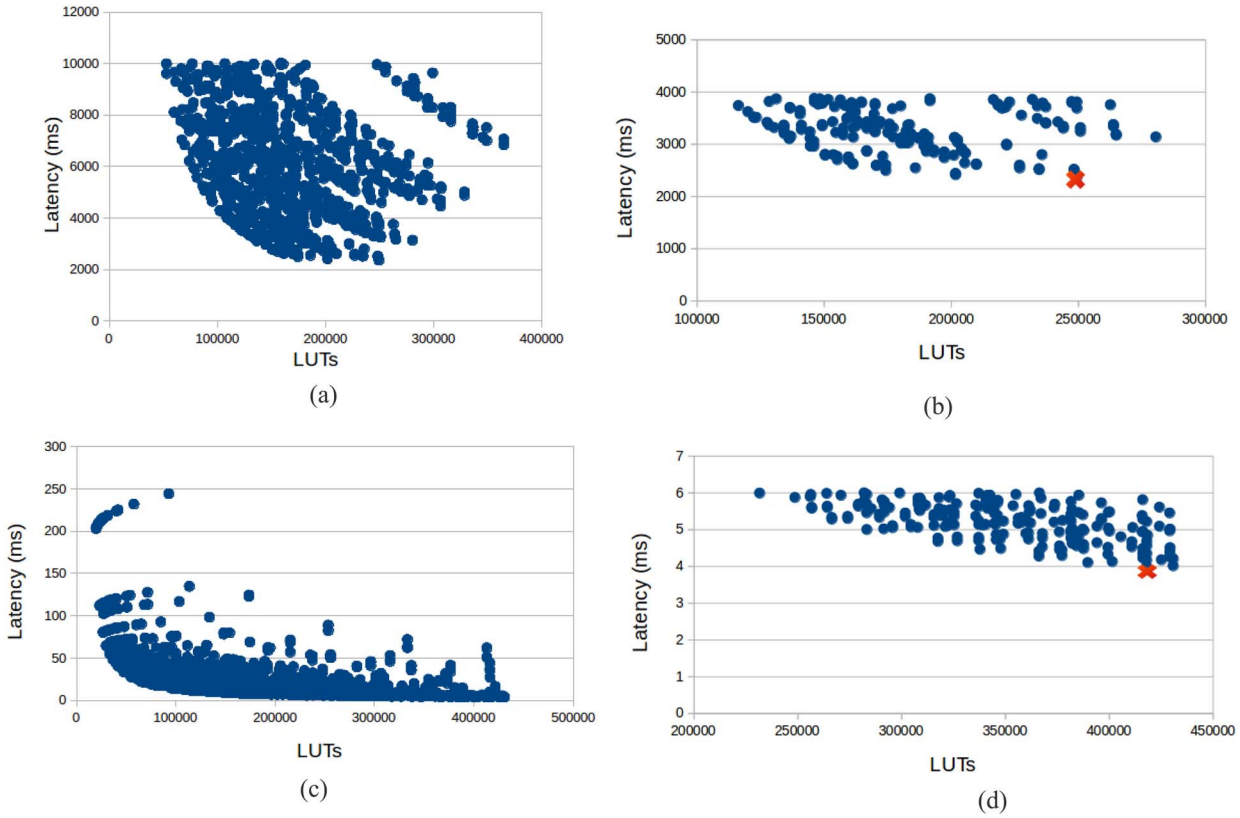


Fig. 17. Multigranularity parallelism design spaces. (a) CP design space. (b) CP design subspace. (c) FWT2 design space. (d) FWT2 design subspace.

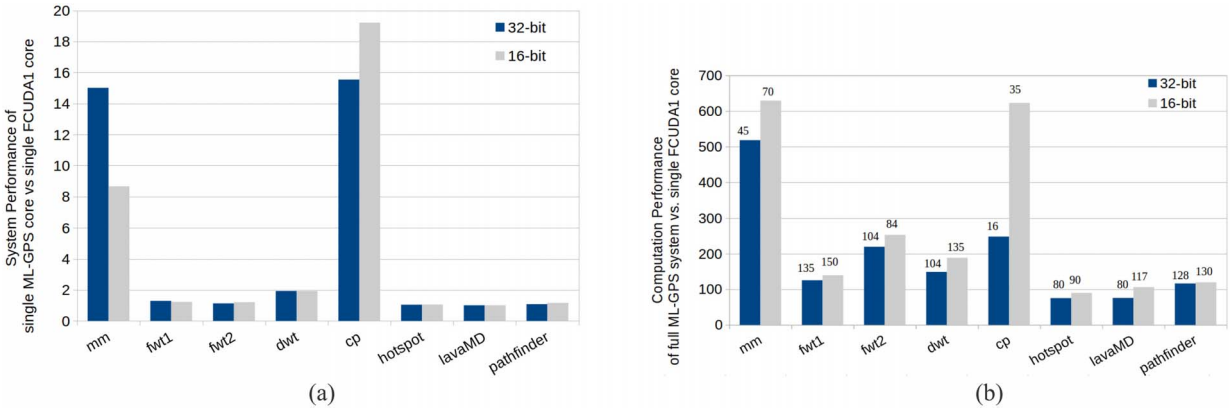


Fig. 18. ML-GPS versus SL-GPS: computation performance. Computation performance of (a) ML-GPS single core versus SL-GPS single core and (b) ML-GPS full system versus SL-GPS single core.

remind that SL-GPS only explored core and core–cluster synthesis parameters.

1) Design Space Exploration Latency: The profiling process for building the analytical model requires a few hours per benchmark; in comparison, assuming the average synthesis time per HLS setting remains the same, exhaustive exploration of all settings would take approximately two months for a single benchmark. The analytical design space exploration requires 10 min per benchmark on average, primarily for Vivado HLS invocations. The analytical exploration together with the profiling represents a substantial acceleration over evaluation of all implementation options.

2) Computation Latency: First, we evaluate the ML-GPS design space exploration core selection. Our analytical models are used to explore core implementation options and determine the best core design to use in a multi-core system implementation. In Fig. 18(a), we compare the performance of the SL-GPS core implementation to a single core of the best system-level implementation chosen by ML-GPS. Note that ML-GPS chooses a system-level design, so there may be better single-core designs not selected by ML-GPS. In several cases, the best core design achieves significant speedup over the SL-GPS design, although the selected core design is sometimes simple either because the application has little benefit from unrolling or partitioning or the benchmark is memory bound and so there is greater benefit through increased memory-level parallelism.

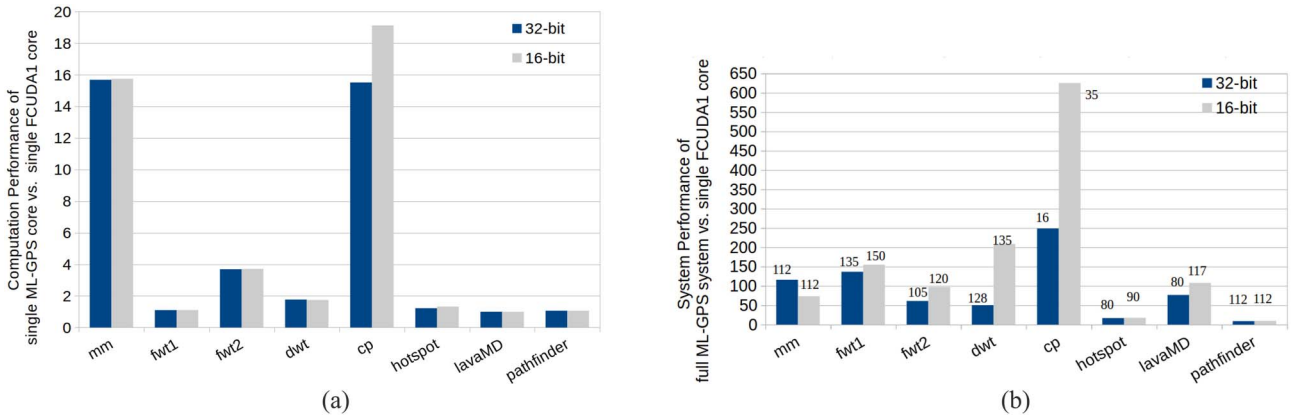


Fig. 19. ML-GPS versus SL-GPS: system performance. System performance of (a) ML-GPS single core versus SL-GPS single core and (b) ML-GPS full system versus SL-GPS single core.

In Fig. 18(b), we evaluate the computation latency of the many-core system, also normalized to the performance of a single FCUDA core. In each case, we instantiate many of the best core design and integrate with the bus-based interconnect. The speedup of many-core designs scales well, with speedup close to the product of per-core speedup and number of cores (annotated above each bar) in all cases. In benchmark FWT2, both 16-bit and 32-bit integer implementations achieve less speedup due to memory bandwidth limitations, where the 16-bit version achieves greater overall speedup with fewer cores due to improved bandwidth utilization compared to the 32-bit version. Similarly, in the cp benchmark, the 16-bit version obtains significantly more speedup than the 32-bit version due to bandwidth use.

3) System Performance: The design space exploration improves computation latency effectively, but as seen in several cases, the memory bandwidth utilization can still significantly affect achieved speedup. For this reason, overlapping computation and memory is particularly important. In this paper, we implement ping-pong buffers in our system-level implementation flow. We now evaluate how these ping-pong buffers affect system performance through overlap of memory and computation.

In Fig. 19(a), we compare the best ML-GPS single core with the FCUDA single core. With only one core, as expected, the overall speedup generally degrades compared to the evaluation of computation-only. A single core is insufficient to utilize all the bandwidth, so even with overlap, the total performance including memory latency is reduced. However, this measurement is more representative of actual performance expectations, whereas the computation-only comparison is less predictive of expected full-system performance.

Fig. 19(b), again compares full-system performance including memory latency. In many applications, we still see significant speedup, with good performance scaling relative to the number of cores. There are a few benchmarks with particularly large benefit due to ping-pong buffers including 16-bit dwt. With a small data-size and effective compute-memory overlap, dwt achieves highly efficient performance scaling with 128 cores. Other benchmarks retain nearly the same level of speedup as the compute-only evaluation despite the addition of system memory latency. Hotspot and pathfinder

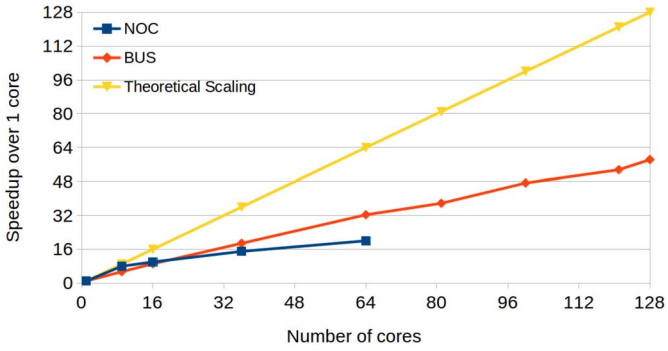


Fig. 20. Scalability of hierarchical bus versus NoC for Mat Mul(mm).

achieve particularly little overall speedup despite many cores; although system-level performance is still 17× and 9×, respectively, they are severely memory-bound and even the maximum number of cores cannot sufficiently overlap memory accesses to achieve better speedup. Despite these examples of memory-bound benchmarks, our system performance using ping-pong buffers achieves good performance scaling across the benchmarks.

D. Bus Scalability

In this paper, we select a bus-based architecture for communications instead of an NoC. In prior work [10], we implemented an NoC with FCUDA cores. However, the NoC introduces two challenges: 1) the NoC routers take significant area that could have been used for additional cores and 2) the significant routing requirements of mesh-based interconnect affects routability of the system design. In Fig. 20, we demonstrate the scalability of the bus-based architecture for the matrix multiplication application, which heavily shares data on the NoC. Although the NoC out-performs the bus with small network sizes, the bus-based communications both scales to larger total number of cores and better performance at the same number of cores due to improved achievable frequency. An NoC-based system can only instantiate up to 64-cores; larger networks are not routable despite available area. In contrast, the bus-based system scalably improves performance up to the maximum number of implemented cores. As stated earlier, the NoC results could be improved substantially through a hard-NoC [11].

VI. Future Work

In the future we plan to extend this paper to consider automated computation and communication scheduling for different cores on the bus for global optimization. For streaming applications with point-to-point connections, an efficient solution was presented in [22]. It is interesting to see how to generate such an approach to our architecture with hierarchical buses. Also, we shall consider bus topology design with physical planning as proposed in [23].

VII. Conclusion

In this paper, we present ML-GPS for automated design space exploration considering parallelism at multiple levels of granularity including internal and external memory bandwidth limitations. We created a framework to automatically generate a hierarchical and scalable AXI-based bus interconnect for FCUDA multicore designs of up to 150 cores.

We demonstrated the scalability of our bus-based interconnect compared to our prior NoC-based work, and the effectiveness of our design space exploration and automated system-level generation to create custom CUDA-to-RTL implementations. Our experimental results show that ML-GPS achieves significant, scalable system speedup with up to 150 cores, and effective memory and computation overlap through automated implementation of ping-pong buffers for all systems. The source code of our framework is available at http://dchen.ece.illinois.edu/tools.html.

References

[1] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, May 2005.

[2] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's J.*, vol. 30, no. 3, pp. 202–210, 2005.

[3] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[4] (Jul. 2015). *OpenCL 1.2 Reference Pages*. [Online]. Available: http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml, accessed Jul. 2015.

[5] NVIDIA. (2012). *CUDA C Programming Guide*. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[6] A. Papakonstantinou *et al.*, "FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs," in *Proc. SASP*, San Francisco, CA, USA, 2009, pp. 35–42.

[7] A. Papakonstantinou *et al.*, "Multilevel granularity parallelism synthesis on FPGAs," in *Proc. FCCM*, Salt Lake City, UT, USA, 2011, pp. 178–185.

[8] S. Pasricha and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. Amsterdam, The Netherlands: Morgan Kaufmann, 2010.

[9] A. H. Lam, "An analytical model of logic resource utilization for FPGA architecture development," M.S. thesis, Elect. Comput. Eng., Univ. British Columbia, Vancouver, BC, Canada, 2010.

[10] Y. Chen *et al.*, "FCUDA-NoC: A scalable and efficient network-on-chip implementation for the CUDA-to-FPGA flow," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2015.2497259

[11] M. S. Abdelfattah and V. Betz, "Design tradeoffs for hard and soft FPGA-based networks-on-chip," in *Proc. IEEE Int. Conf. Field-Program. Technol. (FPT)*, Seoul, South Korea, 2012, pp. 95–103.

[12] J. A. Stratton, S. S. Stone, and W.-M. W. Hwu, "MCUDA: An efficient implementation of CUDA kernels for multi-core CPUs," in *Languages and Compilers for Parallel Computing* (LNCS 5335). Heidelberg, Germany: Springer, 2008, pp. 16–30.

[13] Xilinx, *Vivado High-Level Synthesis*. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, accessed Jul. 2015.

[14] A. Papakonstantinou *et al.*, "Efficient compilation of CUDA kernels for high-performance computing on FPGAs," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 13, no. 2, 2013, Art. no. 25.

[15] Xilinx Inc. *Vivado High-Level Synthesis*. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/hls/index.htm, accessed Jul. 2015.

[16] T. Nguyen, S. Gurumani, K. Rupnow, and D. Chen, "FCUDA-SoC: Platform integration for field-programmable SoC with the CUDA-to-FPGA compiler," in *Proc. FPGA*, Monterey, CA, USA, 2016, pp. 5–14.

[17] Xilinx Inc. (Jul. 2012). *7 Series FPGAs Configurable Logic Block*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[18] W. Zuo *et al.*, "A polyhedral-based SystemC modeling and generation framework for effective low-power design space exploration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Austin, TX, USA, Nov. 2015, pp. 357–364.

[19] *AR# 62720, Vivado Implementation—Placer Reports Higher LUTs Utilization in 'ERROR: [Place 30-380]' Than What Is Seen in the Post-Opt Utilization Report*. [Online]. Available: http://www.xilinx.com/support/answers/62720.html, accessed Jul. 2015.

[20] NVIDIA. *CUDA Zone*. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html, accessed Jul. 2015.

[21] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Austin, TX, USA, 2009, pp. 44–54.

[22] J. Cong, M. Huang, and P. Zhang, "Combining computation and communication optimizations in system synthesis for streaming applications," in *Proc. FPGA*, Monterey, CA, USA, 2014, pp. 213–222.

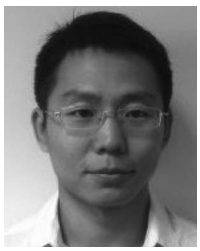
[23] J. Cong, Y. Huang, and B. Yuan, "ATree-based topology synthesis for on-chip network," in *Proc. ICCAD*, San Jose, CA, USA, 2011, pp. 651–658.

Ying Chen received the B.Eng. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2013, and the M.S. degree in computer engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2015.

She is currently an Engineer with Google, Mountain View, CA, USA.

Tan Nguyen received the B.E. degree in computer engineering from Nanyang Technological University, Singapore, in 2015.

He is currently a Software Engineer with Advanced Digital Sciences Center, Singapore. His current research interests include GPU programming, high performance computing, and reconfigurable architecture.



Yao Chen received the B.Eng. degree from Nankai University, Tianjin, China, in 2010, where he is currently pursuing the Ph.D. degree, both in electronic science and technology.

His current research interests include network-on-chip design and high level synthesis.

Swathi T. Gurumani (M'15) received the B.E. degree in electronics and communications from the University of Madras, Chennai, India, and the M.S. and Ph.D. degrees in computer engineering from the University of Alabama in Huntsville, Huntsville, AL, USA.

He is a Principal Research Engineer with Advanced Digital Sciences Center, Singapore, and a Senior Research Affiliate with Coordinated Sciences Laboratory, University of Illinois at Urbana–Champaign, Urbana, IL, USA. His current research interests include high-level synthesis, reconfigurable computing, and hardware/software co-design.

Yun Liang received the B.S. degree from Tongji University, Shanghai, China, in 2004, and the Ph.D. degree in computer science from the National University of Singapore, Singapore, in 2010.

He was a Research Scientist with Advanced Digital Science Center, University of Illinois at Urbana–Champaign, Urbana, IL, USA, from 2010 to 2012. He has been an Assistant Professor with the School of Electrical Engineering and Computer Science, Peking University, Beijing, China, since 2012. His current research interests include GPU architecture and optimization, heterogeneous computing, embedded system, and high level synthesis.

Prof. Liang was a recipient of the Best Paper Award in Field Programmable Custom Computing Machines (FCCM)'11 and the Best Paper Award nominations in CODES+ISSS'08 and DAC'12. He serves as a Technical Committee Member for Asia-South Pacific Design Automation Conference (ASPDAC), Design Automation and Test in Europe, and CASES. He is the TPC Subcommittee Chair for ASPDAC'13.

Kyle Rupnow (M'00) received the B.S. degree in computer engineering and mathematics and the M.S. and Ph.D. degrees in electrical engineering from the University of Wisconsin–Madison, Madison, WI, USA, in 2003, 2006, and 2010, respectively.

He is a Research Scientist with the Advanced Digital Sciences Center, University of Illinois at Urbana–Champaign, Urbana, IL, USA. His current research interests include high level synthesis, reconfigurable computing, and systems management of compute resources.

Dr. Rupnow has served on the program committees of FPGA, Field Programmable Custom Computing Machines, Field Programmable Technology, field programmable logic, and ReConfig, and as an Reviewer for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *Transactions on Design Automation and Embedded Systems* (TODAES), the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and *Transactions on Reconfigurable Technology and Systems*.

Jason Cong (F'00) received the B.S. degree in computer science from Peking University, Beijing, China, in 1985, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 1987 and 1990, respectively.

He is currently a Chancellor's Professor with the Computer Science Department, also with joint appointment from the Electrical Engineering Department, University of California at Los Angeles (UCLA), Los Angeles, CA, USA, the Director of Center for Domain-Specific Computing (CDSC), the Co-Director of UCLA/Peking University Joint Research Institute in Science and Engineering, and the Director of VLSI Architecture, Synthesis, and Technology (VAST) Laboratory. He served as the Chair of the UCLA Computer Science Department from 2005 to 2008. He has graduated 34 Ph.D. students. His current research interests include energy-efficient computing, customized computing for big-data applications, electronic design automation, and highly scalable algorithms. He has over 400 publications in the above areas.

Mr. Cong was a recipient of the ten best paper awards, the two 10-Year Most Influential Paper Awards [from International Conference on Computer Aided Design 14 and Asia-South Pacific Design Automation Conference 15], the 2011 ACM/IEEE A. Richard Newton Technical Impact Award in Electric Design Automation, the 2010 IEEE Circuits and System Society Technical Achievement Award for seminal contributions to electronic design automation, especially in field programmable gate array (FPGA) synthesis, very large scale integration interconnect optimization, and physical design automation, and the 2016 IEEE Computer Society Technical Achievement Award for setting the algorithmic foundations for high-level synthesis of FPGAs. He was an ACM Fellow in 2008.

Wen-Mei Hwu (F'98) received the Ph.D. degree in computer science from the University of California at Berkeley, Berkeley, CA, USA, in 1987.

He is the Walter J. (Jerry) Sanders III-Advanced Micro Devices Endowed Chair of electrical and computer engineering with the University of Illinois at Urbana–Champaign, Urbana, IL, USA. From 1997 to 1999, he was the Chairman of the Computer Engineering Program, University of Illinois at Urbana–Champaign. He is the Principal Investigator (PI) for the Petascale Blue Waters System, the Co-Director of the Intel and Microsoft funded Universal Parallel Computing Research Center, and the PI for the world's first NVIDIA CUDA Center of Excellence, Urbana–Champaign. He is the Chief Scientist of the Illinois Parallel Computing Institute and the Director of the IMPACT Laboratory. His current research interests include architecture, implementation, software for high-performance computer systems, and parallel processing.

Prof. Hwu was a recipient of the 1993 Eta Kappa Nu Outstanding Young Electrical Engineer Award, the 1994 University Scholar Award of the University of Illinois, the 1997 Eta Kappa Nu Holmes MacDonald Outstanding Teaching Award, the 1998 ACM SigArch Maurice Wilkes Award, the 1999 ACM Grace Murray Hopper Award, the 2001 Tau Beta Pi Daniel C. Drucker Eminent Faculty Award, the 2006 Most Influential ISCA Paper Award, and the University of California at Berkeley, Distinguished Alumni in Computer Science Award, for his contributions to the areas of compiler optimization and computer architecture. In 2007, he introduced a new engineering course in massively parallel processing with D. Kirk of NVIDIA. He is a fellow of ACM.

Deming Chen (S'01–M'05–SM'11) received the B.S. degree in computer science from the University of Pittsburgh, PA, USA, in 1995, and the M.S. and Ph.D. degrees in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 2001 and 2005, respectively.

He is a Professor with the ECE Department, University of Illinois at Urbana–Champaign (UIUC), Urbana, IL, USA. His current research interests include system-level and high-level synthesis, nano-systems design and nano-centric CAD techniques, GPU and reconfigurable computing, hardware security, and computational genomics.

Dr. Chen was a recipient of the Achievement Award for Excellent Teamwork from Aplus Design Technologies in 2001, the Arnold O. Beckman Research Award from UIUC in 2007, the National Science Foundation CAREER Award in 2008, the five Best Paper Awards for Asia-South Pacific Design Automation Conference (ASPDAC)'09, SASP'09, FCCM'11, SAAHPC'11, and CODES+ISSS'13, the ACM SIGDA Outstanding New Faculty Award in 2010, and the IBM Faculty Award in 2014 and 2015. He is included in the List of Teachers Ranked as Excellent in 2008. He is a Technical Committee Member for a series of conferences and symposia, including FPGA, ASPDAC, International Conference on Computer Design, International Symposium on Quality Electronics Design, DAC, International Conference on Computer Aided Design, Design Automation and Test in Europe, International Symposium on Low-Power Electronics Design, and field programmable logic. He also served as a TPC Subcommittee or the Track Chair, the Session Chair, the Panelist, the Panel Organizer, or the Moderator for some of these and other conferences. He is the General Chair for SLIP'12, the CANDE Workshop Chair in 2011, the Program Chair for PROFIT'12, and the Program Chair for FPGA'15. He is or has been an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, TODAES, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I: REGULAR PAPERS, *Journal of Circuits Systems and Computers*, and *Journal of Low Power Electronics*. He is the Donald Biggar Willett Faculty Scholar.